Document Title: **SPARK INstructional Guide for KMOS data**

Document Number: **VLT-MAN-KMO-146611-009**

Issue: **0.81**

Date: **10.06.13**

Document Prepared By:	R. Davies A. Agudo Berbel E. Wiezorrek	Signature and Date:
Document Approved By:	N. Förster Schreiber	Signature and Date:
Document Released By:	A. Fairley	Signature and Date:

UK Astronomy Technology Centre

Change Record

Issue	Date	Section(s) Affected	Description of Change/Change Request Reference/Remarks
0.5	31.01.13	All	Draft at end of Comm-2
0.7	04.04.13	All	Update after Comm-3
0.8	08.05.13	All	Intermediate update for SPARK v1.1.2
1.0	15.06.13	All	First Issue for SV

TABLE OF CONTENTS

1 Scope

SPARK is the Software Package for Astronomical Reduction with KMOS. It includes the official pipeline release, as well as some additional perl and shell scripts that can help make using it easier.

This document describes how to get started using SPARK to process KMOS data without reading the full manual. It does not include everything, only the essential things you need to know together with some useful tips. KMOS is a complex instrument and, inevitably, so is the data and the data processing. We have tried to keep it as simple as possible. The guide may seem long, but it takes you through step-by-step, providing examples to follow. So just start, and work your way through it. We hope it is useful, both for beginners and as a reference.

If you use this software, please cite the following reference (**this will be updated during 2013 so please check the newest version of this guide for updates**):

Davies R., Agudo Berbel A., Wiezorrek E., Ott T., Foerster-Schreiber N.M., 2010
in “Ground-based and Airborne Instrumentation for Astronomy III”
eds. McLean I., Ramsay S., Takami H.,
Proc. SPIE, 7735

2 Getting started with ESOREX

The pipeline can be run on the command line using ESOREX (ESO’s recipe execution tool; see http://www.eso.org/sci/software/cpl/esorex.html).

In principle, one can also use GASGANO which provides some file sorting capability and a graphical interface to ESOREX. We prefer to use the tools provided with SPARK: either the SPARKplug to sort files and to create the file lists needed by ESOREX, or the set of easySPARK scripts to do this in an automated way. But file lists can also be made by hand using any editor. If you want to use GASGANO and need help, please contact ESO’s User Support Department.

2.1 Installation

SPARK is distributed by ESO as a kit (`kmos-kit-x.x.x.tar.gz`) containing the official pipeline recipes, some additional tools, the manual, and a set of standard calibration files. The calibration files are huge and are only included for the automatic data processing at Paranal. To avoid users having to download such a large dataset, MPE also distributes their own kit (`spark-kit-x.x.x.tar.gz`) without the calibrations, which uses the newest libraries and includes the most up-to-date fixes. This can be found at https://wiki.mpe.mpg.de/KMOS-spark. In either case, users should create their own calibrations in order to obtain the best results.

The software runs on all major Unix-based operating systems, as well MacOSX. For installation, the sciprt `install_pipeline.sh`, included in the kit, has to be executed. At installation, a target directory (e.g. `/share/KMOSpipeline`) and a calibration directory (e.g. `/share/KMOScalib`) must be specified. Note that throughout this guide we use `/share` as the path to the KMOS directories.

Installing GASGANO is a tad more tricky since it requires the path to the java runtime. We do not discuss the use of GASGANO in this guide. If you also do not need GASGANO, you can delete the gasgano tar-file and the `install_pipeline.sh` script will just skip it.

As a first check to see if all the necessary libraries have been installed correctly, just type:

```
> esorex

     ***** ESO Recipe Execution Tool, version 3.9.6  *****

Libraries used: CPL = 6.1.1, CFITSIO = 3.29, WCSLIB = 4.13.4 (FFTW unavailable)
```

(the FFTW isn't distributed with the kit and isn't needed for KMOS).

After installation, add `/share/KMOSpipeline/bin` (where `/share/KMOSpipeline` should be replaced with your installation target directory) to your `PATH` environment variable (in `.tcshrc` or `.bashrc`). And copy all the scripts from `kmos-kit-x.x.x/kmos-x.x.x/tools/easySPARK` and from `kmos-kit-x.x.x/kmos-x.x.x/tools/SPARKplug` to `/share/KMOSpipeline/bin`. Ensure that all easySPARK scripts and SPARKplug.pl are executable. If they are not then execute
`> chmod a+x easySPARK_* SPARKplug.sh` in `/share/KMOSpipeline/bin`.
For `SPARKplug.pl` to run, the 'perl-tk' module must be installed. (For MacOSX using MacPorts, install the port 'p5-tk').

If ESOREX doesn't behave as described in this manual, some configurations can be done manually. The most comprehensible way is to type:

```
> esorex --create-config=true
```

This creates `.esorex/esorex.rc` in your HOME directory which can be edited in any text editor and provides a multitude of configuration possibilities. For example set
`esorex.caller.suppress-prefix=TRUE`
in order to override the standard ESOREX file-naming convention which defaults to `out_xxx.fits`.

2.2 Using the software

2.2.1 ESOREX & Recipes

Help for esorex is provided by the command:

```
> esorex -help

     ***** ESO Recipe Execution Tool, version 3.9.6  *****
Usage: esorex [esorex-options] recipe [recipe-options] sof
```

And a list of the recipes available is given by:

```
> esorex –recipes

     ***** ESO Recipe Execution Tool, version 3.9.6  *****
List of Available Recipes :

  kmo_arithmetic         : Perform basic arithmetic on cubes
  kmo_combine            : Combine reconstructed cubes
  kmo_copy               : Copy a section of a cube to another cube, image or
                           spectrum
  kmo_dark               : Create master dark frame & bad pixel mask
  kmo_dev_setup          : Create aligned KMOS files out of test frames
  kmo_extract_spec       : Extract a spectrum from a cube.
  kmo_fits_check         : Check contents of a KMOS fits-file
  kmo_fits_stack         : Creates KMOS conform fits-files
  kmo_fits_strip         : Strip noise and/or rotator extensions from a processed
                           KMOS fits frame
```

```
  kmo_fit_profile        : Fit spectral line profiles as well as spatial profiles
                           with a simple function - for example to measure
                           resolution or find the centre of a source
  kmo_flat               : Create master flatfield frame and badpixel map to be
                           used during science reduction
  kmo_illumination       : Create a calibration frame to correct spatial
                           non-uniformity of flatfield.
  kmo_make_image         : Collapse a cube to create a spatial image
  kmo_multi_reconstruct  : Combine reconstructed cubes
  kmo_noise_map          : Generate a noise map from a raw frame
  kmo_reconstruct        : Performs the cube reconstruction using different
                           interpolation methods.
  kmo_rotate             : Rotate a cube spatially
  kmo_sci_red            : Reconstruct and combine data frames dividing
                           illumination and telluric correction.
  kmo_shift              : Shift a cube spatially
  kmo_sky_mask           : Create a mask of spatial pixels that indicates which
                           pixels can be considered as sky.
  kmo_stats              : Perform basic statistics on a KMOS-conform fits-file
  kmo_std_star           : Create the telluric correction frame.
  kmo_wave_cal           : Create a calibration frame encoding the spectral
                           position (i.e. wavelength) of each pixel on the
                           detector.
```

Not all of the recipes are required to run the pipeline; some aim instead to provide useful tools for manipulating KMOS data, which can otherwise be awkward due to the use of numerous extensions.

Detailed help on any individual recipe (an outline of its purpose, a list of input files required, a list of the output files produced, and a description of the various optional parameters) can be found by, for example:

```
> esorex -man kmo_flat

     ***** ESO Recipe Execution Tool, version 3.9.6  *****

NAME
  kmo_flat -- Create master flatfield frame and badpixel map to be used during
              science reduction

SYNOPSIS
  esorex [esorex-options] kmo_flat [kmo_flat-options] sof

DESCRIPTION
<blah>
```

2.2.2 Static Calibration Files

The KMOS data reduction recipes require a number of calibrations that should not need to change. These include, for example, arc-line lists, look-up tables etc. The user should confirm that these are available. A full list of these is:

```
kmos_ar_ne_list_h.fits     kmos_atmos_h.fits     kmos_solar_h_2400.fits     kmos_oh_spec_h.fits
kmos_ar_ne_list_hk.fits    kmos_atmos_hk.fits    kmos_solar_hk_1100.fits    kmos_oh_spec_hk.fits
kmos_ar_ne_list_iz.fits    kmos_atmos_iz.fits    kmos_solar_k_1700.fits     kmos_oh_spec_iz.fits
kmos_ar_ne_list_k.fits     kmos_atmos_k.fits                                kmos_oh_spec_k.fits
kmos_ar_ne_list_yj.fits    kmos_atmos_yj.fits                               kmos_oh_spec_yj.fits

kmos_wave_ref_table.fits   kmos_wave_band.fits   kmos_spec_type.fits
```

2.2.3 SPARKplug

Most recipes require a 'set of files' (sof) as input. The SPARKplug is a graphical data organizer that assists in preparing these lists, and insuring that all the necessary calibration files are included. It is not required – and is not part of the official pipeline release – but does make this step easier. It is started with the command below, with directories for the raw and calibration files set as examples

```
> SPARKplug.pl -cal=/share/KMOScalib -raw=/share/KMOSdata
```

The SPARKplug has sufficient file sorting capability to make this task relatively straightforward as long as the static calibration files, the raw data files, and the processed calibration products are kept in appropriate directories. One just needs to decide for which recipe the sof list is required, and the SPARKplug will show the set of appropriate raw and calibration files from which to choose. Being able to sort the files by name, band, etc, makes this very quick and simple. Files can be selected and de-selected. The sof list can be edited manually, and saved, or used directly with the recipe.

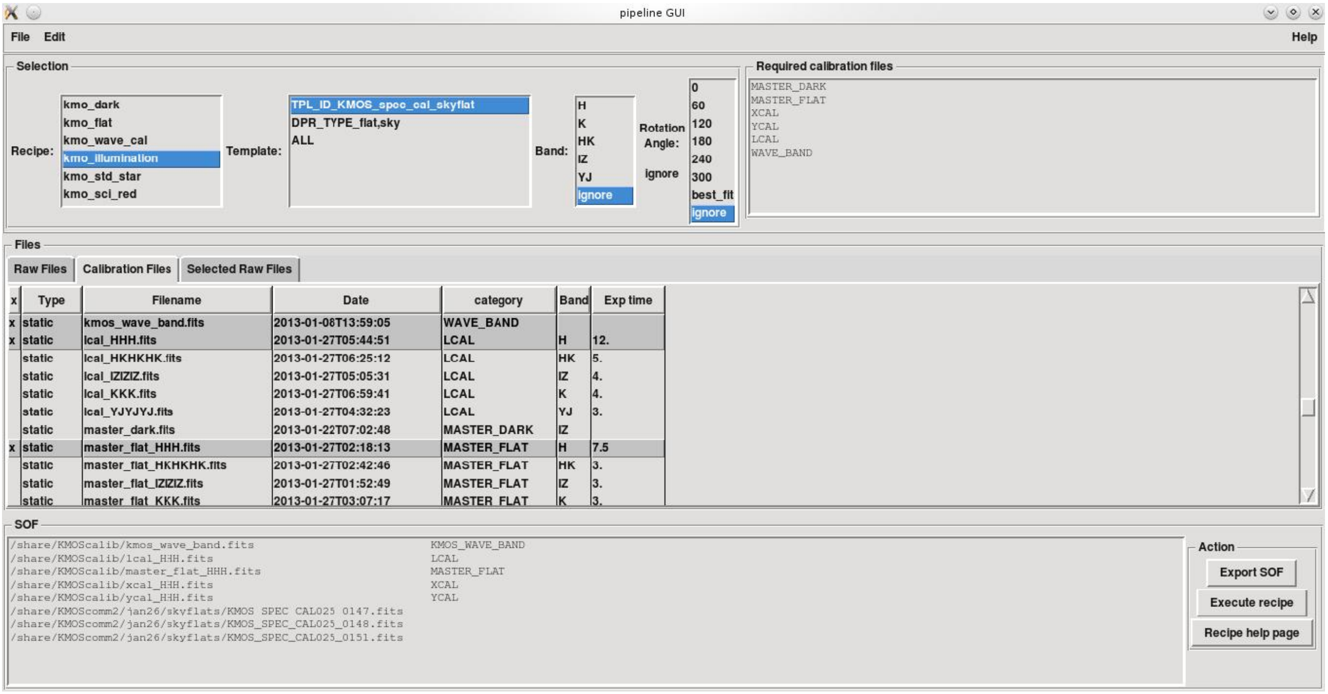


Figure 1: The SPARKplug tool can assist in preparing the 'set of files' required by most recipes.

