JMMC-MAN-2720-0001

Revision 3.0.8

Date: 03/12/2012

JMMC

AMBER Data Reduction Software

User Manual

Authors:
M. Benisty (IPAG/JMMC), F. Malbet (IPAG), F. Millour (OCA), O. Absil (IAGL), G. Duvert (IPAG/JMMC)
Contact author: Miriam Benisty <jmmc-user-support@ujf-grenoble.fr>

Author: M. Benisty, F. Malbet, F. Millour, O. Absil, G Duvert Institute: JMMC	Signature: Date:
Approved by: G. Duvert Institute: JMMC	Signature: Date:
Released by: G. Mella Institute: JMMC	Signature: Date:

Change record

Revision	Date	Authors	Sections/Pages affected
	Remarks		
0.1	01/03/2007	E. Altariba	all
	First draft		
0.2	26/06/2007	O. Absil, F. Malbet	Sects. 2-4
	Major update		
0.3	29/06/2007	O. Absil, F. Millour	all
	Corrected for beta version release of `amdlib v2`		
1.0	30/06/2007	G. Zins	
	Released		
1.1	02/10/2007	O. Absil, F. Millour, F. Malbet	all
	Updated for the beta 2 release of `amdlib v2`		
2.0	17/10/2007	G. Zins	
	Released		
2.1	07/07/2008	E. Altariba	
	Documentation update according to the new filechooser		
3.0b	19/07/2010	F. Malbet	all
	Updated for `amdlib v3`		
3.0	21/07/2010	G. Mella	all
	Released		
3.0.1	29/07/2010	G. Duvert	all
	Revised for `amdlib v3`, highlighting obsolete sections.		
3.0.3	22/06/2011	F. Malbet	all
	Corrected the description of selection criteria: flux which is not flux ratio and piston in microns.		
3.0.5	03/12/2012	M. Benisty	New Sect. 4.2.4
	Add new paragraph on spectral shift.		
3.0.6	5/03/2013	G. Mella	4.3
	Replace amdlibSearchAllDiameters by amdlibSearchAllStarDiameters.		
3.0.7	20/09/2013	G. Mella	
	Ask the user to download alfara data for the demo script. Replace www-laog urls by jmmc ones.		
3.0.8	14/11/2015	G. Mella	front page
	fix user-support contact email.		

Table of contents

1 Introduction **4**
1.1 Scope of the document . . . 4
1.2 Reference documents . . . 4
1.3 Abbreviations and acronyms . . . 5

2 General presentation **6**
2.1 Instrument overview . . . 6
2.2 `amdlib` : the AMBER data reduction software . . . 6
2.3 File management . . . 6
2.4 AMBER raw data . . . 7
2.5 Refining raw data . . . 8
2.6 The Pixel-to-Visibility Matrix (P2VM) . . . 9
2.7 From science data to visibilities . . . 10
2.8 On the AMBER Wavelength Calibration . . . 12
2.9 Improvements in version later than 3.0 . . . 12
2.10 Frame selection . . . 13
2.11 Visibility calibration . . . 13

3 `amdlib` and its `Yorick` plugin **14**
3.1 Installation . . . 14
3.2 Data reduction products . . . 15
3.3 Graphical tools of the `Yorick` plugin . . . 15
3.4 `amdlib` C functions . . . 16
3.5 Troubleshooting . . . 17

4 Cookbook **18**
4.1 Recommended fast data reduction . . . 18
4.2 Going steps by steps into the data reduction process . . . 20
4.2.1 Setting the preferences . . . 20
4.2.2 Loading the Bad Pixel and Flat Field Maps . . . 21
4.2.3 Calibrating the instrument (P2VM computation) . . . 21
4.2.4 On the spectral shifts . . . 22
4.2.5 Checking the data quality . . . 24
4.2.6 Producing the raw OI data . . . 24
4.2.7 Producing averaged OI data . . . 25
4.2.8 Reloading the data . . . 28
4.3 Calibrating OI data (experimental) . . . 28

A Frequently Asked Questions **30**

B Calibrating the visibilities: tips and tricks **31**

1 Introduction

1.1 Scope of the document

The AMBER instrument provides a way to measure the coherence of light collected by a triplet of telescopes of the *Very Large Telescope (VLT)* for various astronomical targets. The raw data collected must be processed to have full access to the useful scientific information. This document has been written to give a general overview of the data reduction with AMBER (Sect. 2), to present a description of the actual software implementation and of its installation (Sect. 3) and to provide a *cookbook* for users who wants to start with an actual example (Sect. 4). Finally Appendix A answers some frequently asked questions (FAQ).

The AMBER Data Reduction Software known as `amdlib` is based on a core library initially developed the AMBER consortium (P.I. G. Duvert, see [1, 2, 3]). It is now maintained and augmented by the Jean-Marie Mariotti Center (JMMC), retaining most of the initial contributors.

The features which are new in `amdlib v3` are presented in special boxes.

Obsolete features relevant to `amdlib v2` are written in medium gray.

1.2 Reference documents

[1] VLT-SPE-AMB-15830-6004, Issue 2.0, 2002, AMBER Data Reduction Software Requirements, G. Duvert

[2] VLT-TRE-AMB-15830-4107, Issue 1, 2006, Amber Data Reduction Software Design Description, E. Altariba

[3] VLT-SPE-AMB-15830-3984, Revision 3, September 07, 2007; AMBER Product Data Specification, G. Zins

[4] VLT-SPE-ESO-15000-2764, 1.0, 3 June 2002, VLTI Data Interface Control Document, P. Ballester

[5] "A Data Exchange Standard for Optical (Visible/IR) Interferometry", Pauls et al. 2005, PASP 117, 1255.

[6] JMMC-MAN-2720-0002, Revision 2.0, 21/07/2010; AMBER Data Reduction Software Installation Guide, G. Mella

[7] "AMBER, the near-infrared spectro-interferometric three-telescope VLTI instrument", Petrov, Malbet, Weigelt et al. 2007, A&A 464, 1.

[8] "Interferometric data reduction with AMBER/VLTI. Principle, estimators, and illustration", Tatulli, Millour, Chelli et al. 2007, A&A 464, 29.

[9] "Optimised data reduction for the AMBER/VLTI instrument", Chelli, Utrera, Duvert 2009, A&A 502, 705.

[10] "AMBER Task Force February 2008 run report", Malbet, Duvert, Chelli, Kern 2008, ESO Doc No. VLT-TRE-AMB-15830-7120, issue 1.2, dated 16/04/2008, (arXiv:0808.1315).

[11] Building the 'JMMC bright star diameter catalog' using the SearchCal VO service, Lafrasse et al. 2010, SPIE 7734, in press. Accessible on VizieR under the reference II/300/jsdc.

[12] SearchCal: a Virtual Observatory tool for searching calibrators in optical long baseline interferometry I: The bright object case, Bonneau et al. 2006, A&A 4469.

[13] A catalog of bright calibrator stars for 200-m baseline near-infrared stellar interferometry, Mérand et al. 2005, A&A 433, 1155.

[14] CHARM2: An updated Catalog of High Angular Resolution Measurements, Richichi et al. 2005, A&A 431, 773.

1.3 Abbreviations and acronyms

AMBER	Astronomical Multi-BEam Recombiner
ATF	AMBER Task Force
BPM	Bad Pixels Map
FFM	Flat Field Map
FITS	Flexible Image Transport System
GUI	Graphical User Interface
JMMC	Jean-Marie Mariotti Center
OI-FITS	Optical Interferometry FITS
P2VM	Pixel-To-Visibility Matrix
PBM	Pixel Bias Map

2 General presentation

2.1 Instrument overview

AMBER, the near-infrared focal instrument of the VLTI, operates in the J, H, and K bands (i.e. 1.0 to 2.5 microns), with spectral resolutions of about 35, 1500 and 10000; the resolution can be changed by selecting either a prism (LR mode, $\mathcal{R} \sim 35$), a medium resolution grating (MR mode, $\mathcal{R} \sim 1500$), or a high resolution grating (HR mode, $\mathcal{R} \sim 10000$). The instrument operates with three input beams[1] and provides measurements of spectrally dispersed interferometric observables of science targets on baselines up to 200 m: square visibility (V^2), differential phase (φ_λ) and phase closures (ϕ_{123}).

2.2 `amdlib` : the AMBER data reduction software

The AMBER Data Reduction Software (called `amdlib`) consists of a *core library* of C functions plus a high-level interface in the form of a `Yorick`[2] plugin. The `amdlib` C functions are used at all stages of AMBER data acquisition and reduction: in the observation software for wavelength calibration and fringe acquisition, in the (quasi) real time display program used during the observations, in the online data processing pipeline customary for ESO instruments, and in various offline front end applications, noticeably the `Yorick` implementation. The `amdlib` library is meant to incorporate all the expertise on AMBER data reduction and calibration acquired throughout the life of the instrument, which are bound to evolve with time.

The amdlib library provides a command-line interface, but we offer it behind a `Yorick`-based GUI interface to make it easier for the user to manipulate and produce plots of the different observables, to compute interferometric observables and/or to perform specific processing on these data in order to check the data quality (like frame selection). Note that the names of the `Yorick` functions are generally copied and pasted from the names of the C routines they interface.

The `amdlib` core library, which is responsible for computing the instantaneous correlated fluxes and all the basic interferometric observables (squared visibilities, differential phases and phase closures), has been completely rewritten between versions 2 and 3. It implements most of the algorithms published by Chelli et al. (2009) [9] as well as workarounds of some of the problems audited by the "ATF team" [10]. The `Yorick` interface has been improved to make the data reduction as automatic as possible.

2.3 File management

In the directories provided in ESO DVDs, file names are not explicit enough to easily and immediately understand what kind of data they contain (calibration, observation...). The ESO utility, `gasgano`[3], or `amdlib` can list the properties of those files. The useful files for data reduction are those with the following ESO observation type:

- SKY, DARK for sky or dark file;
- WAVE,3TEL for internal calibration of the offsets of the photometric channels;
- 3P2V for internal interferometric calibration, leading to the P2VM computation;
- OBJECT for actual observation files either on the calibrator star (CALIB) or the science object (SCIENCE).

[1] AMBER can be operated with 2 beams only, however this degraded mode is not covered specifically in this document, since `amdlib v3` deals transparently with such data, of which only phase closures are absent.

