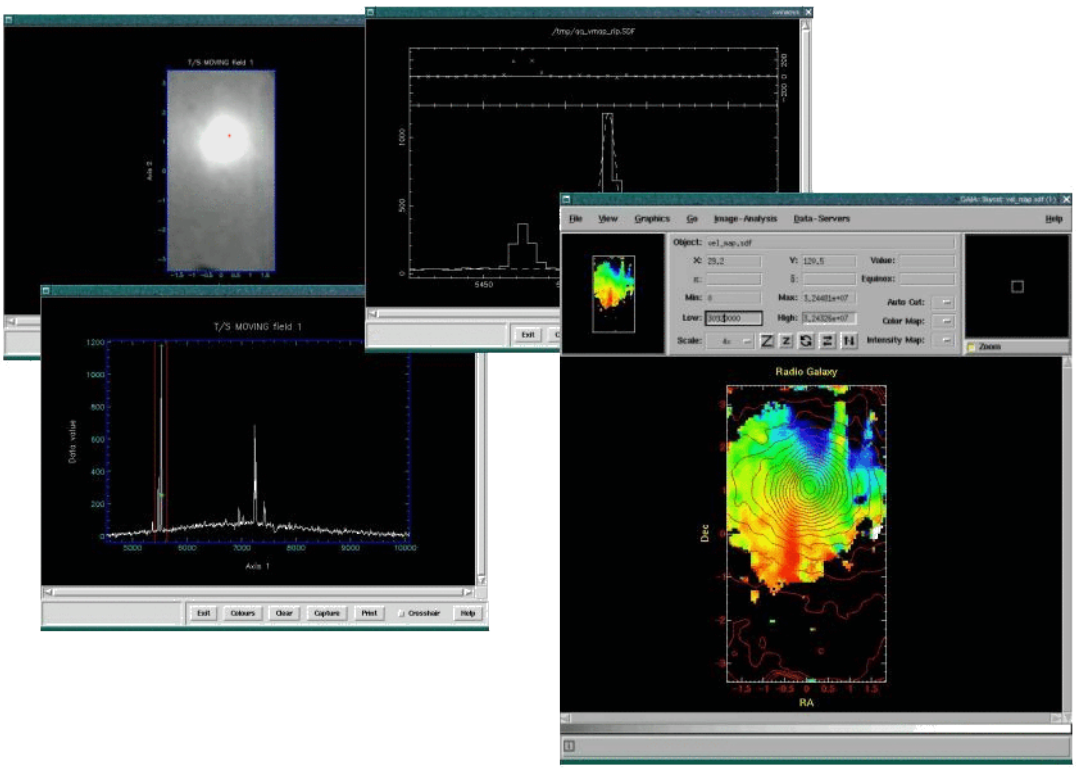
**SC/16.2**

Starlink Project
STARLINK Cookbook 16.2

A. Allan & Malcolm J. Currie

2008 July 4

# The IFU Data-Product Cookbook
# Version 1.3



## Abstract

This cookbook is a collection of material covering IFU data reduction and analysis. Along with this material are pointers to more advanced documents dealing with the various packages, and hints and tips about how to deal with commonly occurring problems.

# Contents

# List of Figures

**Revision history**

(1) 1st September 2000; Version 0.1 Original version (AA)

(2) 24th November 2000; Version 0.2 Added VIMOS information (AA)

(3) 12th December 2000; Version 0.3 Added IDL procedures (AA)

(4) 30th December 2000; Version 0.4 Cleanup for release (AA)

(5) 2nd January 2001; Version 1.0 Release version (AA)

(6) 30th September 2002; Version 1.0-1 Minor changes (AA)

(7) 2005 October; Version 1.1 Updated for DATACUBE version 1.1 and major tidy. (MJC)

(8) 2006 March 17; Version 1.1-1 Add illustrated velmoment and gridspec sections; mention CLINPLOT. (MJC)

(9) 2006 June 16; Version 1.1.-2 Introduce GAIA '3D' in a new illustrated subsection. List CHANMAP, and smoothing of planes by BLOCK and GAUSMOOTH. (MJC)

(10) 2008 July 4; Version 1.2 Add section on locating features with CUPID and STILTS. Update GAIA section and graphics; add new illustrated subsection on three-dimensional rendering. Mention PLUCK and PERMAXES. (MJC)

# 1 Introduction

Integral Field Spectroscopy (IFS) is a technique to produce a spectra over a contiguous two-dimensional field, producing as a final data product a three-dimensional data cube of the two spatial co-ordinate axes plus an additional spectral axis, usually in wavelength. Although existing techniques, such as stepping a longslit spectrograph or scanning a Fabry-Perot device, can produce such a data cube, the IFS technique collects the data simultaneously with obvious savings in observing efficiency. However, IFS has only recently approached maturity as a hardware technique.

The technique started with the use of lenslet arrays without fibres, but the lack of a reformatting ability resulted in a short spectral range. The use of fibres improves on the lenslet-only technique, since the field can be reformatted into a pseudo-slit which can be dispersed by conventional spectrographs, and allows an IFS capability to be retrofitted to existing spectrographs. The earliest versions used bare fibres (*e.g.* INTEGRAL on WHT) but this suffers from inefficient coupling to the telescope and incomplete field coverage due to gaps between the fibre cores. Both these problems can be solved by coupling the fibres to micro-lenses. Despite its greater technical difficulty, this technique has been successfully prototyped, *e.g.* SMIRFS on UKIRT and TEIFU on the WHT. For infrared instruments working in cryogenic or space environments which are hostile to fibres, the technique of image slicing has been developed. Here the field is sliced into one-dimensional sections which are then reformatted into a near-continuous long slit.

# 2 The Datacube Package

The Starlink DATACUBE Package (see SUN/237) of which this cookbook forms a part, mainly consists of C-shell (`csh`) scripts layered on top of various pieces of the Starlink Software Collection (SSC). This approach was a deliberate design decision to allow the maximum amount of flexibility during data visualisation. Due to the relative lack of maturity in this still developing field, it is difficult to say exactly what visualisation tasks you may wish (or be required) to carry out to achieve the underlying science. While I (AA) have attempted to anticipate commonly required tasks, the implementation of the package in easily understandable, and modifiable, scripts allows you to make minor (or even major) changes to the way they behave, although most of the scripts already have command-line arguments which can modify their behaviour to some extent (see SUN/237 for details).

It is to be hoped that enough ground work has been done so that the approach to solving your data visualisation or manipulation problem is obvious, even if DATACUBE doesn't have a script or application to exactly what you require. I would welcome comments, contributions and corrections to the package since I have been very much aware while compiling it my on lack of experience in this fast evolving field, and my own biases. For instance, this document deals only briefly with IRAF, while I am aware that there are IRAF applications available that will deal with spectral data cubes, my own lack of familiarlity with the IRAF package has lead to only spartan coverage. Comments should be sent to the Starlink support mailing list `starlink@jiscmail.ac.uk` .

# 3 Data reduction

Initial data reduction to remove instrumental effects such as flat fielding and cosmic-ray removal, and mapping between the two-dimensional detector co-ordinates and the data cube, is highly instrument dependent. The IFU instruments currently in use, with obvious exceptions, tend not to be *common user* but instead *fast-track* or *in-house* instruments. This has had a strong influence on the available data-reduction software.

## 3.1 Reduction paradigms

There are two paradigms for IFS data reduction. First, the 'traditional' method, adapted from multi-object spectroscopy (MOS), where the output from each fibre is extracted by tracing the spectrum and accounting for wavelength-dependent distortion (normally referred to as the *MOS paradigm*). More recently, with the arrival of TEIFU where the fibre outputs are under-sampled by the detector, an alternative paradigm has arisen (usually referred to as the *longslit paradigm*). Although the independence of the spatial samples is lost due to the under-sampling of the point-spread function (PSF) by the detector, it can be shown that this is irrelevant so long as the target is critically sampled by the IFU; see Allington-Smith & Content (1998). Here the methods adapted from MOS cannot be used and the resulting dataset bears more resemblance to traditional longslit spectroscopy than to MOS data.

## 3.2 INTEGRAL data

INTEGRAL is an integral-field spectroscopic facility deployed at the Nasmyth focus of the WHT, and channels the light into WYFFOS fibre spectrograph. Up to six fibre bundles are available, although only three bundles are used in normal operation, with field sizes ranging from 10 to 40 arcseconds and different fibre core sizes. These options allow observers to make the most efficient use of the prevailing seeing conditions. More information can be found at `http://www.iac.es/proyect/integral/`.

Data-reduction facilities for the instrument are provided by the INTEGRAL IRAFpackage which contains some standard tasks from other IRAF packages (such as ONEDSPEC, SPECRED, and CCDRED) and some custom tasks designed to deal with INTEGRAL data. The package can be downloaded from

`http://andromeda.roque.ing.iac.es/~astrosw/InstSoft/integral/integral-0.3.tar.gz` along with the package user manual, which contains installation instructions and a description of the available reduction software (see Section 7). It should be noted that to compile the INTEGRAL package requires the Starlink public-domain algorithms (PDA) package to be present on your machine (see SUN/194).

## 3.3 OASIS data

OASIS is an integral field spectrograph for use with AOB/PUEO although it can also be used at the direct f/8 Cassegrain focus as a backup mode and for science programs necessitating IFS but with a coarser spatial sampling defined by the natural seeing. OASIS can currently be used in two modes. The imagery mode is used primary for accurate pointing on objects.

Image quality has been optimized so that high spatial resolution (0.″1) images can be obtained. The spectroscopic mode offers low-to-medium spectral resolution with a wide range of spatial samplings performed by an array of hexagonal micro-lenses. Depending of the configuration employed, the spectrographic field diameter varies from 1.5 arcsec to 10 arcsec. More information can be found at `http://www.cfht.hawaii.edu/Instruments/Spectroscopy/OASIS/`.

Data-reduction facilities is provided by the XOASIS software and detailed installation and 'cook-book style' usage instructions are available online.

### 3.4 SAURON data

SAURON is another IFS based on the TIGRE/OASIS concept of a micro-lens array built for the WHT. It has an array of 1500 square lens, and a wide field, either 10 or 35 arcsec$^2$. Data reduction is via pipeline software (XSAURON and PALANTIR) especially developed for the instrument which was modelled after the XOASIS package. More information about the instrument can be found at `http://www.strw.leidenuniv.nl/sauron/`.

### 3.5 GMOS data

The two Gemini Multi-Object Spectrographs (GMOS), one for each Gemini telescope, provides facilities for two-dimensional spectroscopy over a contiguous field of $\sim$50 arcsec$^2$ with 0.2-arcsec sampling. A small background field will be available at fixed separation ($>$ 1 arcmin) from the main object field for accurate background subtraction. Details of the IFU itself can be found at `http://www.gemini.edu/sciops/instruments/gmos/gmosIFU.html`.

Data-reduction facilities for the instrument are provided in the Gemini IRAF package.

### 3.6 CIRPASS data

CIRPASS is a near-infrared spectrograph with a 499-element, lens and fibre, integral-field unit to collect the light from the target object. CIRPASS is available as a visitor instrument on the Gemini telescopes. Users of CIRPASS are strongly encouraged to collaborate fully with the instrument team in Cambridge to get the most out of their Gemini time.

The data-reduction and analysis software for CIRPASS runs under IRAF. There is a cookbook available at `http://www.ast.cam.ac.uk/~optics/cirpass/datared/cookbook_sn1987a.php`.

The final data product for the science data is an $x$,$y$,$\lambda$ data cube (see Section 4.3), which can be visualised as a cube where the $z$-axis is wavelength and each plane is a picture of what the IFU observed at that wavelength.

The first version of the data-reduction package will deal separately with the different types of data (*e.g.* dome flats, sky flats, arc lamps, flux standards and target observations). For each type of data there are one or more pipeline scripts to reduce the data, with each pipeline script running a series of IRAF tasks.

Links to more information on the ongoing development of the reduction software can be found on the web at `http://www.ast.cam.ac.uk/~optics/cirpass/docs.html`.

### 3.7 SMIRFS data

SMIRFS was constructed by the Durham group as a technology demonstrator for the more ambitious integral-field units which Durham has producing and continues to develop for the WHT and Gemini (*e.g.* GMOS). The IFU works with CGS4 on UKIRT to provide IFS for the near infrared (1–2 $\mu$m in the $J$ and $H$ bands). The SMIRFS IFU is available for use in collaboration with the SMIRFS-IFU team, please contact Jeremy Allington-Smith (`J.R.Allington-Smith@durham.ac.uk`).

More information on the technical specifications of the SMIRFS instrument, and the science that can be done with it, can be found at `http://star-www.dur.ac.uk/~jra/ukirt_ifu.html`.

### 3.8 TEIFU data

TEIFU is a system for integral-field spectroscopy using adaptively corrected images produced by the ELECTRA and NAOMI AO systems on the WHT. It is also able to operate in a stand-alone mode without an adaptive-optics system.

Data-reduction facilities will be provided by the IMSPEC IRAF package which is under development at Durham. No detailed information is available at this time.

### 3.9 UIST data

UIST is a general-purpose imager and spectrometer operating in the 1–5 $\mu$m range at UKIRT, It was commissioned in 2002 October, replacing all spectroscopy functions of CGS4 except for echelle spectroscopy, all imaging functions of IRCAM/TUFTI and all imaging functions of UFTI except the Fabry-Perot filter. It also includes a deployable image slicing IFU mounted in the slit wheel. Slicing mirrors are used to reformat a $6.0 \times 3.3$-arcsec region of the sky into fourteen slices (the IFU contains eighteen slicing mirrors but four are currently not usable), each fifty pixels long, offset from one another along their length. This produces a staggered column on slitlets (as shown in Figure 1) which is used as the input for the spectrometer in place of the long slit.

Data acquisition, reduction and control software is provided by the JAC ORAC system. The data-reduction part of the system, ORAC-DR, is provided by JAC and distributed by Starlink, and consists of a fully automated perl-based pipelining software (Economou*et al*. 1999) sitting on top of the Starlink software collection. A general introduction to the ORAC-DR system can be found in SUN/230. The data-reduction recipes are documented in SUN/246 and at the UKIRT web site. An arc spectrum (Ar or Kr from the UIST calibration unit) is used to straighten the staggered slitlets (which correspond to a wavelength displacement from one slice to another) and apply a wavelength calibration to the image. The individual slice images will then be copied to form $y$-$\lambda$ planes of an $x$,$y$,$\lambda$ data cube. Recipes are also provided to carry out tasks such as flat-fielding and flux-calibration. Many of the recipes have specific requirements in terms of, for instance, darks and flats fields which must be acquired before a target observation is obtained and reduced on-line.

### 3.10 VIMOS data

VIMOS has been developed in fast track under ESO contract by the VIRMOS consortium, headed by the Laboratoire d'Astrophysique de Marseille. In IFU mode the field of view is between $13\times13$ arcsec$^2$ and $54\times54$ arcsec$^2$ at 0.33 or 0.67 arcsec fibre$^{-1}$.

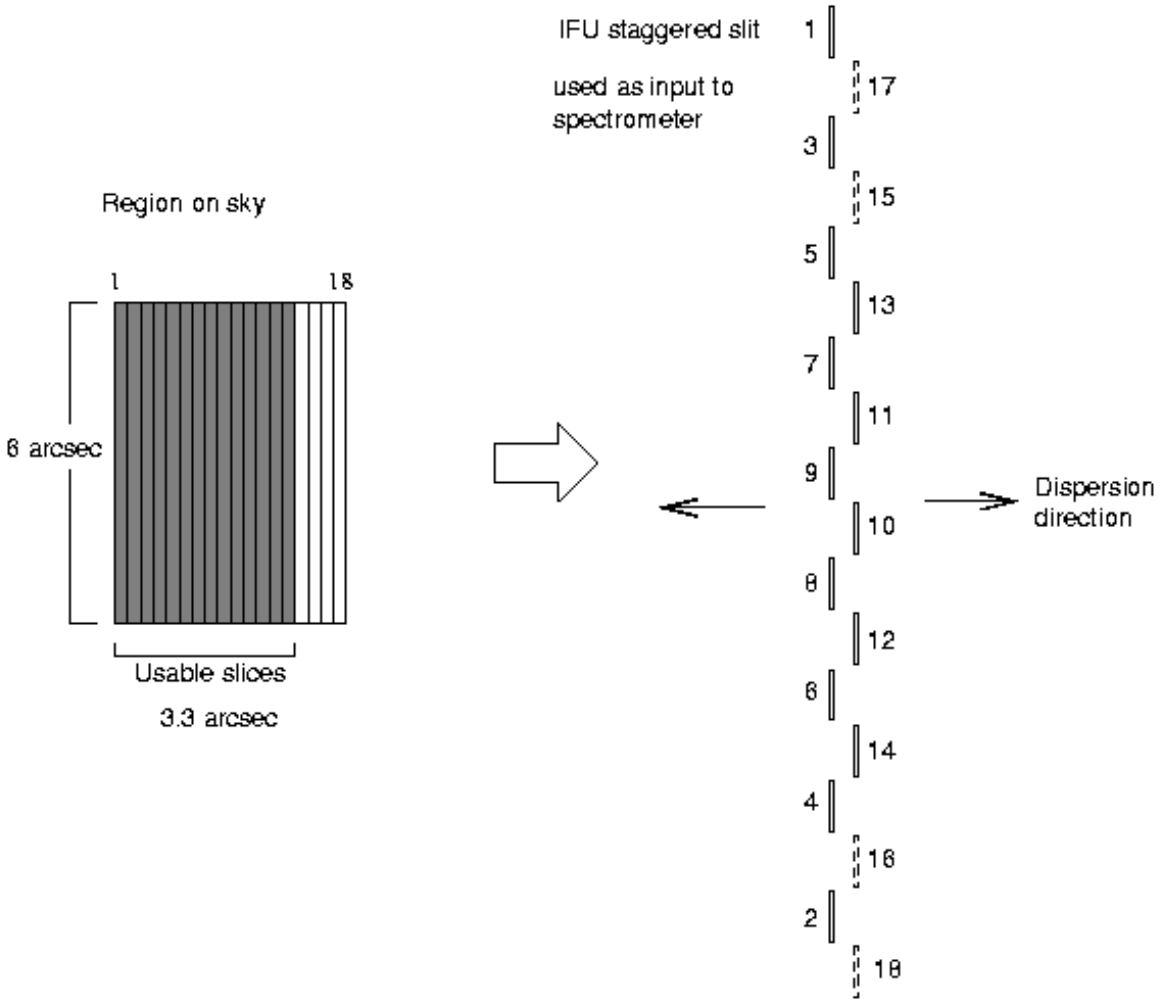



Figure 1: The UIST staggered slitlets.

Data-reduction software (DRS) will be made available by ESO, with the procedures available as standalone packages, or under the VIMOS pipeline-reduction software.

`http://www.oamp.fr/virmos/virmos_publications.htm` contains links to various publications. Two papers are scheduled for the 2005 November *Astronomical Journal*.

### 3.11 Other IFU instruments

Since the list of instruments was compiled for the original version of this cookbook, many new Integral Field Units and area-spectroscopy instruments have come online, such as SINFONI and FLAMES at the VLT, GNIRS-IFU at Gemini-S, FISICA, SNIFS; and several are being designed, for instance KMOS-1, MUSE, and FRIDA.

# 4 File formats

There are two main file format for IFS final data products, and one further prospective format. The first of these file formats is a MOS style multi-extension FITS file, and is being put forward by the GEMINI group (see Section 4.1). The other main format is an $xy\lambda$—data cube which is used by the Durham group (*i.e.* for SMIRFS and TEIFU data) (see Section 4.3).

Of these two formats the more natural for data analysis is the TEIFU style data cube, it has therefore been adopted as the standard format by the major IFS groups in the UK, Durham (SMIRFS and TEIFU), Cambridge (CIRPASS) and the ATC (UIST). Conversion between the GMOS/CIRPASS MEF and UK standard data cube formats(see Section 4.4) will therefore need to be implemented before these instruments are brought into general use. The EOS VIMOS instrument will provide its final data product in both MEF and data-cube formats.

The final IFS format is still in draft, and is the new IRAF spectroscopic file format (see Section 4.2). The specification is intended to provide a general description for two-dimensional spectroscopic image data, and should be able to represent long-slit, multi-object (MOS), integral-field unit (IFU) and slitless spectroscopy. Conversion between this format and the standard data-cube format will be implemented if the standard is adopted.

## 4.1 The GMOS working format

Currently the final data product of the GMOS and CIRPASS data-reduction software is a multi-extension FITS (MEF) file. However, this format may be replaced by the new IRAF spectral format (see Section 4.2) which is currently in development by the IRAF group at NOAO. The MEF is similar to the standard NIRI format now used with GEMINI, and has a binary FITS table with separate data, variance and quality planes.

| No. | Type | Name | Format | BITPIX | INH |
|---|---|---|---|---|---|
| 0 | ifs_data.fits | | | 16 | |
| 1 | BINTABLE | TAB | $16\times$*num. of fibres* | 8 | |
| 2 | IMAGE | SCI | $\lambda\times$*num. of fibres* | $-32$ | F |
| 3 | IMAGE | VAR | $\lambda\times$*num. of fibres* | $-32$ | F |
| 4 | IMAGE | DQ | $\lambda\times$*num. of fibres* | 16 | F |

Table 1: GEMINI MEF file format

The first extension is a binary FITS table with columns: ID, RA, DEC, and SKY. This table would hold information specific to individual lenslets/fibres like relative fibre positions on the sky (RA, DEC), whether the fibre is a sky or object spectrum (SKY), *etc.*

The three image planes are like the IRAF multispec format; each row is a separate spectrum. This is a compact and efficient way of storing the extracted spectra, avoiding having multiple extensions for each individual spectra. From IRAF, ONEDSPEC tasks like **splot** can be used on individual planes, and **ldisplay** should be able to work directly on the MEF.

