FITSH – a software package for image processing

András Páll, 2
1Konkoly Observatory of the Hungarian Academy of Sciences, Konkoly Thege Miklós út 15–17, Budapest H-1121, Hungary
2Department of Astronomy, Loránd Eötvös University, Pázmány Péter sétány 1/A, Budapest H-1117, Hungary

Accepted 2011 September 10. Received 2011 August 30; in original form 2011 May 13

ABSTRACT
In this paper we describe the main features of the software package named FITSH, intended to provide a standalone environment for analysis of data acquired by imaging astronomical detectors. The package both provides utilities for the full pipeline of subsequent related data-processing steps (including image calibration, astrometry, source identification, photometry, differential analysis, low-level arithmetic operations, multiple-image combinations, spatial transformations and interpolations) and aids the interpretation of the (mainly photometric and/or astrometric) results. The package also features a consistent implementation of photometry based on image subtraction, point spread function fitting and aperture photometry and provides easy-to-use interfaces for comparisons and for picking the most suitable method for a particular problem. The set of utilities found in this package is built on top of the commonly used UNIX/POSIX shells (hence the name of the package); therefore, both frequently used and well-documented tools for such environments can be exploited and managing a massive amount of data is rather convenient.

Key words: methods: data analysis – techniques: image processing – techniques: photometric.

1 INTRODUCTION
In general, homogeneous data acquisition and subsequent data processing are essential for obtaining proper measurements and characterizing various observations. This is especially true when observing weak signals and analysing related data series with a relatively low signal-to-noise ratio (S/N). Data-processing steps include

(a) calibration and per-pixel arithmetic operations;
(b) source detection, source-profile characterization and (optionally space-varying) point spread function (PSF) determination;
(c) catalogue and cross-matching, coordinate-list processing and astrometry;
(d) image registration, convolution and combination of multiple images;
(e) instrumental photometry (normal, PSF fitting and based on image subtraction);
(f) refined profile modelling, obtaining centroid and shape parameters;
(g) creation of model and artificial images;
(h) data modelling and regression analysis.

Although several software solutions exist for processing imaging astronomical data (see e.g. IRAF, 1 ISIS, 2 SExtractor, 3 and the utilities and wrappers around these implemented in Python 4 or IDL), our intent was to develop a lightweight package that is both a versatile solution for astronomical image processing (focusing mostly on the processing of optical imaging data) and features many of the most recent data-analysis and interpretation algorithms.

This paper presents a software package named FITSH, which provides a set of independent binary programs (called ‘tasks’) that are designed to be executed from a UNIX command-line shell or shell script. Each of these tasks performs a specific operation (see the various steps above), while the details of a certain operation are specified via command-line switches and/or arguments. Therefore this package does not need any higher level operating environment than a standard UNIX shell; however, processing the related data might require a little more knowledge of the shell used (the documentation and available examples use the bash shell5). Additionally, some of the processing steps might require minor or basic operations performed with other tools like awk6 or text-processing

1 http://iraf.noao.edu/
2 Alard & Lupton (1998) and Alard (2000).
3 Bertin & Arnouts (1996).
4 http://www.stsci.edu/resources/software_hardware/pyraf
5 http://www.gnu.org/software/bash/
6 http://www.gnu.org/software/gawk/
utilities (sort, uniq, paste,...). Another advantage of a ‘plain’ UNIX environment is the option for exploiting other shell-level features such as very easy implementation of remote execution, job control, batched processing (including background processing), higher level ways of running tasks in parallel and integration for autonomous observing systems.9

In order to have a consistent implementation of the procedures required by astronomical image processing, several well-known algorithms (which are ultimately used as standard procedures) have also been implemented in the FITSH package. The details of these are known from the literature. These include image preprocessing and calibration (Chromey 2010), basic source extraction, aperture photometry and PSF modelling (Stetson 1987, 1989) or differential image analysis (Alard & Lupton 1998). The new improvements, including routines related to astrometry and catalogue matching, image transformations, consistent photometry on subtracted (differential) images and various regression-analysis methods, have been discussed in more detail in Pál (2009) and references therein; the reader is referred to this work. In this paper we also refer to some parts of Pál’s work during the discussion of various tasks.

The structure of this paper is as follows. Section 2 describes briefly some aspects of the practical implementation. In Section 3, each of the main tasks is described in more detail, featuring small ‘scriptlets’ as a demonstration of the syntax of the various tasks. In Section 4 we summarize the results. The software package has its own website located at http://fitsh.szofi.net/. This site displays information about the program and the tasks (including documentation and detailed examples), is the primary download source of the package itself and additionally serves a public forum for the program users on various topics.

2 IMPLEMENTATION ASPECTS

This section briefly summarizes the main aspects of the implementation of the package tasks. The tasks can be divided into two well-separated groups with respect to the main purposes. In the first group there are programs that manipulate the (astronomical) images themselves (for instance read an image, generate one or perform a specific transformation on one or more images). In the second group, there are the tasks that manipulate textual data, mostly numerical data presented in a tabulated form.

In general, all of these tasks are capable of the following.

2.1 Logging and versions

The codes give release and version information and additionally the invocation can be logged on demand. The version information can be reported by a single call of the given task, moreover it is logged along with the invocation arguments in the form of special FITS key words (if the main output of the actual code is a processed FITS image) and/or textual comments (if the main output of the code is text data). Preserving the version information along with the invocation arguments makes any kind of output easily reproducible.