[2] `Yorick` is a high-level language environment similar to IDL or Matlab which is public domain. More info can be found at http://yorick.sourceforge.net

[3] http://www.eso.org/sci/data-processing/software/gasgano/

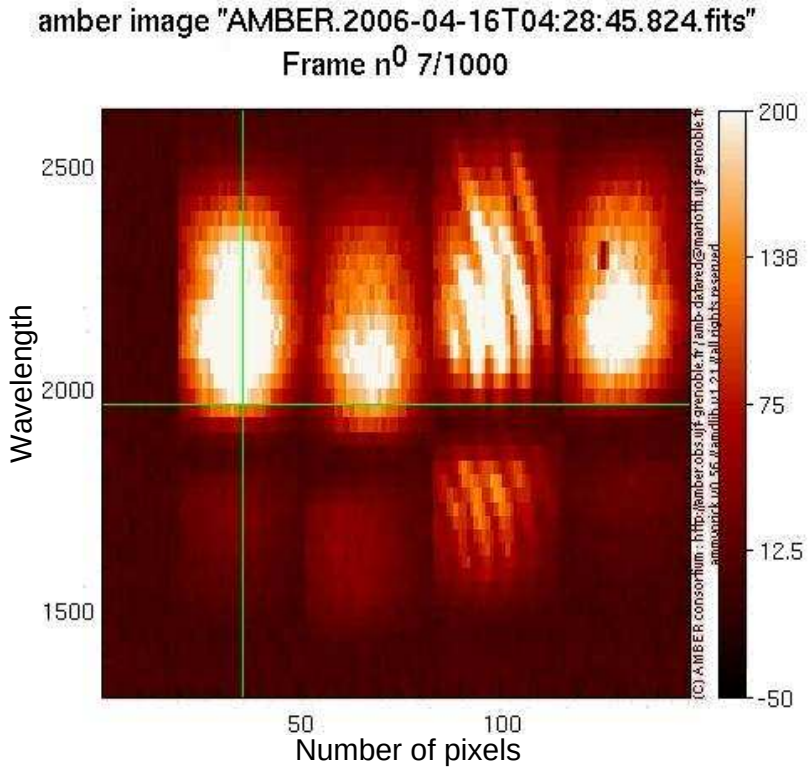

Figure 1: Raw detector image from AMBER in its low-resolution mode [7]. The stellar signal is spectrally dispersed in the vertical direction and each detector line contains a spectral channel. The rows of K and H bands occupy respectively the upper and the lower half of the screen. From left to right, the first, second, and fourth columns represent the photometric beams for the each one of the three telescopes, while the third column contains the interferometric signal, with three superimposed fringe systems.

A flat field map (FFM) and a bad pixel map (BPM) file are also required[4]. The `Yorick` user interface uses a specific automatic logging facility to distinguish the different input files in the directory. This *autolog* can be manually created by using the `amdlibCreateLog` function, but it is usually automatically generated. The automatic log file lists all files belonging to the specified directory identifying for each file name any useful information such as the category (CALIB or SCIENCE), the data type (e.g., OBJECT or 3P2V, ...), etc. This autolog file is also useful to colorize file names in the browser.

2.4 AMBER raw data

The data obtained with AMBER ("raw data") consist of an exposure, i.e. a time series of frames read on the infrared camera, plus all relevant information from AMBER sensors, on the observed object, and about the VLTI and AMBER setup, all of which are stored in FITS ([4]) table format. The detector in the AMBER camera is a cooled HAWAII-I detector, where only one 512×512 pixel quadrant is used. The vertical dimension is aligned with the wavelength direction. The first 20 pixels of each line of the detector are masked and never receive any light, allowing the observation software (**not** the data reduction software) to estimate the readout noise and bias of the exposure. The beams from the 3 telescopes are imaged in 4 panels arranged horizontally. 3 of them are the "photometric" channels, and the last one is the "interferometric"

[4]These files are recalibrated about once a year. They are publicly available at ESO at `http://www.eso.org/observing/dfo/quality/AMBER/qc/qc1.html`. Note that we discourage the use of the ESO Flat field maps. See the comments about Flats and `amdlib v3` in the JMMC release notes `http://www.jmmc.fr/twiki/bin/view/Jmmc/Software/AmberDrsReleaseNotes`. In any case, the same flat file should be used for all the calibration an datat reduction steps.

sub-image where the interference fringes are located. All sub-images are dispersed and spread along the vertical direction. See Fig. 1 for an actual example of an AMBER detector image.

The detector is read in sub-windows. The Raw Data format used by AMBER records individually these sub-windows. Horizontally, these sub-windows are centered on the regions where the beams are imaged, with a typical width of 32 pixels. Vertically, the detector can be set up to read up to three sub-windows (covering up to three different wavelength ranges and referred to as "rows"). These wavelength ranges are defined by the astronomer at the time of the observations.

Saving the raw uncalibrated data, although space-consuming, allow us to benefit afterward, by replaying the calibration sequences and the data reduction anew with all the improvements that can have been deposited in `amdlib` in the meantime, noticeably those defined in [9] .

2.5 Refining raw data

The first step of the raw data processing consists in applying *cosmetics*, i.e. correcting them for detector effects like bad pixels or pixel gains. All defective pixels (called bad pixels) are marked in a so-called Bad Pixel Map (BPM) available at ESO[5]. In the AMBER case, only two types of pixels exist: good and bad pixels (i.e., a pixel is either useful or not and nothing in between).Then all frames pixels are tagged valid if not present in the currently available BPM of the AMBER detector.

`amdlib v3` can update the list of bad pixels defined in the BPM using statistics on DARK observations

Pixels values are then converted into photo-event counts. This step requires, for each frame, to model precisely the spatially and temporarily variable bias added by the electronics. The detector exhibits a pixel-to-pixel (high frequency) bias whose pattern is constant in time but depends on the detector integration time (DIT) and on the size and location of the sub-windows read on the detector. Thus, after each change in the detector setup, a pixel bias map (PBM) is measured prior to the observations by averaging a large number of frames acquired with the detector facing a shutter. This PBM is then removed from all frames prior to any other treatment.

Once this fixed pattern has been removed, the detector may still be affected by a time-variable line bias, i.e., a variable offset for each detector line. This bias is estimated for each scan line and each frame as the mean value of the corresponding line of masked pixels, and subtracted from the rest of the line of pixels. Pixels are then converted into photo-event counts by multiplying by the pixel gain. Currently the map of the pixel gains used is simply a constant value multiplied by a flat field map acquired during laboratory detector calibrations. Finally, the rms of the values in the masked pixel set, that were calibrated as the rest of the detector, gives the frame's detector noise.

Infrared cameras like the AMBER one are far from perfect photon-counters. Depending on the readout mode used, the camera may present a large pixel-dependent bias that must be removed, and it is even necessary to drop a fair number of frames at the beginning of an exposure. `amdlib v3` computes on a pixel-by-pixel basis the readout noise on the DARK or SKY frames (these are equivalent for this purpose), and removes the DARK or SKY mean value from the data, thus removing both the bias, the thermal emission and, eventually, the sky glow. Pixels are then converted into photo-event counts by multiplying by the pixel gain. Currently the map of the pixel gains used is simply a constant value determined during laboratory detector calibrations (eventually multiplied by a flat).

Once the cosmetics on the pixels is performed, `amdlib` corrects the data from the spatial distortions present in the image. Presently, the only effect corrected is the displacement of the spectra acquired in the *photometric channels* with regards to the fringed spectrum in the *interferometric channel*. These offsets of a few pixels in the spectral dispersion direction, is due to geometrical effects in the spectrograph. Correcting

[5] http://www.eso.org/observing/dfo/quality/AMBER/qc/qc1.html

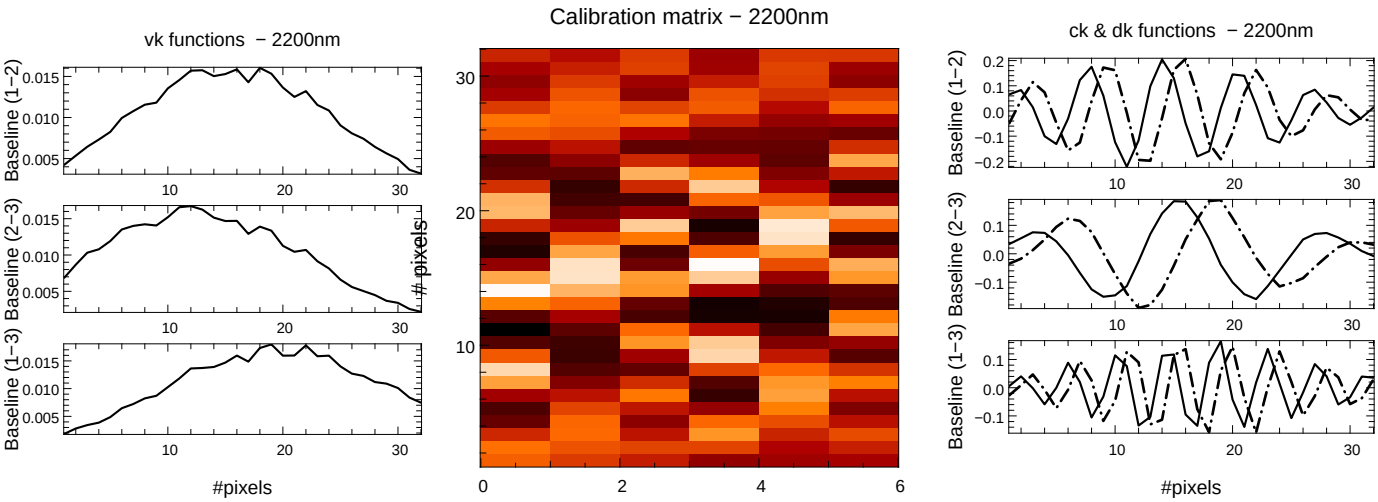

Figure 2: Outputs of the calibration procedures. Left: the photometry-normalized interferometry shape functions. Middle: the matrix containing the carrying waves; the first three columns are the real part and the three last columns are the respective imaginary part. Right: another representation of the carrying waves. From top to bottom, both sinusoidal functions correspond to columns 1-4, 2-5, and 3-6 of the calibration matrix.

from this is mandatory to ensure an exact correspondence between the wavelengths in the interferometric channel with the wavelengths in the photometric ones, especially for Medium or High resolution data exhibiting strong absorption or emission features.

The ATF found that these displacements are fixed as long as the cryostat is not opened or the hardware positioning of the slit mask not changed. They must be re-calibrated after each opening of the cryostat or changes in the hardware positioning of the entrance slit. The data for the different periods are available in [10] or on the AMBER website `http://amber.obs.ujf-grenoble.fr/spip.php?article176`. The effect of this displacement is corrected by `amdlib` using the *spectral calibration offsets* and is done automatically in the `Yorick` interface of `amdlib v3`.

Finally, each frame is converted to the more handy *science data* structure, that contains only the calibrated image of the *interferometric channel* and (up to) three 1D vectors, corresponding to the instantaneous photometry of each beam, corrected from the above mentioned spectral displacement.

2.6 The Pixel-to-Visibility Matrix (P2VM)

The expected outputs of the data reduction process are complex coherent fluxes or a combination of them. The coherent flux are nothing else than the amplitude of a fringe system which is fully characterized by the instrumental system. These fringe systems for each pair of beams, normalized to the unit energy, are called *carrying waves*. The raw data can be decomposed on these waves: we obtain an amplitude and a phase which are by definition the complex coherent flux. Therefore the complex coherent fluxes are related to the raw data by a relatively simple linear relationship which can be addressed by a matrix. This matrix, called the Pixel-to-Visibility Matrix or P2VM in brief, contains the values of all instrumental parameters and requires to be calibrated which is done during the P2VM calibration phase.

The function which computes the P2VM has to process 5 or 10 relevant calibration files depending on the number of telescopes used. It applies on each of them the detector calibration, image alignment and conversion to science data described above, and finally computing all the elements of the P2VM matrix described above. The result is stored in a FITS file ([3]), called the P2VM file.