2.2.4 easySPARK scripts

Normally data are obtained in a well-defined standard procedure and creating sof-files to reduce these is therefore a quite repetitive task. These scripts aim to create sof-files and run ESOREX on them in a fairly automated manner.

All the scripts require a single file (path and name) as input and extract the other associated exposures via the `TPL.START` keyword, which is identical for all exposures generated in a single template. Furthermore, the environment variable `KMOS_CALIB` should be set to a path containing the static calibration files (see Sec. 2.2.2). Then if any dynamic calibration file (like e.g. `XCAL`) is not found in the working directory, `KMOS_CALIB` is queried as well.

In order to obtain help on a specific script, just execute the script without an argument. If just the sof-files should be created without running ESOREX, simply provide `sof` as an additional parameter.

<u>For the calibration recipes, the following scripts are available:</u>

```
easySPARK_dark.sh
easySPARK_flat.sh
easySPARK_wave_cal.sh
easySPARK_illumination.sh
easySPARK_std_star.sh
```

`easySPARK_calibration.sh` Unifies the dark, flat and wave_cal scripts. Here, because the keyword `OBS.START` is also examined, the script only works when all the calibration files have been generated in a single observation block (which is normally the case).

<u>For standard use-cases there exist also scripts which are rather self-explanatory:</u>

`easySPARK_reconstruct.sh`

`easySPARK_mapping.sh` (to come)

etc.

3 Handling KMOS data

3.1 Data Format and Properties

The KMOS instrument has 3 similar segments, and so each exposure yields 3 frames. The data are stored in fits extensions. Since each segment has 8 IFUs, the reconstructed data will have 24 extensions (or 48 if noise is propagated). One can quickly see how many extensions a file has and what format the data is stored as, using:

```
> esorex kmo_fits_check KMOS_SPEC_CAL022_0004.fits

<blah>
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
FORMAT:          RAW
NAXIS:           2
NAXIS1:          2048
NAXIS2:          2048
NOISE:           FALSE
BADPIX:          FALSE
NR. EXT:         3 (excluding primary header)
   NR. DATA:     3
   NR. NOISE:    0
   NR. BADPIX:   0

VALID RAW FILE!
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
[ INFO ] esorex: [tid=000] 0 products created
```

If you want to view the data, we recommend using QFitsView, which can be downloaded from http://www.mpe.mpg.de/~ott/QFitsView. Other viewers that are able to read extensions can also be used.

3.2 Header Keywords

KMOS data has a lot of keywords, both in the primary header and the extension headers. We recommend using `dfits` and `fitsort` (both part of the `qfits` package from ESO) to list relevant keywords in the data. An example of usage is:

```
> dfits -x 0 KMOS*fits | fitsort tpl.id det.seq1.dit ins.filt1.id ocs.rot.naangle

FILE                         TPL.ID                DET.SEQ1.DIT   INS.FILT1.ID   OCS.ROT.NAANGLE
KMOS_SPEC_CAL022_0001.fits   KMOS_spec_cal_stdstar 20.0000000     H              -13.593
KMOS_SPEC_CAL022_0008.fits   KMOS_spec_cal_stdstar 20.0000000     YJ             9.478
KMOS_SPEC_DARK021_0016.fits  KMOS_spec_cal_dark    10.0000000     Block          -60.000
```

Or to list all QC parameters in a processed file:

```
> dfits -x 0 cube.fits | grep QC
```

For these data, the '`-x 0`' is important since it will then look at the headers in all extensions.

Another useful expression allows you to list, for example, the names of the targets assigned to each arm in a single frame. This is otherwise difficult since each arm has a different keyword:

```
> dfits KMOS_SPEC_OBS085_0049.fits | grep OCS.ARM.*NAME
```

Similarly the allocation of the arms in a frame as reference/object/sky (R/S/O) can be listed using

```
> dfits KMOS_SPEC_OBS085_0049.fits | grep OCS.ARM.*TYPE
```

A list of the most useful keywords in the RAW frames, and where to find them, is:

keyword in RAW frame	*location*	*description*
`dpr.type`	p	Type of observation (e.g. object,sky / flat,lamp / dark / etc)
`obs.start`	p	Date/time at which the OB was started
`tpl.start`	p	Date/time at which the template (within the OB) was started
`tpl.id`	p	Name of template used for observations
`date-obs`	p	Date/time at which exposure was started
`obs.id`	p	Unique identifier for OB
`obs.name`	p	Name of OB
`paf.id`	p	Name of KARMA parameter file (PAF) used: *.ins
`obs.targ.name`	p	Name of KARMA catalogue used; typically *.cat
`ocs.arm[1-24].name`	p	Name of target assigned to arm [1-24]
`ocs.arm[1-24].type`	p	Type of exposure for this arm (O / S / R for object / sky / reference)
`det.seq[1-3].dit`	p	Integration time for detector [1-3]
`det.ndit`	p	Number of integrations averaged during exposure
`det.ndsamples`	p	Number of non-destructive samples during integration
`ins.filt[1-3].id`	p	Name of filter [1-3] (IZ / YJ / H / HK / K / Block)
`ins.grat[1-3].id`	p	Name of grating [1-3] (IZ / YJ / H / HK / K)
`ins.lamp1.st`	p	Keyword only included if status of argon lamp is ON
`ins.lamp2.st`	p	Keyword only included if status of neon lamp is ON
`ins.lamp3.st`	p	Keyword only included if status of flatfield lamp is ON
`ocs.rot.offangle`	p	Orientation of KMOS field wrt North
`ocs.rot.naangle`	p	Orientation of KMOS instrument wrt Nasmyth platform
`tel.parang.[start/end]`	p	Parallactic angle at start / end of exposure
`tel.airm.[start/end]`	p	Airmass at start / end of exposure
`tel.targ.alpha`	p	Right ascension of preset telescope pointing (first field

		centre defined in KARMA).
`tel.targ.delta`	p	Declination of preset telescope pointing (first field centre defined in KARMA).
`ocs.targ.alpha`	p	Right ascension of current assigned telescope pointing (field centre). KARMA defines 2 field centres.
`ocs.targ.delta`	p	Declination of current assigned telescope pointing (field centre). KARMA defines 2 field centres.
`ocs.arm[1-24].alpha`	p	Right ascension of pointing assigned to arm [1-24]. KARMA defines 2 pointings for each arm, associated with the 2 field centres.
`ocs.arm[1-24].delta`	p	Declination of pointing assigned to arm [1-24]. KARMA defines 2 pointings for each arm, associated with the 2 field centres.
`ocs.arm[1-24].notused`	p	Keyword only present if arm is not used
`ocs.targ.ditha`	p	Relative offset (right ascension) of dither position with respect to the current assigned pointing, in arcsec. Dither sequences for the 2 KARMA field centres are followed independently.
`ocs.targ.dithd`	p	Relative offset (declination) of dither position with respect to the current assigned pointing, in arcsec. Dither sequences for the 2 KARMA field centres are followed independently.
`ocs.stdstar.mag`	p	Magnitude of standard star, if it is given in the template. Applies only to files created with the stdstar templates.
`ocs.stdstar.type`	p	Spectral type of standard star, if it is given in the template. Applies only to files created with the stdstar templates.
`extname`	x	Name of extension: CHIP[1-3].INT1
`det.chip.gain`	x	Gain in e- per ADU of chip (=2.1)
`naxis`	x	Dimension of data in the extension
`naxis[1-n]`	x	Size of data axis [1-n]

A list of the most useful keywords in the pipeline products, and where to find them, is given below. QC parameters are excluded from this list, and instead a selection of the most useful ones is given in each of the respective sections of this document. To see all QC parameters in the header, use:

```
> dfits -x 0 product.fits | grep QC
```

keyword in processed frame	*location*	*description*
`extname`	x	Name of extension, e.g. IFU.1.DATA / IFU.1.NOISE / etc.
`pro.catg`	p	Type of product
`pro.rot.naangle`	x	Orientation of KMOS instrument (wrt Nasmyth platform) associated with extension; especially useful for master_flat.
`pipefile`	p	Useful when facing data produced by the on-line workstation. This is the human-readable name for the file, which you would get if you processed the data yourself.

If you want to rename the output of the on-line workstation to more useful names, try the following:

```
> dfits *fits | fitsort PIPEFILE > filelist
```

Check that `filelist` has no repeated names, and then rename everything:

```
> awk '{if ($1 != "FILE") {printf("mv %s %s\n",$1,$2)}}' filelist | csh
```

This also works for renaming archive files, but using the `ORIGFILE` keyword instead of `PIPEFILE`.

3.3 IFU orientation, pixel arrangement, resolution

Across the detectors, the IFUs are numbered sequentially from left to right, across detector 1 to 3. The order (from left to right across a detector) of the spatial pixels within a slitlet, and the slitlets within an IFU, is more complex. These are arranged as shown in Figure 2 for the 24 IFU fields. The effect of this arrangement can be seen in the raw data and also sometimes in the reconstructed cubes. The spectral axis is approximately aligned with the columns. Long wavelengths are at the bottom of the detectors; short wavelengths at the top.

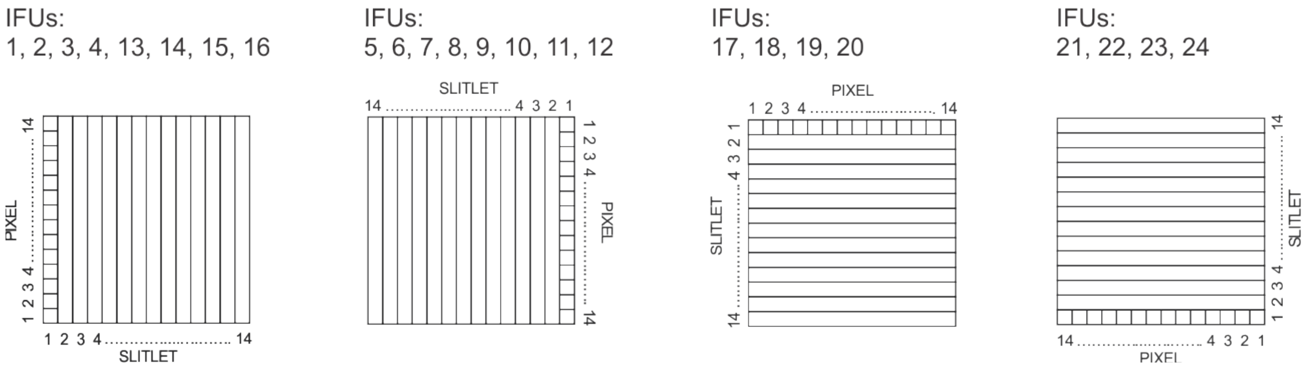


Figure 2: Order (from left to right on the detector) of the spatial pixels and slitlets in each IFU.

If KMOS is oriented to north (`ocs.rot.offangle = 0`), then the IFUs will all have north up and east to the left. If the offset angle is non-zero, then all the IFU fields are rotated by that angle.