## 4.2 The *new* IRAF spectral format

A draft document has been written by the NOAO describing the new IRAF spectroscopic file format. This may, eventually, replace the GMOS working format as the final science data product for GMOS and CIRPASS observations. An example header block is shown below for the CIRPASS instrument.

```
OBJECT  = 'CIRPASS: m51 V 600s' / Observation title
OBJNAME = 'M 51    '           / Target object
OBJRA   = '13:29:24.00'        / Right ascension of object (hr)
OBJDEC  = '47:15:34.00'        / Declination of object (deg)
OBJEPOCH=               2000.1 / Epoch of object coordinates (yr)
EQUINOX =               2000.0 / Default coordinate equinox (yr)
RADECSYS= 'FK5     '           / Default coordinate system
RAUNIT  = 'hr      '           / Right ascension unit
DECUNIT = 'deg     '           / Declination unit
APERTURE= 'CIRPASS IFU'        / Aperture identification
APTYPE  = 'hexlens+fiber'      / Aperture type
APERDIA =                 0.36 / Aperture diameter (arcsec)
APERPA  =                 90.0 / Hexagon angle (deg)
APUNIT  = 'arcsec  '           / Aperture dimension unit
APPAUNIT= 'deg     '           / Aperture position angle unit
APEPOCH =               2000.1 / Aperture coordinate epoch (yr)
CRVAL1  =                  1.1 / Spectrum dispersion center (um)
CRVAL2  =                   0. / Spectrum cross-dispersion center (pixel)
CRPIX1  =               1024.0 / Spectrum center (pixel)
CRPIX2  =               1024.0 / Spectrum center (pixel)
CMIN1   =                  0.9 / Spectrum dispersion limit (um)
CMAX1   =                  1.3 / Spectrum dispersion limit (um)
CMIN2   =                 -1.5 / Spectrum cross-dispersion limit (pixel)
CMAX2   =                  1.5 / Spectrum cross-dispersion limit (pixel)
CTYPE1  = 'WAVE-WAV'           / Spectrum coordinate type
CTYPE2  = 'LINEAR  '           / Spectrum coordinate type
CUNIT1  = 'um'                 / Spectrum coordinate unit
CUNIT2  = 'pixel'              / Spectrum coordinate unit
CD1_1   =             0.00022 / Spec coord matrix (um/pixel)
CD1_2   =                 0.0 / Spec coord matrix (um/pixel)
CD2_1   =                 0.0 / Spec coord matrix (pixel/pixel)
CD2_2   =                 1.0 / Spec coord matrix (pixel/pixel)
SPECFWHM=                 2.0 / Fiber FWHM (pixel)
ARA0001 = '13:29:24.00'        / Aperture right ascension (hr)
ADEC0001= '47:15:34.00'        / Aperture declination (deg)
CRP20001=               500.0 / Spectrum center (pixel)
ARA0002 = '13:29:24.00'        / Aperture right ascension (hr)
ADEC0002= '47:15:34.36'        / Aperture declination (deg)
CRP20002=               504.0 / Spectrum center (pixel)
```

The aperture identification, `APERTURE`, specifies the IFU. The aperture type `APTYPE`, aperture diameter `APERDIA`, and aperture position angle `APERPA` are the same for each spectrum. This information can be used to construct a data cube (see Section 4.4) or spatial/dispersion displays in conjunction with the aperture centres. The position angle is for one of the hexagonal edges and would orient the hexagons (IFU lenslets) when reconstructing a spatial display or data cube.

For a purely fibre IFU (such as SAURON) much the same description would be used except the position angle would be eliminated.

The centre of each spectrum in world co-ordinates is given by the CRVAL keywords. In this example each spectrum is centred at about 1.1 $\mu$m in the dispersion direction (CRVAL1) and zero pixels in the cross-dispersion direction (CRVAL2). The cross-dispersion co-ordinates are defined as pixels from the centre of the fibre profile, since there is no real spatial information. The region the spectra cover in world co-ordinates are given by the CMIN and CMAX keywords. In this example the spectra cover the range 0.9 to 1.3 $\mu$m along the dispersion and $-1.5$ to 1.5 pixels relative to the fibre profile centre.

The CD keywords define the conversion between world co-ordinates and pixels on the detector. They also define any possible tilt of the dispersion path relative to the detector pixels. In this example the dispersion is 0.22 nm per pixel along the first image axis (detector rows) and there is no tilt.

The CRP keywords override the CRPIX keyword and provide the positions of the fibre spectra on the detector.

The fibre full width at half maximum (SPECFWHM) gives the fibre profile FWHM at the detector in the units of the spatial WCS, in this case pixels. This is used to guide the tracing and extraction of blended fibre profiles.

The ARA and ADEC keywords give the centre positions of each lenslet element or fibre. While it is desirable for the absolute co-ordinates to be accurate it is more important that the relative positions be fairly precise. It is these keywords that determine the reconstructed field and gives the IFU sampling pattern and orientation. The relative positions of the lenslets or fibres on the sky is something that should be well-known for each IFU instrument.

### 4.3 The UK data-cube format

The MOS-style MEF format, which is the end product of the GMOS and CIRPASS data-reduction software, is not particularly natural way of handling IFS data. Indeed, under the *longslit* paradigm (used to reduce TEIFU data) these files cannot be generated. The TEIFU-style data cube format has therefore been adopted as the standard UK IFS file format and will be used in the analysis stages for both CIRPASS and UIST. This adoption allows the use of many of the generic applications within the SSC, which due to the adoption of NDF as the standard file interchange format for Starlink applications, has many tasks that can process $N$-dimensional data.

A conversion program for GMOS and CIRPASS data to a more easily analysed data cube, which will involve re-binning the input spectra on to a rectangular array, is therefore desirable (see Section 4.4).

In the case of this format the IFU geometry information is no longer needed, as the input spectra have already been rebinned, but it is likely that (in the finalised file format) such information will be included as a FITS binary table.

An example of the FITS header block from a TEIFU data cube is shown below.

```
SIMPLE  =                    T /  file does conform to FITS standard
BITPIX  =                   16 /  number of bits per data pixel
NAXIS   =                    3 /  number of data axes
```

| No. | Type | Name | Format | BITPIX | Comment |
|---|---|---|---|---|---|
| 0 | ifs_data.fits | | | | |
| 1 | IMAGE | SCI | $x \times y \times \lambda$ | $-32$ | 3-D science array |
| 2 | IMAGE | VAR | $x \times y \times \lambda$ | $-32$ | 3-D variance array |
| 3 | IMAGE | DQ | $x \times y \times \lambda$ | 16 | 3-D data-quality array |

Table 2: TEIFU data-cube format

```
NAXIS1  =                   59 /  length of data axis 1
NAXIS2  =                  110 /  length of data axis 2
NAXIS3  =                  961 /  length of data axis 3
EXTEND  =                    T /  FITS dataset may contain extensions
OBJECT  = 'T/S MOVING field  ' /  Title of the dataset
DATE    = '2000-10-05T17:50:52'/  file creation date (YYYY-MM-DDThh:mm:ss UTC)
BSCALE  =         3.126180E-02 /  True_value = BSCALE * FITS_value + BZERO
BZERO   =         6.089249E+02 /  True_value = BSCALE * FITS_value + BZERO
BLANK   =               -32768 /  Bad value
CD1_1   =               0.0625 / Axis rotation and scaling matrix
CD2_2   =               0.0625 / Axis rotation and scaling matrix
CD3_3   = 5.795317000000068219 / Axis rotation and scaling matrix
CRVAL1  =             -0.03125 / Axis 1 reference value
CRVAL2  =             -0.03125 / Axis 2 reference value
CRVAL3  = 7302.291864499999065 / Axis 3 reference value
CRPIX1  =                 29.5 / Axis 1 pixel value
CRPIX2  =                 55.0 / Axis 2 pixel value
CRPIX3  =                480.5 / Axis 3 pixel value
WCSDIM  =                    3
CTYPE1  = 'LINEAR  '           / Quantity represented by axis 1
CTYPE2  = 'LINEAR  '           / Quantity represented by axis 2
CTYPE3  = 'LAMBDA  '           / Quantity represented by axis 3
CD1_2   =                  0.0 / Axis rotation and scaling matrix
CD1_3   =                  0.0 / Axis rotation and scaling matrix
CD2_1   =                  0.0 / Axis rotation and scaling matrix
CD2_3   =                  0.0 / Axis rotation and scaling matrix
CD3_1   =                  0.0 / Axis rotation and scaling matrix
CD3_2   =                  0.0 / Axis rotation and scaling matrix
LTV3    =                -39.0
LTM1_1  =                  1.0
LTM2_2  =                  1.0
LTM3_3  =                  1.0
WAT0_001= 'system=image'
END
```

Here the the number and size of the cube dimensions is specified by the `NAXIS` keywords, and as with the IRAF spectral format the `CD` keywords define the conversion between world co-ordinates and pixels on the detector, along with the tilt of the dispersion path relative to the detector pixels. While the `CRVAL` keywords defines the central value of each axis in world co-ordinates, *e.g.* in the case the spectral axis is centred on $\sim$7302 Å (`CRVAL3`).

A full dictionary defining FITS header keywords which can be generated by the data-acquisition system is provided on the web by the National Optical Astronomy Observatories (NOAO) at `http://iraf.noao.edu/projects/ccdmosaic/imagedef/fitsdic.html`.

## 4.4 MEF to data-cube format

In the Gemini IRAF package the command **gfcube** converts the GMOS working format, which is currently being used as the science end product file format for the GMOS and CIRPASS instruments, to a UK standard $x$,$y$,$\lambda$ data cube in FITS format.

## 4.5 Format conversion

The Starlink CONVERT package (see SUN/55) can be used to convert to and from the Starlink NDF format. On-the-fly conversion of supported file formats (such as FITS and IRAF) can also be done by most Starlink applications if the CONVERT package has been initialised.

### 4.5.1 GMOS MEF to NDF

The CONVERT package handles the GMOS/CIRPASS MEF working format without complaint, as in the following example.

```
% fits2ndf
IN - Input FITS file(s) > gmos.fits
1 file selected.
OUT - Output NDF data structure(s) > out
%
```

Converting the MEF to a Starlink standard NDF, the FITS binary table is converted into a normal NDF extension. An example of a resulting NDF is illustrated below. The $< >$ indicate the data type of a component. Those beginning with an underscore are primitive types; others are structures.

```
IFS_FILE  <NDF>

   DATA_ARRAY     <ARRAY>
      ORIGIN(2)      <_INTEGER>
      DATA(2010,750)  <_REAL>

   MORE           <EXT>
      FITS_EXT_1      <TABLE>
         NROWS          <_INTEGER>
         COLUMNS        <COLUMNS>
            ID              <COLUMN>
               DATA(750)    <_INTEGER>

            RA          <COLUMN>
               COMMENT      <_CHAR*19>
               DATA(750)    <_REAL>

            DEC         <COLUMN>
```

```
            COMMENT      <_CHAR*19>
            DATA(750)    <_REAL>

         SKY          <COLUMN>
            COMMENT      <_CHAR*19>
            DATA(750)    <_INTEGER>

   FITS(790)       <_CHAR*80>

VARIANCE       <ARRAY>
   DATA(2010,750)  <_REAL>
   ORIGIN(2)       <_INTEGER>

QUALITY        <QUALITY>
   QUALITY         <ARRAY>
      DATA(2010,750)   <_UBYTE>
      ORIGIN(2)        <_INTEGER>
```

Here the right ascension and declination position of each fibre is preserved in the FITS_EXT_1 NDF extension along with and array indicating whether the fibre is 'on sky'.

### 4.5.2 TEIFU FITS to NDF

The CONVERT package also handles the UK standard data cube format (*i.e.* TEIFU style data) without complaint, such as in the example below.

```
% fits2ndf
IN - Input FITS file(s) > teifu.fits
1 file selected.
OUT - Output NDF data structure(s) > out
%
```

The FITS file is be converted it into a three-dimensional NDF which, as discussed earlier (see Section 4.3) can be read by many existing applications in the software collection.

```
IFS_FILE  <NDF>

   DATA_ARRAY     <ARRAY>
      ORIGIN(3)      <_INTEGER>
      DATA(59,110,961)  <_DOUBLE>
      BAD_PIXEL      <_LOGICAL>

   MORE           <EXT>
      FITS(45)       <_CHAR*80>

   TITLE          <_CHAR*18>
   WCS            <WCS>
      DATA(99)       <_CHAR*32>
```

## 4.6 GMOS vs. TEIFU format

While both GMOS (MOS style) and TEIFU (data cube) representations of IFS data are perfectly valid, there are several advantages to using the data-cube format in preference to other options. First, and perhaps most importantly, for 'longslit' paradigm instruments such as TEIFU (and perhaps also CIRPASS) a MOS style data reduction is not possible and therefore it is impossible to produce the first file type without major problems. Additionally a data-cube format is considered, by most people, to be intrinsically easier to visualise. Both these reasons were taken under consideration when the data-cube format was adopted, with consultation of Durham, Cambridge and the ATC, as the UK standard format for this data.

## 4.7 FITS header manipulation

Due to the still developing nature of the IFU file formats it is possible that your data may have missing FITS header keywords, or keywords which contain incorrect information. If this is the case you may need to manually edit your FITS file headers.

### 4.7.1 Native FITS files

A good package to use for FITS header (and data) manipulation is the FTOOLS software, which is released along with XANADU, as part of the HEASOFT package from GSFC. Further information about HEASOFT, along with detailed installation instructions, user manuals and a development guide, can be found at `http://heasarc.gsfc.nasa.gov/docs/software/lheasoft/`.

### 4.7.2 The NDF FITS extension

When a FITS file is converted to an NDF a FITS extension—sometimes called the 'airlock' to avoid confusion with extensions within FITS files—is created. This comprises a one-dimensional array of character strings containing the imported FITS header information. On exporting a file from NDF format back to FITS using **ndf2fits** the airlock contents will be propagated back to the FITS file. However, since the FITS extension is not updated when an NDF is manipulated, any information that can be derived directly from the NDF structure such as dimensionality, units and axis information will replace any equivalent information held in the FITS extension when it is exported.

The KAPPA package provides tools that allow you to read from, and write to, an NDF FITS extension. Example code using some of these tools is shown later in this document (see Section 6.5), and detailed documentation on these tasks is available in SUN/95.

## 4.8 FITS I/O with IDL

FITS I/O with IDL can be accomplished using the IDL Astronomy Library from the GSFC. The IDL Astronomy Library contains four different sets of procedures for reading, writing, and modifying FITS files. The reason for having four different methods of FITS I/O with IDL is partly historical, as different groups developed the software independently. However, each method also has its own strengths and weakness for any particular task. For example, the procedure `MRDFITS()`—which can read a FITS table into an IDL structure—is the easiest procedure for analyzing FITS files at the IDL prompt level (provided that one is comfortable

with IDL structures). But mapping a table into an IDL structure includes extra overhead, so that when performing FITS I/O at the procedure level, it may be desirable to use more efficient procedures such as FITS_READ and FTAB_EXT.

For example a data cube can be read into an IDL array using the FXREAD method.

```
; Read the FITS file
fxread, 'ifu_file.fit', DATA, HEADER

; Determine the size of the image
SIZEX=fxpar(header, 'NAXIS1')
SIZEY=fxpar(header, 'NAXIS2')
SIZEZ=fxpar(header, 'NAXIS3')

; Find the data type being read
DTYPE=fxpar(header, 'BITPIX')
```

As can be seen, various values contained within the FITS header of the original file can be obtained using the FXPAR procedure.

Alternatively the MRDFITS procedure can be used, as in this example.

```
; Read the FITS file
data = mrdfits('ifu_file.fit',0,header)
```

In both examples the image data is read into an IDL array called DATA, while the FITS header information is read into another array, of TYPE STRING, called HEADER.

In addition FITS files can be read into IDL using the CONVERT package's on-the-fly file conversion ability (see SUN/55 for more details) and the READ_NDF IDL function.

### 4.9 NDF I/O with IDL

There are several methods for reading an NDF file into IDL. First the NDF can be converted to a FITS file using the **ndf2fits** application in the CONVERT package,

```
% ndf2fits comp=D
IN - Input NDF data structure(s) /@section/ >
1 NDF selected.
OUT - Output FITS file(s) /@out/ > section.fit
%
```

and then read into IDL using the IDL Astronomy Library (as in Section 4.8). However, there are several other approaches that can be taken.

The easiest approach is to use the READ_NDF IDL procedure available with the CONVERT package. When CONVERT is installed, both the IDL procedures READ_NDF and WRITE_NDF are placed in $CONVERT_DIR so, to make them available to IDL, that directory must be added to the IDL search path. This will be done if the environment variable IDL_PATH has been set.

For example assuming we have a data cube called file.sdf which is of type _REAL

```
IDL> data_array = read_ndf('file')
```

creates an IDL floating array, data_array, with the same dimensions as the NDF and containing the values from its DATA component.

```
IDL> data_array = read_ndf('file', !values.f_nan)
```

As above except that any occurrence of a bad value (VAL__BADR as defined by the Starlink PRIMDAT package) in the NDF will be replaced by NaN in the IDL array.

```
IDL> var_array = read_ndf('file',comp='v')
```

creates an IDL byte array from the VARIANCE component of the same NDF. Output of an IDL array is achieved using the corresponding WRITE_NDF procedure, for example assuming data_array is an IDL floating array then,

```
IDL> write_ndf, data_array, 'file'
```

creates the NDF file.sdf with the same dimensions as the IDL array data_array, and writes the array to its DATA component (of type _REAL). No checks on bad values are made by default, such checks can be carried out, *e.g.*

```
IDL> write_ndf, data_array, 'file', !values.f_nan
```

Here any occurrence of the value NaN in the array will be replaced by the VAL__BADR value as defined by the Starlink PRIMDAT package. While

```
IDL> write_ndf, var_array, 'file', comp='v'
```

writes the IDL array var_array to the VARIANCE component of the NDF created above. A check is made that the size of the array corresponds with the size of the NDF.

There is yet another approach to read NDF data into IDL. Again we make use of the CONVERT package, this time we use the **ndf2ascii** application to convert the NDF to a ASCII text file so that we may use the IDL read_ascii procedure, after generating an associated data template using the ascii_template GUI. For instance reading the file file.dat in the subdirectory ifu_data

```
IDL> data_file = filepath('file.dat', SUBDIR='ifu_data')
IDL> data_template = ascii_template(data_file)
IDL> data = read_ascii(data_file, TEMPLATE=data_template)
```

we create an associated data template using ascii_template GUI and read the data into the IDL structure data.

We can similarly use the **ndf2unf** application to create a sequential unformatted binary file and use the read_binary and assocaited binary_template GUI to read the data into IDL, *e.g.*

```
IDL> udata_file = filepath('binary.dat', SUBDIR='ifu_data')
IDL> udata_template = binary_template(udata_file)
IDL> udata = read_binary(udata_file, TEMPLATE=udata_template)
```

will return an IDL structure variable udata.

The alternative more-low-level approach to reading either the ASCII or binary unformated files can be taken, allowing you to bypass the template GUIs. More details can be found in CONVERT documentation (see SUN/55).

# 5 Data-cube manipulation

While the initial data-reduction software for IFUs is highly instrument dependent, the data analysis of the final science data product for all these instruments should be fairly generic. The end product of the data reduction for IFS is, almost naturally, an $x$,$y$,$\lambda$ data cube. For instruments not working in the optical and infrared regimes, the third axis may also be in some other spectral co-ordinate system, such as frequency. Once assembled, with associated variance and quality arrays, scientifically interesting information can be extracted from the cube.

## 5.1 Existing software

For *long-slit paradigm* instruments a data cube is the only data product available, as overlapping point-spread functions mean that the spectral data must be resampled. For *MOS paradigm* data, while the individual spectra are available, data visualisation is often intrinisically more intuitive if done on resampled data cubes. As such, this cookbook will (in the main) only deal with applications that handle the data-cube format. If you have already sourced the Starlink `/star/etc/login` and `/star/etc/cshrc` files[1] then the commands `kappa`, `figaro`, and `ccdpack` will set up access to the KAPPA, FIGARO, and CCDPACK tasks respectively, including most of the following applications.

### 5.1.1 Arithmetic Operations

Some of the most fundamental operations you might wish to perform on a data cube are the arithmetic operations of addition, subtraction, multiplication and division, both by scalars and other data cubes. All these basic operations, and additional more complicated ones, can be carried out using tasks from the KAPPA package. Detailed documentation for all of these tasks can be found in SUN/95.

- **add**
  Adds two NDF data structures
- **cadd**
  Adds a scalar to an NDF data structure
- **calc**
  Evaluates a mathematical expression
- **cdiv**
  Divides an NDF by a scalar
- **cmult**
  Multiplies an NDF by a scalar
- **csub**
  Subtracts a scalar from an NDF data structure

[1] `/star/etc/` is for a standard Starlink installation, but the Starlink software may be in a different directory tree on your system.