The P2VM matrix (see Fig. 2) is the most important set of calibration values needed to retrieve visibilities. The shape of the carrying waves and into a less extent the coefficients for photometry correction, are

the imprints of all the changes in intensity and phase that the beams suffer between the exit of each fiber and the plane of detection in the infrared camera. Any change in the AMBER optics located in this zone, either by moving a grating or just thermal long-term effects, destroys the P2VM calibration and renders the P2VM unusable.

Thus, the P2VM matrix must be re-calibrated each time one calls a new spectral setup that involves changing the optical path after the fibers output. All the instrument observing strategies and operation are governed by the need to avoid unnecessary optical changes, and care is taken at the operating system level to insure a recalibration of the P2VM whenever a critical motor affecting the optical path is set in action.

To satisfy these needs, the P2VM computation has been made mandatory prior to science observations, and is given an unique ID number. All the science data files produced after the P2VM file inherit of this ID, that associates them with their governing calibration matrix. The `amdlib` library takes the opportunity that the P2VM file is pivotal to the data reduction, and unique, to make it a placeholder of all the other calibration tables needed to reduce the science data, namely the spectral calibration, bad pixels and flat field tables. In some situation it is required to use a P2VM which has been recorded afterward and this is possible with the software by forcing the use of the P2VM.

A typical data reduction process first takes care of all raw data files related to the calibration procedures performed before acquiring the science data, thus performing spectral calibration, then the P2VM file computation. Fig. 3 summarizes the inputs needed for the P2VM computation. One can see two sets of raw data are used. Necessary ones are characterized by the 3P2V observation type keyword. Other represented raw data, with WAVE,3TEL observation type keyword are used if one wants to perform the calibration of spectral internal offsets.

2.7 From science data to visibilities

Once the P2VM file is computed, it contains all the internal calibration quantities needed to process science object observations and to extract the visibilities from the raw data:

1. extract raw visibilities;
2. correct for biases, compute debiased V^2 visibilities;
3. compute phase closures;
4. compute cross spectra;
5. calculate piston values by fitting cross spectra;
6. write the OI-FITS output file.

The visibility computation program is able to perform visibility estimates on a frame-by-frame basis, or over groups of frames, called bins, or over a selection of frames based on photometry or fringe detection criteria. It estimates the instantaneous atmospheric piston from two different methods as well as the photon noise used in debiasing the V^2 estimator either by the variance of the temporal series of instantaneous photometries or as the square root of the instantaneous photometry.

Figure 4 summarizes the inputs needed for the interferometric observables computation. Main input file is an observation raw data file. Bad pixels, flat field maps and P2VM file are mandatory, while dark and/or sky files are not. Computations can be done frame by frame or on groups of frames called *bins*, the size of each (i.e. the number of frames in each group) being given by the user.

It must be stressed that, contrary to `amdlib v2`, `amdlib v3` uses prolifically the FLAG columns of the OI-FITS files.

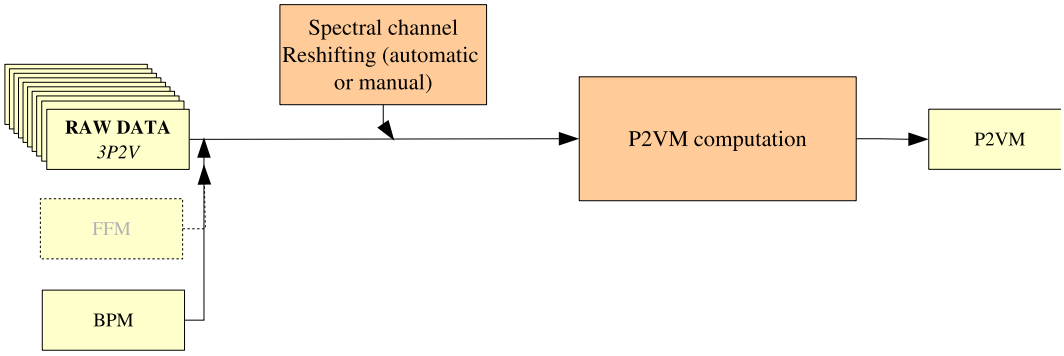

Figure 3: P2VM computation work flow

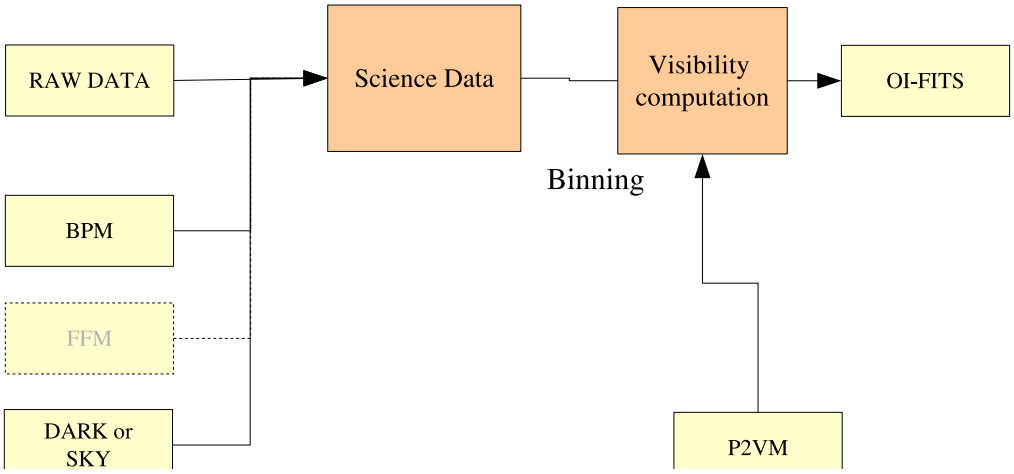

Figure 4: Interferometric observables computation work flow

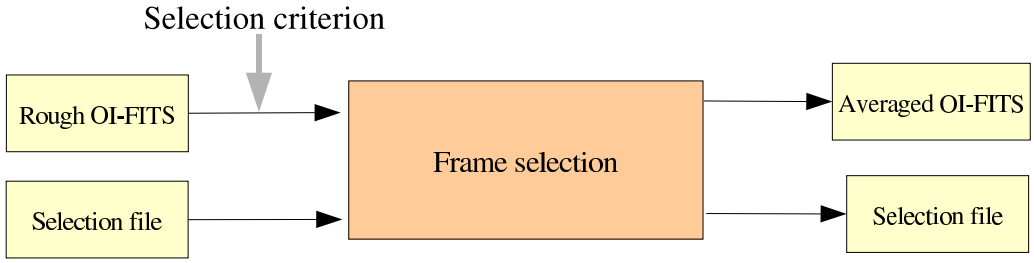

Figure 5: Frame selection

2.8 On the AMBER Wavelength Calibration

There is no wavelength calibration in `amdlib`. The wavelength values used by `amdlib` are those given by the instrument at the time of observations. The dispersion coefficients of the 3 dispersive elements (2 gratings and a prism) does not seem to be precisely known and repositioning errors in the disperser turret can lead to wavelength displacements of up to ~ 10 pixels.

For Medium or High resolution observations, it is advisable to calibrate the wavelengths of data after each new P2VM, using known positions of, e.g., atmospheric lines. Normally this verification should show that there is only a displacement of the true spectrum wrt. the wavelength stored in the FITS files. In this case, `amdlib v3` through the `-S` argument of the `amdlibComputeP2vm` command, can shift the wavelength table accordingly. If the dispersion itself is found wrong, this is a hardware problem that should be reported to ESO. The only solution is to update manually the values of the wavelength table of *the first file of the P2VM raw file set*, values that will be transmitted to all data reduction that uses this P2VM.

For Low Resolution observations, the wavelength dispersion given by the instrument is known to be quite wrong, and we are waiting for the results of a new dispersion experiment, which will be automatically updated by amdlib (contrary to above) and will benefit retroactively all data. In the meantime `amdlib v3` will try to find the position of the H-K discontinuity and automatically recenter the spectrum, keeping the current wavelength list.

2.9 Improvements in version later than 3.0

Starting from version 3, the `amdlib v3` library implements the new algorithms based on an improved data and noise model [9]. In a nutshell, the previous version of `amdlib v2` based upon theoretical assumptions on the instrument gave biased visibilities especially at low flux due to:

- an incomplete data model overlooking both the presence of stray light and optical ghosts in the spectrograph and non-linearity effects in the detector at very low fluxes,
- a too simplistic noise model. Both effects were enhanced at low S/N, where noise estimate and stray light level was becoming dominant.

Additional improvements are also added to the core library:

- The wavelength displacement between the three photometric beams is automatically taken into account (see Sect. 2.2).
- In low resolution mode, the algorithm detects the H-K interband phase shift and compensate the defects of repositioning of the spectrograph prism by displacing the wavelength table of the amount needed to bring the H-K interband to its nominal position.
- `amdlib v3` provides on-the-fly bad pixel detection.
- `amdlib v3` uses a refined algorithm to compute pistons, and an heuristic scheme to evaluate this piston "goodness of fit". Also, we added a piston closure algorithm to improve piston estimate.
- `amdlib v3` uses a goodness of fit test to tag individual visibilities which are not well fitted by the carrying waves of the interferogram.
- similarly, the program tags all visibilities where one of the photometries is below a used-defined value (0 being the default).
- Finally, all bad values of the instantaneous or averaged interferometric observables are tagged in the OIFITS file using the FLAG columns.

2.10 Frame selection

Figure 5 summarizes the inputs needed for the selection of "good" frames. A frame selection consists in selecting a set of "good" frames according to a criterion. This function allows frames to be selected in two ways:

- by defining a predefined selection criterion (flux, fringe contrast SNR or piston excursion), its type (percentage or threshold) and its associated threshold as input parameters;
- by using a customized selection file as an input parameter. The format of this table is described in [3], and further discussed in Section 4.2.7.

Multiple selection criteria can be "chained", either directly using the `Yorick` frame selection routine, or by using an input selection file in addition to a user-specified selection criterion. One can save the selection table obtained, and the result averaged OI-FITS file.

Note that one should be very careful when performing frame selection since this procedure can bias the final results.

2.11 Visibility calibration

All the processing steps described before is aimed at removing most of the instrumental effect on the data, but the effect of the atmosphere still remains. In order to cope with the atmosphere systematics, one of them being seeing variation, it is required to calibrated the observed visibility with those of a star of known diameter. The same procedures should be used on the target and on the calibrator.

The proper calibration procedure to obtain calibrated interferometric quantities is not yet supported in `amdlib v3` but Annex B gives some hints how to perform such final calibration. Preliminary routines to perform such a calibration is provided in the `amdlib` package (see Sect. 4.3), but should be used at one's own risks.

3 amdlib and its Yorick plugin

amdlib is made of:

1. A library of C functions and command-line tools ("core amdlib") that can be used alone to reduce data, or can be called by an external application or shell scripts;
2. A more user-friendly interface, written in Yorick language[6], to ease the use of the core amdlib.
3. Sets of Yorick scripts to go beyond the possibilities allowed by the core amdlib alone:
   - Visualization of raw, reduced and calibration data
   - Automated reduction of whole observing nights
   - Absolute calibration using, e.g., night transfer function
   - File browsers, logs, etc...