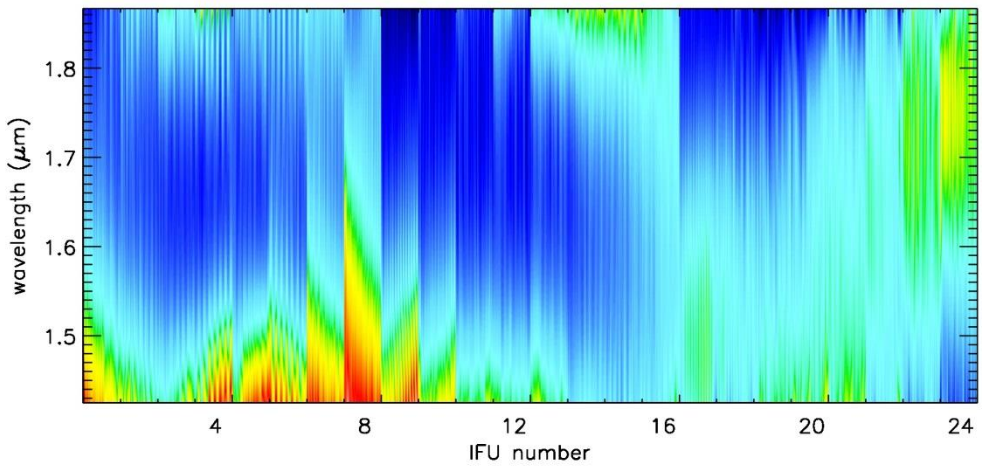


Figure 3: Resolution map of KMOS in H-band as a function of spatial & spectral position (in units of wavelength rather than velocity). Slitlets from the IFUs (labelled 1-24) have been drawn side-by-side. Resolution is indicated by colour from 3A (dark blue) to 6.7A (red).

The image quality across the slitlets is excellent, and is a true representation of the seeing. The image quality along the slitlets is affected by the KMOS optics and adds, in quadrature, about 0.2" to the resolution. This is most noticeable in the best seeing conditions. Note that the image quality of IFUs 23 and 24 is not quite as good as the others; while the spectral resolution (Figure 3) at the short end of IFUs 1-8 is slightly poorer than the rest.

3.4 Impact of Flexure

Like most instruments that have to cope with a changing gravity vector, KMOS has flexure. This has been quantified and amounts to about ±1 pixel both spectrally (shifting the data up and down columns of the detector) and spatially (shifting the data across the rows of the detector). To account for this, standard calibrations are taken at 6 equally-spaced rotator angles. In addition, at the end of each night, calibrations are taken at a set of (at most 6) angles best suited to the observations that have been done. When processing data, the pipeline automatically selects the calibration at the closest available rotator angle to the data. This process is completely transparent to the user.

The residual flexure is relatively small but can still have a measurable impact on the data. This is addressed by the monolithic pipeline (see Section 5.1) using the simple measures described below.

Residual spectral flexure can be measured from the OH lines on each individual frame (as long as the exposure is long enough). The pipeline will fully correct this if an OH line reference spectrum is provided, as described in the options for the `kmo_sci_red` recipe in Section 5.1. Reconstruction with wavelength adjustment is a 2-pass process and so does not lead to additional interpolations for the final data product.

Residual spatial flexure can occur for several reasons: (i) the rotator angle is between those used for calibrations, (ii) the temperature in the cryostat has changed by more than ~1degree since the calibrations were taken, and (iii) the grating has a finite positioning precision. Together, these lead to a non-repeatability of about 0.2 pixel rms, which cannot generally be easily measured (the only exception made in the pipeline is the sky flats, and the `kmo_illumination` recipe described in Section 4.4 does account for it). During reconstruction, the pipeline will interpolate between the calibrations to account for (i) above; and using calibration taken within a day of the science data should always mean that (ii) is negligible.

If (iii) alone is large enough to have a measurable impact on the data (i.e. edge effects, apparent as a slightly brighter or darker row of pixels along the slitlet edges) it can be corrected by the user – for example using OH lines to find the edges of the slitlets on the science data frames and applying a matching offset to the `XCAL` and `YCAL` calibration frames (which then avoids the need for additional interpolations of the science data itself). Note that at the current time, the pipeline does not perform this task.

4 Processing Calibrations

To create a complete set of processed calibrations, the recipes should be executed in the order given below because the later recipes make use of products from earlier ones. In addition, one should use a consistent set of frames (i.e. at least the flat and arc templates should have been executed in a single OB), so that the set of files (sof) grows consistently as one progresses through the recipes. Note that frames that do not belong to a recipe are ignored, so there is no harm in having them propagated.

First set up a 'reduction session' by creating an appropriate directory structure. In the examples shown, the path `/share` is used, and the directories to create are:

`KMOScalib` for static and processed calibration products
`KMOSdata` contains all the raw data files (or links to them)
`KMOSscience` will contain the processed products from the science observations

We recommend creating environment variables `KMOS_CALIB` and `KMOS_DATA` since the first is used in the easySPARK scripts and both of them can anyway be used in manually created sof-files.

Once this is done, copy the static calibrations into `KMOScalib` (or make links to them), move into `KMOScalib`, and run the calibration recipes as described below

HINTS

- Processing calibrations from multiple bands can take several hours. We recommend that the user creates all the 'set of files' (sof) lists first, and then sets all the recipes running overnight. If the directory structure above is followed, and `KMOScalib` is used as the working directory, then all the calibration products will appear there too, ready for the subsequent recipes.
- Alternatively (and because for standard calibrations, the dark, flat, and arc templates are all combined in a single OB) one can simply execute in `KMOScalib`:
  ```
  > easySPARK_calibration.sh /share/KMOSdata/KMOS_SPEC_DARK018_0012.fits
  ```
  where the filename given is the full path of any single frame from that OB. The script will automatically process darks, flats, and arcs (it identifies all the other associated files, makes the necessary sof lists, and executes the recipes).

4.1 Darks

The easiest way to process dark frames is to execute:

```
> easySPARK_dark.sh /share/KMOSdata/KMOS_SPEC_DARK018_0012.fits
```

where the filename given is the full path of any single dark exposure of the appropriate exposure time. The script will automatically identify the other relevant files from that template, generate the sof list, and execute the recipe. Note that to get help about the script simply execute

```
> easySPARK_dark.sh
```

on its own. Or to just generate the sof list, add `sof` as a parameter:

```
> easySPARK_dark.sh /share/KMOSdata/KMOS_SPEC_DARK018_0012.fits sof
```

Alternatively, you can do all this by hand as described below.

Create a file called, for example, `dark_60s.sof` which contains a list of at least several dark exposures with the same exposure time (`det.seq1.dit` and `det.ndit`), together with the identifier `DARK`. The file will look something like this (but typically with 5 `DARK` frames):

```
> cat dark_60s.sof
/share/KMOSdata/KMOS_SPEC_DARK018_0012.fits     DARK
/share/KMOSdata/KMOS_SPEC_DARK018_0013.fits     DARK
/share/KMOSdata/KMOS_SPEC_DARK018_0014.fits     DARK
```

If you have set the environment variable `KMOS_DATA` then this can also be written as

```
$KMOS_DATA/KMOS_SPEC_DARK018_0012.fits    DARK
$KMOS_DATA/KMOS_SPEC_DARK018_0013.fits    DARK
$KMOS_DATA/KMOS_SPEC_DARK018_0014.fits    DARK
```

Then execute the `kmo_dark` recipe:

```
> esorex kmo_dark dark_60s.sof
```

This will create `master_dark.fits`, and a preliminary bad pixel mask `badpixel_dark.fits` that is used by `kmo_flat`. If you want the exposure time appended to the output file name then execute the recipe with an extra parameter:

```
> esorex kmo_dark -file_extensionSet dark_60s.sof
```

HINTS

- If you want to rename the files in a different way, you should do this yourself
- Dark frames can be identified either from the file name or from the `dpr.type` keyword as `DARK`
- Ignore the `ins.grat[1-3].id` keyword in dark frames – it has no meaning for them. Having said this, dark frames can be reconstructed even though there is no associated waveband (see Section 7).
- The parameters `pos_bad_pix_rej` and `neg_bad_pix_rej` can be used to adjust the sigma level at which pixels are flagged as bad; e.g.
  ```
  > esorex kmo_dark -pos_bad_pix_rej=25 dark_60s.sof
  ```
- The dark current is extremely low, ~0.01e-/s.
- The readnoise is lowest (~3.2 ADU) for exposures times in the range 100-600sec; for exposures of 10sec it increases to ~5 ADU.
- We recommend using dark exposures of 60-300sec to identify bad pixels, because the number of bad pixels flagged increases with exposure time up to ~60sec and then stabilises at $48/18/17\times10^3$ for detectors 1/2/3.
- Useful QC parameters include:

  `QC.BADPIX.NCOUNTS`   number of bad pixels (in each extension)

4.2 Flats

As before, the easiest way to process flatfield frames is to execute:

```
> easySPARK_flat.sh /share/KMOSdata/KMOS_SPEC_CAL020_0035.fits
```

where the filename given is the full path of any single flat exposure of the required waveband. The script will automatically identify the other relevant files from the template, generate the sof list, and execute the recipe. Alternatively, you can do it by hand.

Create a sof list containing the lamp on and lamp off flatfield frames, as well as the preliminary bad pixel mask – together with their identifiers. Note that due to flexure, flats and arcs are usually taken at 6 rotator angles so the list may be long. To make a list of the appropriate raw frames with relevant information, use a command like

```
> dfits KMOS*CAL*fits | fitsort tpl.id ins.filt1.id det.seq1.dit ocs.rot.naangle | grep calunitflat
```

The resulting sof may look like this (but with 3 `FLAT_OFF` and 18 `FLAT_ON` frames)

```
> cat flat_k.sof
/share/KMOScalib/badpixel_dark.fits           BADPIXEL_DARK
/share/KMOSdata/KMOS_SPEC_CAL020_0020.fits    FLAT_OFF
/share/KMOSdata/KMOS_SPEC_CAL020_0035.fits    FLAT_ON
```

Note that because `KMOScalib` is the working directory, and the order of the files in the sof list does not matter, `flat_k.sof` could also look like this:

```
/share/KMOSdata/KMOS_SPEC_CAL020_0035.fits    FLAT_ON
badpixel_dark.fits                            BADPIXEL_DARK
/share/KMOSdata/KMOS_SPEC_CAL020_0020.fits    FLAT_OFF
```

Then execute the `kmo_flat` recipe:

```
> esorex kmo_flat flat_k.sof
```

Five files will be produced, which are tagged with the waveband used. The waveband tag is repeated 3 times, once for each of the instrument segments.

`master_flat_###.fits`	Normalised flatfields, typically with 36 extensions (data and noise frames for each detector, and for 6 rotator angles).
`xcal_###.fits,` `ycal_###.fits`	Frames containing the x and y coordinates within an IFU (an integer given in milli-arcsec from the centre of that IFU field) for every illuminated pixel on the detector. The number after the decimal point is the IFU identification. These frames have 18 extensions (3 detectors, 6 rotator angles).
`badpixel_flat_###.fits`	Map of bad or non-illuminated pixels (18 extensions as above).
`flat_edge_###.fits`	Coefficients of fits to the left and right edges of the slitlets in the IFUs (144 extensions for 24 IFUs and 6 rotator angles).

HINTS

- You should not need to rename any of these files.
- Flatfield frames can be identified with the `dpr.type` keyword as `FLAT,LAMP` and `FLAT,OFF`
- Make sure you use frames taken together as a set, rather than mixing data from different dates.
- It is planned that the flat and arc calibrations taken after a night will match the rotation angles used during that night. To get the best results, one should use this matching set of flats and arcs to process the data.
- The flatfield illumination is not uniform, and so it is recommended to include an illumination correction when processing science data (see Section 4.4).
- The badpixel mask created from the flatfield frames (`badpixel_flat_###.fits`) includes also all non-illuminated pixels and so will be of order 90000.
- Useful QC parameters include:

`QC.FLAT.SAT.NCOUNTS`	number of saturated pixels
`QC.FLAT.SN`	mean signal-to-noise of illuminated regions
`QC.SLIT.MEAN`	mean slit width in pixels

4.3 Arcs

Again, the easiest way to process arc frames is to execute:

```
> easySPARK_wave_cal.sh /share/KMOSdata/KMOS_SPEC_CAL020_0130.fits
```

where the filename given is the full path of any single arc exposure of the required waveband. The script will automatically identify the other relevant files from the template, generate the sof list, and execute the recipe. How to do this manually is described below.