2.2 Pipelines

All of the tasks are capable of reading their input from stdin (the standard input on UNIX environments) and writing the results to stdout (the UNIX standard output). Since many of these tasks manipulate a relatively large amount of data, the number of unnecessary hard disk operations can therefore be reduced as much as possible. Moreover, in many cases the output of one of the programs is the input of another one: pipes, available in all of the modern UNIX-like operating systems, are basically designed to perform such bindings between the (standard) output and (standard) input of two programs. Therefore, such a capability of redirecting input/output data streams significantly reduces the overhead of background storage operations.

2.3 Symbolic operations

For the tasks dealing with symbolic operations and functions, a general back-end library10 is provided to make a user-friendly interface to specify arithmetic expressions. This kind of approach is seldom used in software systems, since such a symbolic specification of arithmetic expressions does not provide a standalone language. However, it allows an easy and transparent way to perform arbitrary operations and turned out to be very efficient in higher level data-reduction scripts.

2.4 Extensions

Tasks that manipulate FITS images are capable of handling files with multiple extensions. The FITS standard allows the user to store multiple individual images, as well as (ASCII or binary) tabulated data in a single file. The control software of some detectors produces images that are stored in this extended format.11 Other kinds of detectors (which acquire individual images with a very short exposure time) might store the data in the three-dimensional format called ‘data cube’. The tasks are also capable of processing such data-storage formats.

The list of standalone tasks that come with the package and their main purpose is shown in Table 1. The next section discusses these tasks in more detail.

3 TASKS

In this section we summarize the features and properties of the standalone tasks that are implemented as distinct binary executables.

3.1 Basic operations on FITS headers and key words: fiheader

The main purpose of the fiheader utility is to read specific values from the headers of FITS files and/or alter them on demand.

Although most of the information about the observational conditions is stored in the form of FITS key words, image-manipulation programs use only the necessary ones and most of the image-processing parameters are passed as command-line arguments (examples of such key words and data are the gain, image-centroid

---

9 Available from http://szofi.elte.hu/~apal/utils/libpsn/ and developed by the author.

10 One example is detectors in which the charges from the CCD chip are read out in multiple directions: therefore the camera electronics utilizes more than one amplifier and analogue-to-digital (A/D) converter, thus yielding different bias and noise levels.
The **FITSH** package

Table 1. An overview of the standalone tasks included in the package, displaying their main purposes and the types of input and output data.

| Task    | Main purpose                                                                 | Type of input                                                                 | Type of output                                                                 |
|---------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| fiarith | Evaluates arithmetic expressions on images as operands.                      | A set of FITS images.                                                        | A single FITS image.                                                         |
| ficalib | Performs various calibration steps on the input images.                      | A set of raw FITS images.                                                    | A set of calibrated FITS images.                                              |
| fcombine | Combines (most frequently averages) a set of images.                         | A set of FITS images.                                                        | A single FITS image.                                                         |
| ficonv  | Obtains an optimal convolution transformation between two images or uses an existing convolution transformation to convolve an image. | Two FITS images or a single image and a transformation.                      | A convolution transformation or a single image.                               |
| fiheader | Manipulates, i.e. reads, sets, alters or removes some FITS header key words and/or their values. | A single FITS image (alternation) or more FITS images (if header contents alone are read). | A FITS image with altered header or a series of key words/values from the headers. |
| fiign   | Performs low-level manipulations on masks associated with FITS images.       | A single FITS image (with some optional mask).                               | A single FITS image (with an altered mask).                                  |
| fiinfo   | Gives some information about the FITS image in a human-readable form or creates image stamps in a conventional format. | A single FITS image.                                                         | Basic information or PNM images.                                             |
| fiphot | Performs photometry on normal, convolved or subtracted images.               | A single FITS image (with additional reference photometric information if the image is a subtracted one). | Instrumental photometric data.                                               |
| firandom | Generates artificial object lists and/or artificial (astronomical) images.   | List of sources to be drawn to the image or an arithmetic expression that describes how the list of sources is to be created. | List of sources and one single FITS image.                                   |
| fistar | Detects and characterizes point-like sources from astronomical images.       | A single FITS image.                                                         | List of detected sources and one optional PSF image (in FITS format).         |
| fitrans | Performs generic geometric (spatial) transformations on the input image.    | A single FITS image.                                                         | A single, transformed FITS image.                                             |
| fi[un]zip | Compresses and decompresses primary FITS images.                             | A single uncompressed or compressed FITS image file.                         | A single compressed or uncompressed FITS image file.                         |
| grcollect | Performs data transposition on the input tabulated data or some sort of statistics on the input data. | A set of files containing tabulated data.                                    | A set of files containing the transposed tabulated data or a single file for the statistics, also in a tabulated form. |
| grmatch | Matches lines read from two input files of tabulated data, using various criteria (point matching, coordinate matching or identifier matching). | Two files containing tabulated data (which must be two point sets in the case of point or coordinate matching). | One file containing the matched lines and, in the case of point matching, an additional file that describes the best-fitting geometric transformation between the two point sets. |
| grselect | Selects lines from tabulated data using various criteria.                    | A single file containing tabulated data.                                     | The filtered rows from the input data.                                       |
| grtrans | Transforms a single coordinate list or derives a best-fitting transformation between two coordinate lists. | A single file containing a coordinate list and a file that describes the transformation or two files, each one containing a coordinate list. | A file with the transformed coordinate list in tabulated form or a file that contains the best-fitting transformation. |
| lfit    | General purpose arithmetic evaluation, regression and data-analysis tool,     | Files containing data to be analysed in a tabulated form.                    | Regression parameters or results of the arithmetic evaluation.               |

coordinates and astrometrical solutions). The main reasons why this kind of approach was chosen are the following.