The advantage of the Yorick plugin compared to the shell commands is the use of a specific AMBER file browser and the access to display functions that allows the data quality to be plotted and the data to be examined in an interactive way. The Yorick plugin provides also a direct access to the results by means of Yorick variables allowing them to be manipulated.

3.1 Installation

To install Yorick and the amdlib plugin, please refer to the specific installation guide [6]. When starting the amdlib command, the terminal should be like:

```
gag9143:alfara$$ amdlib

amdlib: Starting rlwrap yorick -i amdlib...

 Copyright (c) 2005.  The Regents of the University of California.
 All rights reserved.  Yorick 2.1.06 ready.  For help type 'help'
 cfitsioPlugin plugin loaded
 yoco - IPAG Yorick Contributions
 yoco package loaded
 yocoPlugin plugin loaded
 oiFitsUtils package loaded
 amdlib plugin loaded

 --==< amdlib, version 3.0 >==--
 Type "amdlib" if you want some help

 \\\///////////////////////////////////////////////////////////////////////////
 ---                                                                        ---
 ---       PLEASE ACKNOWLEDGE THE USE OF AMBER DATA                         ---
 ---          REDUCTION SOFTWARE FROM THE JMMC,                             ---
 ---            FOR EACH USE IN A PUBLICATION.                              ---
 ---      http://www.jmmc.fr/amberdrs/acknowledgement                       ---
 ---                                                                        ---
 ///\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

>
```

[6] Other high level packages like IDL or matlab can also be used but no interfaces has been developed on these platforms.

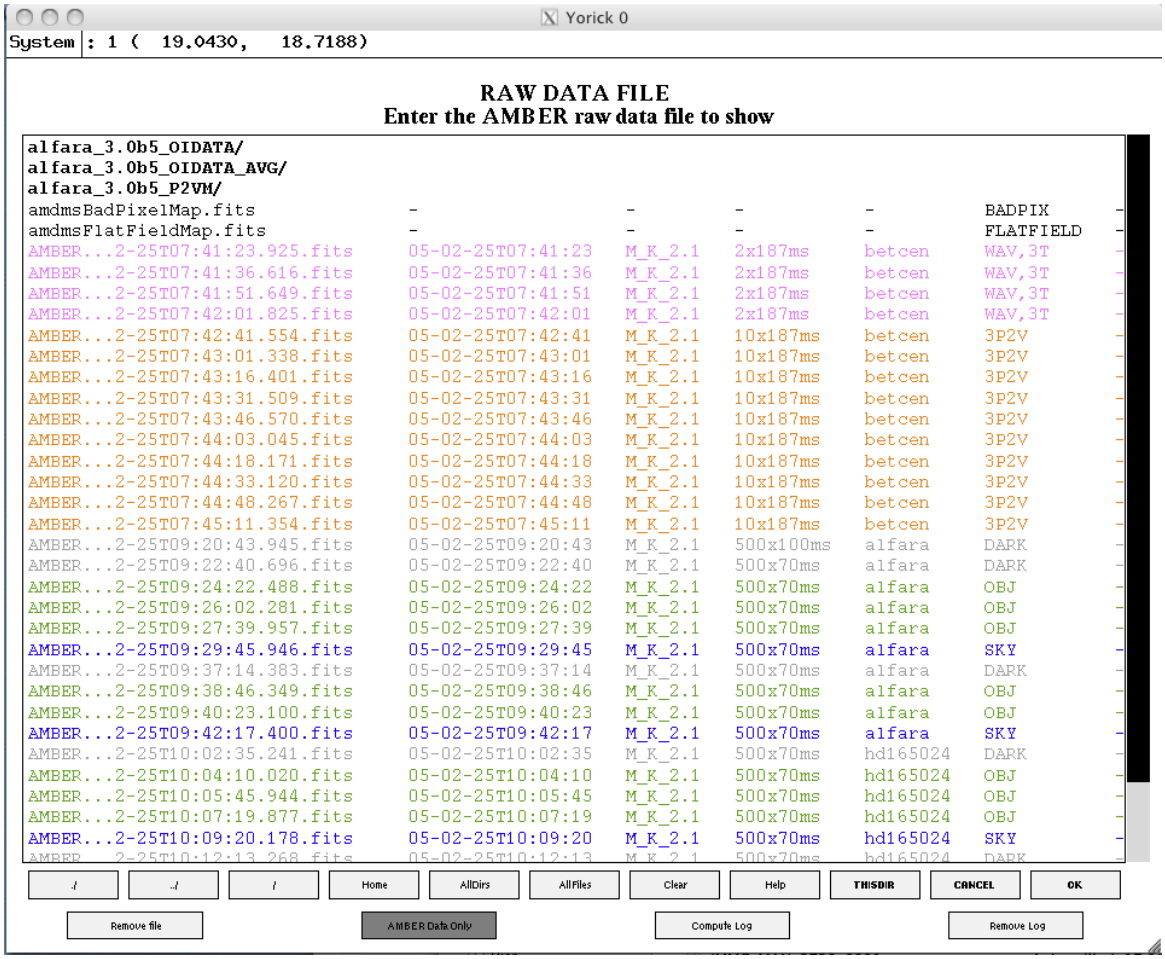

Figure 6: `Yorick` plugin file chooser

3.2 Data reduction products

The raw data files are stored in FITS format [4]. There are two types of data reduction output products provided by `amdlib` :

- the P2VM files which contains most instrumental calibration (spectral and interferometric);
- the OI-FITS files that store all interferometric and spectrophotometric observables.

The format of these two products are described in more details in [3].

3.3 Graphical tools of the `Yorick` plugin

The functions `amdlibShowRawData`, `amdlibShowP2VM` and `amdlibShowOiData` are graphical tools. They display information respectively on raw data, P2VM, and interferometric observables.

Any time that no file and/or directory is given in the input parameters, a file chooser is opening (see Fig. 6). Directories can be browsed and files or directories can be selected. The file chooser displays all files and directories located in the current directory (which name is written at the top of the window) with the keywords of interest.

At the bottom of the window, there are two rows of buttons. The first 4 left buttons on the top line allow us to move to specific directories: the current directory (./), the parent directory (../), the root directory (/) or the user's home directory (home). The next 3 buttons on the same line allow us to select all directories

or all files in the current directory. A single-click on a file (un)selects the file, shift-mouse (un)selects several files, control-click (un)selects only the clicked file, etc... The "clear" button allows to unselect all files. The last 3 buttons of the top line which are displayed in bold face are select actions. The button "THISDIR" is to select the current directory and no files in it, whereas "OK" selects the selected files.

The last row of buttons are other actions. The left one is used to remove selected files (to be used with care), the second one is a filter and displays only the data important in the AMBER data reduction, the last two buttons refresh the log file which is at the base of the browser and remove it.

The browser needs a log file in order to display the right information. If the information about the files are missing, please click on the "Compute Log" button: it will compute a file whose name is `NameOfDirectory_LOG.txt`.

3.4 `amdlib` C functions

The `amdlib` C library allows the user to compute directly interferometric observables, but also to decompose each step of the data reduction flux. The C functions can be called directly from a Unix shell but it is advised to use the `Yorick` user-friendly interface introduced in Section 4. In general the `Yorick` plugin commands bear a resemblance with the C "command-line" functions. However when the command-line functions use traditional shell options (i.e., one-letter preceded by `‘‘-’’` like `‘‘-S’’`), the `Yorick` plugin sets or unsets values in preferences using the `‘‘amdlibSetPreference’’` command. The list of shell `amdlib` function is:

```
amdlibComputeP2vm           amdlibAppendOiData  amdlibComputeSpectralCalibration
amdlibPerformFrameSelection amdlibAppendOiFits  amdlibBtbl2Fits
amdlibCalibrateRawData      amdlibMergeP2vm     amdlibSelectFrames
amdlibComputeOiData         amdlibP2vm2OiData
```

Sending the command without arguments gives the usage of the function.

Computing the P2VM

The function `amdlibComputeP2vm` computes the P2VM for the 2 or 3 telescope configuration, depending of the number of input files provided. The resulting P2VM is saved into the file given as parameter. Note that the possibility is given to re-do the spectral calibration at the beginning of the run. The `amdlibComputeAllP2VM` function, which is only available in the `Yorick` interface, computes all P2VMs which are possible in a specific directory. Automatic spectral offset calibration is performed in this program.

Computing the interferometric observables

The function `amdlibComputeOiData` computes the interferometric observables corresponding to an input file using a given P2VM, and provide as a result one OI-FITS file. The `amdlibComputeAllOiData` function, only available in the `Yorick` interface, takes all observation files and compute all associated OI-FITS files.

Performing a frame selection

The `amdlibPerformFrameSelection` function performs frame selection of AMBER OI-FITS file using different selection criteria and threshold or a selection file. The output can be averaged or use at the entry to another selection process. The `amdlibPerformAllFrameSelection` function, only available in the `Yorick` interface, make frame selections on all OI-FITS data present in the directory using either the default selection criteria or specified values.

3.5 Troubleshooting

You can request help by contacting the JMMC user support at `jmmc-user-support@ujf-grenoble.fr` or at `http://www.jmmc.fr/support.htm`. Please attach the full output of the "`amdlib c`" command located into the bin directory of the expanded binary package: `/your/path/to/amdlib-3.0.../bin/amdlib -c`

4 Cookbook

In this section, we provide a usage example of the `Yorick` plugin. The new `amdlibYorick`} user is encouraged to follow these elementary steps by reducing him/herself the α Ara (`alfara`) data provided on the AMBER web site. These data were obtained in the K band with 3 telescopes. They can be downloaded from the download page accessible from the AMBER page on the JMMC website:

http://www.jmmc.fr/data_processing_amber.htm

New in version 3!

4.1 Recommended fast data reduction

This is the recommended way of proceeding starting with `amdlib v3`. In next sections, we will redo the steps one-by-one.

1. Move to the `alfara/` directory and start `amdlib` (see the installation manual [6] if you have problems).
2. Compute all P2VM calibration files in the directory by using the `amdlibComputeAllP2vm` command. You will be asked for the bad pixel map (BPM) and the flat filed map files. Select one after the other (by clicking after selecting each one) respectively the `amdmsBadPixelMap.fits` and `amdmsFlatFieldMap.fits` files, then select the current `alfara/` directory.

   ```
   > amdlibComputeAllP2vm
   ```

   A sub-directory which ends with `..._P2VM/` is now created in `alfara/` and contains all P2VM files that could be computed.
3. Compute the complex visibilities from the data files present in the directory by using the `amdlibComputeAllOiData` command. You will be asked for the directory to process. Select again the current `alfara/` directory where the data files are. The function will determine automatically which raw data files and which P2VM files should be used.

   ```
   > amdlibComputeAllOiData
   ```

   There is now a sub-directory in `alfara/` which ends with `..._OIDATA/` and contains all OI-FITS files which have been computed with a binning of 1 and contains all frames.
4. Perform frame selection and averaging on all files using the `amdlibPerformAllFrameSelection` command. You will be asked the name of the directory to process. Use the sub-directory which finished with `..._OIDATA/` by using the button "THISDIR" when you are in the file browser in this directory.