Create a sof list containing the on/off arc-lamp frames, together with the required calibration products produced by `kmo_flat` and a few static calibration files. To make a list of the appropriate raw frames with relevant information, use a command like

```
> dfits KMOS*CAL*fits | fitsort tpl.id ins.filt1.id det.seq1.dit ocs.rot.naangle | grep cal_wave
```

The resulting sof may look like this (but with 6 `ARC_ON` frames for the 6 rotator angles). The order of the files does not matter, but they must be tagged correctly.

```
> cat arc_iz.sof
/share/KMOSdata/KMOS_SPEC_CAL020_0125.fits    ARC_OFF
/share/KMOSdata/KMOS_SPEC_CAL020_0130.fits    ARC_ON
/share/KMOScalib/badpixel_flat_IZIZIZ.fits    BADPIXEL_FLAT
```

```
/share/KMOScalib/flat_edge_IZIZIZ.fits        FLAT_EDGE
/share/KMOScalib/master_flat_IZIZIZ.fits      MASTER_FLAT
/share/KMOScalib/xcal_IZIZIZ.fits             XCAL
/share/KMOScalib/ycal_IZIZIZ.fits             YCAL
/share/KMOScalib/kmos_ar_ne_list_iz.fits      ARC_LIST
/share/KMOScalib/kmos_wave_band.fits          WAVE_BAND
/share/KMOScalib/kmos_wave_ref_table.fits     REF_LINES
```

Then execute the `kmo_wave_cal` recipe:

```
> esorex kmo_wave_cal arc_iz.sof
```

Two files will be produced, which are tagged with the waveband used.

`lcal_###.fits`	Frame containing wavelength (in microns) for every illuminated pixel on the detector. This frame has 18 extensions (3 detectors, 6 rotator angles).
`det_img_wave_###.fits`	Reconstructed arc-lamp frame, reformatted so that slitlets and IFUs are side-by-side (a pseudo detector image). This is wavelength calibrated, so arc lines should exactly follow the rows, allowing one to quickly and easily verify that the recipe has been successful.

HINTS

- Arclamp frames can be identified with the `dpr.type` keyword as `WAVE,LAMP` and `WAVE,OFF`
- Typically arcs are taken together with flats, and we strongly recommend using those frames – it is very important to ensure you have a consistent set of calibration products.
- The order of the files in the sof list does not matter; and it is not necessary to specify the full path for files that are in the working directory. One can also make use of environment variables instead of writing out the full path each time.
- Useful QC parameters include:

`QC.ARC.SAT.NCOUNTS`	number of saturated pixels
`QC.ARC.AR.POS.MEAN`	mean offset (in km/s) of reference argon line
`QC.ARC.AR.FWHM.MEAN`	mean FWHM (in km/s) of reference argon line
`QC.ARC.NE.POS.MEAN`	mean offset (in km/s) of reference neon line
`QC.ARC.NE.FWHM.MEAN`	mean FWHM (in km/s) of reference neon line

4.4 Illumination Correction

This too can be done with a single command with one of the appropriate files given as a parameter:

```
> easySPARK_illumination.sh /share/KMOSdata/KMOS_SPEC_CAL018_0209.fits
```

but is described more fully below.

Prepare a sof list containing the sky-flat frames, together with suitable dark frames, and the required calibration files. To make a list of the appropriate raw frames with relevant information, use a command like

```
> dfits KMOS*CAL*fits | fitsort tpl.id ins.filt1.id det.seq1.dit | grep skyflat
```

The `MASTER_FLAT` and `XCAL/YCAL/LCAL` files should match the wavelength of the sky flats. The resulting sof may look like this (but with typically 3 `SKY_FLAT` frames):

```
> cat skyflat_h.sof
/share/KMOSdata/KMOS_SPEC_CAL018_0209.fits      FLAT_SKY
/share/KMOScalib/master_dark.fits               MASTER_DARK
/share/KMOScalib/master_flat_HHH.fits           MASTER_FLAT
/share/KMOScalib/flat_edge_HHH.fits             FLAT_EDGE
/share/KMOScalib/xcal_HHH.fits                  XCAL
```

```
/share/KMOScalib/ycal_HHH.fits              YCAL
/share/KMOScalib/lcal_HHH.fits              LCAL
/share/KMOScalib/kmos_wave_band.fits        WAVE_BAND
```

Then execute the `kmo_illumination` recipe:

```
> esorex kmo_illumination skyflat_h.sof
```

One file will be produced, which is tagged with the waveband used.

`illum_corr_###.fits`	Frame containing images of the internal flatfield uniformity for each IFU. This frame has 48 extensions (data and noise for each of the 24 IFUs.)

HINTS

- Skyflat frames can be identified with the `dpr.type` keyword as `FLAT,SKY`.
- The first sky flat in a series is a test exposure to set the integration time. The subsequent 3 are the ones to use. Check that the count levels in the raw data are at least several hundred (and ideally more than 1000 cts per pixel) for a significant fraction of the spectral traces.
- The rotator angle does not matter since the flatfield spatial uniformity is independent of the orientation of the KMOS instrument.
- The parameter `range` can be used to specify a particular (set of) wavelength range(s) over which the illumination correction should be derived, e.g.
  ```
  > esorex kmo_illumination -range='1.50,1.75' skyflat_h.sof
  ```
  But the default ranges ought to be fine.
- It is important to include the `FLAT_EDGE` frame because, in order to minimize edge effects, the recipe shifts the data on the raw frame so the slitlet edges match those of the flatfield. Without this frame, the cross-correlation cannot be done.
- Useful QC parameters include:

`QC.SPAT.UNIF`	RMS spatial uniformity of the internal flatfield

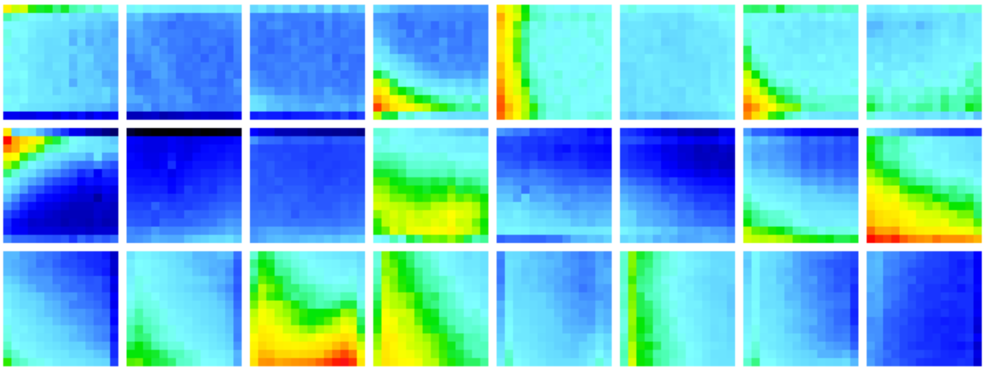


Figure 4: Images portraying the illumination correction in H-band (note that this was measured before adjusting the arms to their final positions and so may look different to what you find). Given the large gradients across some IFUs, this is definitely worth applying. The calibration positions of the arms have now been adjusted so that the gradients are much smaller than shown here.

4.5 Standard Stars

The final calibration is the standard star, and the recipe for this is basically a full science reduction (as in Section 5.1) with some extra bits added on. Reduction is the same for both the `KMOS_spec_cal_stdstar` and `KMOS_spec_cal_stdstarscipatt` templates. The only difference here is whether 3 or 24 IFUs are processed – and therefore whether there are 3 or 24 raw files (flagged as

STD) in the sof list. Note that at least 2 STD frames are required to enable sky subtraction, but the recipe will also work if only one STD frame is provided.

There is a script available also for this recipe, which can be executed in the usual way:

```
> easySPARK_std_star.sh /share/KMOSdata/KMOS_SPEC_CAL333_0023.fits
```

If you wish to do this manually, begin by making the sof list, which will look something like this:

```
> cat std_hip012345_k.sof
/share/KMOSdata/KMOS_SPEC_CAL333_0023.fits      STD
/share/KMOSdata/KMOS_SPEC_CAL333_0024.fits      STD
/share/KMOSdata/KMOS_SPEC_CAL333_0025.fits      STD
/share/KMOScalib/xcal_KKK.fits                  XCAL
/share/KMOScalib/ycal_KKK.fits                  YCAL
/share/KMOScalib/lcal_KKK.fits                  LCAL
/share/KMOScalib/master_flat_KKK.fits           MASTER_FLAT
/share/KMOScalib/illum_cor_KKK.fits             ILLUM_CORR
/share/KMOScalib/kmos_wave_band.fits            WAVE_BAND
/share/KMOScalib/kmos_solar_k_1700.fits         SOLAR_SPEC
/share/KMOScalib/kmos_atmos_k.fits              ATMOS_MODEL
/share/KMOScalib/kmos_spec_type.fits            SPEC_TYPE_LOOKUP
```

Note that the last 3 lines are not mandatory. If they are excluded, the recipe will not create a telluric spectrum. The situations for which they can be used, and their impact, are described in Section 4.5.2.

Execute the kmo_std_star recipe:

```
> esorex kmo_std_star –save-cubes std_hip012345_k.sof
```

When the save-cubes option is set, the reduced cubes will be written to file, so that 5 files are created. Writing out the cubes allows you to extract the spectra yourself if, for example, you want to use a different aperture or to do additional cosmetic cleaning on the cubes.

std_cube_###.fits	Frame containing cubes of the standard star. There are 48 extensions (data and noise for 24 IFUs), but not necessarily all will contain data.
std_image_###.fits	Collapsed images of the standard stars (24 extensions). Only the extensions for IFUs used to observe the star will contain data.
std_mask_###.fits	Spatial masks showing which pixels were used to extract the spectra (which are those within the FWHM).
star_spec_###.fits	The extracted (integrated) spectra. No scaling is applied here, so the units are the total counts measured in the exposure time used.
telluric_###.fits	The derived telluric correction spectrum – see Section 4.5.2 for details. Always check this to make sure you are satisfied with the correction of the stellar features. In some cases, it may require additional interactive work.

HINTS

- Standard star frames can be identified with the dpr.type keyword as OBJECT,SKY,STD,FLUX (all one designation).
- The recipe selects a sky exposure for each IFU independently (the closest in time to the star exposure), from among the exposures given in the sof list. This is reported in the output text, which is also saved in the esorex.log file.
- The recipe will choose the calibrations at the closest available rotator angle (OBS.ROT.NAANGLE) to the observations.

- Reconstruction is done using cubic-spline method (see Section 7.1) by default. We would recommend not changing this for the standard star.
- Flux calibration (see Section 4.5.1) is performed using the total flux in the IFU; but the extracted spectrum – from which the telluric spectrum is made – is integrated only from pixels within the measured FWHM (which encloses about half the total flux). If this doesn't look good, you can make a new extraction using `kmo_extract_spec`.
- Useful QC parameters include:
  - `QC.SPAT.RES` FWHM of the star in arcsec for each IFU (in `std_image_###.fits`)
  - `QC.ZPOINT` zeropoint for each IFU (in `star_spec_###.fits`) – see Section 4.5.1

4.5.1 Flux Calibration

If a magnitude for the star is given, the same recipe will perform a flux calibration and calculate the zeropoint. The magnitude should match the band used for the observations and can be set in the template using P2PP, in which case this calculation is done automatically. You can check this by looking for the magnitude keyword:

```
> dfits KMOS_SPEC_CAL026_0029.fits | fitsort ocs.stdstar.mag
```

Alternatively, it can be set as a parameter when executing the recipe (which will override the magnitude keyword):

```
> esorex kmo_std_star -save-cubes -mag=6.61 std_hip012345_k.sof
```

Note that for the HK band, the magnitudes for both bands should be given (H first, K second) separated by a comma with no spaces:

```
> esorex kmo_std_star -save-cubes -mag='6.71,6.61' std_hip012345_hk.sof
```

The zeropoint is written as the QC parameter `QC.ZPOINT` and is defined so that

$$\mathrm{mag} = \mathrm{qc.zpoint} - 2.5\log_{10}(\mathrm{cts/sec})$$

where `mag` is the magnitude of a source that has a mean count rate of `cts/sec` per spectral pixel. You can then convert the magnitude to a flux density. Putting these steps together you have

$$\mathrm{flux\ density} = \mathrm{cts/sec} \times F_0 \times 10\hat{}[-0.4 \times \mathrm{qc.zpoint}]$$

where F_0 is the zero magnitude flux density taken from the table below in whichever units are preferred. If you want a line flux, integrate the counts over the line, convert the result to a flux density, and multiply it by the spectral size of a pixel (given by the `CDELT3` keyword in the cubes or the `CDELT1` keyword in the extracted spectra).