(i) First, interpreting many of the standard key words leads to false information about the image in the case of wide-field or heavily distorted images. Such a parameter is the gain, which can be highly inhomogeneous for images acquired by an optical system with non-negligible vignetting; the gain itself cannot be described by a single real number, but rather a polynomial or some equivalent function. Similarly, the standard World Coordinate System information, describing the astrometrical solution of the image, has been designed for small field-of-view images, i.e. the number of coefficients is insufficiently few to constrain properly the astrometry of a distorted image.

(ii) Secondly, altering the meanings of standard key words leads to incompatibilities with existing software. For example, if the format of the key word **GAIN** were changed to be a string of finite real numbers (describing a spatially varied gain) then other programs would not be able to parse this redefined key word.

Therefore, our conclusion was not to alter the syntax of the existing key words but to define some new ones (wherever necessary). The **fiheader** utility enables the user to read any of the key words and allows higher level scripts to interpret the values read from...
the headers and pass their values to other programs in the form of command-line arguments.

### 3.2 Basic arithmetic operations on images: fiarith

The task `fiarith` allows the user to perform simple operations on one or more astronomical images. Supposing all of the input images have the same size, the task allows the user to perform **per pixel** arithmetic operations as well as manipulations that depend on the pixel coordinates themselves. Unlike utilities like `imarith` in IRAF, `fiarith` is capable of processing both symbolic arithmetic operations (carried out via the `libpsn` library, see Section 2.3) and per-pixel operations via a single call.

### 3.3 Basic information about images: fiinfo

The aim of the task `fiinfo` is twofold. First, this task is capable of gathering some statistics and masking information about the image. These include:

1. General statistics, such as mean, median, minimum, maximum and standard deviation of the pixel values;
2. Statistics derived after rejecting the outlier pixels;
3. Estimations for the background level and its spatial variations;
4. Estimations for the background noise;
5. The number of masked pixels, detailed for all occurring mask types.

The most common usage of `fiinfo` in this statistical mode is to deselect those calibration frames that seem to be faulty (e.g., saturated sky flats, aborted images).

Secondly, the task is capable of converting astronomical images into widely used graphics file formats. Almost all of the scaling options available in the well-known `DS9` program (see Joye & Mandel 2003) have been implemented in `fiinfo`; moreover, the user can define arbitrary colour palettes as well. In practice, `fiinfo` creates only images in PNM (portable anymap) format. Images stored in this format can then be converted to any of the widely used graphics file formats (such as JPEG, PNG), using existing software (e.g., `convert`/imagemagick\(^\text{13}\)). Figures in the current paper displaying stamps from real (or mock) astronomical images have also been created using this mode of the task.

### 3.4 Combination of images: ficombine

The main purpose of image combination is to create either a single image from individual images (mainly in order to create higher signal-to-noise ratio frames and/or images with smaller statistical noise) or a mosaic image covering a larger area (mainly from images with smaller fields of view). The task `ficombine` is intended to perform this averaging of individual images. This task has several possible applications in a data-reduction pipeline. For instance, it is used to create the master calibration frames (see also Section 3.5); the reference frame required by the method of image subtraction is also created by averaging individual registered object frames (see also Section 3.11 for details of image registration).

In the actual implementation, such a combination is employed as a **per pixel** averaging, where the method of averaging and its fine-tuning parameters can be specified via command-line arguments. The most frequently used ‘average values’ are the mean and median values. In many applications, rejection of outlier values is required: for instance, omitting pixels affected by cosmic-ray events. The respective parameters for tuning the outlier rejection are also given as command-line options. See Section 3.5 for an example of the usage of `ficombine`, demonstrating its usage in a simple implementation of a calibration pipeline.

### 3.5 Calibration of images: ficalib

In principle, the task `ficalib` implements the evaluation of the equations required by image calibration in an efficient way. It is optimized for the assumption that all master calibration frames are the same for all input images. Because of this assumption, the calibration process is much faster than if it were performed independently on each image, using the task `fiarith`.

The task `ficalib` automatically performs the overscan correction (if the user specifies overscan regions) and also trims the image to its designated size (by clipping these overscan areas). The output images inherit the masks from the master calibration images, as well as additional pixels that might be masked from the input images if these were found to be saturated and/or bloomed. In Fig. 1 a shell script is shown that demonstrates the usage of the tasks `ficalib` and `ficombine` in brief.

### 3.6 Rejection and masking of nasty pixels: fiign

The aim of the program `fiign` is twofold. First, it is intended to perform low-level operations on masks associated with FITS images, such as removing some masks, converting between layers of the masks and merging or combining masks from separate files. Secondly, various methods exist with which the user can add additional masks based on the image itself. These additional masks can be used to mark saturated or blooming pixels, pixels with unexpectedly low and/or high values or extremely sharp structures, especially pixels that result from cosmic-ray events.

This program is a crucial piece in the calibration pipeline if it is implemented using purely the `fiarith` program. However, most of the functionality of `fiign` is also integrated in `ficalib` (see Section 3.5). Since `ficalib` implements the operations of the calibration much more efficiently than would individual calls of `fiarith`, `fiign` is used only occasionally in practice.