   ```
   > amdlibPerformAllFrameSelection
   ```

   The selection criterion by default is the SNR and 100% of the frames, which means that all possible files have been used, although there are some frames that are automatically dropped (30 frames at the beginning in this example). The reduced data are now in the sub-directory ending by `..._AVG/`. This the end product. However, you may want to change or apply new selections by resending the `amdlibPerformAllFrameSelection` command with other values of the selection (use `help, amdlibPerformAllFrameSelection` for more details).
5. To see the results on the first reduced file `AMBER.2005-02-25T09:24:22.488_OIDATA_AVG.fits`, you use the command `amdlibShowOiData` by selecting this file in the browser under the sub-directory `..._AVG/`.

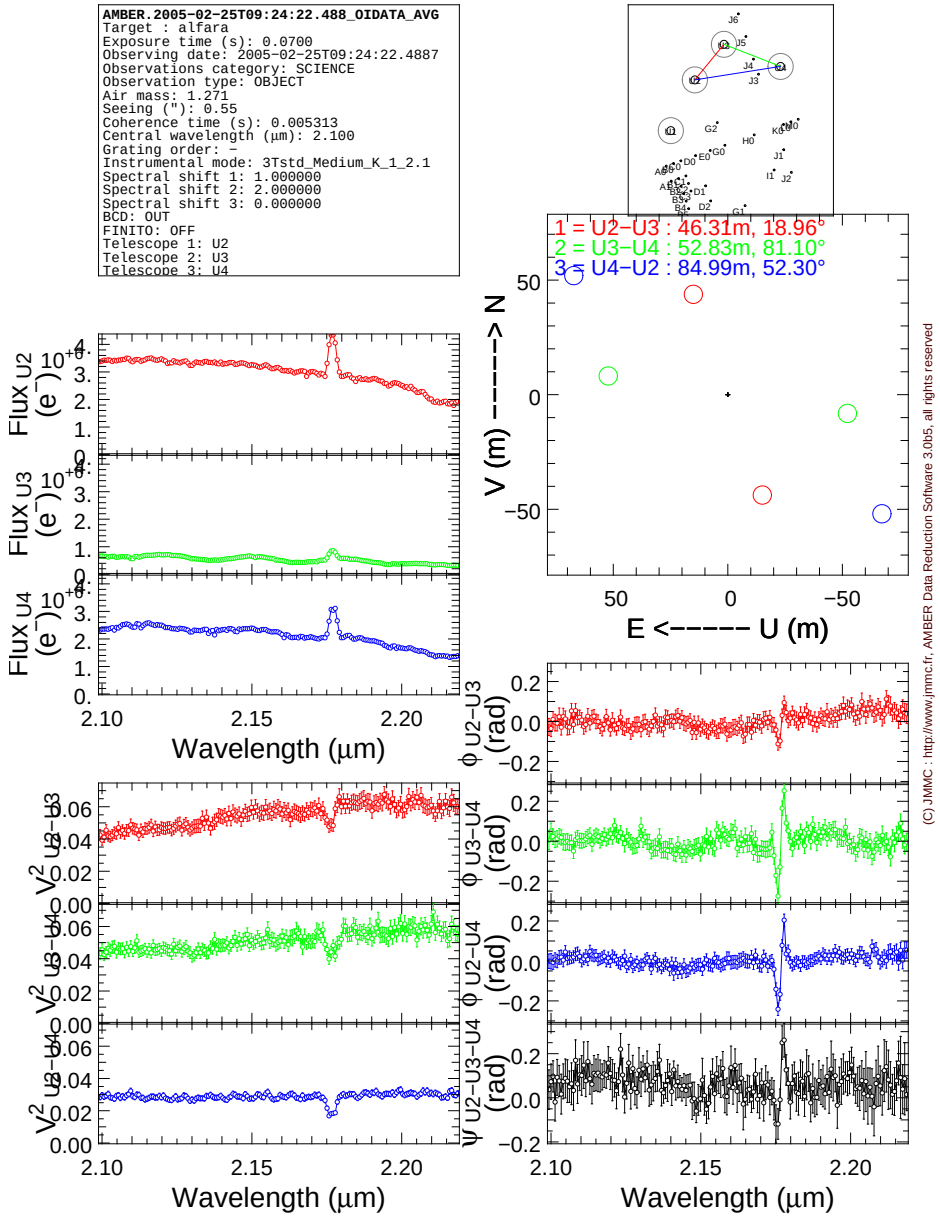

Figure 7: Result of `amdlibShowOiData` on the file `AMBER.2005-02-25T09:24:22.488_OIDATA_AVG.fits`.

```
> amdlibShowOiData
```

The display that pops up is the one presented in Fig. 7. If you are happy with the results, you are done. You just need to load the results from your OIFITS data in your favorite software. If you want to continue using `Yorick` then use the `amdlibLoadOiData` command (see Sect. 4.2.8).

Note: remember that, at anytime, typing `amdlib` or `help, amdlib` gives you an overview of all `amdlib` commands. Do not hesitate also to use `help, amdlib`*`Command`*.

4.2 Going steps by steps into the data reduction process

A demo file, called ***amdlibAlfAraDemo.i****, can be downloaded from the JMMC AMBER site (do also download the* ***alfara_3T.tgz*** *data archive). To execute it, send the command* ***#include "amdlibAlfAraDemo.i"*** *and follow the instructions. The figures of this cookbook come mainly from this demo.*

4.2.1 Setting the preferences

Before starting the data reduction, it is advisable to check the preferences that will be used throughout the process, and change them if deemed necessary. The current status of all variables stored in the preferences can be inspected by using the `amdlibShowPreferences` command, which returns the following information to the screen:

```
> amdlibShowPreferences
                      Value           Default         Possible values
 overwrite            0               0               0 or 1
 calibrateRaw         1               1               0 or 1
 convertToScience     0               0               0 or 1
 specCalShifts        "FIX"           "FIX"           [an array of 3 floats] or "FIX" or "MAN"
 splitBands           0               0               0 or 1
 selValue             100             20              a non-complex scalar number
 selType              "percent"       "percent"       "percent" or "threshold"
 selCriterion         "snr"           "snr"           "snr", "flux" or "piston"
 binSize              1               1               a positive non-zero integer
 pistonType           "phasor"        "phasor"        "phase" or "phasor"
 errorType            "statistic"     "statistic"     "statistic" or "theoric"
 band                 "all"           "all"           "all", "J", "H" or "K"
 useSky               0               0               0 or 1
 useDark              1               1               0 or 1
 dropFrames           -1              -1              [-1..1000]
 forceP2VM            0               0               0 or 1
 smoothPhaseGauss     4               4               [0..30]
 normalizeP2vm        1               1               0 or 1
 autoShiftP2vm        1               1               0 or 1
 zapDiscontinuities   0               0               0 or 1
 shiftSpectralAxis    0               0               a non-complex scalar number
 newSpectralResolution 0              0               a non-complex scalar number
 continuumCorrection  1               1               0 or 1
 minPhotometry        0               0               a non-complex scalar number
 maxChi2              -1              -1              a non-complex scalar number
 pistonClosure        1               1               0 or 1
 globalBias           1               1               0 or 1
 bpmUpdate            0               0               0 or 1
 fastMode             2               2               0 or 1 or 2
 normalizeSpectrum    0               0               0 or 1
 logEnable            1               1               0 or 1
 logLevel             "info"          "info"          "warning", "info", "test" or "debug"
 helpEnable           0               1               0 or 1
 debugEnable          0               0               0 or 1
 errorEnable          0               1               0 or 1
 flatFieldFile        []              "-"             a valid flat-field filename
 badPixelFile         []              "-"             a valid bad-pixels filename
 fileColors           [[160,160,160], "-"             a list of RGB colors for each AMBER file type
 graphicsDpi          79              "-"             [30..100]
 fileChooser          "new"           "new"           "old" or "new"
 calibTextFile        "/home/me/myfil "/home/me/myfil Should be a single filename
 catalogsDir          "/home/me/"      "/home/me/"    Should be a directory
```

Any of the above values can be changed by using the `amdlibSetPreference` routine, e.g. in the following way for the selection criterion:

```
> amdlibSetPreference, "selCriterion", "snr";
```

Each variable can also be queried individually by using the `amdlibGetPreference` function:

```
> amdlibGetPreference, "selCriterion", currValue, defValue, hlp;
```

where `currValue` and `defValue` respectively store the current and default value of the parameter, while `hlp` stores a message describing the possible values, as in the table above.

Part of the variables stored in the amdlib preferences are related to the use of specific data reduction functions. Their purpose and usage will be described in the following sections. The other part mostly specifies the way amdlib interacts with the user, e.g. by returning error messages, allowing to enter debug mode, logging various information, etc. A short description of all these variables can be obtained by typing:

```
> help, amdlibSetPreference
```

4.2.2 Loading the Bad Pixel and Flat Field Maps

The very first step in the data reduction process is to specify the Bad Pixel and Flat Field Maps (BPM and FFM[7]) that will be used throughout the data reduction process. Normally with advanced commands like `amdlibComputeAllP2vm` or `amdlibComputeAllOiData`, if the BPM and FFM are not loaded, then you will be asked for the location of the files.

If you want to realod the BPM and FFM or make a separate step, then the following command lines should be used:

```
> amdlibLoadBadPixelMap, inputBadPixelFile="amdmsBadPixelMap.fits"
> amdlibLoadFlatFieldMap, inputFlatFieldFile="amdmsFlatFieldMap.fits"
```

These two maps are included in the .tgz archives containing the α Ara data. Note that these maps **must not be used** to reduce your own data[8], which should have been delivered by ESO with their set of appropriate calibration maps. You can also find them at the ESO Quality Control website[9] or at the AMBER website[10] in the *Data Processing* menu. The BPM and the FFM can be visualized by using the following commands:

```
> amdlibShowBadPixelMap
> amdlibShowFlatFieldMap
```

The resulting panels are illustrated in Fig. 8.

4.2.3 Calibrating the instrument (P2VM computation)

The next step consists in computing the Pixel To Visibility Matrices, that will be used to calibrate the interferometric data. You can do it using the `amdlibComputeAllP2vm` (see Sect. 4.1), or do it by controlling what you do.