Near-infrared magnitudes for stars are widely available from 2MASS. And so for estimating the zeropoint in the YJ, J, HK, and K bands, the 2MASS bandpasses are used. In addition, 2MASS zero magnitude flux densities are used for the throughput estimates. These are taken from Cohen, Wheaton, & Megeath (2003; AJ, 126, 1090). Since the *z* band is poorly defined, for the IZ band we use a pseudo-monochromatic 1μm flux density. One way to estimate this is to interpolate it from the KHJIR magnitudes, where the latter 2 come from the USNO-B1 catalogue. The parameters used for KMOS are summarised in the table below.

KMOS band	*2MASS band*	*Band pass for calibration*	*Zero magnitude flux density*	
K	K	2.028 –2.290 μm	4.283×10^{-10} W/m^2/μm	4.65×10^{9} ph/s/m^2/μm
HK	H & K	1.5365 – 1.7875 μm +	1.133×10^{-9} W/m^2/μm &	9.47×10^{9} ph/s/m^2/μm &

		2.028 - 2.290 μm	4.283×10^{-10} W/m^2/μm	4.65×10^{9} ph/s/m^2/μm
H	H	1.5365 – 1.7875 μm	1.133×10^{-9} W/m^2/μm	9.47×10^{9} ph/s/m^2/μm
YJ	J	1.154 – 1.316 μm	3.129×10^{-9} W/m^2/μm	1.944×10^{9} ph/s/m^2/μm
IZ	—	0.985 – 1.000 μm		3.81×10^{10} ph/s/m^2/μm

4.5.2 Telluric Calibration

If the spectral type is provided then it may be possible to create a normalised telluric spectrum. This can be set in the template using P2PP. You can check whether that has been done by looking for the spectral type keyword:

```
> dfits KMOS_SPEC_CAL026_0029.fits | fitsort ocs.stdstar.type
```

Alternatively, it can be set as a parameter when executing the recipe:

```
> esorex kmo_std_star –save-cubes –startype='B8III' std_hip022112_k.sof
```

There are only a limited number of cases for which this software attempts to make a telluric spectrum:

G (ideally G2V) stars in the H, HK, or K bands: the recipe will divide out a solar spectrum and correct for the blackbody temperature associated with the spectral type.

O, B, A, and F stars in any band: the recipe will fit and subtract the strongest H absorption lines (making use of an approximate atmospheric model to help). This works best if there are no more than a few lines across the band; so be aware that if there are many lines close together, the result from this non-interactive procedure is unlikely to be satisfactory.

To process the data for G stars, you need to include the following 2 lines in the sof list:

```
/share/KMOScalib/kmos_solar_h_2400.fits   SOLAR_SPEC
/share/KMOScalib/kmos_spec_type.fits      SPEC_TYPE_LOOKUP
```

To process the data for OBAF stars, you need to include the following 2 lines in the sof list:

```
/share/KMOScalib/kmos_atmos_k.fits        ATMOS_MODEL
/share/KMOScalib/kmos_spec_type.fits      SPEC_TYPE_LOOKUP
```

If you decide to try this, the recipe will create an additional product:

`telluric_###.fits` The normalised telluric spectrum. Always check this to make sure you are satisfied with the correction of the stellar features. In some cases, additional interactive work may be required.

5 Science Reduction

Once you have a full set of calibrations, you are ready to process the science frames. This can be done either using the monolithic pipeline (Section 5.1) which is the most straightforward way; or using the recipes one at a time (5.2) which gives more flexibility, as well as enabling you to use your own routines – which may be extra processing steps or replacements for pipeline recipes.

Before starting work on the science observations, move into `KMOSscience`, which is now your working directory.

5.1 Monolithic pipeline

This recipe performs all the standard processing steps: sky subtraction, flat fielding, illumination correction, reconstruction, telluric correction, shifting, and finally combining. It is straightforward to use, and also allows the user some degree of flexibility. To use this recipe, the first step is, as for the calibrations, to create a sof list.

This can be done with an easySPARK script:

```
> easySPARK_sci_red.sh /share/KMOSdata/KMOS_SPEC_OBS023_0006.fits sof
```

which will make an sof list called `sci_red_###.sof` (where `###` is the date/time in the `TPL.START` keyword) that includes all the science exposures from the same template as the frame given. Omitting the `sof` parameter will also allow the script to execute the recipe. But you may want to check the sof list first, rename it, or add additional observations from other OBs.

The sof list will look something like that here, but most likely with more `science` frames:

```
> cat sci_obs.sof
/share/KMOSdata/KMOS_SPEC_OBS023_0006.fits      SCIENCE
/share/KMOSdata/KMOS_SPEC_OBS023_0007.fits      SCIENCE
/share/KMOScalib/xcal_YJYJYJ.fits               XCAL
/share/KMOScalib/ycal_YJYJYJ.fits               YCAL
/share/KMOScalib/lcal_YJYJYJ.fits               LCAL
/share/KMOScalib/master_flat_YJYJYJ.fits        MASTER_FLAT
/share/KMOScalib/illum_cor_YJYJYJ.fits          ILLUM_CORR
/share/KMOScalib/telluric_YJYJYJ.fits           TELLURIC
/share/KMOScalib/kmos_wave_band.fits            WAVE_BAND
```

All the observations are called `SCIENCE`, with no differentiation between sky and object. This is because any particular frame may include both object and sky data, depending on the arm assignments.

The `ILLUM_CORR` and `TELLURIC` files are optional. If `ILLUM_CORR` is omitted, there will be no correction for spatial uniformity of the internal flatfield; if `TELLURIC` is omitted, there can be no correction for atmospheric transmission.

Execute the `kmo_sci_red` recipe:

```
> esorex kmo_sci_red sci_obs.sof
```

There are 2 sets of output:

`sci_reconstructed_###.fits`	Processed and reconstructed cubes, matching the input `SCIENCE` files. The tag is the name of the input file. Only input files with at least one object or reference IFU (`OCS.ARMi.TYPE='O'` or `'R'`) will appear as an output file; and in these files, only object or reference IFUs will be processed, the other extensions will be empty.
`sci_combined_#####.fits`	A set of files containing the final combined cubes (1 data and 1 noise extension in each), constructed by shifting and combining the data for each IFU. There is one output file for each named object or reference source found in the input `SCIENCE` files, and this name is used as the tag.

OPTIONS

Since this is a work-horse recipe, there are a number of options which you may find useful (see also Section 5.3). These can be used together if it is appropriate.

- The `pix_scale` parameter allows you to set the spatial pixel scale for the reconstructed cube. The default (natural scaling) is 0.2arcsec, but any scale can be set. The example here is what you may want to use if, in the observing template, you set the dithering pattern to be at half-pixel offsets:
  ```
  > esorex kmo_sci_red -pix_scale=0.1 sci_obs.sof
  ```

Don't forget to also create the illumination correction frame at the same pixel scale.

- The `no_combine` parameter will stop the recipe after the `sci_reconstructed_###.fits` frames have been created. It suppresses the creation of combined cubes.
`> esorex kmo_sci_red -no_combine sci_obs.sof`

- The `no_subtract` option will process each `SCIENCE` frame given in the sof independently of the others, without looking for or subtracting any sky exposures. With this option, all active IFUs (including sky IFUs) will be processed and reconstructed. In addition, the `no_combine` option is also implicitly set. This is the default behaviour if only 1 `SCIENCE` frame is listed in the sof.
`> esorex kmo_sci_red -no_subtract sci_obs.sof`

- If you are interested in only 1 object or only in specific IFUs, and want to save time processing, these can be specified as parameters.
`> esorex kmo_sci_red -name='gal21' sci_obs.sof`
will process only IFUs labelled with `OCS.ARM[1-24].NAME='gal21'`.
`> esorex kmo_sci_red -ifus="3;14;3;14" sci_obs.sof`
will process only IFU 3 from the 1st `SCIENCE` frame, IFU 14 from the 2nd, IFU 3 again from the 3rd, and IFU 14 again from the 4th. In this example, there must be exactly 4 `SCIENCE` exposures given. In both cases, sky frames are identified as before.

- Various options are available for specifying the offsets when shifting the cubes (see Section 5.2 for details). This is done with the `method` parameter, which can be set to `'header'`, `'none'`, `'center'`, or `'user'`. For example, if you are processing data taken at different times, and the sources are clearly visible in individual exposures, you may prefer to derive shifts from the sources themselves. In this case you might try:
`> esorex kmo_sci_red -method='center' sci_obs.sof`

- Setting the `edge_nan` parameter will, as part of the shift-and-combine stage, set the single row or column of pixels at each edge of the slitlets to be `NaN`. This is an effective way of avoiding 'edge effects'. Because (see also 5.3.4) this is done as part of the `kmo_combine` recipe, you will not see any change in the `sci_reconstructed_###.fits` frames.
`> esorex kmo_sci_red -edge_nan sci_obs.sof`

- Setting the background parameter will, as an additional step before combining frames, subtract a single constant value from each IFU. This is calculated as the mode of the pixel values in the cube after excluding the brightest 25%, and is the best approximation to a uniform background level that can be made from the data itself. You should use your judgement about whether this is appropriate for your data (e.g. whether the source is too extended for this to yield a reliable estimate).
`> esorex kmo_sci_red -background sci_obs.sof`

- Flux conservation is not applied during interpolations unless the parameter `flux` is explicitly set (due to sky frames typically being subtracted before reconstruction, and complications arising from the changing background level). In extreme cases this can make as much as 10% difference to the derived fluxes, so you should consider using it. But note that it will be anyway disabled for any cubes in which the total flux is not sufficiently greater than the noise. The flux is calculated simply as the total counts in each cube, and it can be done with or without the `background` option (i.e. the flux is calculated with/without background subtraction, as described above, from both the input and output data).
`> esorex kmo_sci_red -flux sci_obs.sof`

- Wavelength correction may be necessary to account for spectral flexure in order to, for example, subtract OH lines well. This has been implemented in the `kmo_reconstruct` recipe, and the action propagated into `kmo_sci_red`. No parameter needs to be set, but the appropriate static calibration file (`kmos_oh_spec_#.fits`) should just be included in the sof list and tagged as `OH_SPEC`. For each reconstruction, the recipe will derive and apply a modification to `lcal_###.fits` based on the measured wavelengths of prominent OH lines (see Davies et al. 2007: MNRAS, 375, 1099). This is done internally, and the file itself remains unchanged. It requires a double-pass (a preliminary reconstruction is done to derive the wavelength offset, and then the proper reconstruction is done using the corrected calibration) so that each product has been interpolated only once. With `kmo_sci_red`, this only makes sense if combined with the `no_subtract` option because object and sky frames will most likely require different corrections:

```
> esorex kmo_sci_red -no_subtract sci_obs.sof
```

where the sof list may now look like:

```
> cat sci_obs.sof
/share/KMOSdata/KMOS_SPEC_OBS023_0006.fits    SCIENCE
/share/KMOSdata/KMOS_SPEC_OBS023_0007.fits    SCIENCE
/share/KMOScalib/kmos_oh_spec_YJYJYJ.fits     OH_SPEC
/share/KMOScalib/xcal_YJYJYJ.fits             XCAL
/share/KMOScalib/ycal_YJYJYJ.fits             YCAL
/share/KMOScalib/lcal_YJYJYJ.fits             LCAL
/share/KMOScalib/master_flat_YJYJYJ.fits      MASTER_FLAT
/share/KMOScalib/illum_cor_YJYJYJ.fits        ILLUM_CORR
/share/KMOScalib/telluric_YJYJYJ.fits         TELLURIC
/share/KMOScalib/kmos_wave_band.fits          WAVE_BAND
```

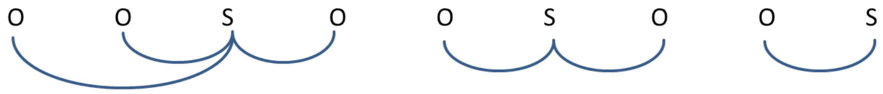
Future release

- Enhanced OH removal, which will require a new recipe.