### 3.6.1 Masking

As was mentioned above, pixels having some undesirable properties must be masked in order to exclude them from further processing. The `FITSH` package and therefore the pipeline of the whole reduction supports various kind of masks. These masks are transparently stored in the image headers (using special key words) and preserved if independent software modifies the image. Technically, this mask is a bitwise combination of Boolean flags, assigned to various properties of the pixels. Here we briefly summarize our masking method.

Actually, the latest version of the `FITSH` package supports the following masks:

1. **Mask for faulty pixels.** These pixels show strong non-linearity. These masks are derived occasionally from the ratios of flat-field images with low and high intensities.

\(^{13}\) [http://netpbm.sourceforge.net/](http://netpbm.sourceforge.net/)

\(^{14}\) [http://www.imagemagick.org/script/index.php](http://www.imagemagick.org/script/index.php)
Figure 1. A shell script demonstrating the proper usage of the ficalib and ficombine tasks for the example of a simple calibration pipeline. The names for the files containing the input raw frames (both calibration frames and object frames) are stored in the arrays $BIASLIST[*]$, $DARKLIST[*]$, $FLATLIST[*]$ and $OBJLIST[*]$. In this example, these frames are taken from the directory $SOURCE$ and their types are identified by the file names themselves (hence the wildcard-based selection during the declaration of the above arrays). The variable $COMMON_ARGS$ contains all necessary common information related to the frames (geometry, saturation level, etc.) The individual calibrated bias, dark and flat frames are stored in subdirectories of the $TMP$ directory. These files are then combined to a single master bias, dark and flat frame, which are used in the final step of the calibration when the object frames themselves are calibrated. The final calibrated scientific images are stored in the directory $TARGET$. Note that each flat frame is scaled after calibration to have a mean value of 20 000 ADU. In the case of dome flats this scaling is not necessary, but in the case of sky flats, this step corrects for variations in the sky background level (during dusk or dawn). Of course, there are may ways to make this simple pipeline more sophisticated, depending on the actual optical and instrumental set-up.

(ii) **Mask for hot pixels.** The mean dark current for these pixels is significantly higher than the dark current of normal pixels.

(iii) **Mask for cosmic rays.** Cosmic rays cause sharp structures; these structures mostly resemble hot or bad pixels, but do not have a fixed structure that is known in advance.

(iv) **Mask for outer pixels.** After a geometric transformation (dilation, rotation, registration between two images), certain pixels near the edges of the frame have no corresponding pixels in the original frame. These pixels are masked as ‘outer’ pixels. Usually, the pixel values assigned to these ‘outer’ pixels are zero.

(v) **Mask for oversaturated pixels.** These pixels have an analogue-to-digital unit (ADU) value that is above a certain limit defined near the maximum value of the A/D conversion (or below if the detector shows a general non-linear response at higher signal levels).

(vi) **Mask for blooming.** In the cases when the detector has no antiblooming feature or this feature is turned off, extremely saturated pixels causes ‘blooming’ in certain directions (usually parallel to the readout direction). The A/D conversion value of the blooming pixels does not reach the maximum value of the A/D conversion, but these pixels should also be treated as somehow saturated. The ‘blooming’ and ‘oversaturated’ pixels are commonly referred to as ‘saturated’ pixels, i.e. the logical combination of these two respective masks indicates pixels that are related to the saturation and its side effects.

(vii) **Mask for interpolated pixels.** Since the cosmic rays and hot pixels can be easily detected, in some cases it is worth replacing these pixels with an interpolated value derived from the neighbouring pixels. However, these pixels should only be used with caution; therefore these are indicated by a suitable mask for further processes.

We found that the above categories of seven distinct masks are feasible for all kind of applications appearing in the data processing. The fact that there are seven masks – all of which can be stored in a single bit for a given pixel – makes the implementation quite easy. All bits of the mask corresponding to a pixel fit in a byte and we still have an additional bit. This is rather convenient during the implementation of certain steps (e.g. the derivation of the blooming
mask from the oversaturated mask), since there is temporary storage space for a bit that can be used for arbitrary purpose. Fig. 2 shows an example for a mask on a saturated star and the respective encoding as it appears in the FITS header. The interpretation of the encoding is displayed in Table 2.

3.7 Generation of artificial images: firandom

The main purpose of the program firandom is to create artificial images. These artificial images can be used to create either model images for real observations (for instance to remove fitted stellar PSFs) or mock images that are intended to simulate some of the influence related to one or more observational artefacts and realistic effects. In principle, firandom creates an image with a given background level on which sources are drawn. Additionally, firandom is capable of adding noise to the images, simulating the effects of both readout and background noise as well as photon noise. In the case of mock images, firandom is also capable of generating the object list itself. Moreover, firandom is capable of drawing stellar profiles derived from PSFs (by the program fistar, see also Section 3.8).

The program features symbolic input processing, i.e. variations in background level, spatial distribution of the object centroids (in the case of mock images), profile shape parameters, fluxes for individual objects and noise level can be specified not only as a tabulated data set but also in the form of arithmetic expressions. In these expressions one can involve various built-in arithmetic operators and functions, including random number generators. Of course, the generated mock coordinate lists can also be saved in tabulated form. In Fig. 3, some examples are shown that demonstrate the usage of the program firandom.