[7] See the comments about Flats and `amdlib v3` in the JMMC release notes http://www.jmmc.fr/twiki/bin/view/Jmmc/Software/AmberDrsReleaseNotes

[8] See the comments about Flats and `amdlib v3` in the JMMC release notes http://www.jmmc.fr/twiki/bin/view/Jmmc/Software/AmberDrsReleaseNotes

[9] http://www.eso.org/observing/dfo/quality/AMBER/qc/qc1.html

[10] http://amber.obs.ujf-grenoble.fr

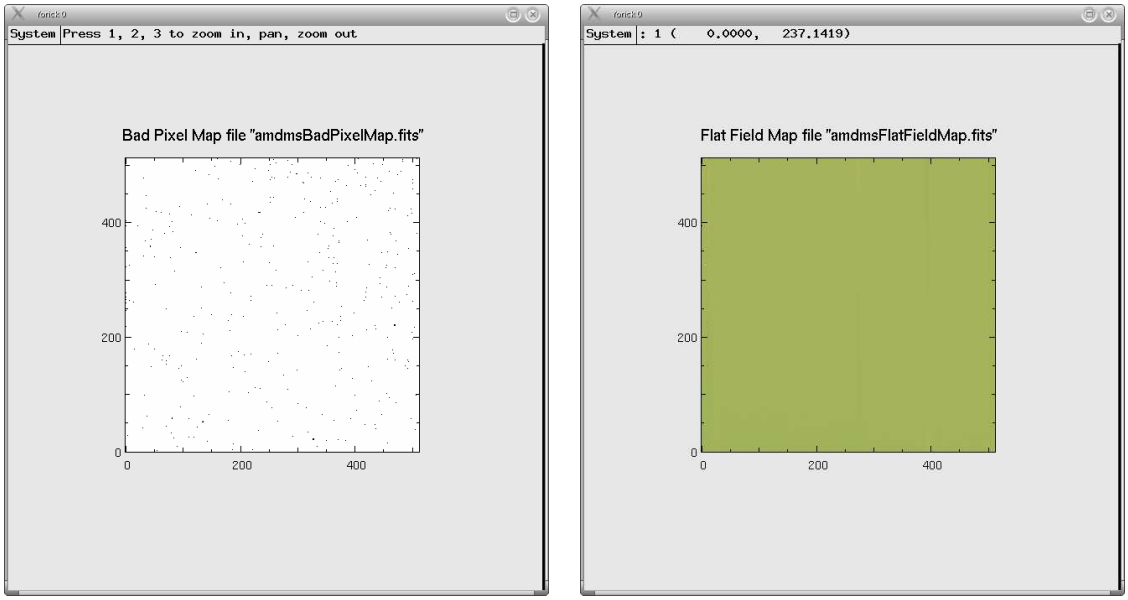

Figure 8: Panels showing the Bad Pixel and Flat Field Maps relevant to the α Ara data.

If `logLevel` is set to `"test"` by using the command `amdlibSetPreference, "logLevel", "test"`, you can have an echo of the amdlib command actually sent to the system. Useful if you want to grab them and execute in the shell.

To compute the P2VMs one by one, one needs to use the `amdlibComputeP2vm` routine. Apart from the BPM and FFM, to compute the P2VM file you will need the 5 (for 2 telescopes) or the 10 (for 3 telescopes) P2VM raw data files (keywords 2P2V or 3P2V).

The quality of the resulting P2VM files can be inspected by using the `amdlibShowP2vm` routine. The user will be prompted to choose a P2VM file through a file browser window. In the case you performed the automated P2VM computation, a subdirectory `..._P2VM` has been created. Once the P2VM is chosen, the `Yorick` plugin displays two panels containing various plots of the photometry, the instrumental contrast, the phases introduced by the piezoelectric actuator during the calibration, the carrying waves (C_k, D_k) and the ratio V_k between the photometric fluxes and the DC component of the interferogram. These panels are illustrated in Fig. 9. A diagnose graph is included in the bottom-right corned of the first panel to check whether the acquisition process was correctly performed: it represents the level of flux in the four channels (photometric and interferometric) during the P2VM acquisition sequence, so that the user can check that there is flux in the appropriate channels at all times.

4.2.4 On the spectral shifts

If you use `amdlibComputeAllP2vm`, the spectral shifts are computed automatically. If you want to manually find the spectral shifts, you can do the following:

```
> amdlibComputeP2vm,inputRawFile=raw,inputBiasFile=bias,specCalShifts="MAN"
```

where raw is your raw data file and bias, the dark file. A pop-up window will help you determine the spectral shifts (integers).

You can then apply these shifts (in this example, [4,4,-1]) by doing :

```
> amdlibComputeP2vm, inputRawFile=raw, inputBiasFile=bias, specCalShifts=[4,4,-1].
```

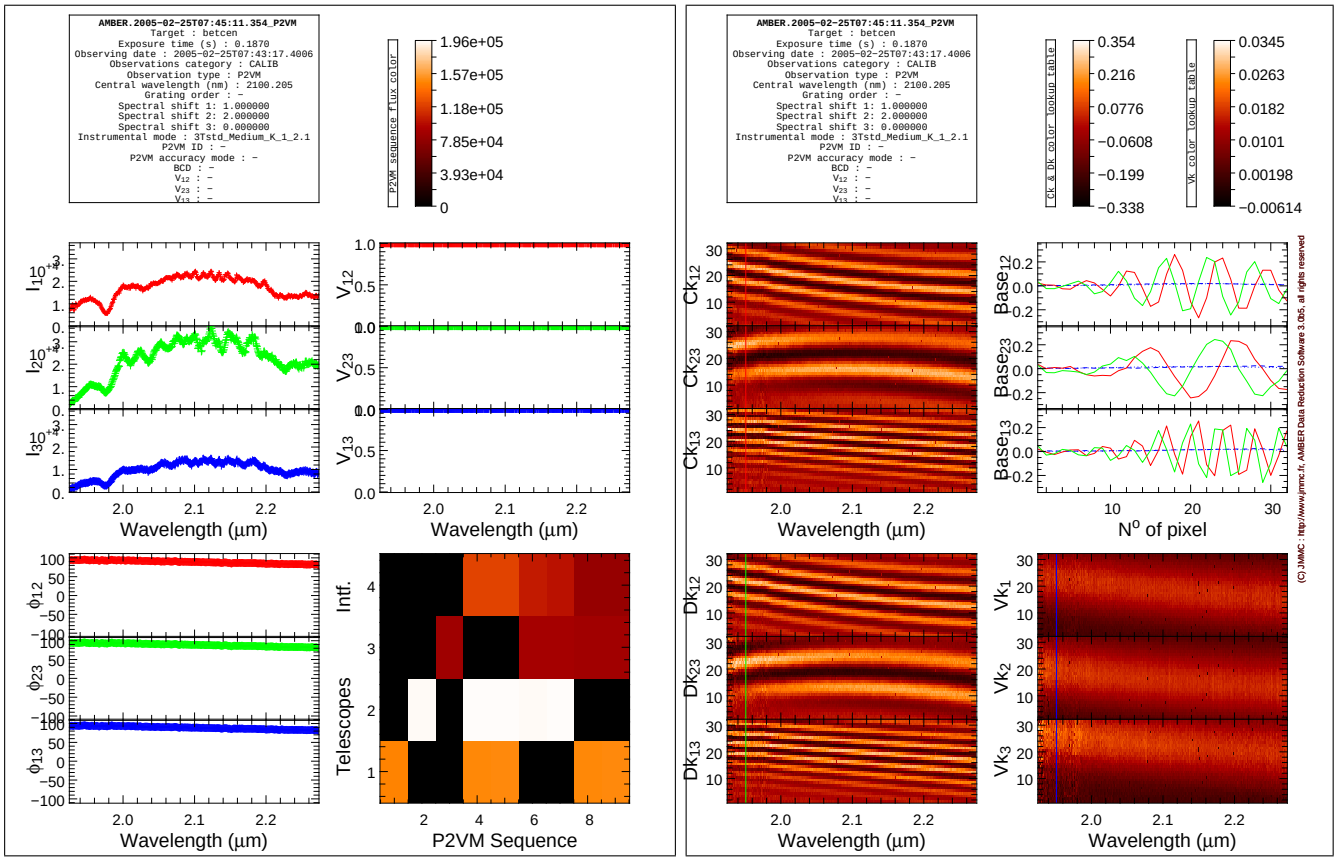

Figure 9: The two panels produced by the `amdlibShowP2vm` routine. In the left-hand side are shown the intensities, visibilities and phases recorded during the P2VM exposures, as well as the flux level in the photometric and interferometric channels during the P2VM acquisition sequence. In the right-hand side are displayed the carrying waves (C_k, D_k) and the ratio between photometric and interferometric channels (V_k). These quantities are also represented in the top-right plot for a given wavelength, which can be selected by a left-click on any of the three figures.

You can also find (sub-pixel) shifts values in the headers (FITS Keywords "`HIERARCH ESO QC Px OFFSETY`" where x is the beam number), as they are computed in Paranal, but because we have experienced issues with some of these values, we strongly recommend to double check using the previous method.

Finally, the user can use two new options to completely redefine the spectral calibration, by using the options 'newSpectralResolution' (option -w in shell, i.e., not in yorick) and 'shiftSpectralAxis' (option -S in shell) when computing the P2vm. 'newSpectralResolution' assigns a new value in nm/pixel for the spectral resolution. 'shiftSpectralAxis' assigns a value (in pixels) by which the wavelength table will be shifted, a positive value producing a shift towards longer wavelengths.

This could be used to correct from grating turret repositional error that affect all AMBER data. Use only in Medium or High Resolution mode, since in Low resolution Mode, the **default option should do this job automatically** (check anyway). The value to use depends on the particulars of your observation, and should you need it, it can only be found by means (yet) external to amdlib, ask `jmmc-user-support@ujf-grenoble.fr` for help.

We have found that in Medium Resolution from H to K band, a constant value per pixel of 0.6858 nm gives perfect results. This "linear dispersion mode" is not automatically enforced by amdlib.

Overall, we warn the users that the computation of the phase shifts is very sensitive to the quality of the Flat field map. At the JMMC, we recommend using a FFM that is really flat, and not the ones provided by ESO that are not always flat. You can find a good FFM here :

`http://www.jmmc.fr/twiki/pub/Jmmc/Software/AmberDrsReleaseNotes/VraimentFlat.fits.gz`

*`amdlibCheckSpectralShift` allow to check that the spectral calibration has been performed correctly during the computation of the P2VM, the user may use this function independently from the P2VM computation to check the spectral calibration has been done correctly. Please refer to the help of the function for more details (`help, amdlibCheckSpectralShift`).

4.2.5 Checking the data quality

The problems issuing from fringing in the "Old" AMBER detector are now correctly treated with `amdlib v3`. Unless further notice, there is no longer need to use the AMDC utility, developed by G. Li Causi.

Before producing the reduced OI data, it is important to check the quality of the raw data. As a first step, the user may want to check whether there is flux and fringes in a particular raw data file, which can be done with a command line similar to the following one:

```
> amdlibShowRawData, inputRawFile="AMBER.2005-02-25T09:38:46.349.fits",
inputBiasFile="AMBER.2005-02-25T09:37:14.383.fits"
```

By default, the raw data will be calibrated from detector artifacts. If the Bad Pixel and Flat Field Maps have not been loaded yet, the `Yorick` plugin will prompt the user to specify them using the file browser window. A Pixel Bias File (i.e., a dark file) will also be asked if not specified in the call to the routine. Three possible levels of "calibration" are offered within the `amdlibShowRawData` routine:

- no calibration (use `calibrateRaw = 0`): the raw data files are displayed without any treatment;
- detector calibration (default): the raw data are corrected for known detector imperfections;
- conversion to science data (use `convertToScience = 1`): in addition to the previous detector calibration, the data are compressed into science data format (the photometry is shrunk to one column per beam and the dead spectral zones are removed) and the spectra are calibrated so that each row corresponds to the same wavelength for the three photometric and the interferometric channels. The user will be prompted to specify a P2VM file, in which the spectral calibration information is contained. In case the data contains bright spectral lines, the user can check that spectral calibration has been performed correctly by a simple inspection (the spectral line must be located in the same raw for all columns). A slight offset is visible in Figure 10, where conversion to science data has not been performed.