- **div**
  Divides one NDF data structure by another
- **maths**
  Evaluates mathematical expressions applied to NDF data structures
- **mult**
  Multiplies two NDF data structures
- **normalize**
  Normalises one NDF to a similar NDF by calculating a scale factor and zero-point difference
- **pow**
  Takes the specified power of each pixel of a data array
- **rift**
  Adds a scalar to a section of an NDF data structure to correct rift-valley defects
- **sub**
  Subtracts one NDF data structure from another
- **thresh**
  Edits an NDF such that array values below and above two thresholds take constant values

### 5.1.2 Cube manipulation

Slightly more complex is manipulation and resampling of the data cube itself. The most important utilities available to do this within KAPPA are the **collapse**, **chanmap**, **ndfcopy**, and **pluck** applications. **collapse** can produce white-light and passband images (see Section 5.8) and velocity maps by the appropriate selection of statistic. **chanmap** creates a grid of passbands. **ndfcopy** can extract a single spectrum (along pixel axes) or arbitrary cube sections (again see Section 5.8 for details), while **pluck** uses interpolation to extract at *arbitrary* positions, such as a spectrum at nominated equatorial co-ordinates, or an image at a given wavelength or frequency. Some applications require the spectral axis to be the first or the third axis; **permaxes** allows shuffling of the pixel axes.

- **chanmap**
  Creates a channel map from a cube NDF by compressing slices along a nominated axis
- **collapse**
  Reduces the number of axes in an $N$-dimensional NDF by compressing it along a nominated axis
- **compadd**
  Reduces the size of an NDF by adding values in rectangular boxes
- **compave**
  Reduces the size of an NDF by averaging values in rectangular boxes
- **compick**
  Reduces the size of an NDF by picking equally spaced pixels

- **flip**
  Reverses an NDF's pixels along a specified dimension
- **ndfcopy**
  Copies an NDF, or an NDF section, to a new location
- **permaxes**
  Permutes an NDF's pixel axes
- **pixdupe**
  Expands an NDF by pixel duplication
- **pluck**
  Plucks slices from an NDF at arbitrary positions
- **regrid**
  Applies a geometrical transformation to an NDF
- **segment**
  Copies polygonal segments from one NDF into another
- **slide**
  Realigns an NDF using a translation.

### 5.1.3 Two-dimensional manipulation

While IFU data is intrinsically three dimensional there are times when it is necessary to deal with two dimensional images during analysis (*e.g.* velocity maps). Some potentially useful KAPPA applications which are restricted to handling two-dimensional files are

- **look**
  Outputs the values of a sub-array of a two-dimensional data array to the screen or a text file
- **median**
  Smooths a two-dimensional data array using a weighted median filter

Some tasks operate in two dimensions, but apply processing to a series of planes in a cube. They permit, for example, to smooth spatially while retaining the spectral resolution.

- **block**
  Smooths an NDF using a one- or two-dimensional square or rectangular box filter
- **gausmooth**
  Smooths a one- or two-dimensional NDF using a Gaussian filter
- **rotate**
  Rotates two-dimensional NDF about its centre through any angle

### 5.1.4 Pixel Operations

There are several applications which can carry out operations on individual pixels, the most general of these is **chpix** which can replace the pixel value of a single, or region, of pixels with an user defined value (including the bad magic value).

- **chpix**
  Replaces the values of selected pixels in an NDF
- **errclip**
  Removes pixels with large errors from an NDF
- **fillbad**
  Removes regions of bad values from an NDF
- **nomagic**
  Replaces all occurrences of magic value (see SUN/95 for details) pixels in an NDF array with a new value
- **numb**
  Counts the number of elements of an NDF with values or absolute values above or below a threshold
- **substitute**
  Replaces all occurrences of a given value in an NDF array with another value

### 5.1.5 Other tools and file manipulation

Building processing scripts from the Starlink applications can involve you manipulating or querying information about structures within the NDF file itself, three useful KAPPA commands to do this are listed below.

- **ndftrace**
  Displays the attributes of an NDF data structure
- **parget**
  Obtains the value or values of an application parameter
- **stats**
  Computes simple statistics for an NDF's pixels

In some cases NDF information must be modified after a processing step (*e.g.* the NDF title) or, in the case of a newly created NDF (see Section 6.6), we must generate inital values. KAPPA provides various tools to manipulate NDF extensions.

- **axlabel**
  Sets a new label value for an axis within an NDF data structure
- **axunits**
  Sets a new units value for an axis within an NDF data structure

- **setaxis**
  Sets values for an axis array component within an NDF data structure
- **setbad**
  Sets new bad-pixel flag values for an NDF
- **setbb**
  Sets a new value for the quality bad-bits mask of an NDF
- **setbound**
  Sets new bounds for an NDF
- **setext**
  Manipulates the contents of a specified NDF extension
- **setlabel**
  Sets a new label for an NDF data structure
- **setmagic**
  Replaces all occurrences of a given value in an NDF array with the bad value
- **setnorm**
  Sets a new value for one or all of an NDF's axis-normalisation flags
- **setorigin**
  Sets a new pixel origin for an NDF
- **settitle**
  Sets a new title for an NDF data structure
- **settype**
  Sets a new numeric type for the data and variance components of an NDF
- **setunits**
  Sets a new units value for an NDF data structure
- **setvar**
  Sets new values for the variance component of an NDF data structure

### 5.1.6 Visualisation

Combined with the graphics devices commands (see Section 5.3) the following applications, especially **clinplot**, **display**, **linplot** and **contour** act as the backbone for image display, and some complex effects can be generated using these elemental tasks.

- **clinplot**
  Draws a spatial grid of line plots for an axis of a cube NDF
- **contour**
  Contours a two-dimensional NDF
- **cursor**
  Reports the co-ordinates of positions selected using the cursor

- **display**
  Displays a one- or two-dimensional NDF
- **drawsig**
  Draws +/-n standard-deviation lines on a line plot
- **linplot**
  Draws a line plot of the data values in a one-dimensional NDF
- **profile**
  Creates a one-dimensional profile through an *N*-dimensional NDF

### 5.1.7 Mosaics

Mosaicking multiple IFU data cubes together is something you may well wish to consider, unfortuantely there are problems involved in doing so (see Section 5.9), however, the following tasks from KAPPA may be useful

- **wcsalign**
  Aligns a group of NDFs using World Co-ordinate System information
- **wcsmosaic**
  Tiles a group of NDFs using World Co-ordinate System information

along with the following tasks from CCDPACK.

- **makemos**
  Makes a mosaic by combining and (optionally) normalising a set of images
- **drizzle**
  Resamples and mosaics using the drizzling algorithm

### 5.1.8 Spectral fitting

Some of the most useful applications available for spectral fitting and manipulation live inside FIGARO as part of SPECDRE. Some of these applications work on individual spectra, however, others read a whole cube at once and work on each row in turn. A possible problem at this stage is that many of these applications expect the spectroscopic axis to be the first in the cube, whereas for the current generation of IFU data cubes the spectral axis is typically the third axis in the cube. KAPPA **permaxes** can reconfigure the cube as needed by the various packages. A full list of SPECDRE applications can be found in SUN/86.

One of the fundamental building blocks of spectral analysis is gaussian fitting, amougst other tools, FIGARO provides the **fitgauss** application (part of SPECDRE) to carry out this task.

FITGAUSS is especially well suited to automation inside a script. For example,

```
fitgauss \
  in=${spectrum} mask1=${low_mask} mask2=${upp_mask} \
  cont=${cont} peak=${peak} fwhm=${fwhm} reguess=no remask=no \
  ncomp=1 cf=0 pf=0 wf=0 comp=${component} fitgood=yes \
  centre=${position} logfil=${fitfile} device=xwin \
  dialog=f
```

we call the **fitgauss** routine here specifying all the necessary parameters, and suppressing user interaction, to allow us to automatically fit a spectrum from inside a shell script. Here we have specified an input file, `${spectrum}` and the lower and upper boundaries of the fitting region, `${low_mask}`, and `${upp_mask}`, respectively. Various initial guesses for the fitting parameters have also been specified: the continuum level `${cont}`, peak height `${peak}`, full-width half-maximum `${fwhm}` and the line centre `${position}`. By specifying `ncomp=1 cf=0 pf=0 wf=0` we have told the application that we want it to fit a single gaussian with the central line position, peak height and FWHM being free to vary.

In addition we have turned off user interaction with the application, by setting the following parameters, `reguess=no`, `remask=no`, `dialog=f` and `fitgood=yes`.

This allows **fitgauss** to go about the fit without further user intervention, displaying its resulting fit in an X-display, logging the fit characteristics to a file (`${fitfile}`) and saving the fit in the SPECDRE extension (see SUN/86 for details) of the NDF where it is available for future reference and manipulation by other SPECDRE applications.

The **velmap** and **peakmap** scripts are based around the SPECDRE **fitgauss** application.

For data with a significantly varying continuum **mfittrend** is available in KAPPA to fit and remove the continuum signal, thereby making **fitgauss**'s job easier.

## 5.2 Locating Features

The ability to identify and measure the properties of features can play an important role in spectral-cube analysis. For example, you may wish to locate emission lines and obtain their widths as initial guesses to spectral fitting, or to mask the lines in order to determine the baseline or continuum. Extending to three dimensions you may want to identify and measure the properties of clumps of comoving emission in a velocity cube. The CUPID package addresses these needs.

The **findclumps** is the main command. It offers a choice of clump-finding algorithms with detailed configuration control, and generates a catalogue in FITS format storing for each peak its centre and centroid co-ordinates, its width, peak value, total flux and number of contributing pixels. Information from the catalogue can be extracted into scripts using STILTS and in particular its powerful **tpipe** command.

```
findclumps in=cube out=clumps outcat=cubeclump perspectrum \
          config="^ClumpFind.par" accept
set peak = ‘stilts tpipe in=cubeclump.FIT cmd="select index==1" \
          cmd=’keepcols Cen3’ omode=out ofmt=csv-nohead‘
```

Here we find the clumps in the three-dimensional NDF called cube using the configuration stored in the text file `ClumpFind.par` that will specify things like the minimum number of pixels in a clump, the instruments's beam FWHM, and tuning of the particular clump-finding algorithm chosen. The PERSPECTRUM parameter requests that the spectra be analysed independently. It would be absent if you were looking for features in a velocity cube. The contents of output NDF clumps is algorithm dependent, but in most cases its data array stores the index of the clump in which each pixel resides, and all contain clump information and cut-outs of the original data about each clump in an extension called CUPID; and its QUALITY array has flags to indicate if

the pixel is part of a clump or background. The last is useful for masking and inspecting the located clumps.

Some experimentation with **findclumps** algorithms and configuration to obtain a suitable segmentation for your data's characteristics is expected. That is why we have not listed any configuration parameters or even chosen a clump-finding algorithm in the example.

Continuing with the example, the catalogue of clumps detected and passing any threshold criteria are stored in `cubeclump.FIT`. (See the CUPID manual for details of all the column names.) We then use **tpipe** to select the third axis centroid co-ordinate (`Cen3`) of the first clump (`index==1`) and store the numerical value in shell variable `peak`. The commands `cmd=` are executed in the order they appear.

This could be extended to keep other columns `keepcols` and return them in an array of the line with most flux given in column Sum.

```
set parline = ‘stilts tpipe in=cubeclump.FIT cmd=’sort -down Sum’ \
              cmd="select index==1" cmd=’keepcols "Cen3 Width3"’ \
              omode=out ofmt=ascii‘
set centre = $parline[3]
set width = $parline[4]
```

Here **tpipe** sorts the fluxes, picks that clump, and writes an ASCII catalogue containing the centroid and width along the third axis storing the one-line catalogue to shell variable `parline`. In this format the names of the columns appear first, hence the required values are in the third and fourth elements. We could have used the csv-nohead output format to give a comma-separated list of values as in the previous example, and split these with **awk**.

**tpipe** has many command options that for example permit the selection, addition, and deletion of columns; and the statistics of columns. The selections can be complex boolean expressions involving many columns. There are even functions to compute angular distances on the sky, to say select a spatial region. See the STILTS manual for many examples.

CUPID also includes a background-tracing and subtraction application **findback**. You may need to run this or **mfittrend** to remove the background before feature detection. Note the if you are applying **findback** to the spectra independently, the spectral axis must be first, even if this demands a re-orient the cube.

```
$KAPPA_DIR/permaxes in=scube out=cubeperm perm=\[3,1,2\]
$CUPID_DIR/findback in=scubeperm sub=no out=backperm box=\[$box,1,1\] \
                    ilevel=0 rms=$noise $
$KAPPA_DIR/permaxes in=backperm out=back perm=\[2,3,1\]
$KAPPA_DIR/sub in1=cube in2=back out=cube_bs
```

In this example, the spectral axis is the third axis of NDF cube. **findback** removes structure smaller than `$box` pixels along each spectrum independently. The resulting estimated backgrounds for each spectrum are stored in NDF backperm, which is re-oriented to the original axis permutation to allow subtraction from cube.

## 5.3 Dealing with Graphical Devices

### 5.3.1 Devices and Globals

KAPPA and other Starlink applications using PGPLOT graphics have a single graphics device for line and image graphics. These can be specified as either PGPLOT or Starlink device names. The current device may be set using the **gdset** command as in the example below.

```
% gdset xwindows
```

You can use the **gdnames** command to query which graphics devices are available, and your choice of device will remain in force, and can be inspected using the **globals** command, *e.g.*

```
% globals
The current data file is             : /tmp/ifu_data
The current graphics device is       : xwindows
The current lookup table file is     : <undefined>
The current transformation is        : <undefined>
The current interaction mode is      : <undefined>
```

unless unset using the **noglobals**, or overriden using the DEVICE parameter option in a specific application. Predicatably, the **gdclear** command can be used to clear the graphics device.

More information on these and other graphics topics can be found in SUN/95.

### 5.3.2 The Graphics Database

Each Starlink application which makes use of the standard Starlink graphics calls, which is most of them, creates an entry in the *graphics database*. This allows the applications to interact, for instance you can display an image to an X-Window display device using the **display** command, and later query a pixel position using the **cursor** command.

The graphics database is referred to as the AGI database, after the name of the subroutine library used to access its contents, and exists as a file stored in your home directory. In most circumstances it will be named for the machine you are working on.

```
% ls ~/*.sdf
-rw-r--r--  1 aa   users  2083328 Jan 02 12:50  /home/aa/agi_pc10.sdf
```

An extensive introduction sprinkled with tutorial examples to making full use of the graphics database can be found in the KAPPA manual (SUN/95) in the sections entitled *The Graphics Database in Action* and *Other Graphics Database Facilities*. There is little point in repeating the information here, however learning to manipulate the graphics database provides you with powerful tools in visualising your IFU data, as well as letting you produce pretty publication quality plots. For instance, the **compare** shell script make fairly trivial use of the graphics database to produce multiple image and line plots on a single GWM graphics device.

### 5.3.3 Pseudo Colour and LUTs

The different display types, such as pseudo colour and true colour, are explained in detail in the Graphics Cookbook (SC/15).

On pseudo-colour displays KAPPA uses a number of look-up tables (commonly refered to as LUTs) to manipulate the colours of your display. For instance you may want to have your images displayed in grey scale (**lutgrey**) or using a false colour 'heat' scale (**lutheat**). KAPPA has many applications to deal with LUTs (see SUN/95), these applications can easily be identified as they all start with "lut", *e.g.* **lutable**, **lutcol**.

## 5.4 Dealing with WCS Information

World Co-ordinate System (*i.e.* real world co-ordinates) information is a complex topic, and one that many people, including the author, find confusing at times.

Starlink applications usually deal with WCS information using the AST subroutine library (see SUN/210 for Fortran and SUN/211 for C language bindings), although notably some parts of FIGARO (such as SPECDRE) have legacy and totally independent methods of dealing with WCS information. See Section 6.6 for an example of how to overcome this interoperability issue.

This general approach has an underlying effect on how Starlink applications look at co-ordinate systems and your data in general.

Starlink applications therefore tend to deal with co-ordinate 'Frames'. For instance, the PIXEL Frame is the frame in which your data is considered in the physical pixels with a specified origin, *i.e.* for a simple two-dimensional example, your data frame may have an $x$ size of 100 pixels and a $y$ size of 150 pixels with the origin of the frame at the co-ordinates (20,30). Another frame is the SKY frame, which positions your image in the real sky—normally right ascension and declination but other sky co-ordinate systems are available and easily transformed. A 'mapping' between these two frames will exist, and will be described, inside the WCS extension of your NDF. The KAPPA **wcscopy** application can be used to copy WCS component from one NDF to another, optionally introducing a linear transformation of pixel co-ordinates in the process. This can be used to add WCS information back into an NDF which has been stripped of WCS information by non-WCS aware applications. Further details about WCS Frames are available in SUN/95 *Using World Co-ordinate Systems*.

Why is this important? Well, for instance, the **display** command will automatically plot your data with axes annotated with co-ordinates described by the current WCS frame, so if your data contains a SKY frame it can (and much of the time will) be automatically be plotted annotaed with the real sky co-ordinates, usyally right ascension and declination, of the observation. It is also critical for mosaicking of data cubes, as explained later in Section 5.9.

Both KAPPA and CCDPACK contain commands to handle WCS NDF extensions. In KAPPA we have the following applications.

- **wcsadd**
  Adds a new co-ordinate Frame into the WCS component of an NDF
- **wcsattrib**
  Manages attribute values associated with the WCS component of an NDF

- **wcscopy**
  Copies WCS information from one NDF to another
- **wcsframe**
  Changes the current co-ordinate Frame in the WCS component of an NDF
- **wcsremove**
  Removes co-ordinate Frames from the WCS component of an NDF
- **wcstran**
  Transforms a position from one NDF co-ordinate Frame to another

while CCDPACK has

- **wcsedit**
  Modifies or examines image co-ordinate system information

which is a very useful utility for handling frames within the extension. For instance,

```
% wcsedit ifu_file

   WCSEDIT
   =======
 1 NDF accessed using parameter IN
MODE - Action to perform (CURRENT,ADD,REMOVE,SET,SHOW) /'SHOW'/ >

   Index Cur  Domain            Title
   ----- ---  ------            -----

ifu_file:
     1        GRID              Data grid indices; first pixel at (1,1,1)
     2        PIXEL             Pixel coordinates; first pixel at (0.5,0...
     3    *   AXIS              Axis coordinates; first pixel at (-1.812...

%
```

here we see that `ifu_file.sdf` has three WCS frames, the base GRID frame with origin (1,1,1), a PIXEL frame with origin (0.5,0.5,0.5) and an AXIS frame with real world co-ordinate mapped on to the PIXEL frame.

**ndftrace** also contains a useful option at this point: `FULLFRAME`. In the following example, most of the **ndftrace** output is excised for clarity, denoted by a vertical ellipsis.

```
% ndftrace mms6_co fullframe

NDF structure /data/mjc/datacube/mms6_co:
                .
                .
                .

Shape:
  No. of dimensions:  3
```

```
  Dimension size(s):  5 x 5 x 83
  Pixel bounds     :  -2:2, -2:2, -54:28
  Total pixels     :  2075
                  .
                  .
                  .

World Coordinate Systems:
  Number of coordinate Frames: 4

  Current coordinate Frame (Frame 4):
    Index               : 4

    Frame title         : "Compound coordinates describing celes..."
    Domain              : SKY-SPECTRUM
    First pixel centre  : 5:36:52.8, -7:26:11, 345.7623

       Axis 1:
          Label: Right ascension
          Units: hh:mm:ss.s

       Axis 2:
          Label: Declination
          Units: ddd:mm:ss

       Axis 3:
          Label: Frequency (LSB)
          Units: GHz
                  .
                  .
                  .

    NDF Bounding Box:

       Axis 1: 5:36:51.0 -> 5:36:53.0
       Axis 2: -7:26:14 -> -7:25:44
       Axis 3: 345.762 -> 345.8139
```

The important information here is the boundary of the image in the PIXEL and CURRENT frame of the NDF, the CURRENT frame being the current 'default' frame which applications accessing the NDF will report co-ordinates from (the CURRENT frame of the NDF can be changed using the KAPPA **wcsframe** command and is usually the last accessed frame in the extension). In this case we can see that the current frame is the SKY-SPECTRUM frame, with extent 5:36:51 to 5:36:53.0 in right ascension, −7:26:14 to −7:25:44 in declination, and 345.762 to 345.8139 Ghz in frequency.

## 5.5 GAIA data visualisation

GAIA is a display and analysis tool widely used for two-dimensional data. For many years the GAIA tool had little to offer for cube visualisation. At the appropriately named Version 3.0, came a toolbox for cubes to permit inspection of planes individually, or as an animation, or as a passband image by collapsing. Now at the time of writing, Version 4.2 GAIA offers further facilities, such as rendering, for cube analysis. GAIA is also under active delevopment. Check the `$STARLINK_DIR/news/gaia.news` file for the latest features extending upon those summarised below.

### 5.5.1 Cube toolbox

If you start up GAIA with a cube

```
% gaia orion_mos_msk.sdf &
```

the cube toolbox appears (see Figure 2). In the upper portion you may choose the axis perpendicular to the axes visible in the regular two-dimensional display; this will normally be the spectral axis, and that is what is assumed this manual. You can select a plane to view in the main display by giving its index or adjusting the slider.