3.8 Detection of stars or point-like sources: fistar

The source-detection and stellar-profile modelling algorithms (see Pál 2009) are implemented in the program fistar. The main purpose of this program is therefore to search for and characterize point-like sources. Additionally, the program is capable of deriving the point-spread function of the image and spatial variations of the PSF can also be fitted up to arbitrary polynomial order.

The list of detected sources, their centroid coordinates, shape parameters (including FWHM) and flux estimations are written to a previously defined output file. This file can have arbitrary format, depending on our needs. The best-fitting PSF is saved in FITS format. If the PSF is supposed to be constant throughout the image, the FITS image is a normal two-dimensional image.

Table 2. Interpretation of the tags found in MASKINFO key words in order to decode the respective mask. The values of $M, h, v$ and $w$ must always be positive.

| Value | Interpretation |
|-------|----------------|
| $T$   | Use type $T$ encoding. $T = 0$ implies absolute cursor movements, $T = 1$ implies relative cursor movements. Other values of $T$ are reserved for optional further improvements. |
| $-M$  | Set the current bitmask to $M$. $M$ must be between 1 and 127 and it is a bitwise combination of the numbers 1, 2, 4, 8, 16, 32 and 64 for faulty, hot, cosmic, outer, oversaturated, blooming and interpolated pixels, respectively. |
| $x, y$ | Move the cursor to the position $(x, y)$ (in the case of $T = 0$) or shift the cursor position by $(x, y)$ (in the case of $T = 1$) and mark the pixel with the mask value of $M$. |
| $x, y; h$ | Move/shift the cursor to/by $(x, y)$ and mark the horizontal line having the length $h$ and left endpoint at the actual position. |
| $x, y; -v$ | Move/shift the cursor to/by $(x, y)$ and mark the vertical line having the length $v$ and lower endpoint at the actual position. |
| $x, y; h, w$ | Move/shift the cursor to/by $(x, y)$ and mark the rectangle having a size of $h \times w$ and lower left corner at the actual cursor position. |
Figure 3. Three mock images generated using the program firandom. The first image (globular.fits) on the left shows a 'globular cluster' with some field stars as well. For simplicity, the distribution of the cluster stars is Gaussian and the magnitude distribution is quadratic, while field stars are distributed uniformly and their magnitudes are derived from assuming uniformly distributed stars of constant brightness. The second image (coma.fits) simulates a nearly similar effect on the stellar profiles to that of comatic aberration. The shape parameters $\delta$ and $\kappa$ (referred to as $d$ and $k$ in the command-line argument of the program) are specific functions of the spatial coordinates. The magnitude distribution of the stars is the same as for the field stars in the previous image. The third image (grid.fits) shows a set of stars positioned on a grid. The background of this image is not constant. The shell script below the image stamps is used to create these FITS files. The body of the last iterator loop in the script converts the FITS files into PGM format, using the fiinfo utility (see Section 3.3) and the well-known zscale intensity scaling algorithm (see DS9, Joye & Mandel 2003). The images yielded by fiinfo are instantly converted to EPS (encapsulated Postscript) files, the preferred format for many typesetting systems such as Latex.

Otherwise, the PSF data and the associated polynomial coefficients are stored in 'data cube' format and the size of the z ($NAXIS3$) axis is $(N_{\text{PSF}} + 1)(N_{\text{PSF}} + 2)/2$, where $N_{\text{PSF}}$ is the polynomial order used for fitting the spatial variations.

3.9 Basic coordinate-list manipulations: grtrans

The main purpose of the program grtrans is to perform coordinate-list transformations, mostly related to stellar-profile centroid coordinates and astrometrical transformations. Since this program is used exhaustively with the program grmatch, examples and further discussion of this program can be found in the next section, Section 3.10.

3.10 Matching lists or catalogues: grmatch

The main purpose of the grmatch code is to implement the point-matching algorithm that is the key point in the derivation of the astrometric solution and source identification. See Pál & Bakos (2006) or Pál (2009) for more details on the algorithm itself. We note here that although the program grmatch is sufficient for point-matching and source-identification purposes, one may need other codes to interpret or use the output of this program conveniently. For instance, a tabulated list of coordinates can be transformed from one reference frame to another using the program grtrans, while the program fitrans is capable of applying these derived transformations to FITS images, in order to register images to the same reference frame, for example.

3.10.1 Typical applications

As was mentioned earlier, the programs grmatch and grtrans are generally involved in a complete photometry pipeline right after star detection and before instrumental photometry. If the accuracy of the coordinates in the reference catalogue is sufficient to yield a consistent plate solution, one can obtain the photometric centroids
by simply invoking these programs. A more sophisticated example for these programs is shown in Fig. 4. In this example these programs are invoked twice in order both to derive a proper astrometric solution$^{15}$ and to identify properly those stars with larger intrinsic proper motion.$^{16}$

3.11 Transforming and registering images: fitrans

As is known (Pál 2009, Section 2.8), the image convolution and subtraction process requires the images to be in the same spatial reference system. The details of this registration process and the underlying algorithms have been explained in section 2.7 of Pál (2009). The purpose of the program fitrans is to implement these various image-interpolation methods.