Once the raw data file is loaded, the two panels illustrated in Fig. 10 will appear. The user can then browse the various frames, animate them, display a waterfall diagram, change the cuts, etc, or even convert raw data to science data, following the on-screen instructions.

4.2.6 Producing the raw OI data

This is the actual data reduction step, where the raw data are converted into actual interferometric observables, which are stored in a compatible, ESO-defined, variant of the IAU standard OI-fits file format [5]. The easiest way to produce the OI-fits file is to use the following command (see Sect. 4.1):

```
> amdlibComputeAllOiData
```

The routine automatically chooses the appropriate DARK, SKY and P2VM files to process all OBJECT files present in the specified directory, and places the resulting OI fits file in the `..._OIDATA` sub-directory. Alternately, the data reduction process can be applied to any individual OBJECT file, using the command `amdlibComputeOiData`. The user must then specify the DARK, SKY and P2VM files that are to be used.

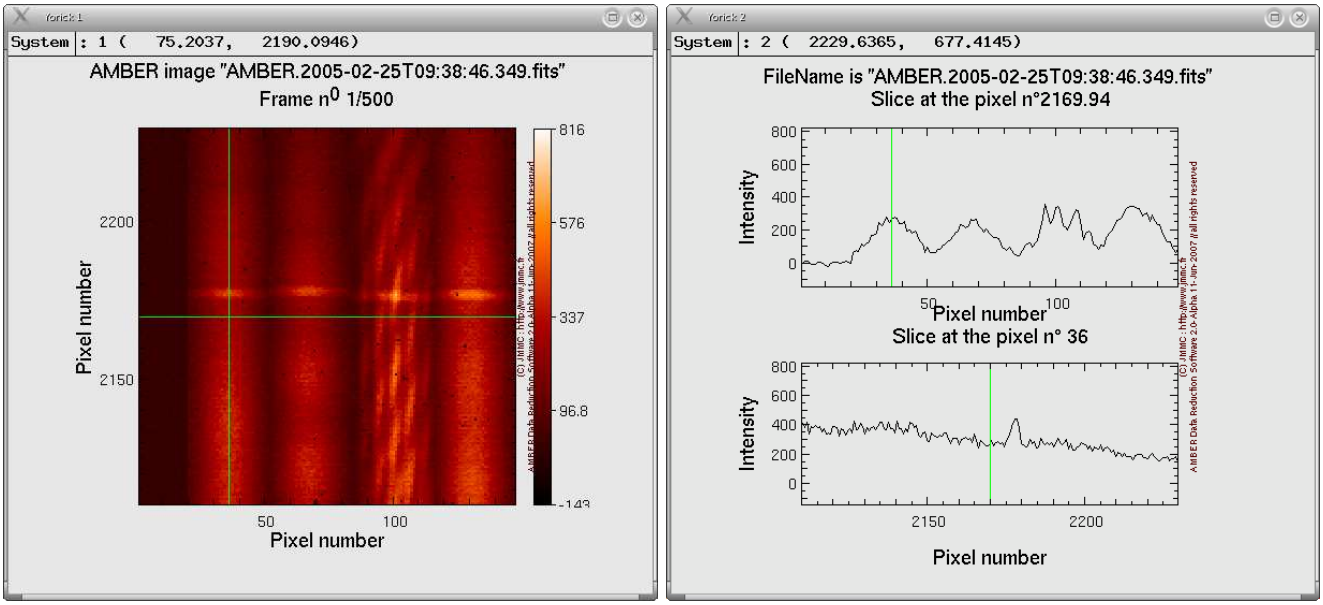

Figure 10: The two panels produced by a call to `amdlibShowRawData` for 3T data.

Note that a different P2VM from the one associated to the OBJECT file can be used, provided that the `forceP2VM` parameter is set to 1.

These two versions of the data reduction routine contain a few important options, which are detailed in the routine documentation. We highlight some of them here below:

- `binSize`: this parameter allows contiguous frames to be binned, by specifying the numbers of frames that should be grouped together. Please note that frame selection will not be possible after binning data.
- `splitBands`: if set to 1, splits the result into one file per spectral band, and adds a suffix _J, _H or _K to the file name (only in LR mode).
- `band`: specifies the spectral band for which the OI data must be computed.
- `useDark`: if set to 1, the software finds and uses dark exposures as pixel bias maps, to remove detector bias before further computations. If set to 0, no dark file is used and the software tries to guess the pixel bias map from the data file itself (which is not recommended). It is set to 1 by default.
- `useSky`: if set to 1, finds and use sky exposures to remove the thermal background from the data. This correction is useful to get rid of the thermal bias at longer wavelength (basically just at the end of the K band, between 2.3 and 2.5 μm). However, this option is set by default to 0 (see FAQ).

  Note that with the command-line interface you can always use a SKY in place of a DARK.

The main contents of the resulting OI data files can be displayed with the `amdlibShowOiData` routine. The two output panels produced by a call to this routine are displayed in Figure 11. The second one is save as a PS file.

4.2.7 Producing averaged OI data

The next step in the data reduction procedure is to perform frame selection on the raw OI data files: due to VLTI vibrations and other extraneous perturbations, not all frames are relevant for the final result, and bad frames need to be removed before averaging the visibilities, closure phases, etc. Frame selection can be

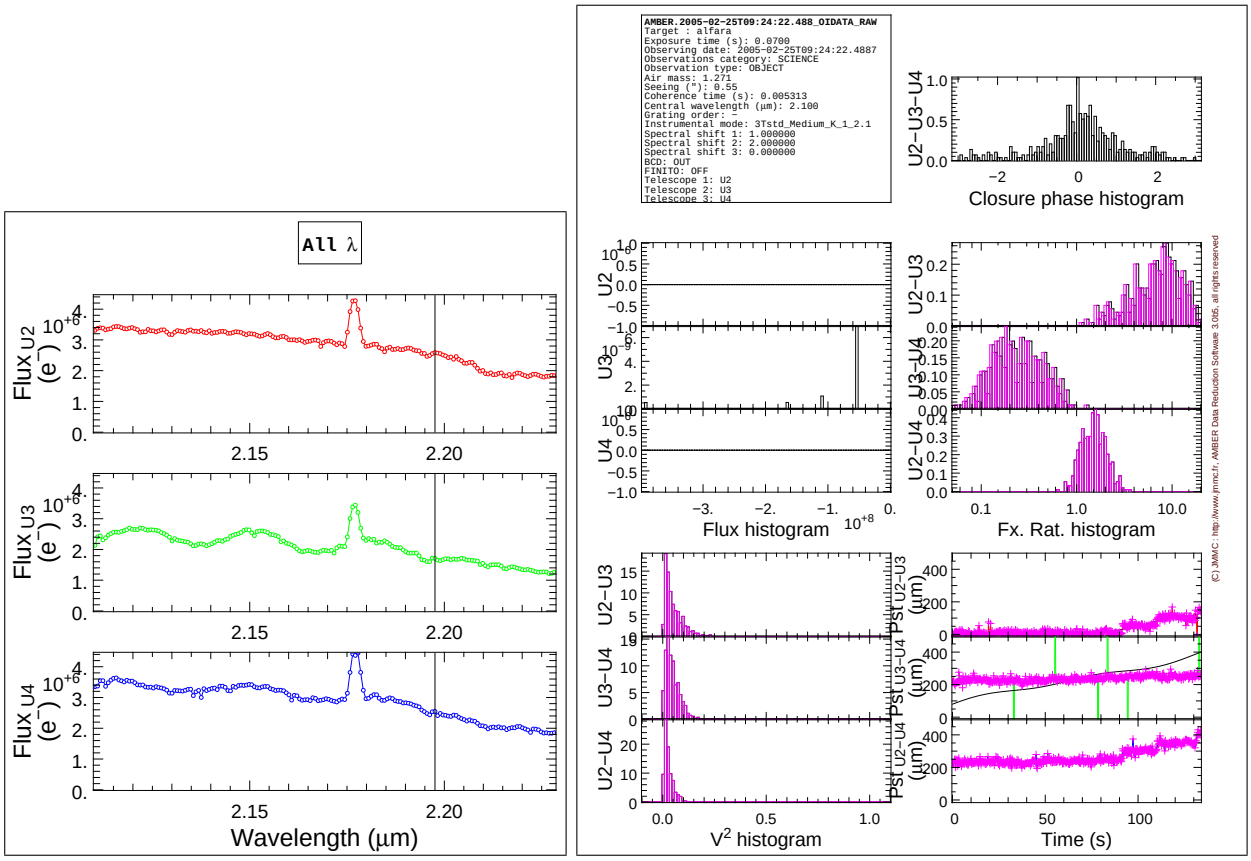

Figure 11: The two panels produced by a call to `amdlibShowOiData` for 3T data.

performed based on various criteria (flux, fringe SNR, or piston), and can be done by specifying either a threshold or the percentage of frames to be kept. An usual way to perform this step is the following:

```
> amdlibPerformAllFrameSelection,
selCriterion=["flux","piston","snr"], selType=["threshold","threshold","percent"],
selValue=[10,15,20];
```

This command line performs multiple chained selection: it first selects all the frames that have a baseline flux SNR larger than 10 (meaning that the baseline flux $\sqrt{P_i P_j}$ is 10 times larger than the associated noise), then it selects among the remaining frames those that have an estimated absolute piston smaller than 15 μm, and finally keeps the 20% of the remaining frames with the highest fringe SNR. As a result of this process, a new OI-fits file is produced, for which the suffix "RAW" is replaced by "AVG". This output OI file can be displayed with the `amdlibShowOiData` routine, as illustrated in Fig. 12.

Selection can be performed on any number of criteria, and is done in the order specified by the `selCriterion` keyword. If no selection criterion is specified, frame selection will be done using the (single) criterion specified in the preferences (multiple chained frame selection is not possible yet by default). After performing frame selection, the user can check which frames have been actually selected by displaying the original raw OI data file with the `amdlibShowOiData` routine: the selected frames are highlighted in purple.

Frame selection can also be performed on a whole directory with the `amdlibPerformAllFrameSelection`. However, this routine should be used with care, as optimal frame selection criteria can change from file to file. Another way to perform frame selection is to specify a user-prepared selection file, by using the `inputSelFile` keyword. The format of the input selection file is specified in [3], and consists in a FITS table filled with zeros and ones depending whether the frame is to be selected or not. Selection files can be saved in the fits format with the `amdlibSaveFrameSelection` function, or loaded to the workspace with the `amdlibLoadFrameSelection` function (see the online help for more information).