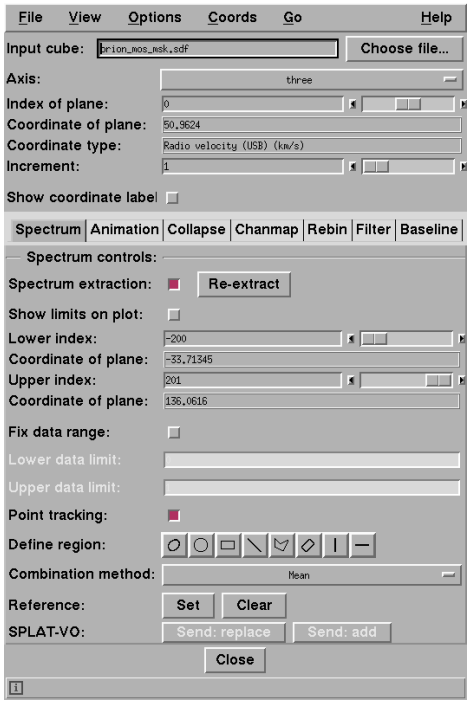



Figure 2: The GAIA toolbox for displaying a cube.

The lower portion of the toolbox has a tabbed interface to a selection of methods. The controls change for each method. Of particular note are the following methods.

`Animation` **tab** This steps through a range of planes automataically at a controlled rate. The animation can be captured to a GIF.

`Spectrum` **tab** This extends the basic behaviour of the spectrum plot (see Section 5.5.2 including defining the spectral limits. You can also define spatial regions graphically, and thereby form a composite spectrum for an object.

`Collapse` **tab** This allows you collapse along the spectral axis over a defined range and offers a wide selection of combination options. Most will form a 'white-light' image in some form, but `Iwc` gives a form of velocity map, and `Iwd` estimates the line dispersions.

`Chanmap` **tab** This forms a channel map comprising a grid of passband tiles, each tile being the result of collapsing a range of planes using one of the Collapse methods. You can inspect the spectral co-ordinate of each plane and with the cursor mark equivalent positions in every tile.

`Rebin` **tab** This allows you to increase the signal-to-noise by rebinning along one or more dimensions.

`Filter` **tab** This controls smoothing of image planes.

If you need to access the Cube toolbox, go to the `Open cube...` option in the `File` menu of the main display.

Once you have an image displayed, be it a plane, passband image or channel map, you can then use GAIA's wide selection of image-display facilities described in SUN/214 (also see SC/17) to enhance the elements under investigation. There are also many analysis capabilities mostly through the (`Image-Analysis` menu), including masking and flux measurement.

### 5.5.2 Spectral plot

One of the most useful facilities for cube analysis is a dynamic spectrum display. Once you have a representative image displayed, click on the mouse over the image a line plot along the spectral axis appears. As you drag the cursor holding down the first mouse button, the spectrum display updates to dynamically to reflect the current spatial location while the data limits are unchanged. If you click again the range will reset to the current spectrum. You can enforce autoscaling as you drag too by enabling `Options`→`Autoscale` in the spectral plot's control bar. This is slower although it allows an intensity independent comparison of the spectra. The **Spectrum** tab mentioned earlier also allows control of the plotting range along both axes. The `Reference: Set` button lets you nominate the current spectrum as a reference spectrum. Then as you subsequently drag the mouse you can compare any of the spectra with the reference spectrum. See Figure 3.

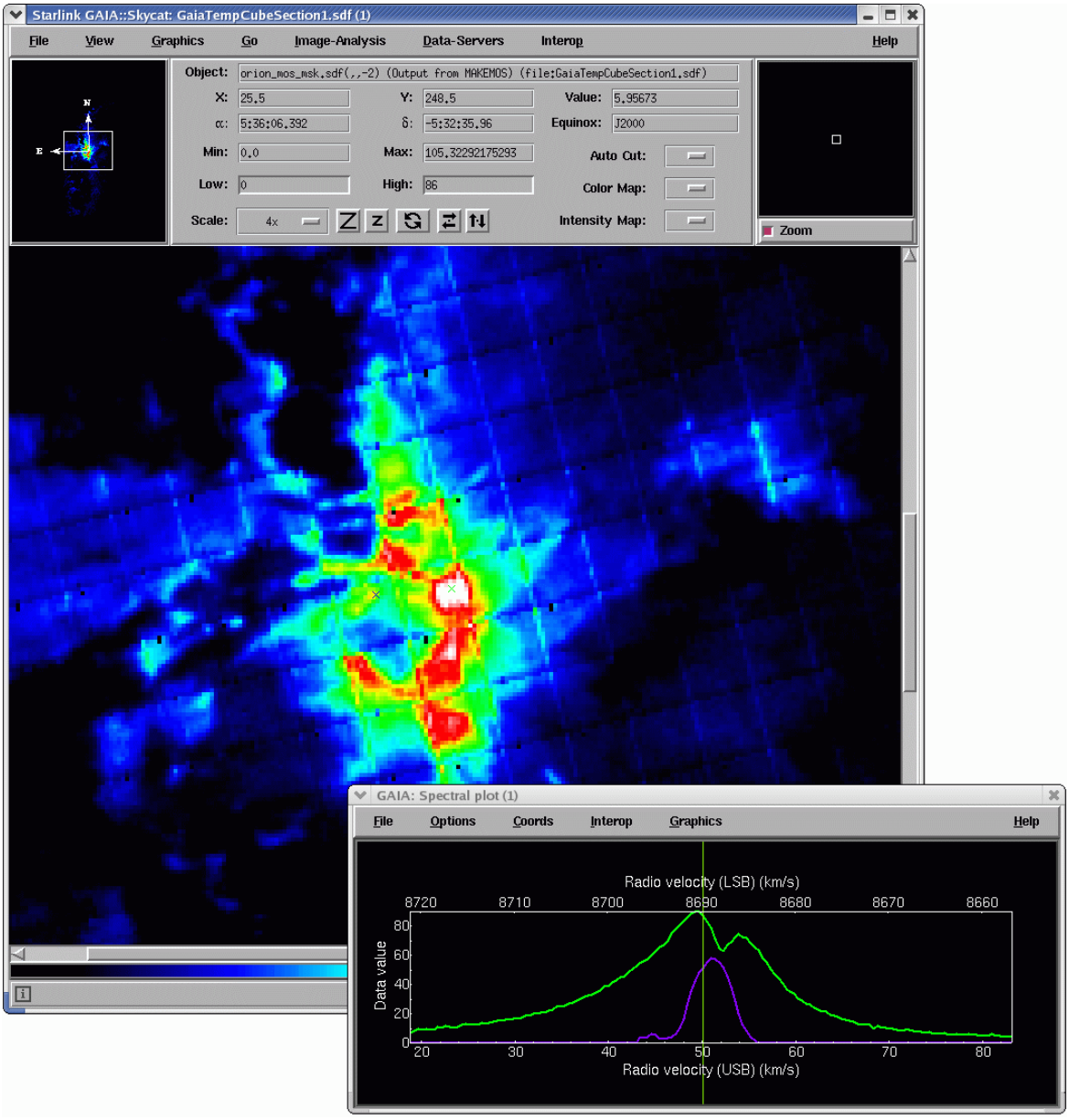



Figure 3: GAIA displays a logarithmically scaled, collapsed cube, from which two spectra are shown. The lighter reference spectrum is from the spatial position marked by the central cross, and the other spectrum corresponds to the left cross.

The vertical red line shows the plane being displayed in the main viewer. You can drag that line with the mouse to adjust the plane on view, say to inspect where emission at a chosen velocity lies spatially.

Further features of the spectral viewer can be found in the online help.

### 5.5.3 Volume Visualisation

The `View` menu in the cube toolbox offers two interactive rendering functions that let you explore your cube in three dimensions.

The first of these is iso-surface. It's akin to contouring of an image extended to three dimensions. Each coloured surface corresponds to a constant data value. In order to see inside a surface, each surface should have an opacity that is less than 1.0. An example using the same Orion dataset is in Figure 4.

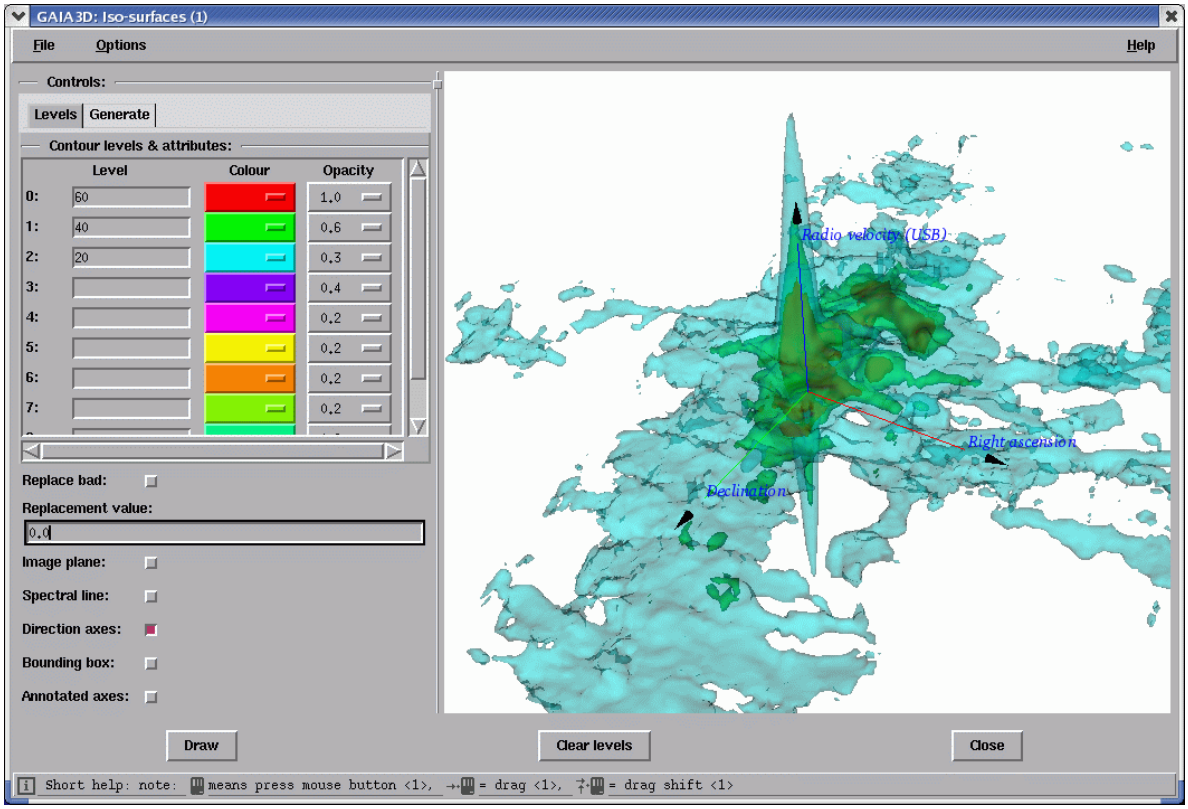



Figure 4: Isophotal contours with various colours and opacities allows you to see different depths.

Just as with the contouring in GAIA, there are different methods to generate iso-surface levels authomatically, as well as manually.

You can adjust the viewing angle and zoom factor either with the mouse, or with the keyboard for finer control. The online help lists the various controls. There are many options to control the appearance, such as directional and annotated axes. GAIA can also display the current slice and displayed spectrum. It is also possible to display two cubes simultaneously, say to compare data from different wavelengths measuring different molecular species. GAIA provides a number of alignment options in this regard.

The second function is volume rendering. It displays all the data within two data limits as a single volume. You assign a colour and opacity to each limit. The controls and options are the same as for the iso-surfaces. See Figure 5.

These visualisation functions place heavy demands on computer memory and CPU. Also a modern graphics card with hardware support for OpenGL makes a huge difference in interactive performance. So you will need a modern machine to get the best out of these tools. Of the two tools, iso-surface is quicker to render and uses less memory.

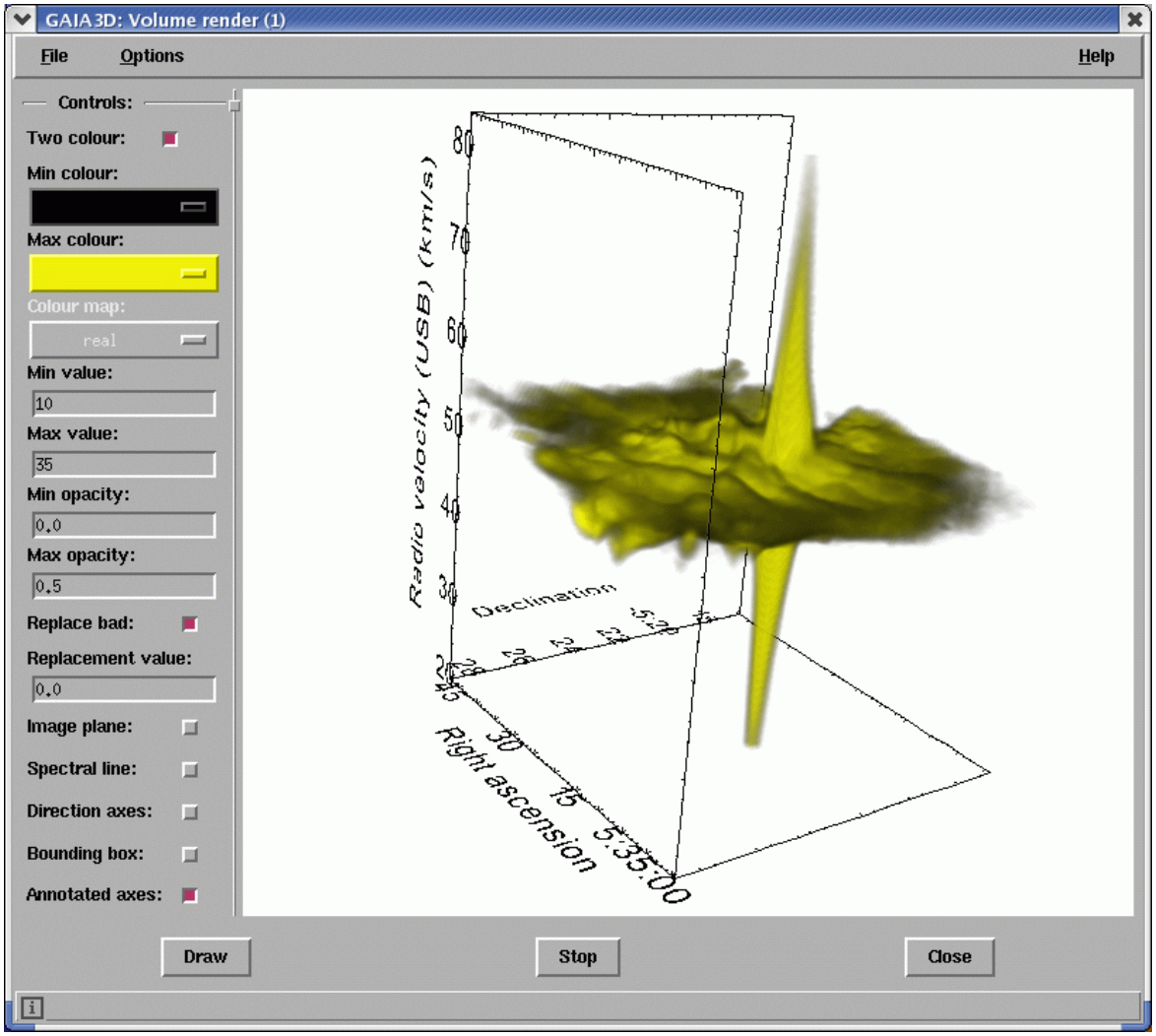



Figure 5: Volume rendering of the Orion dataset. The colour transfer function is a simple mapping between two selected colours between a supplied data range.

## 5.6 IDL and data visualisation

IDL has extensive visualisations capabilities and has many of the tools needed to analyse IFU data cubes available 'off the shelf'. Unfortunately, due to the large file sizes involved, some of the more useful tools availble in IDL can be very slow on machines with small amounts ($<$ 512 Mb) of memory.

### 5.6.1 Display problems

Like many modern UNIX applications IDL suffers problems coping with pseudo and true colour displays. When writing IDL scripts it is important to bear in mind the display type you are using, for pseudo colour (8 bpp) displays you should set the X device type as follows

```
device, pseudo_color = 8
```

while for true-colour displays, commonly found on modern machines running Linux, you should set the X device type to have the appropriate display depth, *e.g.* for a 24-bpp display.

```
device, true_color = 24
```

It should be noted that while the IDL Development Environment (IDLDE) will run in UNIX on a 16 bpp display, only 8 bpp and 24 bpp display are supported for graphics output. If your IDL script or procedure involves graphics display you **must** run IDL either under an 8-bpp pseudo-colour display, or a 24-bpp true-colour display. Ask your system administrator if you are in any doubt as to whether you machine is capable of producing a 24-bpp true-colour display.

Using a true-colour display if you wish to make use of the colour look-up tables (LUTs), and the `loadct` procedure, you should also set the `decomposed` keyword to 0, *e.g.*

```
device, true_color = 24
device, decomposed = 0
```

or alteratively, if you wish to make of 24-bit colour rather than use the LUTs then you should the `decomposed` keyword to 1, *e.g.*

```
device, true_color = 24
device, decomposed = 1
```

For more general information about this issue you should consult the Graphics Cookbook (SC/15).

### 5.6.2 Slicer3

Slicer3 is a GUI widget based application to visualize three-dimensional data which comes with the IDL environment, a simple script to read an IFU data cube into the GUI is shown below,

```
pro display_slicer

   ; Read in TEIFU data cube
   ifu_data = read_ndf('ifu_file', !values.f_nan)

   ; create a pointer to the data array
   ifu_ptr = ptr_new(ifu_data)

   ; run slicer3
   slicer3, ifu_ptr
end
```

here we create a pointer to our data array and pass this pointer to the `slicer3` procedure. The Slicer3 GUI is shown in Figures 6 and 7. These display a projection of an IFU data cube and the user operating on it with the cut and probe tools.

One of the interesting things about the Slicer3 GUI is that it is entirely implemented as an IDL procedure and the code can therefore be modified by the user for specific tasks. More information on Slicer3 can be found in the online help in IDL.

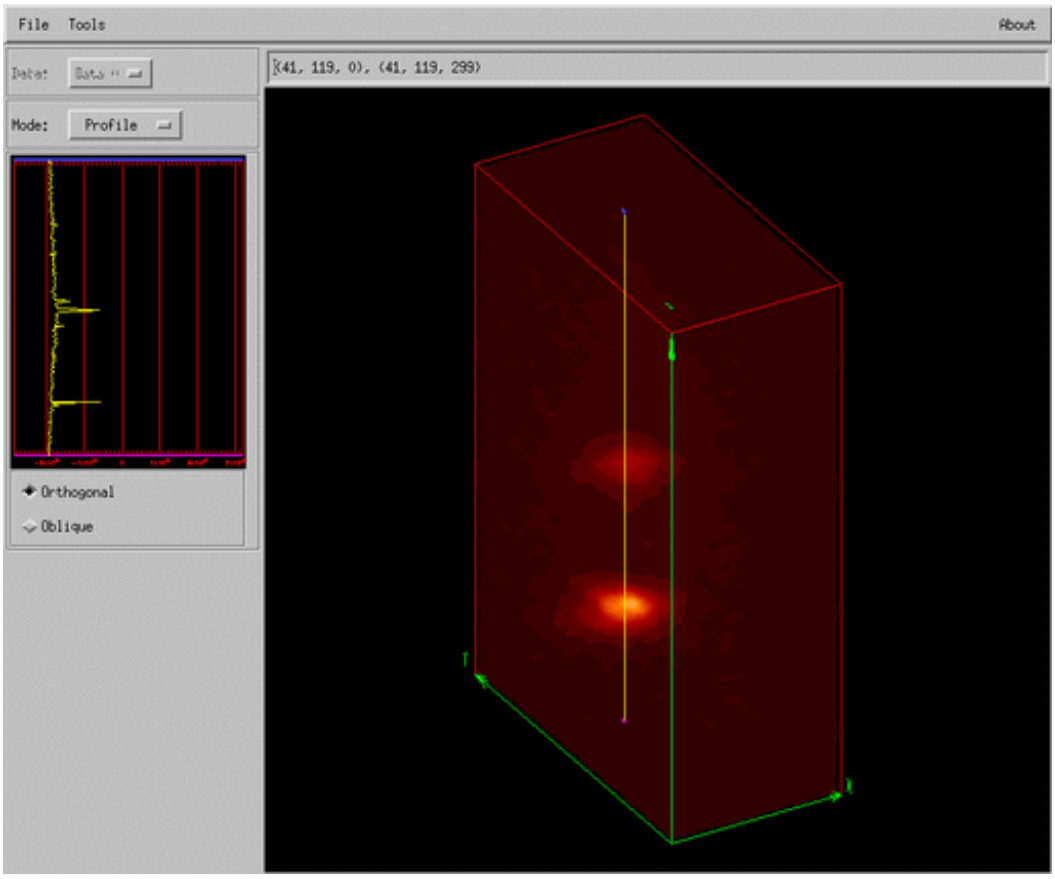



Figure 6: The IDL Slicer3 GUI showing a projection of an NDF data cube, with a cut running in the $\lambda$ direction.

### 5.6.3 The IDL Astronomy Library

No discussion of astronomy data visualisation in IDL can be complete without reference to the IDL Astronomy Library. This IDL procedure library, which is maintained by the GSFC, provides most of the necessary tools to handle your data inside IDL. The library is quite extensive, and fairly well documented. A list of the library routines, broken down by task, can be found at `http://idlastro.gsfc.nasa.gov/contents.html`.