In principle, fitrans reads an image and a transformation file, performs the spatial transformation and writes the output image to a separate file. Image data are read from FITS files while the transformation files are presumably derived from the appropriate astrometric solutions. The output of the grmatch and grtrans programs can be directly passed to fitrans. Of course, fitrans takes into account the masks associated with the given image as well as deriving the appropriate mask for the output file. Pixels that cannot be mapped from the original image always have a value of zero and these are marked as outer pixels (see also Section 3.6.1). Modern imaging systems are deployed with high-resolution detectors, therefore the spatial transformation involving exact integration on biquadratic interpolation surfaces might be a computationally expensive process (see section 2.6.3 of Pál 2009). However, distinct image transformations can be performed independently (i.e. a given transformation does not have any influence on other transformations), thus the complete registration process can easily be performed in parallel.

3.12 Convolution and image subtraction: ficonv

This member of the FITSH package is intended to implement tasks related to kernel fit, image convolution and subtraction. In principle, ficonv has two basic modes. First, assuming an existing kernel solution, it evaluates the basic convolution equations (Pál 2009, equation 67) for an image and writes the convolved result to a separate image file. Secondly, assuming a base set of kernel functions (Pál 2009, equation 73) and some model for the background variations (Pál 2009, equation 75) it derives the best-fitting kernel solution for the basic convolution equations. Since this fit yields a linear equation for these coefficients, the method of classic linear least-squares minimization can be efficiently applied. However, the least-squares matrix can have a relatively large dimension in cases where the kernel basis is also large and/or higher order spatial variations are allowed. In the fit mode, the program yields the kernel solution and optionally the convolved and subtracted residual image can also be saved into separate files without additional invocations of ficonv and/or fiarith.

The program ficonv also implements the fit for cross-convolution kernels (Pál 2009, equation 79). In this case, the two kernel solutions are saved to two distinct files. Subsequent invocations of ficonv and/or fiarih can then be used to analyse various kinds of outputs.

Section 2.9 of Pál (2009) discusses the relevance of the kernel solution in the case in which the photometry is performed on

```bash
for base in ${LIST_OF_FRAMES[*]} ; do
gmatch --reference $CATALOG --col-ref $COL_X,$COL_Y --col-ref-ordering -$COL_MAG 
--input $AST/$base.stars --col-inp 2,3 --col-inp-ordering +8 
--weight reference, column=$COL_MAG,magnitude,power=2 
--order $AST.ORDER --max-distance $MAX_MACHDIST 
--output-transformation $AST/$base.trans --output $AST/$base.match || break

gtrans $CATALOG 
--col-xy $COL_X,$COL_Y --input-transformation $AST/$base.trans 
--col-out $COL_X,$COL_Y --output - 1 \ 
grmatch --reference - --col-ref $COL_X,$COL_Y --input $AST/$base.stars --col-inp 2,3 \n--match-coords --max-distance $MAX_MACHDIST --output - 1 \ 
grtrans --col-xy $COL_X,$COL_Y --input-transformation $AST/$base.trans --reverse 
--col-out $COL_X,$COL_Y --output $AST/$base.match

done
```

---

15 By taking into account only the stars with negligible proper motion.

16 These would otherwise significantly distort the astrometric solution.
The \text{FITSH} package

3.13 Photometry: \text{fiphot}

The program \text{fiphot} is the main code in the \text{FITSH} package that performs the raw and instrumental photometry. In the current implementation we were focusing on the aperture photometry performed on normal and subtracted images. Basically, \text{fiphot} reads an astronomical image (FITS file) and a centroid list file, where the latter should contain not only the centroid coordinates but also the individual object identifiers.

In the case of image subtraction-based photometry, \text{fiphot} requires also the kernel solution (derived by \text{ficonv}). Otherwise, if this information is omitted, the results of the photometry are not reliable and consistent (Pál 2009, Section 2.9).

Currently, PSF photometry is not implemented directly in the program \text{fiphot}. However, the program \text{lfit} (Section 3.8) is capable of carrying out PSF fitting on the detected centroids, although its output is not compatible with that of \text{fiphot}. Alternatively, \text{lfit} (see Section 3.16) can be used to perform profile fitting, if the pixel intensities are converted to ASCII tables in advance,\footnote{If the proper object identification is omitted, \text{fiphot} assigns some arbitrary (but indeed unique) identifiers to the centroids; however, in practice it is almost useless.} however it is not computationally efficient.

3.14 Transposition of tabulated data: \text{grcollect}

Raw and instrumental photometric data obtained for each frame are stored in separate files by default as was discussed earlier (see also Section 3.13). We refer to these files as \textit{photometric files}. In order to analyse the per-object outcome of our data reduction, one has to have the data in the form of \textit{light-curve files}. Therefore, the step of photometry (including magnitude transformation) is followed immediately by the step of transposition. See Fig. 6 for an example of how this step looks in a simple case of three photometric files and four objects.

The main purpose of the program \text{grcollect} is to perform this transposition on the photometric data in order to have measurements stored in the form of light curves and therefore to be adequate for further per-object analysis (such as light curve modelling). The invocation syntax of \text{grcollect} is also shown in Fig. 6. Basically, a small amount of information is needed for the transposition process: the name of the input file, the index of the column in which the object identifiers are stored and the optional prefixes and/or suffixes for the individual light-curve file names. The maximum memory that the program is allowed to use is also specified in the command-line argument. In fact, \text{grcollect} does not need the original data to be stored in separate files. The second example in Fig. 6 shows an alternate way of performing the transposition, namely when all data are read from the standard input (and the preceding command \text{cat} dumps all the data to the standard output; these two commands are connected by a single unidirectional pipe).