As always, if any of these options doesn't work as expected, please let us know.

HINTS

- Science frames can be identified with the `dpr.type` keyword as `OBJECT,SKY` (all one designation).
- For each IFU, object and sky exposures are distinguished by the `OCS.ARMi.TYPE` keyword. For each object exposure in an IFU, the pipeline automatically selects the sky exposure in the same IFU taken closest in time (and reports this in the log). An example of this way of deciding the Object/Sky pair assignments for a sequence of frames is:



  This is not necessarily what one might choose, so it will soon be possible for the user to edit this 'object/sky assignment table'. Note that the templates provide sequences such as: OS OS OS, OSO OSO OSO, OOSOO OOSOO, etc.
- Only frames tagged with `OCS.ARMi.TYPE = O` (object) or `R` (reference source) are reconstructed by the monolithic pipeline; sky frames (`S`) are not. The exception is if only a single `SCIENCE` frame is listed in the sof, or if the `no_subtract` option (described above) is set. Note also that using `kmo_reconstruct` directly will reconstruct all IFUs regardless of their tag.
- The default interpolation method is cubic spline. Depending on the circumstances, other methods may be preferred – see Section 7.1.
- The parameters and input files used by this, or any other, recipe to generate the output files can be found by looking for `PRO` keywords in the primary header of the products:

```
> dfits sci_reconstructed_KMOS_SPEC_OBS023_0006.fits | grep PRO
```

5.1.1 Mapping & Mosaics

The mapping modes of KMOS have specific templates to perform the observations. But the data are treated by the pipeline in exactly the same way as for any other science observation. This means that they can be reduced by the monolithic pipeline with the single command

```
> esorex kmo_sci_red sci_obs.sof
```

as given above. However, the user should note that, other than the options already described, the pipeline makes no attempt to perform any matching (scalings, offsets, etc) between the individual IFUs and pointings. This rather complex task is left to the user, since how they are done depends on the individual data set.

It is often useful to know which IFUs in which exposures makes up the various parts of the patchwork mosaic. Figure 5 and Figure 6 show this information for the 8-arm and 24-arm mapping modes respectively.

21	2	3	8
20	15	14	9

A	B	C
D	E	F
G	H	I

Figure 5: Left – Arrangement of the IFUs used for the Mapping8 mosaic mode. Right – order (from A to I) of the 9 dithers performed during the Mapping8 mode. The IFUs are separated by 8.1” and each dither is 2.7” so that, at the end, there is a 0.1” (half-pixel) overlap between adjacent pieces.

23	24	1	3	5	6
21	22	2	4	8	7
19	20	16	14	10	9
18	17	15	13	12	11

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Figure 6: Left – Arrangement of the IFUs used for the Mapping24 mosaic mode. Right – order (from A to P) of the 16 dithers performed during the Mapping24 mode. The IFUs are separated by 10.8” and each dither is 2.7” so that, at the end, there is a 0.1” (half-pixel) overlap between adjacent pieces.

5.2 Work-flow one step at a time

The monolithic pipeline performs the standard steps for a scientific reduction. These steps can be performed one at a time. The following example shows how. But before embarking on this, we recommend you check whether the options available for `kmo_sci_red` will do what you want, since that would be a much easier path to follow.

First find out the Nasmyth angle at which the observations were taken and extract the matching flatfield frames from the `MASTER_FLAT`. Here the -71° (=289°) of the data most closely matches 300° in the calibration frames. The data at this angle are extracted using the recipe `kmo_fits_strip`.

```
> dfits KMOS_SPEC_OBS025_0034.fits | fitsort ocs.rot.naangle
FILE                            OCS.ROT.NAANGLE
KMOS_SPEC_OBS025_0034.fits      -70.724
> esorex kmo_fits_strip -noise=TRUE -angle=300 master_flat_YJYJYJ.fits
> mv strip.fits flat_YJ_300.fits
```

Then for each object frame, subtract the sky, divide by the flatfield, apply the illumination correction, and reconstruct it. The first 3 steps are done using `kmo_arithmetic`. The default output from this is called `arithmetic.fits`. It can either then be renamed using a shell command or, as done here, you can use the `file_extension` parameter to specify a name suffix.

```
> esorex kmo_arithmetic -op='-' -file_extension='34sub35' KMOS_SPEC_OBS025_0034.fits
       KMOS_SPEC_OBS025_0035.fits
> kmo_arithmetic -op='/' -file_extension='divflat' arithmetic_34sub35.fits flat_YJ_300.fits
> kmo_arithmetic -op='/' -file_extension='preproc34' arithmetic_divflat.fits
       illum_corr_YJYJYJ.fits
```

You are now ready to reconstruct the data. This can be done with an easySPARK script (see Section 5.2.1):

```
> easySPARK_reconstruct.sh arithmetic_preproc34.fits
```

Or you can create an sof list (e.g. called `reconstruct_0034.sof`) which contains the following files:

```
arithmetic_preproc34.fits             OBJECT
$KMOS_CALIB/xcal_YJYJYJ.fits          XCAL
$KMOS_CALIB/ycal_YJYJYJ.fits          YCAL
$KMOS_CALIB/lcal_YJYJYJ.fits          LCAL
$KMOS_CALIB/kmos_wave_band.fits       WAVE_BAND
```

And then execute the 2 commands

```
> esorex kmo_reconstruct reconstruct_0034.sof
mv cube_object.fits cube_OBS025_034.fits
```

The method used for the interpolation can be specified using the `method` parameter, for example

```
> esorex kmo_reconstruct -method='CS' reconstruct_0034.sof
```

A short discussion of the interpolation methods available is given in Section 7.1.

Once all the frames have been reconstructed, the cubes can then be combined by listing them in a sof (no tag is required), and executing `kmo_combine`. For this recipe, the default action is to combine objects by name or, if a mapping template was used, to combine all IFUs together. One can instead specify either which IFUs to combine (1 from each file in the sof list) or an object name (the recipe then finds which IFUs have this object name):

```
> esorex kmo_combine -method='header' -name='gal21' -cmethod='median' objectcubes.sof
```

Or you can specify which IFUs to combine by giving a list with a number in the range 1-24 for each frame in the sof list. In the example here, the object is always in IFU 1:

```
> esorex kmo_combine -method='header' -ifus='1;1;1;1;1;1' -cmethod='median' objectcubes.sof
```

The `method` parameter specifies how the dither offsets should be determined; and the `cmethod` parameter indicates how the pixel values should be combined.

However, better results can often be achieved if the user includes a few special routines to optimise some of the steps. Some options are described in Section 5.3.

5.2.1 easySPARK_reconstruct

A shell script `easySPARK_reconstruct.sh` included in the KMOS kit provides a convenient way to reconstruct cubes by automatically generating the required sof list, and renaming the output file. To use it, first make sure the environment variable `KMOS_CALIB` is set to be the full path of the `KMOScalib` directory (in the examples here `/share/KMOScalib`), and check that `easySPARK_reconstruct.sh` is in your path (see Section 2.2.4). Then simply execute a command like:

```
> easySPARK_reconstruct.sh KMOS_OBS025_0103.fits
```

It will create an sof list called `reconstruct_OBS025_035.sof` which contains the following files:

```
KMOS_SPEC_OBS025_0035.fits          OBJECT
$KMOS_CALIB/xcal_YJYJYJ.fits        XCAL
$KMOS_CALIB/ycal_YJYJYJ.fits        YCAL
$KMOS_CALIB/lcal_YJYJYJ.fits        LCAL
$KMOS_CALIB/kmos_wave_band.fits     WAVE_BAND
```

And then perform the 2 commands

```
> esorex kmo_reconstruct reconstruct_OBS025_035.sof
mv cube_object.fits cube_OBS025_035.fits
```

The method used for the interpolation can be specified as a parameter, for example

```
> easySPARK_reconstruct.sh KMOS_OBS025_0103.fits CS
```

This script can also be used on pre-processed data (although watch out for how it re-names the output cube), for example:

```
> easySPARK_reconstruct.sh frame0034sfi.fits
```

5.3 Alternatives & Optimisation

5.3.1 Residual Sky Subtraction

If the atmospheric OH lines do not subtract out well, you will need to use a routine to apply a wavelength dependent scaling to a sky cube to correct the residuals. The method we use is described in Davies 2007 (MNRAS, 375, 1099). We are currently implementing this as a recipe, but at the moment is still only available as an IDL script. Please contact the authors of this document if you wish to use it.

First you need to create a sky cube for all 24 IFUs, by subtracting a matched dark frame, flatfielding the result, applying the illumination correction, and reconstructing. The steps are exactly as described above in Section 5.2. Having done this, run the `skysubxtn.pro` script. Instructions are given with the script.

5.3.2 Wavelength Corrections

Flexure within KMOS as it rotates means that, even when applying the correction above, the OH lines do not always subtract out and can sometimes leave P-Cygni like residuals. If this happens, it is necessary to apply a wavelength correction to the spectral calibration.

The recipe `kmo_reconstruct` is able to apply this correction (and see also the options for `kmo_sci_red` in Section 5.1). When doing so, it reconstructs each cube individually, measures the wavelength offset from the OH lines, applies this internally to the `LCAL` calibration and then does the reconstruction again. This 2-step process means that the final cube has only been interpolated once. To activate this option, simply include the appropriate static calibration file (`kmos_oh_spec_#.fits`) in the sof list and tag it as `OH_SPEC`.

This procedure is also available in the IDL script `skysubxtn.pro` mentioned above (although there it is applied as a correction to the reconstructed cube, and hence involves an additional interpolation).

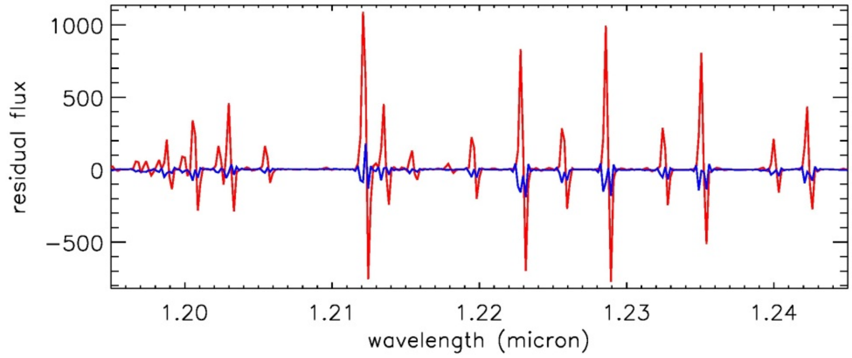


Figure 7: Example of spectrum segment without (red) and with (blue) application of the OH wavelength and scaling corrections described above.

5.3.3 Background Matching

The varying sky brightness may mean that there is a small residual background difference between exposures. If a source within a given IFU is either spatially or spectrally compact, it is possible to measure the background level and correct it.
Until a recipe is implemented, an IDL script called `kmosbkg.pro` is available to do this. Please contact the authors of this document if you wish to use it.

. **CAUTION**

This routine cannot be applied blindly, since its success depends on how much an object fills a data cube. In particular, it will not work with spatially extended continuum sources. The user must decide whether it is appropriate for their data.