### 5.6.4 ATV Image Viewer

ATV is a frontend for the IDL `tv` procedure. Like the Slicer3 GUI discussed previously, ATV is entirely implemented as an IDL procedure so it is simple to add new routines, buttons or menus if additional functionality is needed. The interface, deliberately, resembles SAOimage so that users can quickly start using it. Detailed usage instructions are available online at `http://www.physics.uci.edu/~barth/atv/instructions.html`. However, to display an a plane in a data cube you can pass an array directly to `atv` as follows.

```
; display plane i in the data cube
atv, array(*,*,i)
```

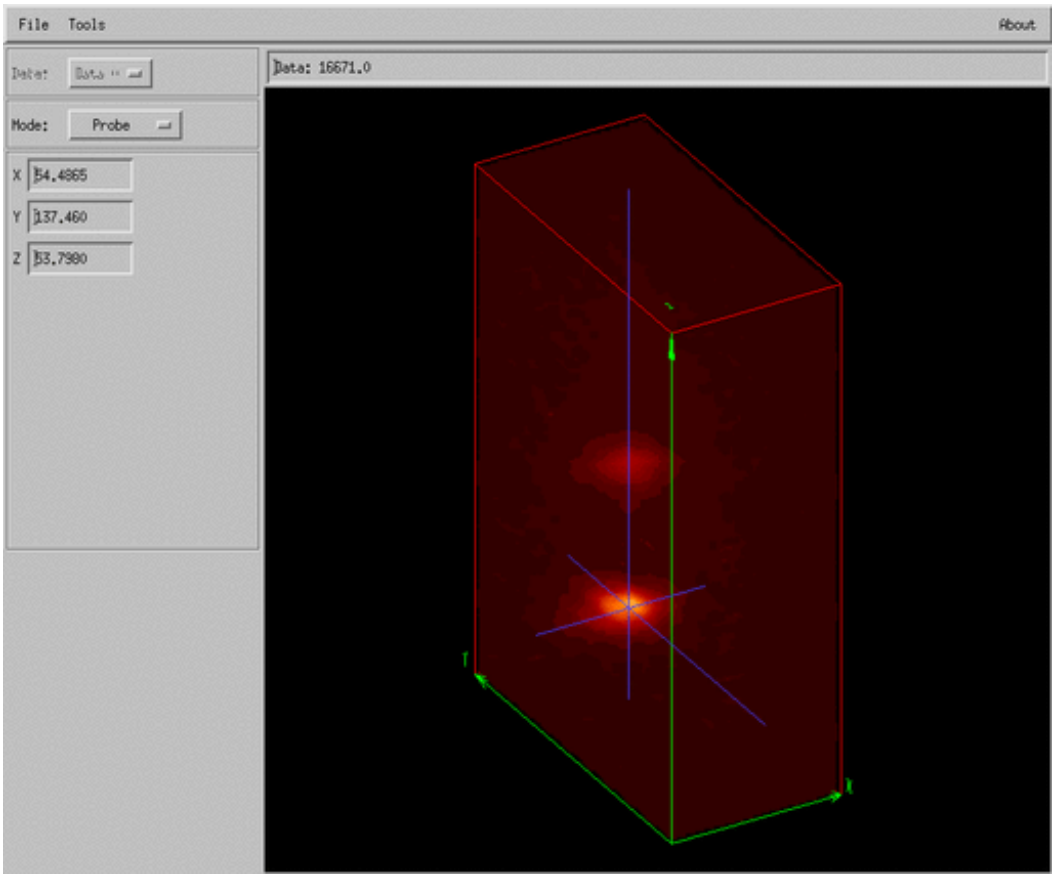



Figure 7: The IDL Slicer3 GUI showing a projection of and NDF data cube, along with the data probe.

## 5.7 IRAF and the Starlink software

Most of the packages we have discussed, *e.g.* KAPPA, FIGARO, and CCDPACK, are available from the IRAF command-line interface (up to the 2004 Spring release) and can be used just like normal IRAF applications (see SUN/217 for details) and IRAF CL scripts can be built around them as you would expect to allow you to analyse your IFU data cubes using their capabilities.

However, it should be noted that Starlink and IRAF applications use intrinsically different data formats. When a Starlink application is run from the IRAF CL, the application will automatically convert to and from the IRAF `.imh` format on input and output. This process should be transparent, and you will only see native IRAF files. However, you should be aware, if you are used to using the Starlink software, that the native NDF format is more capable than the IRAF format and some information (such as quality and variance arrays) may be lost when running the Starlink software from IRAF.

## 5.8 Visualisation using the DATACUBE scripts

The scripts shipped within the DATACUBE package are described in SUN/237. The example dataset has a spectral axis in the wavelength system with units of Ångstrom, however DATACUBE can handle other spectral systems and units, as supported by the FITS standard.

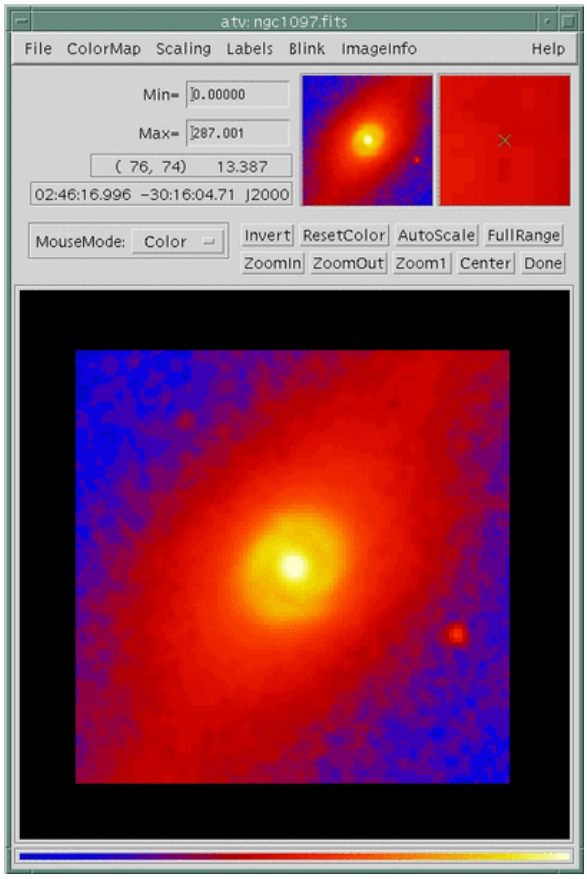



Figure 8: The ATV viewer interface.

### 5.8.1 How do I create a white-light image?

You can make use of the DATACUBE **squash** shell script which is a user friendly interface to the KAPPA **collapse** application allowing you to create both white-light and passband image, *e.g.*

```
% squash -p
NDF input file: ifu_file
     Shape:
       No. of dimensions: 3
       Dimension size(s): 59 x 110 x 961
       Pixel bounds     : 1:59, 1:110, 1:961
       Total pixels     : 6236890
       Wavelength bounds : 4526:10089.8

Lower Wavelength-axis bound : 4526
Upper Wavelength-axis bound : 10089.8

     Collapsing:
       White-light image: 59 x 110
       Wavelength range: 4526--10089.8 Angstrom

NDF output file: out
```

```
      Output NDF:
        File: out.sdf
%
```

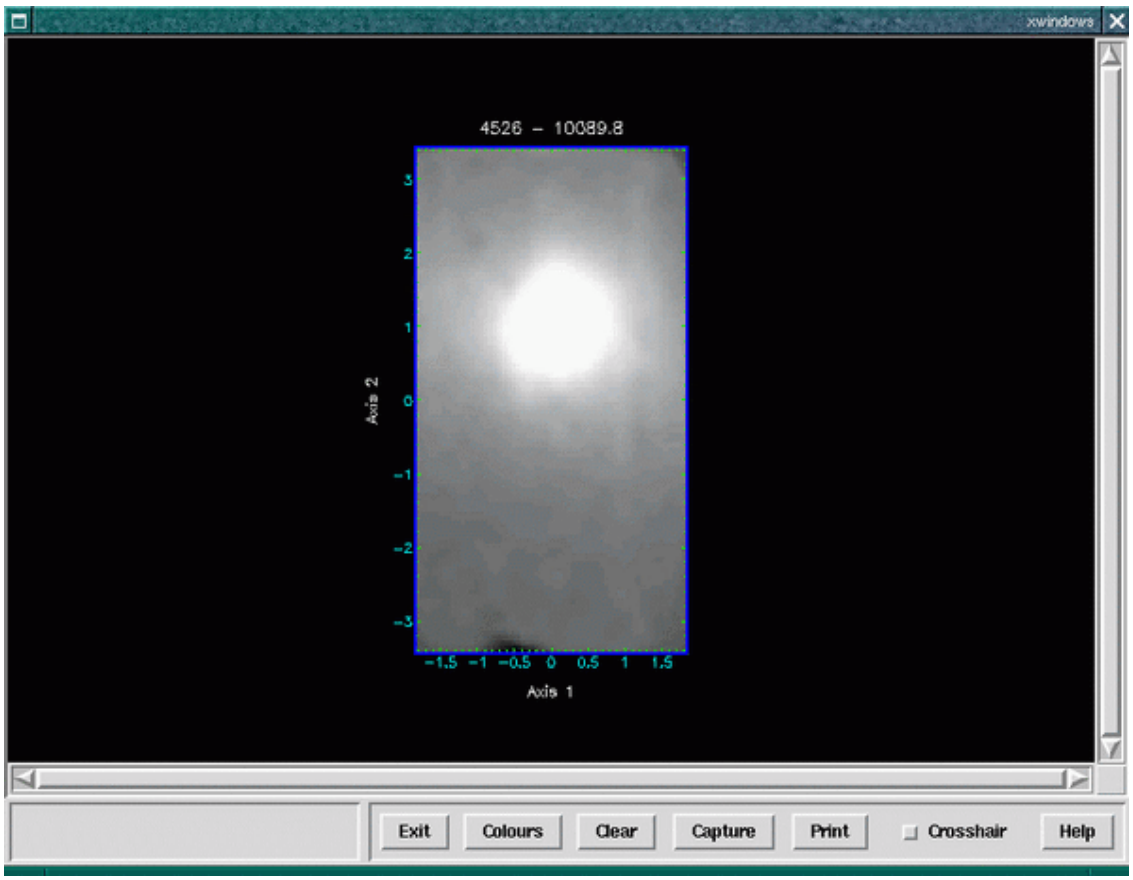



Figure 9: The **squash** script.

Here we make a white-light image from the input data file `ifu_file.sdf`, saving it as a two-dimensional NDF file `out.sdf` as well as plotting it in a GWM window (see Figure 9). Alternatively we can make use of scripts command-line options and specify the input and output files, along with the wavelength bounds, on the command line, as in this example.

```
% squash -i ifu_file -o out -l 4526 -u 10089.8 -p
```

Alternatively we can make direct use of the **collapse** application.

```
% collapse
IN - Input NDF /@/tmp/aa_squash_collapse/ > ifu_file
AXIS - The axis to be collapsed /'1'/ > 3
  Collapsing pixel axis 3 from pixel 1 to pixel 961 inclusive...
OUT - Output NDF > out
%
```

Here we collapse the cube along the third ($\lambda$) axis.

### 5.8.2 How do I create a passband image?

The DATACUBE package offers two ways to create passband images: first we may use (as before) the **squash** shell script, this time specify more restrictive $\lambda$ limits, *e.g.*

```
% squash -i ifu_file -o out -l 5490 -u 5690 -p
```

would create a two-dimensional passband image, collapsing a 200 Å-wide section of the spectral axis.

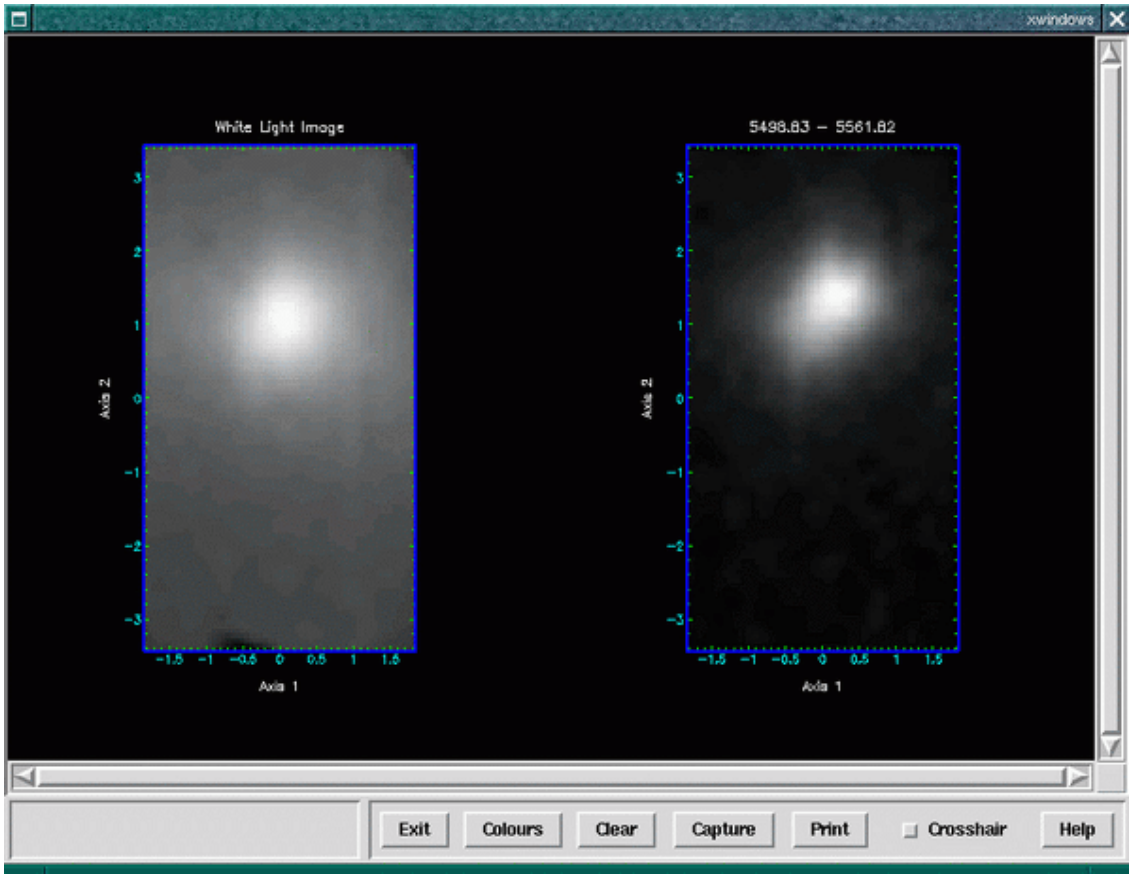



Figure 10: The **passband** script.

Alternatively, we may choose to generate our passband image interactively using the **passband** shell script.

```
%passband
NDF input file: ifu_file

     Input NDF:
       File: ifu_file.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 59 x 110 x 961
       Pixel bounds     : 1:59, 1:110, 1:961
       Total pixels     : 6236890
```

```
      Collapsing:
        White-light image: 59 x 110

 Left click to extract spectrum.

      Extracting:
        (X,Y) pixel             : 32,71
      NDF array analysed        : DATA

         Pixel sum              : 47734.279694
         Pixel mean             : 49.671466903226
         Standard deviation     : 62.703311643991
         Minimum pixel value    : -21.781915
            At pixel            : (31)
            Co-ordinate         : (4697.297)
         Maximum pixel value    : 916.4472266
            At pixel            : (174)
            Co-ordinate         : (5526.027)
         Total number of pixels : 961
         Number of pixels used  : 961 (100.0%)


Zoom in (yes/no): yes

 Left click on lower zoom boundary.
 Left click on upper zoom boundary.

      Zooming:
        Lower Boundary: 5237.92
        Upper Boundary: 5756.03

 Left click on lower boundary.
 Left click on upper boundary.

      Passband:
        Lower Boundary: 5498.83
        Upper Boundary: 5561.82
      Collapsing:
        White-light image: 59 x 110
        Wavelength range: 5498.83--5561.82 Angstrom
      Plotting:
        Left: White-light image.
        Right: Passband image (5498.83--5561.82 Angstrom)
%
```

Here the script presents us with a white-light image and prompts us to click on it to select a good signal-to-noise spectrum, it then asks us whether or not we want to zoom in on a certain part of the spectrum. Let us zoom, and then the script allows us to interactively select a region to extract to create a passband image. It then plots this next to the white-light image for comparison.

Alternatively we can again we can make direct use of the **collapse** application, upon which both the **squash** and **passband** have been built, as shown below.

```
% collapse in=ifu_file out=out axis=3 low=5490 high=5560
```

### 5.8.3 How do I step through passband images?

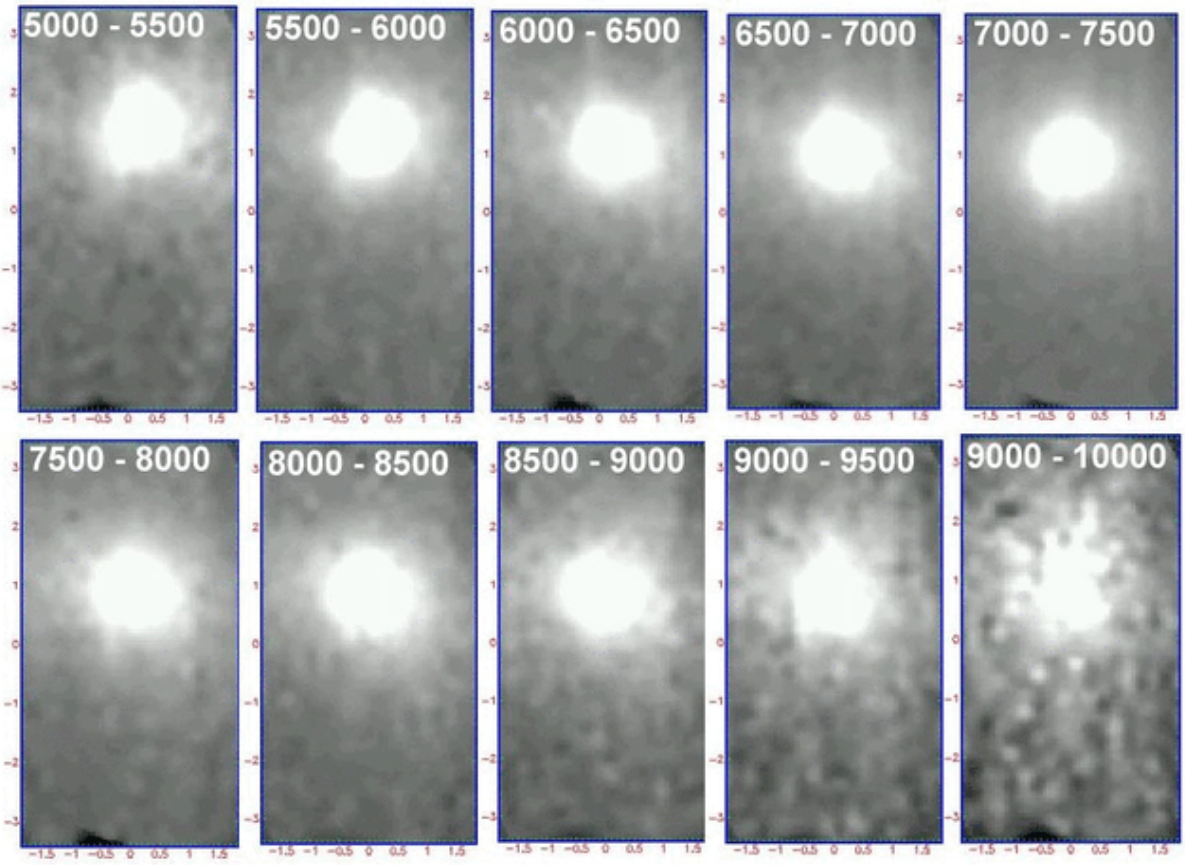



Figure 11: A series of 500Å passband images of 3C 27 produced by the **step** shell script.

The DATACUBE package provides the **step** shell script to carry out this task.

```
% step
NDF input file: ifu_file
     Input NDF:
       File: ifu_file.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 21 x 21 x 961
       Pixel bounds     : 20:40, 60:80, 1:961
       Total pixels     : 423801
       Wavelength bounds : 4526:10089.8 Angstrom

Lower Wavelength-axis bound: 5000
Upper Wavelength-axis bound: 8000
Wavelength step size: 1000

    Stepping:
       Range: 5000--8000 Angstrom
       Step: 1000
```

```
      Collapsing:
        White-light image: 21 x 21
        Wavelength range: 5000--6000 Angstrom
      Output NDF:
        File: chunk_1.sdf
        Title: Setting to 5000--6000

      Collapsing:
        White-light image: 21 x 21
        Wavelength range: 6000--7000 Angstrom
      Output NDF:
        File: chunk_2.sdf
        Title: Setting to 6000--7000 Angstrom

      Collapsing:
        White-light image: 21 x 21
        Wavelength range: 7000--8000 Angstrom
      Output NDF:
        File: chunk_3.sdf
        Title: Setting to 7000--8000 Angstrom
%
```

Here we are asked for the lower and upper bounds of the desired wavelength range, and a step size. The script then generates a series of NDF two-dimensional passband images named `chunk_*.sdf`.