The actual implementation of the transposition inside \text{grcollect} is very simple: it reads the data from the individual files (or from the standard input) until these data fit in the available memory. If this temporary memory is full of records, this array is sorted by object identifier and the sorted records are written/concatenated to distinct files. The output files are named based on the appropriate object identifiers. This procedure is repeated until there are no available data. Although this method creates the light-curve files, it means that neither the whole process nor the access to these light-curve files is effective. If either the number of the files to be transposed or the number of the records in a single file exceeds a certain limit (which can be derived from the available memory, the record size and the

\begin{verbatim}
SELF=$0; base="$1"
if [ -n "$base" ] ; then
  fitrans $(FITS)/$base.fits \
    --input-transformation $(AST)/$base.trans --reverse -k -o $(REG)/$base-trans.fits
else
  pexec -f BASE.list -e base -o -u -c -- "$SELF $base"
fi

SELF=$0; base="$1"
if [ -n "$base" ] ; then
  KERNEL="i/4; b/4; d=3/4"
ficonv --reference ./photref.fits \
    --input $(REG)/$base-trans.fits --input-stamps ./photref.reg --kernel "$KERNEL" \
    --output-kernel-list $(AST)/$base.kernel --output-subtracted $(REG)/$base-sub.fits
else
  pexec -f BASE.list -e base -o -u -c -- "$SELF $base"
fi
\end{verbatim}

\textbf{Figure 5.} Two shell scripts demonstrating the invocation syntax of \text{fitrans} and \text{ficonv}. Since the computation of the transformed and convolved images requires a significant amount of CPU time, the utility \text{pexec} (http://www.gnu.org/software/pexec) is used to run the jobs in parallel on multiple CPUs.

\footnote{The program \text{fiinfo} is capable of producing such tables with three columns: a list of $x$ and $y$ coordinates followed by the respective pixel intensity.}
Figure 6. The schematics of the data transposition. Records for individual measurements are written initially to photometry files (having an extension of *.phot, for instance). These records contain the source identifiers. During the transposition, photometry files are converted to light curves. In principle, these light curves contain the same records but sorted into distinct files by the object names, not the frame identifiers. The command lines on the lower panel show some examples of how this data transposition can be employed involving the program grcollect.

Figure 7. Storage schemes for photometric data. Supposing a series of frames, on which nearly the same set of stars have individual photometric measurements, the figure shows how these data can be arranged for practical usage. The target stars (their identifiers) are arranged along the abscissa while the ordinate shows the frame identifiers to which individual measurements (symbolized by dots) belong. Raw and instrumental photometric data are therefore represented here as rows (see the marked horizontal stripe for frame #3, for instance) while the columns refer to light curves. In practice, native ways of transposition are extremely ineffective if the total amount of data does not fit into memory. The transposition can be speeded up by using an intermediate stage of data storage, so-called macroblocks. In the figure, each macroblock is marked by an enclosing rectangle. See text for further details.

3.15 Archiving: fizip and fiunzip

Due to the large amount of disk space required to store the raw, calibrated and derived (registered and/or subtracted) frames, it is essential to compress and archive image files that are barely used. The purpose of the fizip and fiunzip programs is to compress and decompress primary FITS data, by keeping the changes in the primary FITS header minimal. The compressed data are stored in a one-dimensional 8-bit (BITPIX=8, NAXIS=1) array, therefore these key words do not reflect the original image dimension or data type.

All of the other key words are untouched. Some auxiliary information on the compression is stored in the key words starting with 'FIZIP'; the contents of these key words depend on the compression method involved. fizip rejects compressing the FITS file where such key words exist in the primary header.

In practice, fizip and fiunzip refer to the same program (namely, fiunzip is a symbolic link to fizip) since the algorithms involved in the compression and decompression refer to the same code base or external library. fizip and fiunzip support well-known compression algorithms such as GNU zip (’gzip’) and the block-sorting file compressor (also known as ‘bzip2’) algorithm.

These compression algorithms are lossless. However, fizip supports rounding the input pixel values to the nearest integer or to the nearest fraction of some power of 2. Since the common representation of floating-point real numbers yields many zero bits if the number itself is an integer or a multiple of power of 2 (including fractional multiples), the compression is more effective if this kind of rounding is done first. This ‘fractional rounding’ yields data loss. However, if the difference between the original and the rounded values is comparable to or less than the readout noise of the detector,
such compression does not affect the quality of further processing (e.g. photometry).

3.16 Generic arithmetic evaluation, regression and data analysis: `lfit`

Modelling of data is a prominent step in the analysis and interpretation of astronomical observations. In this section, a stand-alone command-line-driven tool, named `lfit`, is introduced, designed for both interactive and batch-processed regression analysis as well as generic arithmetic evaluation (see e.g. Fig. 8).

Similarly to the task `fiarith`, this tool is built on the top of the `libpsn` library (see Section 2.3). This library provides both the back end for function evaluation and analytical calculations of partial derivatives. Partial derivatives are required by most of the regression methods (e.g. linear and non-linear least-squares fitting) and uncertainty estimations (e.g. Fisher analysis). The program features many built-in functions related to special astrophysical problems. Moreover, it allows the end user to extend the capabilities during run time using dynamically loaded libraries. Sophisticated functions can be implemented and passed to `lfit` in this way (see e.g. Pál 2010, for a practical case).

The built-in regression methods (Table 3), the built-in functions related to astronomical data analysis (see also Table 4) and some of the more sophisticated new tools for non-linear regression analyses (e.g. extended Markov chain Monte Carlo (XMMC)) are discussed in Pál (2009) in more detail.