5.3.4 Edge Effects

These may become apparent in some circumstances due to a slight mismatch between the position on the detectors of flatfield traces and science data (residual spatial flexure). A simple way to deal with this is to trim off the edges of the slitlets (i.e. the top and bottom rows of IFUs 1-16 and the left and most columns of IFUs 17-24). The easiest way to do this is to set them to `NaN`, and this is included as an option for `kmo_combine` (see also options for `kmo_sci_red` in Section 5.1):

```
> esorex kom_combine -edge_nan -name='gal21' objectcubes.sof
```

Note that using this option for data taken in a mapping mode is not recommended since it will probably result in a grid of `NaN` values outlining each IFU pointing in the combined map.

5.3.5 Improving Cosmetics

For a number of reasons, there may well be many deviant pixels in the reconstructed cubes. These can be effectively cleaned using a 3D version of van Dokkum's L.A.Cosmic routine (van Dokkum P., 2001; PASP, 113, 1420). We note that because of the strong OH lines in the raw data, and the possible

presence of continuum sources, the routine is more effective (and safer to use) when applied to the reconstructed cubes.
An IDL script called `lac3dxtn.pro` is available for this. Please contact the authors of this document if you wish to use it.

CAUTION

While the routine has been tested successfully with its default parameters on a variety of sources, it is the user's responsibility to check that it removes only bad pixels and does not impact the source itself.

6 Other Useful Recipes

The full list of recipes has already been given in Section 2.2.1. Here we highlight a few that might be useful.

6.1 Simple Mathematics

The recipe `kmo_arithmetic` has already been encountered in Section 5.2. It can be used in many situations. A few examples are given here. The output frame is called `arithmetic.fits` by default but can have a suffix added if one uses the `file_extension` parameter.

Subtract a (raw) sky frame from an object frame:
```
> esorex kmo_arithmetic -op='-' objectframe.fits skyframe.fits
```

Divide the spectrum at each spatial position in cube by a telluric spectrum:
```
> esorex kmo_arithmeitc -op='/' cube.fits telluric.fits
```

Add 2 cubes together:
```
> esorex kmo_arithmetic -op='+' cube1.fits cube2.fits
```

Raise a spectrum by some power, to account for differing airmass between object and standard star:
```
> esorex kmo_arithmetic -op='^' telluric.fits 1.1
```

Multiply a cube by a constant:
```
> esorex kmo_arithmetic -op='*' cube1.fits 6.3
```

6.2 Basic Statistics

Basic statistical properties of the data can be calculated using
```
> esorex kmo_stats KMOS_SPEC_CAL022_0004.fits
```

```
<blah>
[ INFO ] kmo_stats: [tid=000] -------------------------------------------
[ INFO ] kmo_stats: [tid=000]                   |DET.1.DATA|DET.2.DATA|DET.3.DATA|
[ INFO ] kmo_stats: [tid=000] 1.  #pixels:      |  4194304 |  4194304 |  4194304 |
[ INFO ] kmo_stats: [tid=000] 2.  #finite pix.: |  4194304 |  4194304 |  4194304 |
[ INFO ] kmo_stats: [tid=000] 3.  mean:         | 100.0005 | 11.13203 | 8.771374 |
[ INFO ] kmo_stats: [tid=000] 4.  stdev:        |  996.045 |    416.4 | 362.2933 |
[ INFO ] kmo_stats: [tid=000] 5.  mean w. rej.: | 1.187879 |-0.063260 |-0.250587 |
[ INFO ] kmo_stats: [tid=000] 6.  stdev w. rej.:| 1.997403 | 1.518527 | 1.530504 |
[ INFO ] kmo_stats: [tid=000] 7.  median:       | 1.416667 |0.0133333 |-0.186666 |
[ INFO ] kmo_stats: [tid=000] 8.  mode:         |0.7498566 |-0.136713 |-0.317835 |
```

```
[ INFO ] kmo_stats: [tid=000] 9.  noise est.:   | 1.592248 | 1.447667 | 1.460024 |
[ INFO ] kmo_stats: [tid=000] 10. min. value:   |-385.6167 |  -271.98 |-2016.683 |
[ INFO ] kmo_stats: [tid=000] 11. max. value:   | 107723.6 |  95957.1 | 110734.2 |
[ INFO ] kmo_stats: [tid=000] ------------------------------------------
```

Obviously, if one uses this on a reconstructed cube, there will be 24 or 48 columns of data. Since the lines will wrap, this is going to be tricky to follow. So the data are written into a fits file called `stats.fits` which has extensions to match the input file. And one can also re-direct the output to a text file:

```
> esorex kmo_stats cube_object.fits > cube_object_stats.txt
```

6.3 Make Images

The recipe `kmo_make_image` allows one to combine spectral slices of a cube (collapse the cube) to make an image. It is possible to specify one or more spectral ranges to use; and if an OH spectrum is provided, one can specify that spectral regions close to bright OH lines should be omitted. As an example, the command

```
> esorex kmo_make_image -range='1.52,1.54;1.57,1.59' cube_OBS022_0049_NN.fits
```

will create the file `make_image.fits` from `cube_OBS022_0049_NN.fits`, using data within the spectral ranges 1.52-1.54µm and 1.57-1.59µm.

6.4 Extract Spectra

The recipe `kmo_extract_spec` allows one to extract a spectrum from each cube in a file. Several methods are available to integrate the pixels: one can provide a spatial mask (which is multiplied into each spectral slice of the cube before spatially integrating the result); one can request that such a mask is generated automatically from the data; or one can specify a circular aperture (pixels whose centres lie within this aperture are used). These 3 options are specified by the `mask_method` parameter as '`mask`', '`optimal`', or '`integrated`' (the default) respectively. An example could be

```
> esorex kmo_extract_spec -mask_method='optimal' -save-mask cube_OBS022_0049_NN.fits
```

In this example, the recipe creates 2 output files: the extracted spectrum, as well as the mask that was derived from the data and used to generate the spectrum.

Of course, there are other ways to extract spectra – for example, one could keep adding spectra from individual pixel (in order of brightness) until the signal-to-noise stops increasing. The methods here are designed to be simple and flexible.

6.5 Rotate Cubes

While the pipeline can handle shifting and combining data that is not aligned with north, it cannot (yet) deal with data at a variety of offset angles. The recipe `kmo_rotate` can be used to rotate cubes so that they north points in the same direction for all of them, or to rotate cubes so that north points upwards. An example of its usage is:

```
> esorex kmo_rotate -rotations=35 cube_OBS022_0049.fits
```

It is important to note that by default this recipe (and also `kmo_shift`) do not extrapolate. Thus, the spatial extent of the region with finite data values will decrease. If you do not want to this to happen, you can specify the `extrapolate` parameter:

```
> esorex kmo_rotate -rotations=35 -extrapolate cube_OBS022_0049.fits
```

6.6 Copy Cube Sections

To extract a section of a cube, you can use the recipe `kmo_copy`, specifying the starting point and size in each dimension in pixels. To extract a cube covering the same spatial extent, but only a limited wavelength range, one can do:

```
> esorex kmo_copy -z=1500 -zsize=300 cube_OBS022_0049.fits
```

This recipe also has a useful feature that it can strip off any edges that contain just NaN values:

```
> esorex kmo_copy -autocrop cube_OBS022_0049.fits
```

7 Notes on Reconstruction

The `XCAL`, `YCAL`, and `LCAL` files contain all the information necessary to reconstruct a cube. They provide the (x,y,λ) location in the cube of each illuminated detector pixel. The x and y locations are integer distances in milliarcsec along the horizontal and vertical axes from the centre of each IFU field; the λ location along the spectral axis is given in microns. The IFU identification is encoded in the `XCAL` and `YCAL` frames as the number after the decimal point. These 3 files are used to perform reconstruction as described below. You can also use them with your own reconstruction algorithm.

Note: a variety of tags can be used for the file to be reconstructed. For example, dark frames can be reconstructed even though they have no associated waveband. The `DARK` tag will cause the recipe to skip the waveband checks so that it can use the calibrations provided.

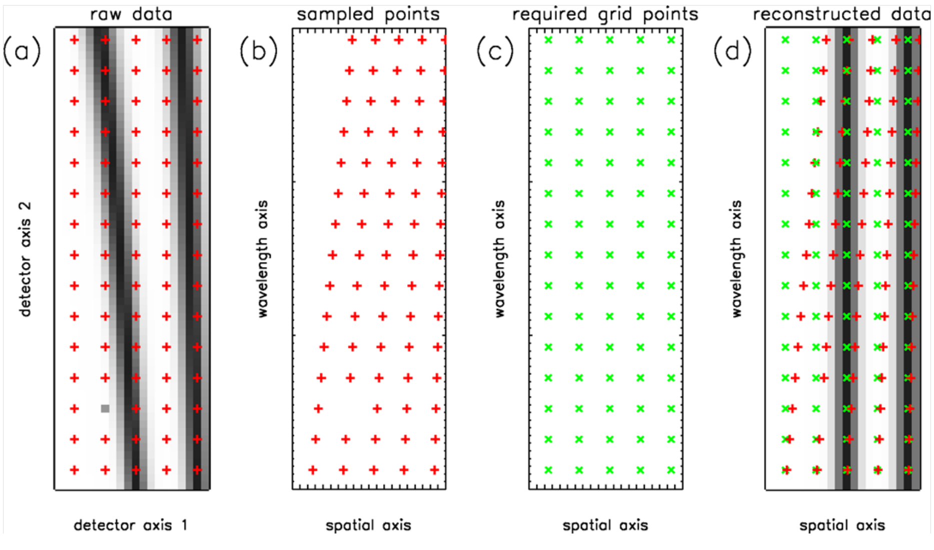


Figure 8: Concept for calibrating and interpolating, illustrated in 2D. (a) Observed data are sampled regularly (red points) on the detector frame; (b) These map to an irregular grid in the reconstructed cube (with bad pixels simply omitted from the set of valid data points); (c) The regular sampling in the final cube (green points) is, in principle, independent of the original sampling on the detector; (d) Each required grid point (green) is interpolated from the neighbouring sample points (red).

7.1 Interpolation Methods

Several interpolation methods are available. But if in doubt, use the default interpolation method which is cubic spline (CS). In this section, we write a few notes about the pros and cons of the various methods.

Nearest neighbour (NN) is a fast but approximate method, simply re-arranging the measured data values without actually performing any interpolation. It may be a good choice for noisy data (since it does not degrade the signal-to-noise at all) when the highest spatial or spectral resolution is not required. But points may be offset up to 0.5 pixels (spatially or spectrally) since this is how far away the neighbour can be – so the fidelity is poor.

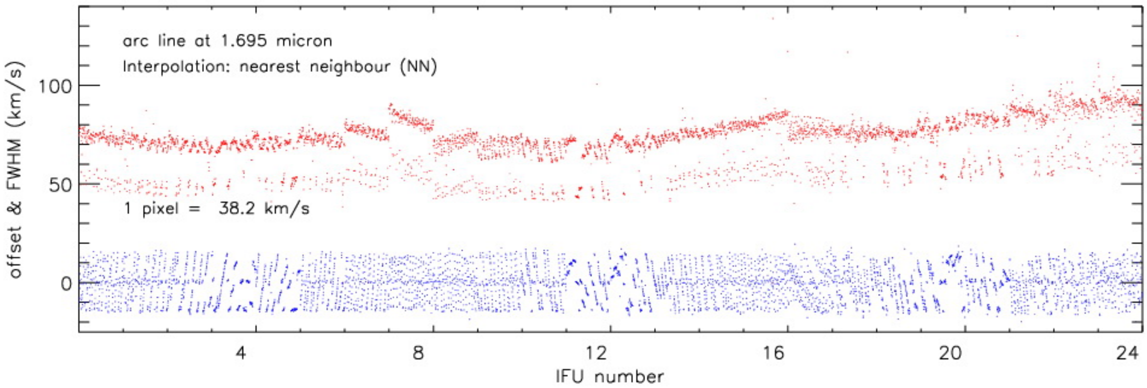


Figure 9: Spectral fidelity of arc-line in the H-band as a function of spatial position across all 24 IFUs, reconstructed using the nearest neighbour. The offset from its nominal position and FWHM are shown. The fainter trace of FWHM at higher resolution are those locations where the undersampled line is almost exactly centered on a pixel. While there can be 0.5pixel offsets in wavelength, this is still relatively small compared to the FWHM resolution. The performance of this method will improve significantly with the multi-reconstruct algorithm described in Section 7.2 which combines several exposures during the reconstruction itself.