Alternatively, a very simplistic IDL script to step through an TEIFU data cube (stored in an NDF called `ifu_file.sdf`) is shown below.

```
pro step

   ; Read in TEIFU data cube
   ifu_data = read_ndf('ifu_file', !values.f_nan)

   ; Create a window of the right size
   window,xsize=236,ysize=440

   ; Step through the data in the lambda direction
   for i = 0, 960 do begin
      tvscl,congrid(ifu_data(*,*,i),236,440)
   endfor

end
```

The script reads the NDF file in using the `READ_NDF` procedure, creates an IDL graphics window, and steps through the data cube in the wavelength direction a pixel at a time.

There is now a KAPPA command **chanmap** task that forms an image of abutting channels or passbands of equal depth. It is also available from the cube toolbox in GAIA. Once created you may view the passband image with GAIA or **display**. To read off the three-dimensional co-ordinates of features, in GAIA use `Image-Analysis`→`Change Coordinates`→`Show All Coordinates...`, and in KAPPA run **cursor**.

### 5.8.4 How do I extract individual spectra?

The **ripper** shell script in the DATACUBE package was designed as a user-friendly interface over the KAPPA **ndfcopy** application.

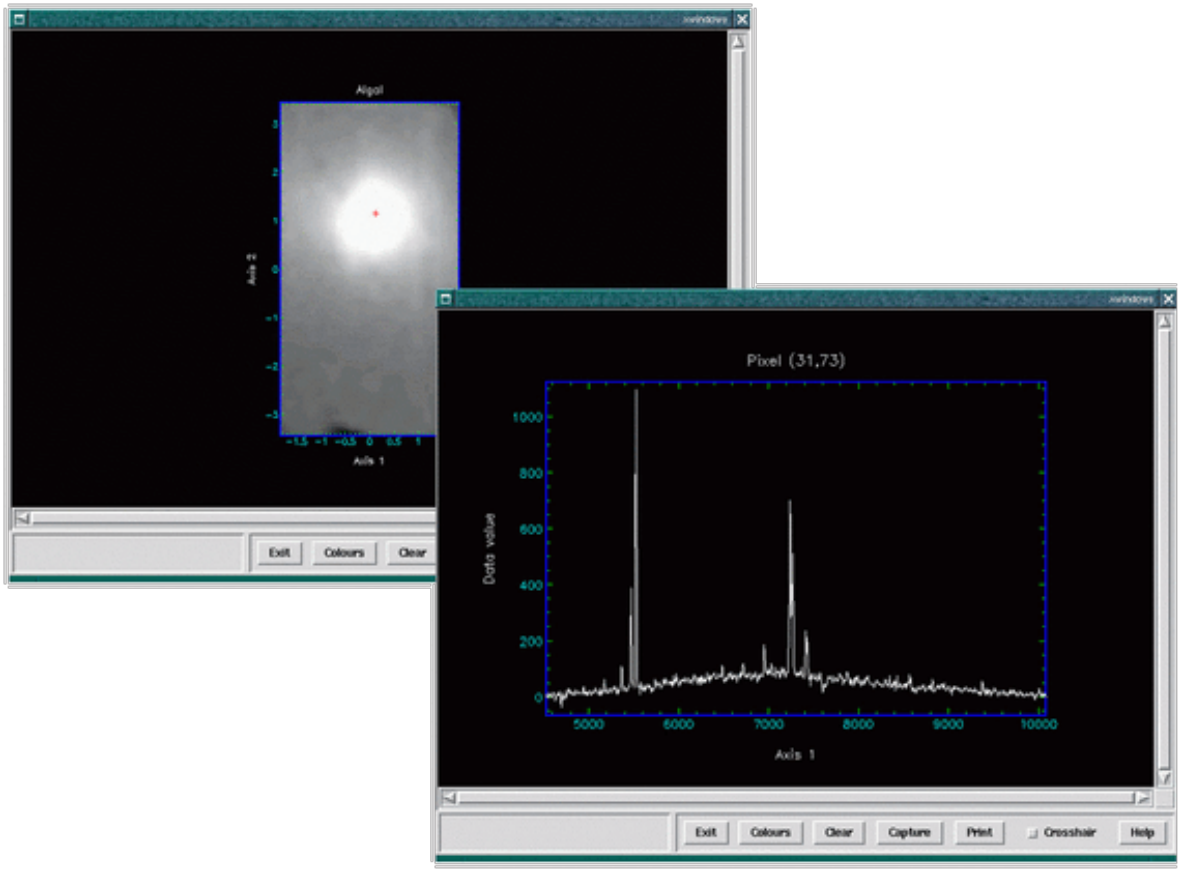



Figure 12: The **ripper** script.

```
% ripper -p
NDF input file: ifu_file
     Input NDF:
       File: ifu_file.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 59 x 110 x 961
       Pixel bounds     : 1:59, 1:110, 1:961
       Total pixels     : 6236890
     Collapsing:
       White-light image: 59 x 110

 Left click to extract spectrum.

     Extracting:
       (X,Y) pixel: 31,73

NDF output file: out
```

```
Output NDF:
  File: out.sdf
```

Here we read in the data cube, `ifu_file.sdf`, and are prompted to click on a pixel to extract the spectrum, see Figure 12.

Alternatively we can use **ndfcopy** directly as underneath.

```
% ndfcopy in="ifu_file(31,73,)" out=out trim trimwcs
```

Here we extract the same spectrum by using NDF sections to specify a region of interest, and the `TRIM` and `TRIMWCS` parameters to reduce the dimensionality of the file to only one dimension.

### 5.8.5 How do I compare spectra?

The **compare** script was written to give you this capability. It allows you to continually select spectra from different parts of the cube, plotting the two most recent to the right of a white-light image of the cube. See Figure 13.

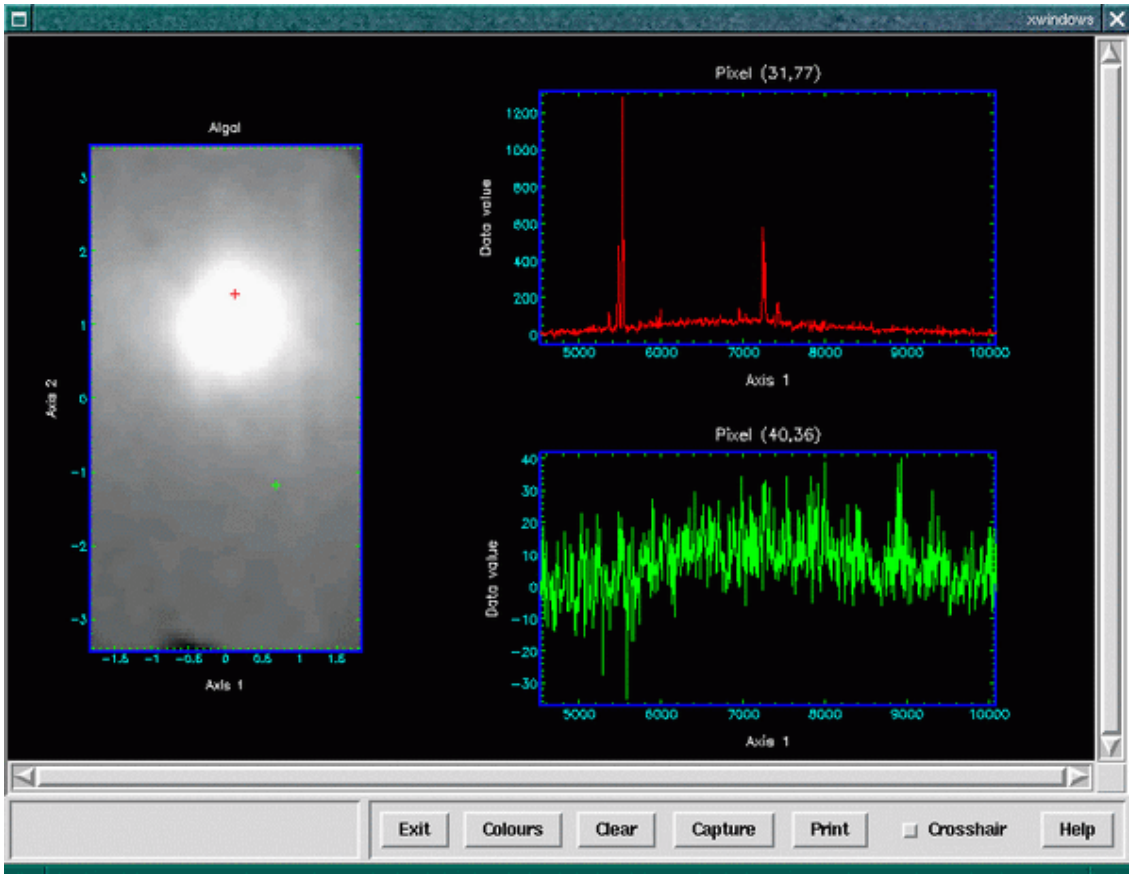



Figure 13: The **compare** script.

For instance,

```
% compare
  NDF input file: ifu_file

      Input NDF:
        File: ifu_file.sdf
      Shape:
        No. of dimensions: 3
        Dimension size(s): 59 x 110 x 961
        Pixel bounds     : 1:59, 1:110, 1:961
        Total pixels     : 6236890
      Collapsing:
        White-light image: 59 x 110

 Left click to extract a spectrum.
 Right click to exit program.

      Extracting:
        (X,Y) pixel             : 31,77
      NDF array analysed        : DATA

         Pixel sum              : 37022.3614004
         Pixel mean             : 38.524829761082
         Standard deviation     : 51.712134738114
         Minimum pixel value    : -11.8406626
            At pixel            : (7)
            Co-ordinate         : (4558.209)
         Maximum pixel value    : 927.2638094
            At pixel            : (174)
            Co-ordinate         : (5526.027)
         Total number of pixels : 961
         Number of pixels used  : 961 (100.0%)

 Left click to extract a spectrum.
 Right click to exit program.

      Extracting:
         (X,Y) pixel            : 40,36
      NDF array analysed        : DATA

         Pixel sum              : 7506.1577888
         Pixel mean             : 7.8107781361082
         Standard deviation     : 8.9202431481454
         Minimum pixel value    : -38.7883342
            At pixel            : (133)
            Co-ordinate         : (5288.419)
         Maximum pixel value    : 47.7443282
            At pixel            : (670)
            Co-ordinate         : (8400.505)
         Total number of pixels : 961
         Number of pixels used  : 961 (100.0%)


 Left click to extract a spectrum.
 Right click to exit program.
```

```
%
```

Here the script presents us with a white-light image and asks us to click on a pixel. It then extracts and displays the spectral axis associated with that pixel in the upper right of the display window. We then have the opportunity to select another pixel, the corresponding spectrum being displayed in the lower right of the display window. We extract only two spectra during this run of the script (Figure 13), pressing the right-hand mouse button to terminate. However, if we carried on and selected a further spectrum it would replace our original in the upper-right panel of the display window. Selecting a further spectrum would replace the lower-right panel. The location of each new spectrum plot alternates.

Also see how GAIAcompares spectra Section 5.5.2.

### 5.8.6 How do I plot stacked spectra?

The DATACUBE package provides two tasks to carry out this process, the **stacker** (see Figure 14) and **multistack** shell scripts.

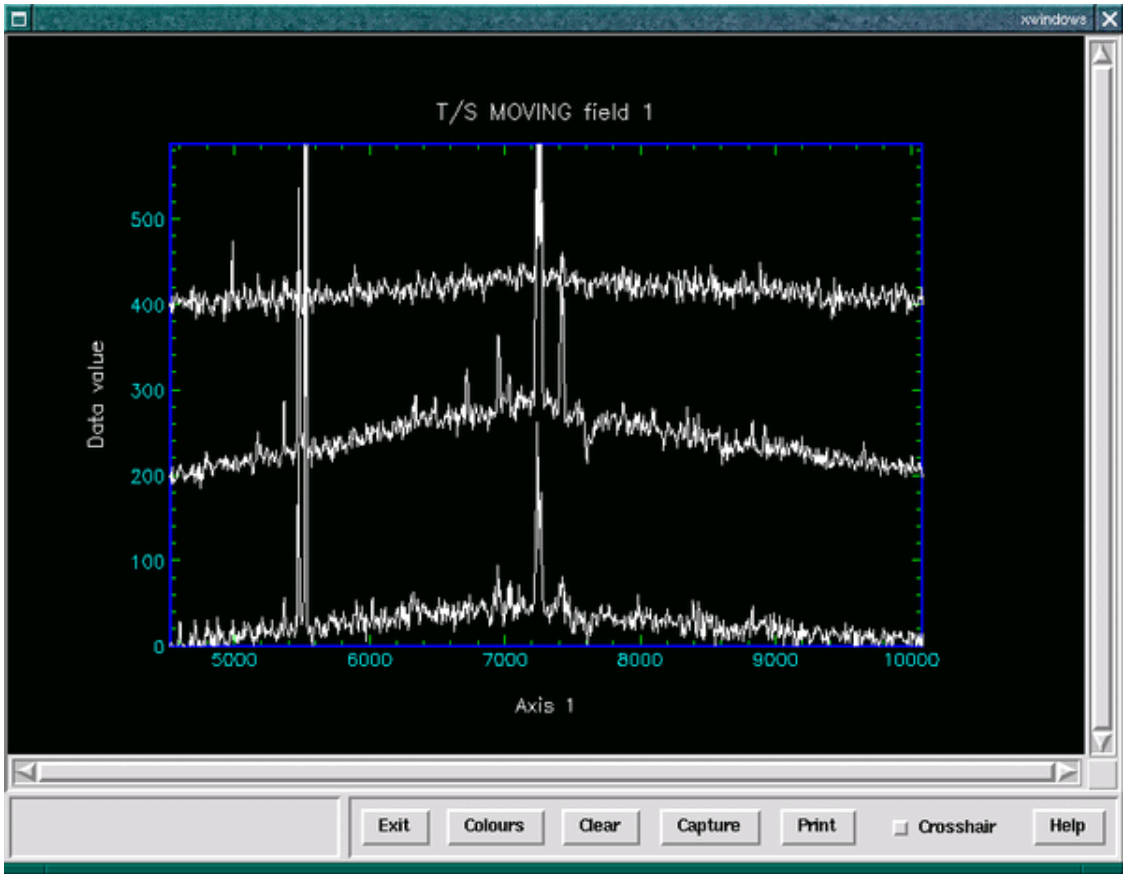



Figure 14: The **stacker** script.

A run of the **stacker** script is shown below.

```
% stacker
NDF input file: ifu_file
```

```
      Input NDF:
        File: ifu_file.sdf
      Shape:
        No. of dimensions: 3
        Dimension size(s): 59 x 110 x 961
        Pixel bounds     : 1:59, 1:110, 1:961
        Total pixels     : 6236890
      Collapsing:
        White-light image: 59 x 110

Number of spectra: 3

 Left click on pixel to be extracted.

      Extracting:
        (X,Y) pixel: 24,80

 Left click on pixel to be extracted.

      Extracting:
        (X,Y) pixel: 30,71

 Left click on pixel to be extracted.

      Extracting:
        (X,Y) pixel: 30,55

Offset: 200

      Adding:
        Adding 0 to spectrum 1
        Adding 200 to spectrum 2
        Adding 400 to spectrum 3

      Plotting:
        Spectrum: 3
        Spectrum: 2
        Spectrum: 1

Zoom in (yes/no): yes

 Left click on lower zoom boundary.
 Left click on upper zoom boundary.

      Zooming:
        Lower Boundary: 6792.26
        Upper Boundary: 7745.58

      Plotting:
        Spectrum: 3
        Spectrum: 2
        Spectrum: 1
%
```

Here we extract three spectra by clicking on a white-light image of the data cube, and these are then plotted with an offset of 200 counts between each spectrum. We then get the opportunity to zoom into a region of interest, and the three spectra are then re-plotted.

The **multistack** script operates in a similar manner, however, here we are prompted for the number of spectral groups required, and the number of spectra in each group. The mean spectrum of for each 'group' of spectra is calculated, and then all the mean spectra are plotted in a stack as before, as seen in the following example.

```
% multistacker
NDF input file: ifu_file

     Input NDF:
       File: ifu_file.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 59 x 110 x 961
       Pixel bounds     : 1:59, 1:110, 1:961
       Total pixels     : 6236890
     Collapsing:
       White-light image: 59 x 110

Number of groups: 3

Number of spectra in group: 4

 Left click on pixel to be extracted.
            .
            .
            .
```

Here we request three groups of four spectra each, *i.e.* we'll get three mean spectra plotted as a stack in the final display.

### 5.8.7 How do I create a grid of spectra?

For an overview visual inspection of the spectra in a cube, it is useful to plot many spectra simultaneously, albeit at a lower resolution, in their respective spatial locations. While KAPPA provides the **clinplot** command to make such a grid, sometimes the sheer number of spatial pixels can make the spectra unreadable and will take some time to plot. Therefore the DATACUBE package offers the **gridspec** shell script. It has an option to average the spectra in the spatial domain, thereby reducing the number of spectra plotted by several times, generating more practical graphics quicker. See Figure 15. Below is an example. The `-z` option requests that the white-light image be shown and a subset of the cube selected with the cursor. The `-b` option sets the spatial blocking factor. Different factors may be given for $x$ and $y$, the two values separated by a comma.

```
% gridspec -b 4 -i ifu_file -z

     Input NDF:
       File: ifu_file.sdf
     Shape:
```

```
        No. of dimensions: 3
        Dimension size(s): 59 x 110 x 961
        Pixel bounds     : 1:59, 1:110, 1:961
        Total pixels     : 6236890

      Collapsing:
        White-light image: 59 x 110

Left click on lower zoom boundary.
Left click on upper zoom boundary.

      Zooming:
      Extracting:
        Lower (X,Y): 20,61
        Upper (X,Y): 42,82

      Plotting:
        Clinplot: Grid of spectra, averaged 4 x 4
```

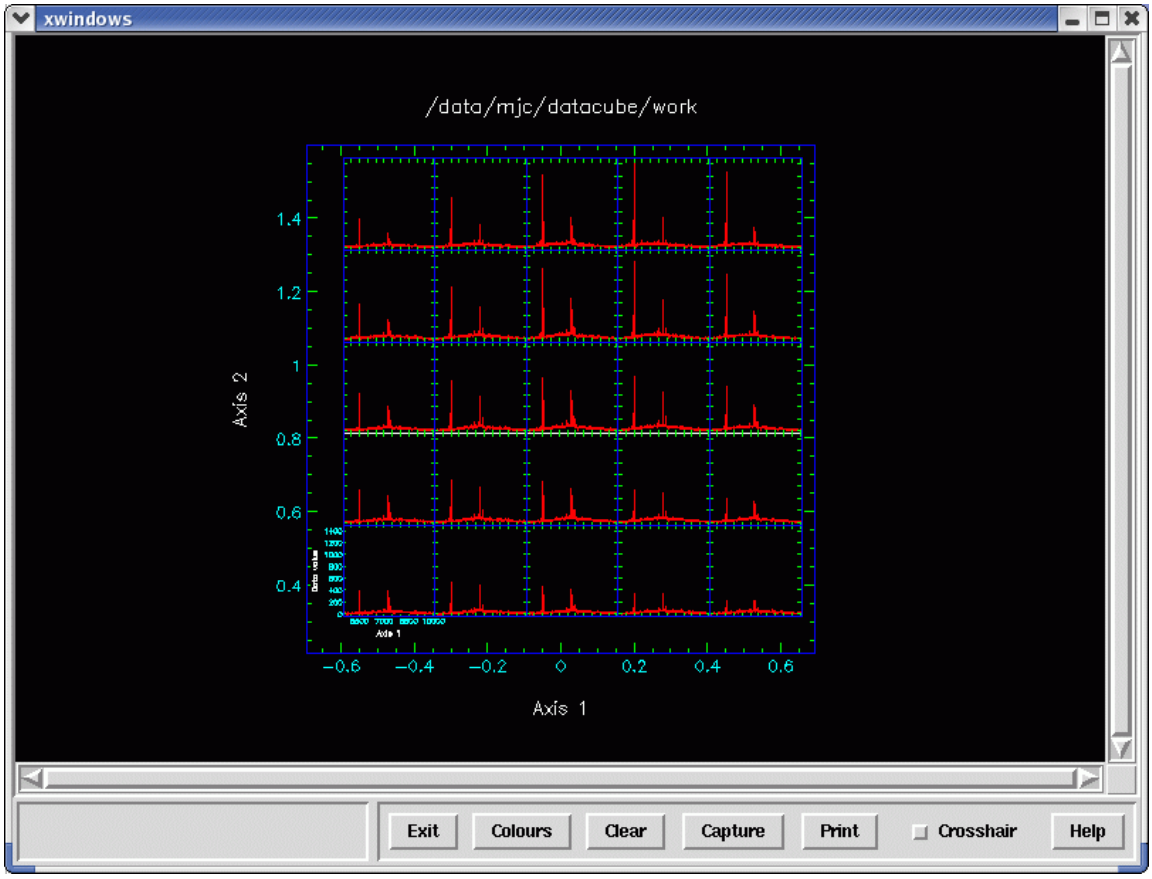



Figure 15: The **gridspec** script, operating on a subset of the 3C 27 observation consisting of the central core of the galaxy, averaging sixteen spectra in the cube for each spectrum plotted. The exterior axes indicate the average spatial co-ordinates of each block of averaged spectra.

A useful strategy is to select a blocking factor that gives no more than ten plots[2] along an axis. Then focus on regions of interest—you either supply an NDF section or select from the white-light image—by decreasing the blocking, and thus increasing both the spatial and spectral plotting resolutions.