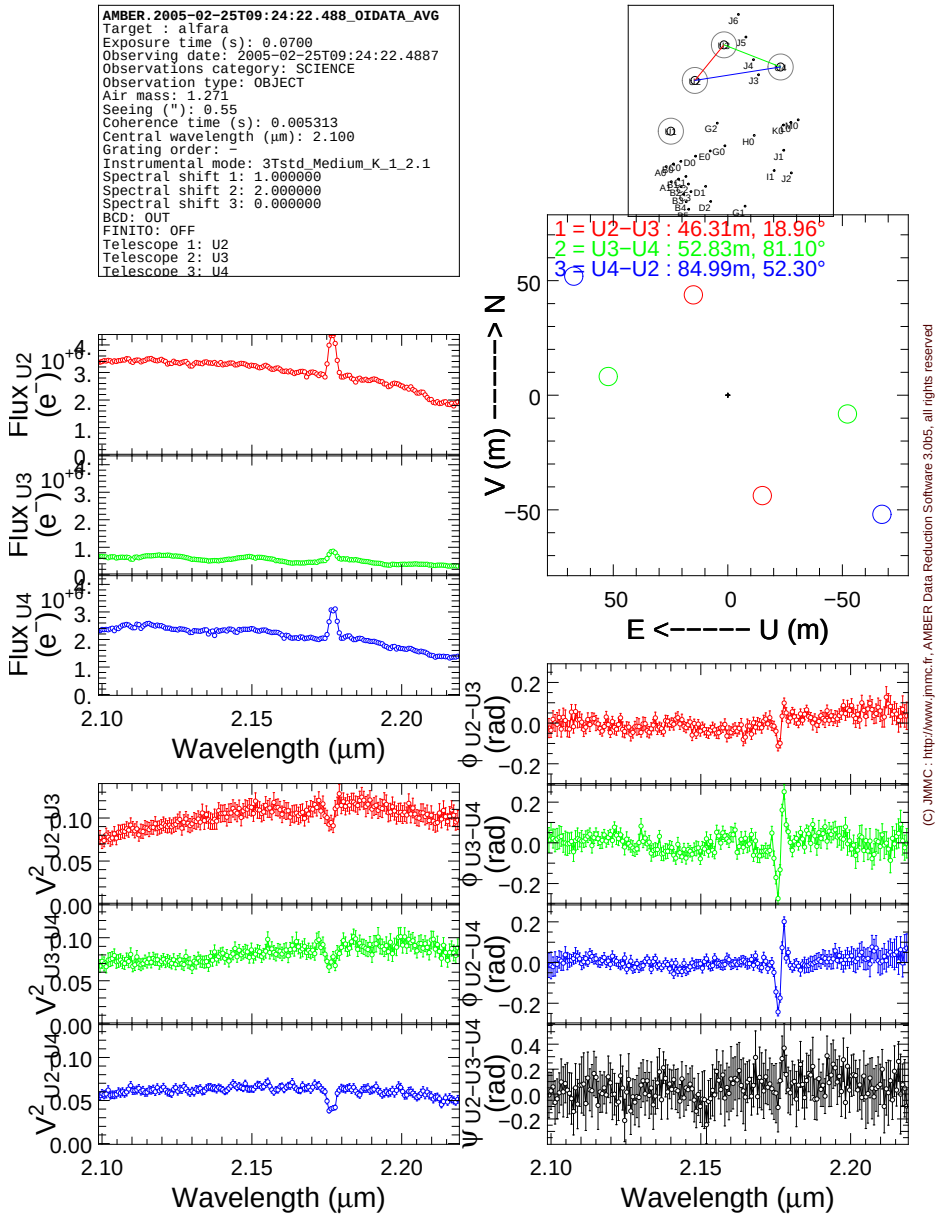

Figure 12: The panel produced by a call to `amdlibShowOiData` for 3T OI data after frame selection. To be compared with Fig. 7.

4.2.8 Reloading the data

The final OI-fits files can be read under `Yorick` using the `amdlibLoadOiData` routine, or using standard OI-fits file manipulation routines under your favourite programming language. A summary of their content can also be displayed using the `amdlibShowOiData` routine, as illustrated in Figure 12. Visually examining the content of the final OI AVG files is very important as the result of frame selection can be quite sensitive to the selection criteria. Selecting too many and/or too few frames may lead to large error bars, absence of phase closure, etc.

Important note: Not all the quantities stored in the final OI fits file have a physical meaning. For instance, the triple amplitude (`vis3Amp`) is not pertinent in the case of AMBER. Furthermore, some of the interferometric quantities are currently not computed in a correct manner by `amdlib` , in particular the amplitude of differential visibility (`diffVisAmp`).

4.3 Calibrating OI data (experimental)

We still do not provide an official way to calibrate data and we still refer to the annex B for details explanations on how to calibrate your data.

However we are testing new ways to automate the visibility calibration in order to provide this feature in a forthcoming version of `amdlib` . We provide in this early release of `amdlib v3` a library called `amdlibCalibrate` and two alternative ways to calibrate:

- search the diameter of the calibrators in the JMMC Stellar Diameter Catalog (JSDC) [11] using `amdlibSearchAllStarDiameters` and then using the software suite provided by F. Millour consisting in: `amdlibComputeAllTransferFunction`, `amdlibShowTransferFunctionVsTime`, `amdlibShowTransferFunctionVsWlen`, `amdlibCalibrateOiData`.
- or search the diameter of the calibrators in the JMMC Stellar Diameter Catalog (JSDC) using the same `amdlibSearchAllStarDiameters` and then using the `amdlibCalibrateAllOiData` provided by J.-B. Le Bouquin.

Use the `help` of these functions to learn how to use them. We will appreciate any comments on these features for later release of a well defined routine.

Annexes

A Frequently Asked Questions

Here are the most frequently asked questions about the amdlib software.

- **Where to find help?** By contacting the JMMC user support at http://www.jmmc.fr/support.htm or send an email to `jmmc-user-support@ujf-grenoble.fr`. Please attach the full output of the "`amdlib c`" command located into the bin directory of the expanded binary package.
- **Where is ammyorick?** ammyorick does not exist anymore. All the stuff previously in ammyorick is now in the amdlib `Yorick` plugin.
- **Where's the "Banana" plot?** There is no more "banana" plot in the `Yorick` plugin of amdlib. You should now get use with the histograms, which give you a better insight of the data structure. For exemple when looking at the visibilities squared histogram:
  - If it looks strange with a bipolar distribution (almost always the case if you have Unit Telescopes - UTs - observations), then it is just OK. Use frame selection for getting your visibilities and to improve SNR.
  - if it looks like a log-normal distribution, then you must not worry about your data: it corresponds to good UT observations.
  - if it is gaussian, centered to a non-zero visibility, then it should be nice ATs (Auxillary Telescopes) observations.
  - If it is a very wide gaussian, centered on zero, then you have no signal at all.

  If you are very sad about your favourite "banana" plot then you can plot it typing the following commands :

```
amdlibLoadOiData,wlen,bandWidth,time,fringeSNR,sqVis;
window,1,style="amdlib3horiz.gs";
amdlibPlotCorrelation,yocoMathSignedSqrt(sqVis(,avg,)), fringeSNR(,,3),
                      YXnames=["Vis","SNR"], startSys=1;
```

  Considering the number of people asking for the "banana" plots, we have implemented back a routine called `amdlibShowBanana` that produces the "banana" plot. Note that this routine is not supported at all and one should consider using the histograms of `amdlibShowOiData` instead (they contain the same information).
- **Why using darks and not skys in the `amdlibComputeAllOiData` function?** In an ideal world, the AMBER instrument would record cold darks to remove pixel bias map and warm skys to measure and remove thermal background. However, technical problems in the dark files prevent from getting cold (i.e. -196°) darks and therefore a thermal contamination occurs in the pixel bias map. Today, no "miraculous" solution exists and one has to cope with these warm darks. This is the main reason why we use darks and not skys and why the region between 2.3 and 2.5μm must be taken with caution.
- **How to reduce the data directly from my DVD?** If you do not want to copy the whole data contained on your ESO DVD to your hard disk, you can reduce your data directly from the DVD by creating a virtual image of your DVD on your hard disk through a symbolic link, by using the `lndir` command (in package `lndir`, which is not installed by default on some Linux Distributions).

B Calibrating the visibilities: tips and tricks

Please see Sect. 4.3 for additional information especially on retrieving calibrator diameters with `amdlibSearchAllStarDiameters`.

Visibility calibration is a very delicate step in interferometric data reduction since one perform a division in the Fourier space of the object brightness distribution, equivalent in the image space to a deconvolution. Therefore the calibration itself, but also the error bars estimation are of crucial importance to have usable calibrated data.

The first step is the correction of the calibration star(s) visibilities by its estimated visibilities, related a uniform disk (or limb-darkened) model of the stellar surface. For that one need to get the different parameters to input to the model: the diameter and in some cases the limb-darkening parameters. Different tools and references exists to get such parameters and we will just cite them here, so that one can browse the respective documentation of the tools:

- `SearchCal` from the JMMC (http://www.mariotti.fr/searchcal_page.htm), based on the work of Bonneau et al. [12], allows to compute stellar diameters from fits of colour-colour relations. JMMC offers also a visibility computation tool called `aspro`[11].
- `CalVin` (http://www.eso.org/observing/etc) is an ESO tool based on Merand et al. [13] and CHARM2 [14] catalogues queries. This ESO angular diameters database is often updated using new interferometric measurements and publications. ESO offers also a visibility computation tool called `VisCalc`.
- `getCal`, another similar tool from the MSI (http://msc.caltech.edu/software/getCal).
- Catalogs, such as those of JMMC[12], the CHARM2 and Merand et al. catalogues, etc, through the Vizier[13] service of CDS.

Please note that these tools are also useful for proposals preparation and therefore the time spent to learn how they work will be very much valued at all steps of interferometric observations.

Once one has found the diameter of the calibration star(s) (and eventually it's associated error), the calibration star(s) visibility correction can be computed in order to measure the instrumental+atmospheric transfer function at the time of the calibration star(s) observation. In the case of an uniform disk model of the calibrator, the transfer function can be computed like that:

$$T^2 = \frac{V^2_{\rm cal}}{\left(2\frac{\mathrm{J}_1\left(\frac{\pi B\theta_{\rm cal}}{\lambda}\right)}{\frac{\pi B\theta_{\rm cal}}{\lambda}}\right)^2} \quad (1)$$

where J_1 is a 1^{st} order Bessel function, B is the projected base length, λ is the wavelength, $\theta_{\rm cal}$ is the apparent stellar diameter of the calibrator and $V^{\rm cal}$ is the measured visibility of this calibration star.

Of course, this step must be repeated as many times as different calibration stars exist during one observing night and one get an instrumental transfer function database all over the night (mandatory to reliably estimate error bars).

Then, one has to divide the science star visibility by the transfer function estimate at the time of the observation. Several possibilities exist to get the value at this observing time such as interpolating between several stars, or using weighted averages of the measured transfer function points, etc.

[11] http://www.mariotti.fr/aspro_page.htm
[12] http://www.mariotti.fr/databases.htm
[13] http://vizier.u-strasbg.fr

$$V^2_{\text{calibrated}} = \frac{V^2_{\text{star}}}{T^2} \tag{2}$$

In order to standardize observations, ESO proposes a simplified method using only one calibration star to measure the transfer function and computes visibilities just by considering that the transfer function is not changing between the star and calibrator. This is implemented in the `amdlib Yorick` function `amdlibDivideOiData` (see Sect. 4.3 for more details). Of course this gives just a first glance of data quality but is not sufficient to fully calibrate the data. One has to check with various calibrators during the same night that the resulting calibrated visibility is unbiased. The ESO DVDs does not contain such data so the user has to retrieve the calibration stars by himself through the ESO science archive facility (http://archive.eso.org/eso/eso_archive_main.html).

Please note all these steps are mandatory to get fully calibrated data. `amdlib` currently do not offer such calibration process but we are developing such tool and it will be present in a next version of the software.