Linear Distance Weighted Nearest Neighbours (lwNN) makes use of all pixels within a specified range – up to 26 neighbours for a neighbourhood range of 1 – and combines them, using weights that decrease linearly with distance. It is a simple method with simple error propagation, which yields reasonable results.

Square Distance Weighted Nearest Neighbours (swNN) is similar, but applies a weighting that depends on the inverse square of the distance, truncated at the edge of the neighbourhood range. The Modified Shepard's method (MS) is similar, but scales the weighting to zero at the edge of this range. This can be a good method if the spatial/spectral resolution elements are well enough sampled (e.g. if one combines data from many exposures during the reconstruction). Note that the swNN and MS methods lead to slightly better spectral resolution but slightly poorer wavelength fidelity than lwNN. Drizzling is expected to yield results much like these methods.

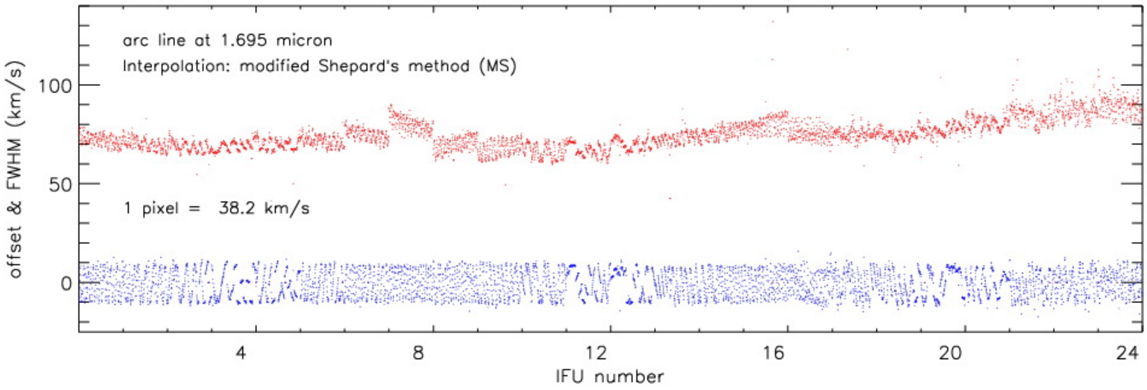


Figure 10: As for Figure 9 but using modified Shepard's method for the interpolation. This is a robust method with simple noise propagation. The 3 similar methods lwNN, swNN, and MS described above have, in this order, improving spectral resolution but decreasing wavelength fidelity. The results of these methods are expected to improve significantly with the multi-reconstruct algorithm described in Section 7.2 which combines several exposures during the reconstruction itself.

Cubic Spline (CS) actually performs 3 successive 1-D interpolations. As such, it is the only method that is not a true 3D interpolation. However, it gives the best results in terms of fidelity – in terms of maintaining the native resolution, and achieving precision. On the other hand error propagation is non-trivial, and is not done for this method.

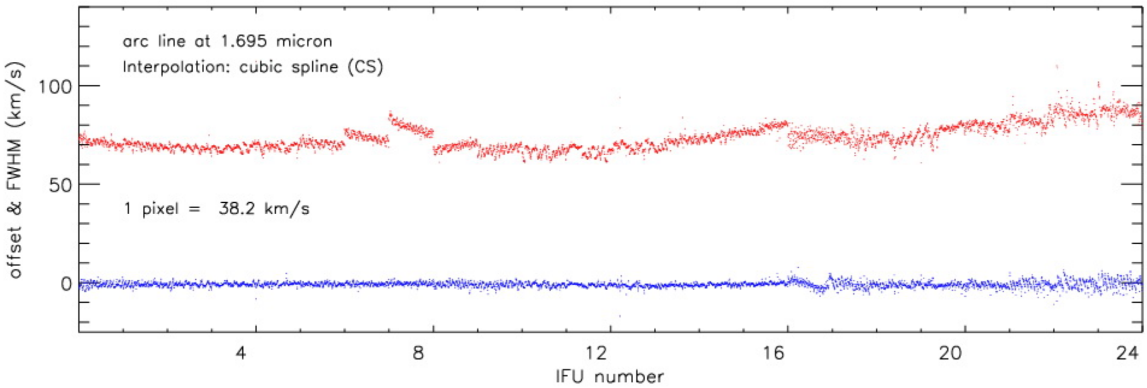


Figure 11: As for Figure 9 but using cubic spline interpolation. This provides the best quality in terms of resolution and wavelength precision, but has complex noise propagation properties.

7.2 Multi-reconstruct

The standard processing steps first reconstruct each cube, and then afterwards shift and combine them. Instead of performing this 2-step process, the calibration files `XCAL/YCAL/LCAL` allow one to put all raw data into a huge 'meta-detector' frame with their respective 'meta-calibrations' and reconstruct the entire dataset in one go. There may be some advantages to doing things this way. Most obviously, it avoids the additional interpolation during sub-pixel shifting, and it makes better use of dithered observations (which is especially important for the true 3D interpolation methods described above). If you want to experiment with this, then use the `kmo_multi_reconstruct` recipe. However, note that this only performs the basic reduction steps included in `kmo_sci_red`; we are working on including the optimisations described in Section 5.3, but this will take time.

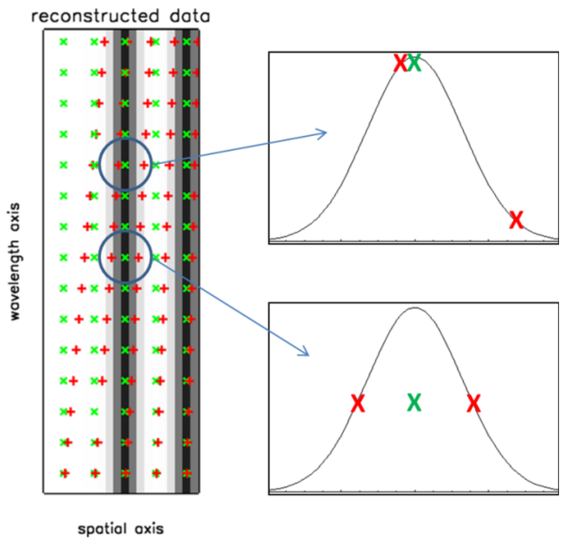


Figure 12: the multi-reconstruct recipe can help to overcome this major disadvantage of 3D interpolations. The upper circle shows a region where the interpolated point is close to a sample point; the lower circle shows the opposite situation where these are far from each other. In both cases, one is trying to interpolate the peak of a Gaussian trace; but the interpolated values are very different. Because 3D methods all make use of weighted sums of neighbouring pixels, the interpolated value can never be greater than the maximum of the surrounding sample points. For spatially compact continuum sources (e.g. stars), this can lead to a slow ripple pattern (or in the worst case several strong discontinuities, along the spectrum) as seen in Figure 14. This feature can only be overcome by better sampling, which is one advantage of the multi-reconstruct recipe.

8 Troubleshooting

In this section, we show a few features we've noticed that we'd rather not have. Some have a simple solution, others are more tricky.

8.1 Detector Readout Channels

Do your reconstructed data have stripes like those shown in Figure 13?

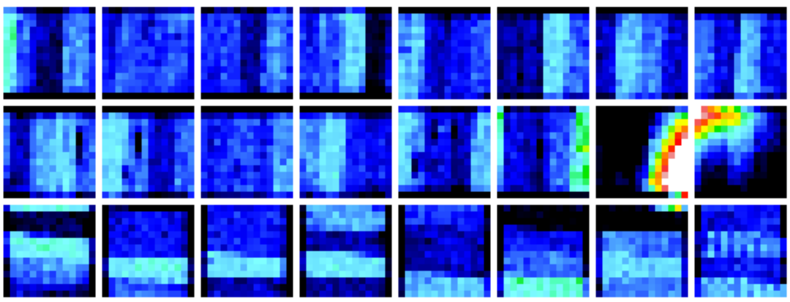


Figure 13: images created by collapsing cubes from one H-band exposure in a mosaic. IFUs 1-8 are from left to right along the top row; IFUs 9-16 along the middle row, and IFUs 17-24 along the bottom row. There are (parts of) sources in only IFUs 15 & 16. The striping effect (with a period of 3-4 slitlets) is apparent in nearly all of the other others. And in IFU 24, one can see an odd-even effect across a few slitlets.

This is caused by temporally variable levels in the read-out channels of the detectors. The effect is only ~1 count or so, but is an issue when observing very faint sources. We have developed an experimental routine to correct for this – but it involves processing the data twice: once so that the effect can be measured; then a second time after it has been corrected (which has to be done as the first step). One also needs to be cautious that any objects in the IFUs are compact so that the effect can be measured properly. If you wish to try the routine, please contact the authors of this document.

8.2 Undersampling

Does your spectrum have a slow ripple pattern superimposed on it, like that shown in Figure 14?

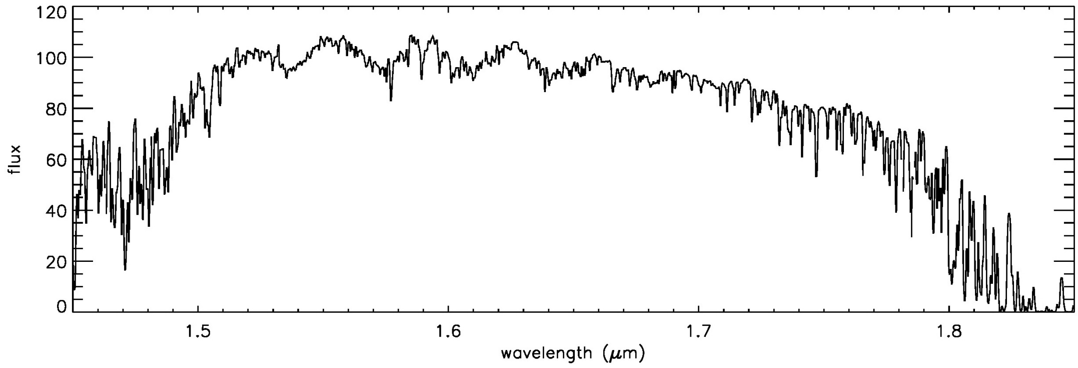


Figure 14: Rather extreme example of the ripples (with a period of 150-200 pixels, or about 0.05μm) that can be seen in stellar spectra if the seeing is so good that the star is spatially undersampled. This example is for cubic spline interpolation in 0.35” seeing. For nearest neighbour reconstruction the effect is more severe and appears as discontinuities. In either case, it can only be avoided by better sampling – which is what the multi-reconstruct recipe provides (see Section 7.2).

8.3 Mismatched Calibrations

Do your reconstructed images look offset, like those in Figure 15?

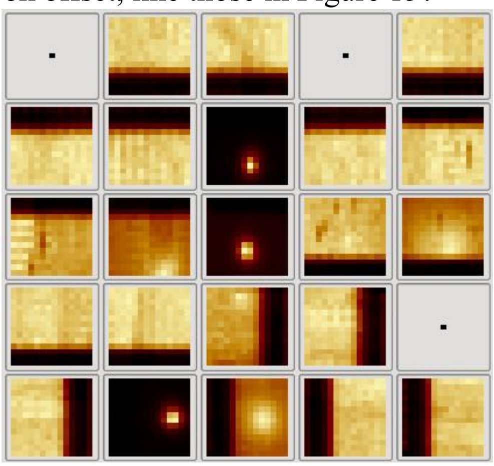


Figure 15: Example of images that appear rather offset. This is a classical effect if the calibrations are not matched to the data. The reconstruction has done its job, but the data were not in the locations on the detector where the calibrations indicated they should be – perhaps because the grating is in a slightly different position (here by about 4 pixels). This can easily be corrected by taking new calibrations.

___oooOOOooo___