### 5.8.8 How do I create a velocity map?

The DATACUBE package provides the **velmap** and **velmoment** shell scripts to manage this fairly complex task. **velmap** fits Gaussians to a selected line, and involves some graphical interaction to select the template spectrum, and initial fitting parameters. For data well characterised by a Gaussian, **velmap** can produce excellent results. However, it is an expensive algorithm to apply to each spatial pixel. Also Gaussian fitting is not appropriate for all data.

The alternative offered **velmoment** collapses along the spectral axis by deriving the intensity-weighted mean co-ordinate, and converts this to a velocity. This is turbo-charged compared with fitting. The downside is that the results will not be as accurate as line fitting, and care must be taken to select regions of the spectra populated by a single significant emission line.

#### By fitting

**velmap** allows you to select the highest signal-to-noise spectrum from a white-light image. You can then interactively fit a line in this spectrum. The script will attempt to automatically fit the same line in all the remaining spectra in the cube, calculate the Doppler velocity of this line from a rest-frame co-ordinate you provide or is read from the NDF WCS, and create a velocity map of that line in the cube. See Figure 16. Below is an example.

```
% velmap -v -p
NDF input file: section

     Input NDF:
       File: section.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 21 x 21 x 961
       Pixel bounds     : 20:40, 60:80, 1:961
       Total pixels     : 423801
     Collapsing:
       White-light image: 21 x 21

 Left click on pixel to be extracted.

     Extracting:
       (X,Y) pixel: 31,72
       Variances: Extension present.

Zoom in (yes/no): yes

 Left click on lower zoom boundary.
 Left click on upper zoom boundary.
```

[2]This is a guide figure. The limit will depend on your hardware and the size of your plotting window. It will be more for a higher-resolution hardcopy.

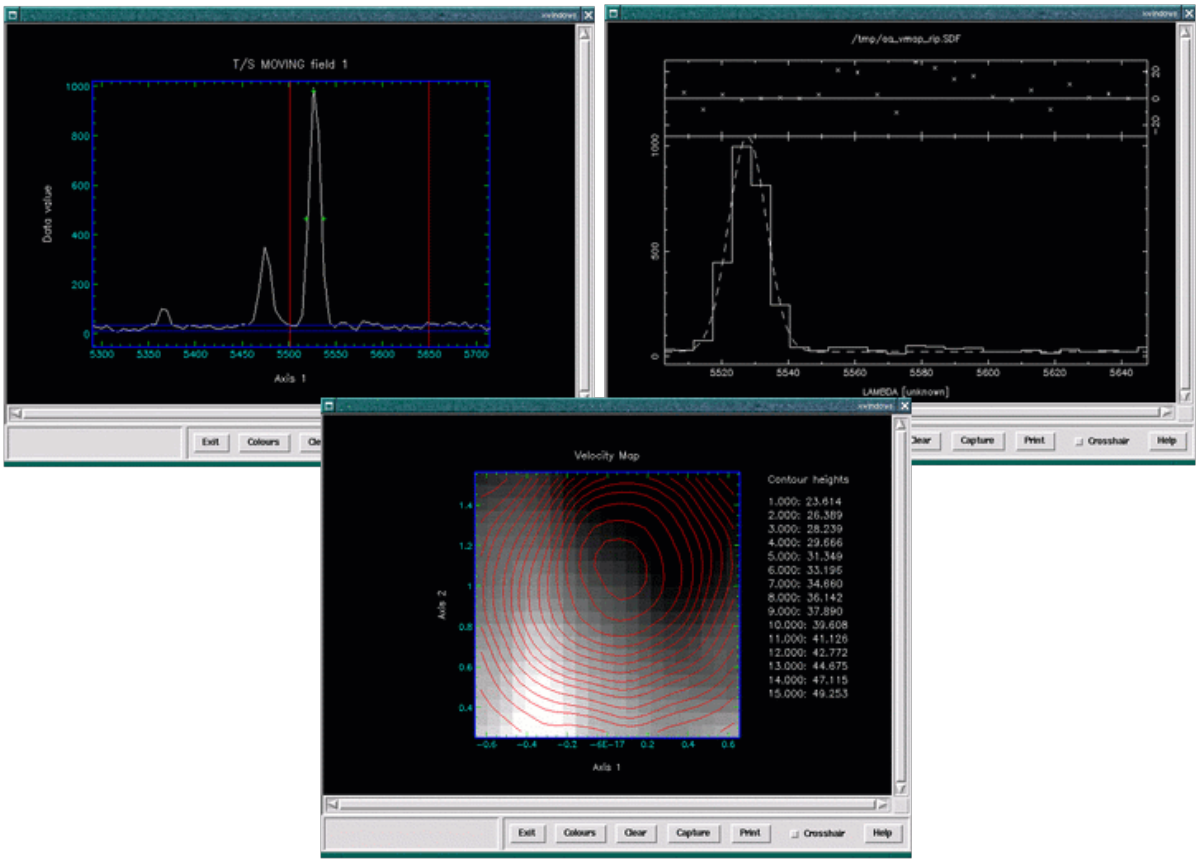



Figure 16: The **velmap** script, operating on a sub-cube of the 3C 27 observation consisting of the central core of the galaxy.

```
    Zooming:
      Lower Boundary: 5289.73
      Upper Boundary: 5714.58

Left click on the lower limit of the fitting region.
Left click on the upper limit of the fitting region.

    Fit Mask:
      Lower Mask Boundary: 5500.97
      Upper Mask Boundary: 5649.73

Left click on your first estimate of the continuum.
Left click on your second estimate of the continuum.

    Continuum:
      First Estimate: 34.2915
      Second Estimate: 10.2744
      Average Value: 22.5659

Left click on the line peak.
```

```
     Line Position:
       Peak Position: 5526.29
       Peak Height: 979.966

 Left click on the left-hand edge of the FWHM.
 Left click on the right-hand edge of the FWHM.

     FWHM:
       Lower Bound: 5518.38
       Upper Bound: 5535.79
       FWHM: 17.41

Rest Wavelength: 5007

     Rest Wavelength:
       Wavelength: 5007 Angstrom
     Fitting:
       Centre Position: 5527.768 +- 0.7394548E-01
       Peak Height: 1020.167 +- 11.59994
       FWHM: 13.28973 +- 0.1748629
       Line integral: 14431.77 +- 163.9841

Fit okay (yes/no): yes

NDF output file: output
     Fitting:
       Spectrum at (20,60): 5530.940 +- 0.5387596
                            31392.451 +- 32.26783 km/s
       Spectrum at (21,60): 5531.210 +- 0.5329667
                            31408.628 +- 31.94727 km/s
                          .
                          .
                          .
       Spectrum at (39,80): 5526.000 +- 0.7465050E-01
                            31096.465 +- 4.475734 km/s
       Spectrum at (40,80): 5526.596 +- 0.8262261E-01
                            31132.175 +- 4.943080 km/s

     Output NDF:
       Converting: Creating NDF from data.
       Origin: Attaching origin (20,60).
       Converting: Attaching VARIANCE array.
       Axes: Attaching AXIS extensions.
       WCS: Attaching WCS information.
       Title: Setting title.

     Plotting:
       Display: Velocity map using percentile scaling.
       Contour: White-light image with equally spaced contours.

Refit points (yes/no): no
%
```

As no automatic process is ever perfect the script allows you to manually refit spectra where it had difficulties in obtaining a fit. If the script was unable to fit an ($x$,$y$) position this value will be marked as `VAL__BADD` in the final velocity map. Should you choose to `Refit points`, clicking on these (normally black) data points in the final output image will drop you into an interactive fitting routine. However, you are not restricted to just refitting those points where the script was unable to obtain a fit, you may manually refit any data point in the velocity map.

There is a `-a` option where you can review each fit at each spatial pixel. The fit parameters can be logged to a Small Text List with the `-l` option.

**By moments**

The second script for generating a velocity map is **velmoment**. This first allows you to select a region of interest. For large regions or noisy spectra you can also request spatial averaging (`-b` option) to reduce the spatial dimensions by integer factors. If your dataset has a WCS SPECTRUM or DSBSPECTRUM Domain, **velmoment** then switches co-ordinate system to one of four velocities, such as optical or radio. It can also cope with NDFs in the UK data-cube format too.

Then comes the heart of the script—the **collapse** task acting upon the spectral axis. It derives the intensity-weighted velocity moment. For simple spectra, *i.e.* a single or dominant emission line, **collapse** finds a representative velocity for the line.

The final stage is to display the map of the velocities. It uses a colour table that runs from blue to red in order to give a visual clue of the relative motions. You may choose to superimpose optional contours of the white-light image upon the velocity map.

Here is an example. Rather than supply a spatial subset as in the **velmap** example, we choose to select the spatial region for analysis by interacting with a 2×2 spatially averaged (`-b 2`) 'white-light' image that the script presents. The wavelength bounds in the NDF section `5300.:5750.` restricts the spectral compression to the range 5300 to 5750 Ångstrom and brackets the [OIII] line. This means that the 'white-light' image is effectively a map of the [OIII] emission. The script converts the intensity-weighted wavelengths into velocities that it displays with a key and suitable colour map. Finally it overlays a ten-level contour plot of [OIII] map on the velocity map. See Figure 17. Below is an example.

```
% velmoment -b 2 -i ifu_file"(,,5300.:5750.)" -r 5007 -p -c 10

     Input NDF:
       File: ifu_file.sdf
     Shape:
       No. of dimensions: 3
       Dimension size(s): 59 x 110 x 79
       Pixel bounds     : 1:59, 1:110, 135:213
       Total pixels     : 512710

Display white-light image to select subset (yes/no): y
     Collapsing:
       White-light image: 59 x 110

 Left click on lower zoom boundary.
 Left click on upper zoom boundary.
```

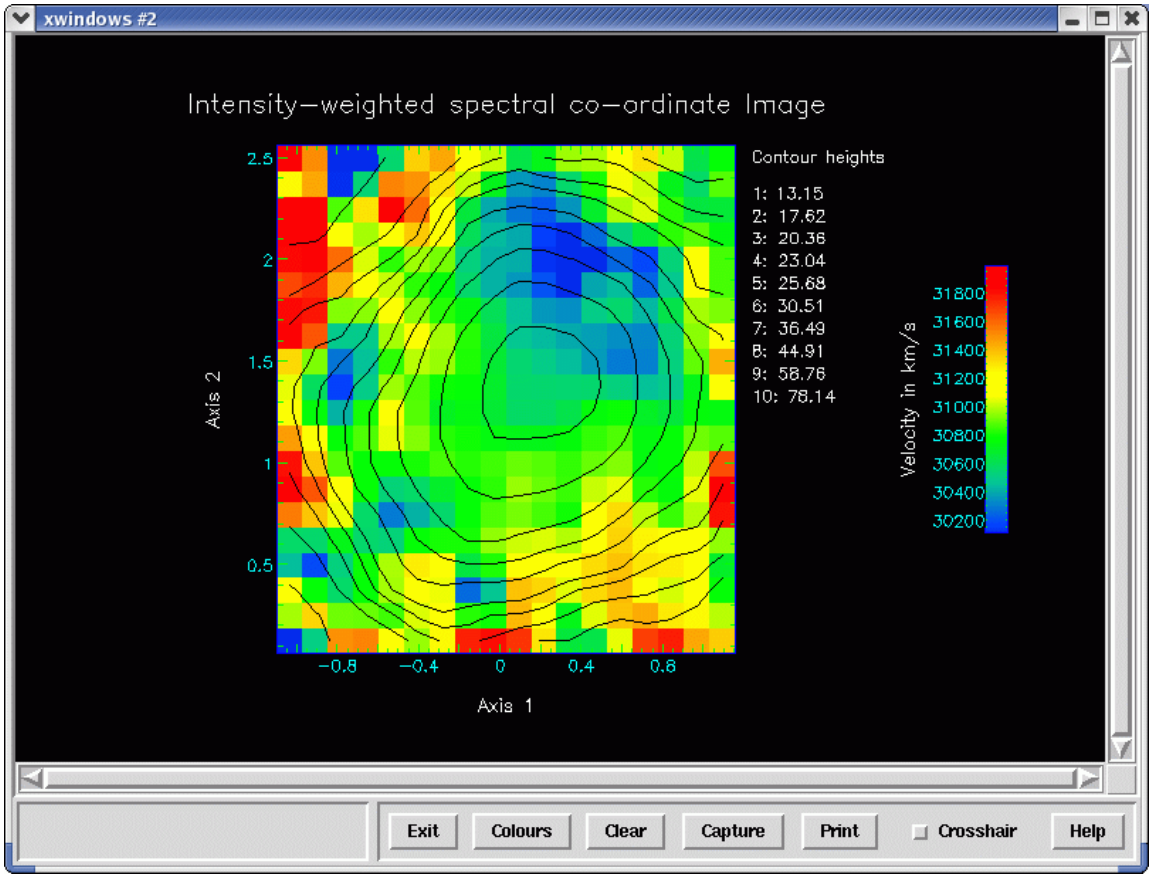



Figure 17: The **velmoment** script, operating on a sub-cube of the 3C 27 observation consisting of the central core of the galaxy.

```
      Zooming:
      Extracting:
        Lower (X,Y): 14,56
        Upper (X,Y): 46,97

      Rest Wavelength :
        Wavelength  : 5007 Angstrom

NDF output file: moment

      Collapsing:
        Intensity-weighted co-ordinate image: 16 x 21
      Plotting:
        Display: Velocity map using percentile scaling.
        Contour: White-light image with equally spaced contours.
```

### 5.8.9 How do I create line-strength map?

The DATACUBE **peakmap** script will generate a line-strength map. The interface to this script is very similar to that of the **velmap** script discussed in Section 5.8.8, and it also generates final

output in a similar form, see Figure 18. Much like the **velmap** script the **peakmap** script allows you to manually refit any spectrum that you think may have been poorly fitted by the automatic process.

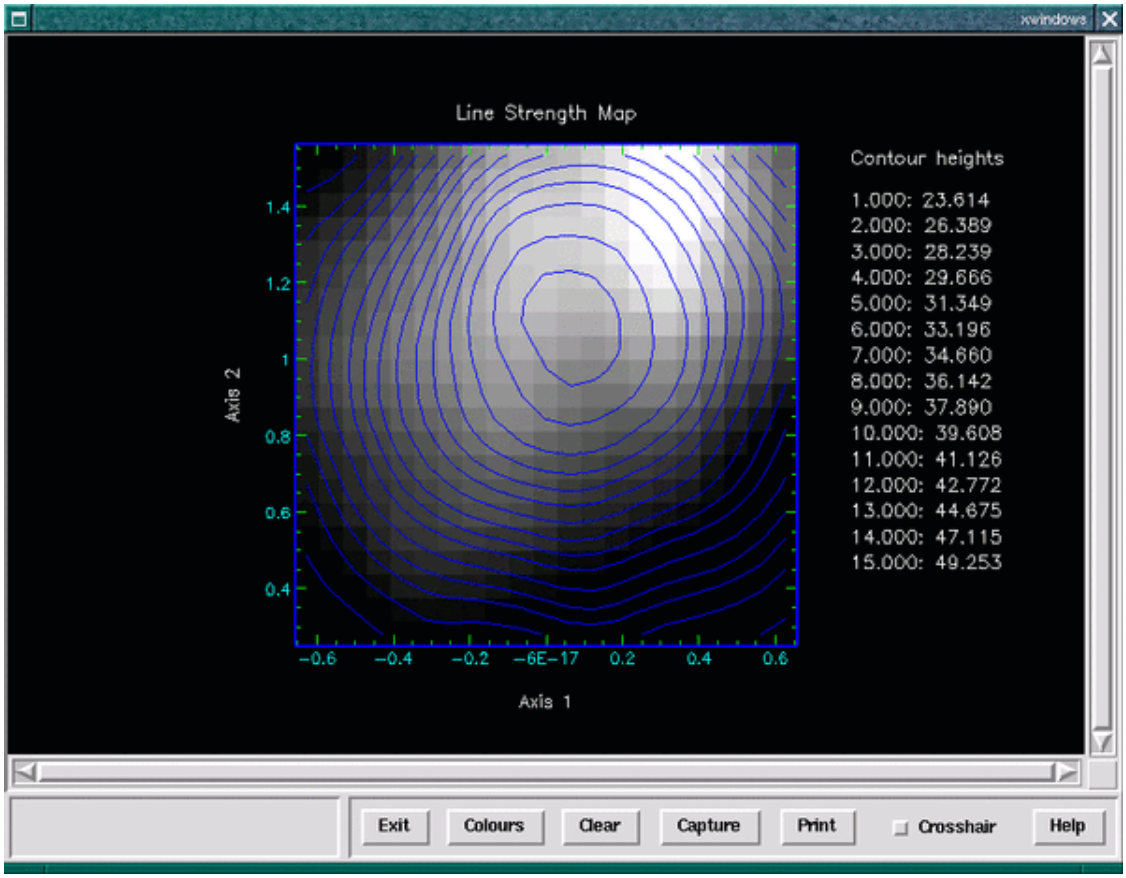



Figure 18: The `peakmap` script.

It should be noted that generating a passband image of a line region, and a line-strength map using the **peakmap** script, should yield similar results. If you are worried about how accurately the automatic fitting of gaussians is doing on a particularly noisy image, then generating a line-strength map and making this comparison is an easy way of deciding a level of trust in your velocity maps, as these are generated using the same fitting algorithms.

### 5.8.10 But they don't handle blended lines!

No, neither **peakmap** nor **velmap** handle multiple gaussians or blended lines. While the FIGARO **fitgauss** application on which these scripts are based can handle fitting blended lines—up to six through the `NCOMP` parameter—automating this process reliably proved to be extremely difficult and made the fitting routine very fragile to signal-to-noise problems.

### 5.8.11 How do I create line-ratio map?

Make a line-strength map of the both lines using **peakmap** or passband images using **squash**, and then use the KAPPA **div** task to divide one by the other to create a ratio map. Here is an example.

```
% div out=ratio_map in1=image1 in2=image2 title="Ratio Map"
```

## 5.9 Mosaicking

Mosaicking IFU data cubes poses unique problems. Firstly the field of view of all the current generation of instruments can be measured in arcseconds, far too small for the traditional approach of image registration to allow the cubes to be matched up the *x*,*y* plane, additionally, the wavelength calibration of the two cubes you wish to mosaic may be entirely different, certainly the case for cubes coming from different instruments.

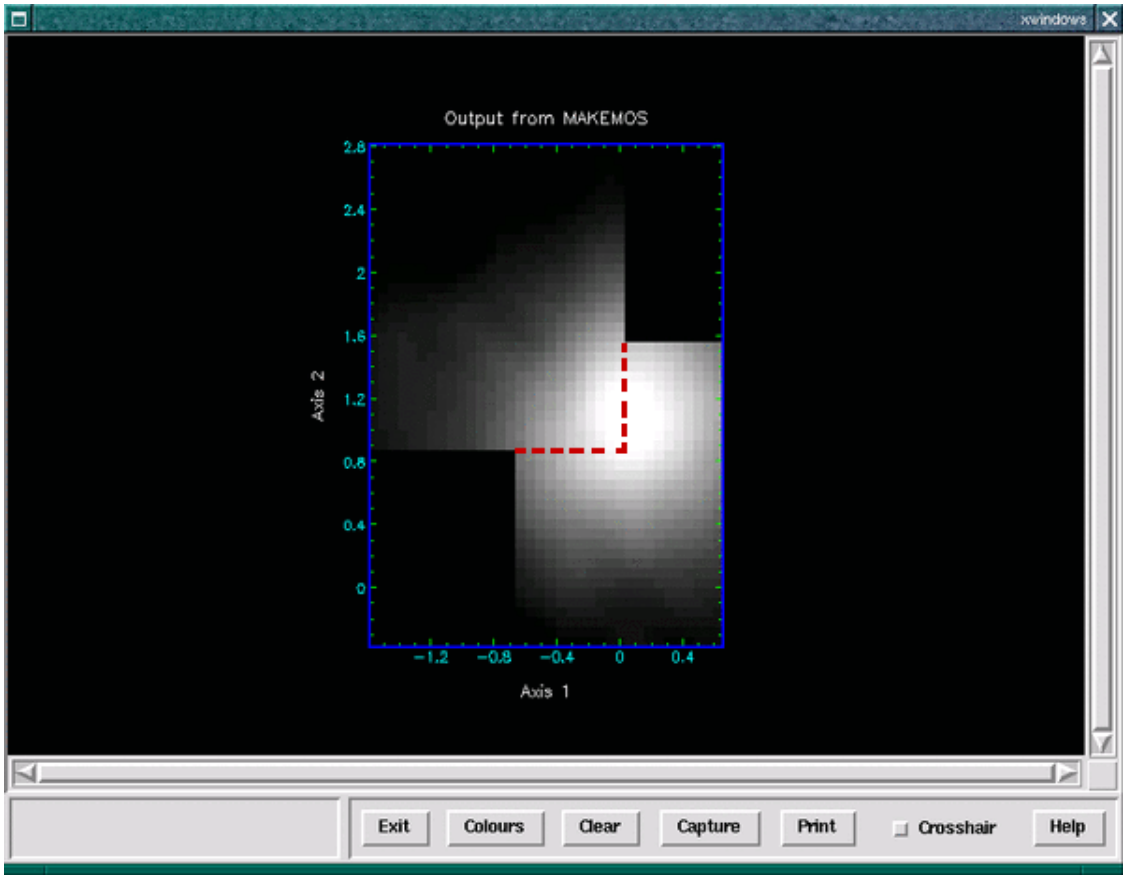



Figure 19: The white-light image of a mosaic of two data cubes created using **makemos**; a dashed line has been drawn on to the image for clarity.

Unfortunately mosaicking therefore relies critically on WCS information provided in the cube FITS headers. Currently the form this information takes varies between cubes from different instruments; and sometimes where active development work is ongoing, between different cubes produced by the same instrument. It is therefore very difficult to provide a 'catch all' script or even recipe to allow you to mosaic two cubes together as yet. The agreement of a standard for the spectroscopic world co-ordinates promulgated in FITS (Greisen *et al.*, 2006, *Representations of spectral coordinates in FITS*, Astronomy & Astrophysics 446,747) should diminish the problem. At the time of writing the Starlink AST already supports spectral frames (these are used to compute the velocities in **velmap**), and most features of this FITS standard.

If the data cubes to be combined have valid WCS information, you should try the **wcsmosaic** task. If your spectral co-ordinates are only present in an AXIS component, see the section *Converting an AXIS structure to a SpecFrame* in SUN/95.

Without valid WCS information we offer a possible approach to the problem. If the two data cubes have identical spectral-axis, *e.g.* wavelength, calibrations and, rather critically, the same number of pixels along the spectral axis, (*i.e.* they are from the same instrument); then the approach we take to the problem is to determine the right ascension and declination of the centre of the *x*,*y* plane and work out the offset between the two frames in pixels. You can probably use the AXIS frame to determine the arcsecond-to-pixel conversion factor, or this may be present in the FITS headers.

Then make use of the CCDPACK **wcsedit** application to modify the origin of the PIXEL frame of one of the cubes such that the two cubes are aligned in the PIXEL frame. Next we suggest you change the current frame to the PIXEL frame (with **wcsframe**) and use **makemos** to mosaic the cubes together (see Figure 19). It should be noted that **makemos** pays *no attention* to the WCS information in the third axis (being designed for two-dimensional CCD frames) which is why having an identical wavelength calibration over the same number of pixels is rather crucial.

Alternatively use can be made of the CCDPACK **wcsreg** application to align the cubes spatially.

Due to the differences in WCS content between instruments, if you want to mosaic cubes from two different instruments together, the only additional advice we can currently offer you is that you should carefully inspect the WCS information provided by the two cubes using (for instance) **wcsedit** and try to find a way to map a frame in the first cube to a frame in the second. It may then prove necessary to re-sample one of the cubes to provide a similar wavelength scale. This may involve using KAPPA tasks **wcsadd** to define a mapping between frames, and **regrid** to resample.

# 6 Writing csh scripts

An excellent introduction to writing csh scripts can be found in the C-shell Cookbook (SC/4).

## 6.1 How do I get pixel positions using the cursor?

Seemingly a trivial problem with a simple solution this turns out to be slightly more complicated than you would initially expect. Presuming we need an *integer* pixel position to give to another application we are using in our script, we can use the following code block to do so.

```
# Grab x,y position.
cursor showpixel=true style="Colour(marker)=3" plot=mark \
      maxpos=1 marker=2 device=xwin frame="PIXEL" > /tmp/cursor_lock

# Wait for cursor input.
while ( ! -e /tmp/cursor_lock )
  sleep 1
end
rm -f /tmp/cursor_lock

# Retrieve the position from the parameter file.
set pos = ‘parget lastpos cursor‘

# Get the pixel co-ordinates and convert to grid indices.  The
# exterior NINT replaces the bug/feature -0 result with the desired 0.
set xpix = ‘calc exp="nint(nint($pos[1]+0.5))" prec=_REAL‘
set ypix = ‘calc exp="nint(nint($pos[2]+0.5))" prec=_REAL‘
```

Here we run the **cursor** application, requesting it to return co-ordinates in the PIXEL Frame, creating a lock file to stop further execution of the script until the user has clicked on the graphics display window. We then delete the lock file and retrieve the cursor position using **parget**. **cursor** returns each co-ordinates as a floating-point value, whereas we require an integer pixel index. We therefore use **calc** to convert from co-ordinates to indices by adding 0.5 and finding the nearest integer.

This functionality has been encapsulated within the script `$DATACUBE_DIR/getcurpos.csh`.

## 6.2 How do I get real world co-ordinate positions using the cursor?

A similar, but slightly easier, problem is to grab real world co-ordinates from a cursor click on a graphics display window.

```
# Grab one position.
cursor showpixel=true style="Colour(marker)=3" plot=mark \
      maxpos=1 marker=2 device=xwin > /tmp/cursor_lock

# Wait for cursor input.
while ( ! -e /tmp/cursor_lock )
  sleep 1
```

```
end
rm -f /tmp/cursor_lock

# Retrieve WCS co-ordinates.
set pos = ‘parget lastpos cursor‘
set x = $pos[1]
set y = $pos[2]
```

Here we run the **cursor** application, again creating a lock file to stop further execution of the script. We then delete the lock file and retrieve the cursor position using **parget**.

This functionality has been encapsulated within the script `$DATACUBE_DIR/getcurpos.csh`.

### 6.3 How do I overplot contours from one image on to another?

The following code will overplot the contours of one dataset on to the greyscale image of another of the same spatial size using the KAPPA **display** and **contour** tasks.

```
# Display greyscale image.
display in=file1 device=xwin mode=per percentiles=[15,98] \
       axes=yes lut=$KAPPA_DIR/grey_lut margin=!

# plot contours
contour ndf=file2 device=xwin clear=no mode=equi axes=no \
       ncont=${numcont} pens='colour=2' margin=!
```

This is useful in many circumstances, for instance, plotting the contours of a white-light image over a passband image, or velocity map.

### 6.4 How do I use scientific notation in bc?

If you want to make use of the **bc** utility to carry out floating-point calculations in `csh` scripts you may come across a problem with scientific notation. In many cases applications return scientific notation in the form `3.0E+08`, or `3.0E-0.8`, while **bc** requires these numbers to be of the form `3.0*10^08`, or `3.0*10^-08`. The following example segment of code takes two numbers in the first format and passes them to **bc** in the correct manner so that it can do some arithmetic with them. **bc** will return a floating-point number (not in scientific notation).

```
# first number
set num1 = ‘echo ${num1} | sed 's/E/\\*10\\^/' | sed 's/+//'‘

# second number
set num2 = ‘echo ${num2} | sed 's/E/\\*10\\^/' | sed 's/+//'‘

# answer
set num3 = ‘echo "scale = 15; ${num1}-${num2}" | bc‘
```

The `scale` specifies the number of decimal places.

An alternative to using **bc** inside your scripts is the KAPPA **calc** command which can evaluate many arbitrary mathematical expressions, as in this extract.

```
# calculate velocity
set delta_lambda = \
    ‘calc exp="’${centre_fit} - ${line_centre}’" prec=_double‘

set velocity = \
    ‘calc exp="’${delta_lambda}/${line_centre}’" prec=_double‘

set velocity = \
    ‘calc exp="’${velocity}*3.0E+08’" prec=_double‘
```

here we evaluate the Doppler velocity of an emission line using three calls to **calc**, although the velocity could have been derived in a single expression.

## 6.5 My file has been converted to NDF. How do I access FITS header keywords?

You may need to access information that was contained in the FITS header to carry out your data analysis, having converted your data to NDF format to make use of the Starlink software the FITS header keywords are still accessible as part of an NDF extension. KAPPA has a number of tools specifically written to handle FITS keywords (see SUN/95 for details).

The recommended way to find the value of a FITS header keyword is by using the **fitsval** application. For instance, you could obtain the value of the FITS keyword CRVAL3 (see Section 4.3) as follows.

```
% fitsval ifu_file CRVAL3
7302.2918645
%
```

This could be used in a script to take appropriate action along with other KAPPA FITS manipulation tools.

```
# Get the input file name.
echo -n "NDF input file: "
set infile = $<

# Check to see if the CTYPE3 keyword exists.
set status = ‘fitsexist ${infile} CTYPE3‘

# If the keyword exists...
if ( ${status} == "TRUE" ) then

  # get the value of CTYPE3.
  set ctype3 = ‘fitsval ${infile} CTYPE3‘

  # Warn the user if its value is not LAMBDA.
  if ( ${ctype3} != "LAMBDA" ) then
     echo "Warning: Axis 3 not type LAMBDA"
  endif

# The keyword does not exist.
else
```

```
  # Warn the user that the keyword is missing.
  echo "Warning: Axis 3 type keyword missing"

endif
```

Here we use KAPPA **fitsexist** comtmnd to check that the the keyword we are interested in exists, then the **fitsval** command to query its value and act on the information.

## 6.6 How do I create an NDF file from an ASCII file?

If it possible to build an NDF file from a flat ASCII text file using the CONVERT application **ascii2ndf** and then use KAPPA and DATACUBE applications to modify the structure and contents of the NDF extensions until it has the correct specifications.

```
# Create a basic NDF from an ASCII file.
ascii2ndf in=${datfile} out=${tmpfile} shape=[${dims[1]},${dims[2]}] \
         maxlen=1024 type='_double'

# Set bad pixels to the magic value VAL__BADD.
setmagic in=${tmpfile} out=${outfile} repval=-9999.99
rm -f "${tmpfile}.sdf"

# Set the origin of the output file.
setorigin ndf=${outfile} origin=[${lbnd[1]},${lbnd[2]}]

# Attach the variance array.
ascii2ndf in=${varfile} comp="Variance" out=${outfile} \
            shape=[${dims[1]},${dims[2]}] maxlen=1024 type='_double'

# We have a similar shaped NDF from which we want to clone the WCS
# and AXIS information and attach to our newly created ${outfile}.

# Clone the AXIS information from an similar shaped NDF.
setaxis ndf=${outfile} like=${likefile}

# Clone the WCS information.  This will be done incorrectly if the
# AXIS structures does not exits before the WCS extension is cloned
wcscopy ndf=${outfile} like=${likefile}
```

Here we take a flat ASCII data file `$datfile` and create an NDF of dimensions `$dims[1]` $\times$ `$dims[2]`, with data type `_DOUBLE`, using **ascii2ndf**. We then cal upon **setmagic** to flag all pixels that have the value $-9999.99$ in the NDF with the standard bad 'magic' value, in this case `VAL__BADD`. **setorigin** sets the pixel origin to `($lbnd[1],$lbnd[2])`. We then use **ascii2ndf** again to attach a variance array, from the ASCII flat file `$varfile`, to our newly created NDF.

For both invocations of **ascii2ndf** we make use of the `MAXLEN` parameter. This is the maximum record length (in bytes) of the records within the input text file, the default value being 512. If you attempt to generate an NDF from a file where many of the entries are double-precision numbers, it might be necessary to set this value higher than the default value otherwise some records (lines) may become truncated leading to 'stepping' effects within your output NDF.

After including our variance array, we attach AXIS information using the **setaxis** application, and incorporate WCS information with `wcscopy` application, both from KAPPA. We clone this

information from an NDF whose world co-ordinate information is the same as our newly created NDF. If we wanted to avoid copying the AXIS information (or if the NDF from where we were cloning had no WCS information) we could make use of the **wcscopy** command's TR parameter to provide a transformation matrix.

Alternatively, we could make use of the KAPPA **setaxis** command to create an AXIS structure within the NDF being cloned obtained from its existing WCS extension via the `$likefile` parameter. **setaxis** should be invoked for each axis. Then we copying the AXIS and WCS structures to our new NDF.

```
# Create an AXIS structure from a WCS extension
setaxis ndf=${likefile} mode=wcs comp=Centre dim=1
setaxis ndf=${likefile} mode=wcs comp=Centre dim=2

# Copy the AXIS structure to our new NDF.
setaxis ndf=${outfile} like=${likefile}

# Copy the WCS extension to our new NDF
wcscopy ndf=${outfile} like=${likefile}
```

The above usage of **setaxis** is sometimes needed when including non-WCS compliant legacy applications, such as **fitgauss**, in scripts, as these legacy tasks do recognise the AXIS structure.

## 6.7 How to make a simple GUI

The **Xdialog** program is designed as a drop-in replacement for the **dialog** and **cdialog** programs. It can convert a simple terminal based script into a program with an X-Windows (GUI) interface. The requirements to install **XDialog** is that you must have the X11 libraries and GTK+ libraries (version 1.2.×) installed on your machine. GTK+ comes installed by default with most recent Linux distributions (being necessary to run the GNOME desktop), however, it can be compiled under both Solaris and Tru64 UNIX, and with the adoption of GNOME as the standard SUN desktop will increasing come installed by default on platforms other than Linux.

Most shell scripts are easily converted to use **Xdialog**, for example we quickly modified the **velmap** script to use it, see Figures 20 and 21. More information about **XDialog** can be found at `http://xdialog.dyns.net/`.

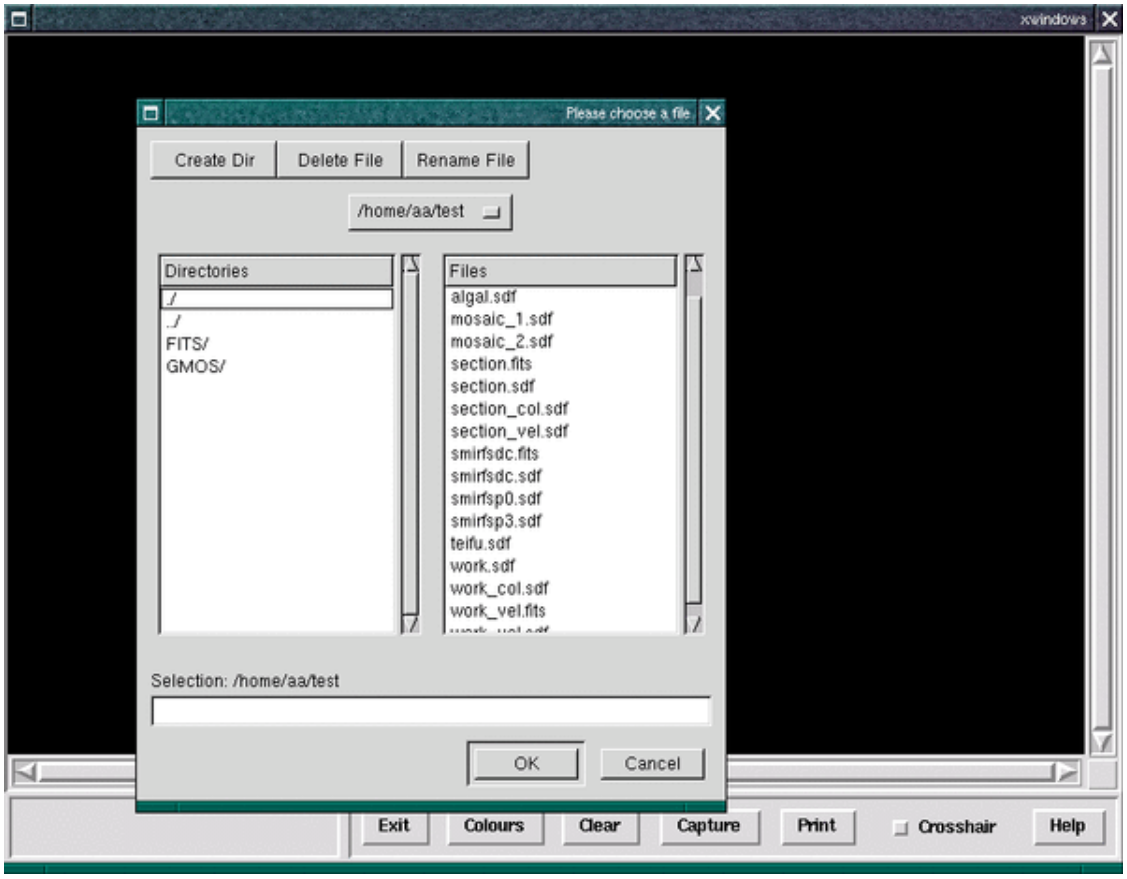



Figure 20: An **XDialog** script based on **velmap**; here it asks for an input file.

# 7 Instrument information sources

- **CIRPASS**
  `http://www.ast.cam.ac.uk/~optics/cirpass/`
- **GMOS**
  `http://www.gemini.edu/sciops/instruments/gmos/gmosIndex.html`
- **INTEGRAL**
  `http://andromeda.roque.ing.iac.es/~astrosw/InstSoft/integral/integral-0.3.tar.gz`
  `http://andromeda.roque.ing.iac.es/~astrosw/InstSoft/integral/manual_de_reducciones.ps.gz`
- **OASIS**
  `http://www.cfht.hawaii.edu/Instruments/Spectroscopy/OASIS/`
- **SAURON**
  `http://www.strw.leidenuniv.nl/sauron/`
- **SMIRFS**
  `http://star-www.dur.ac.uk/~jra/ukirt_ifu.html`
- **TEIFU**
  `http://star-www.dur.ac.uk/~jra/teifu.html`

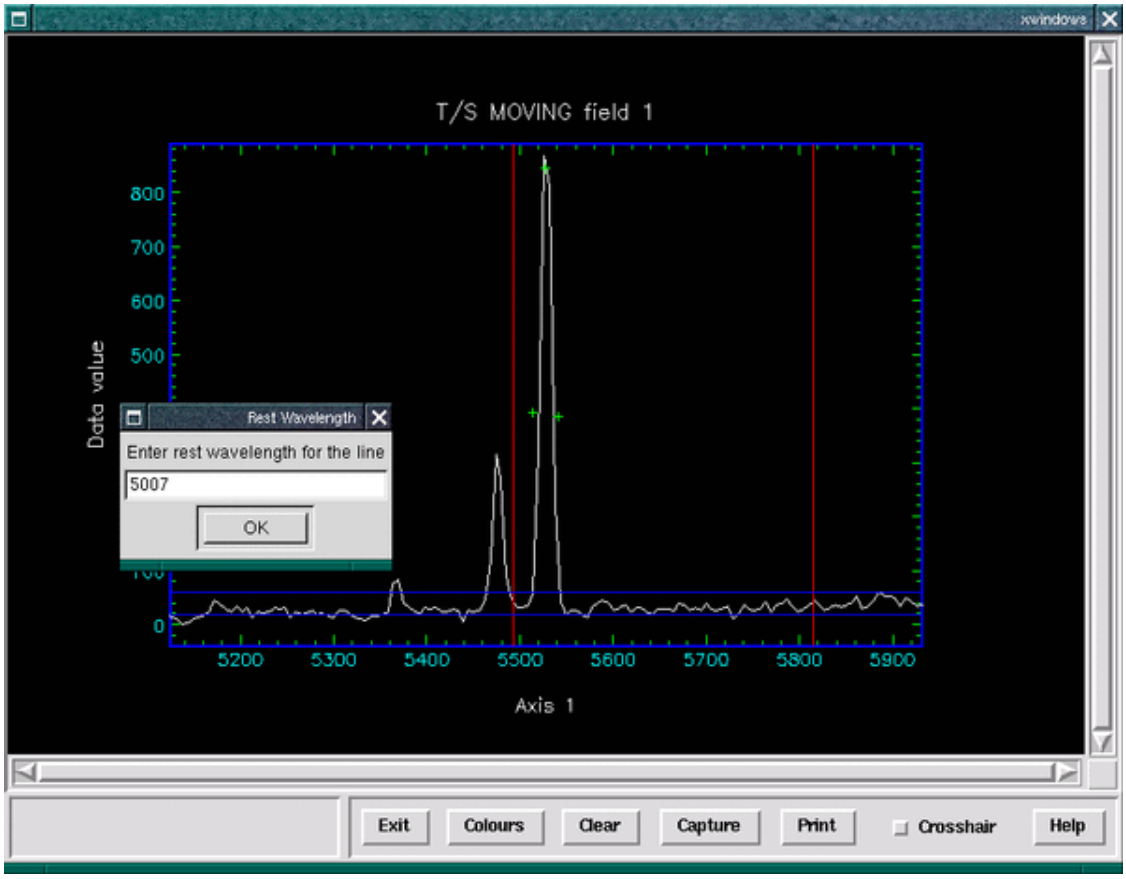



Figure 21: The same **XDialog** script later in the run; here we are prompted for the central wavelength of the line.

- **UIST**
  `http://www.roe.ac.uk/ukatc/projects/uist/`
  `http://www.ukirt.hawaii.edu/instruments/uist/ifu/uistoracdr.html`

# 8 Other information sources

- **ATV Image Viewer**
  `http://www.physics.uci.edu/ barth/atv/`
- **IDL Astronomy Library**
  `http://idlastro.gsfc.nasa.gov/homepage.html`
- **HEASOFT**
  `http://heasarc.gsfc.nasa.gov/docs/software/lheasoft/`
- **XDialog**
  `http://xdialog.free.fr/`
- **GTK+**
  `http://www.gtk.org/`

## References

Economou F., Bridger A., Wright G.S., Jenness T., Currie M.J., Adamson A., *Astronomical Data Analysis Software and Systems VIII*, Mehringer D.M., Plante R.L., Roberts D.A. (eds.), 1999, p.p. 11, ASP Conf. Ser., Vol. 172

## Acknowledgments

In compiling this document I (AA) have again leant heavily on already available material, and the help of many people in the IFS community. However, special thanks should go to Rachel Johnston, Jeremy Allington-Smith, James Turner and Frank Valdes for their co-operation and contributions.

Up-to-date information about UIST data reduction was provided by Stephen Todd who wrote the original ORAC-DR IFU recipes for the instrument.