Although the task `lfit` does not perform direct operations on images, this can also be an essential part of a photometric pipeline.

See e.g. Fig. 9 for a complete shell script that involves many of the `FITSH` tasks for data processing.

4 SUMMARY

In this paper we described a software package named `FITSH` intended to provide a complete solution for many problems related to astronomical image processing, including calibration, source extraction, astrometry, source identification, photometry, light-curve processing and regression analysis. The implementation scheme enables not only simple and portable processing but easy cooperation with other existing related data-processing software packages. This package has an open-source code base, so any details related to the actual execution of the various tasks and algorithms can be traced. Although the current implementation allows the user fast accomplishment of the previously listed exercises, there are some features that need to be improved, or else some implementational aspects should be reconsidered. These include the clean-up of the code related to PSF analysis and some user-interface functionality and homogeneity (a more similar syntax for some related tasks, more sophisticated output formatting as provided by the users and so on). In order to be more compatible with existing software solutions, some improvements are considered related to the mask handling and built-in compression algorithms. In addition, implementation of features currently in the testing stage that aid the processing of data not related directly to ‘CCD imaging’ is also considered. These include processing of grism observations or applications for post-processing images acquired in (near- or far-) infrared spectral regimes (e.g. Herschel Space Observatory). The web page of the
1836  A. Pál

#!/bin/sh
CATALOG=input.cat # name of the reference catalog
COLID=1 # column index of object identifier (in the $CATALOG file)
COLX=2 # column index of the projected X coordinate (in the $CATALOG file)
COLY=3 # column index of the projected Y coordinate (in the $CATALOG file)
COLMAG=4 # column index of object magnitude (in the $CATALOG file)
COLCOLOR=5 # column index of object color (in the $CATALOG file)
THRESHOLD=4000 # threshold for star detection
GAIN=4.2 # combined gain of the readout electronics and the A/D converter in electrons/ADU
MAGFLUX=10,10000 # magnitude/flux conversion
APERTURE=6:8:8 # aperture radius, inner radius and thickness of the background annulus (all in pixels)

mag.param=c0.00,c1.10,c0.01,c0.20,c0.11,c0.02,c1.00,c1.01,c1.10
mag.funct="c0.00+c1.10*x+c0.01*y+0.5*(c0.20*x^2+2*c0.11*x*y+c0.02*y^2)+color*(c1.00+c1.10*x+c1.01*y)"

for base in $[@] ; do
  fistar $[CATALOG]/$base.fits --algorithm uplink --prominence 0.0 --model elliptic
    --flux-threshold $THRESHOLD --format id,x,y,s,d,k,amp,flux --o $[AST]/$base.stars
  gmatch --reference $CATALOG --col-ref $COLX,$COLY --col-ref-ordering --$COLMAG
    --input $[AST]/$base.stars --col-imp 2,3 --col-imp-ordering 8
    --weight-reference-column=$COLMAG,magnitude,power=2
    --triangulation-maxim=100,maxref=100,conformable,auto,unitarity=0.002
    --order 2 --max-distance 1
    --comment --output-transformation $[AST]/$base.trans || continue
  grtrans $CATALOG --col-xy $COLX,$COLY --col-out $COLX,$COLY
    --input-transformation $[AST]/$base.trans --output - |
  fphot $[CATALOG]/$base.fits --input-list --col-xy $COLX,$COLY --col-xy-list --$COLID
    --gain $GAIN --mag-flux $MAGFLUX --aperture $APERTURE --disjoint-annuli
    --sky-fit-mode,iterations=4,sigma=3 --format IXY,MmBbS
    --comment --output $[PHOT]/$base.phot
  paste $[PHOT]/$base.phot $[PHOT]/$REF.phot $CATALOG |
  1fit --columns mag=4,err=5,mag0=12,x=10,y=11,color=$(2*8+COLOR) |
    --variables $mag.param --function "$mag.funct" --dependent mag0-mag --error err |
    --output-variables $[PHOT]/$base.coeff
  paste $[PHOT]/$base.coeff $[PHOT]/$REF.coeff |
  1fit --columns mag=4,err=5,mag0=12,x=10,y=11,color=$(2*8+COLOR) |
    --variables $(cat $[PHOT]/$base.coeff) |
    --function "$mag.funct" --format %9.5f --column-output 4 |
  awk '{ print $1,$2,$3,$4,$5,$6,$7,$8; }' > $[PHOT]/$base.tphot
done
for base in $[@] ; do test -f $[PHOT]/$base.tphot & & cat $[PHOT]/$base.tphot ; done |
grcollect --col-base 1 --prefix $LC/ --extension .lc

Figure 9. A shell script demonstrating a complete working pipeline for time-series aperture photometry. The input FITS files are read from the directory $[CATALOG] and their base names (without the *.fits extension) are expected to be in the array $[LIST[*]]. These base names are then used to name the files storing data obtained during the reduction process. Files created by subsequent calls of the fistar and gmatch programs are related to the derivation of the astrometric solution and the respective files are stored in directory $[AST]. The photometry centroids are derived from the original input catalogue (found in the file $CATALOG) and the astrometric transformation (plate solution, stored in the *.trans files). The results of the photometry are put into the directory $[PHOT]. Raw photometry is followed by a magnitude transformation. This branch involves additional common UNIX utilities such as paste and awk in order to match the current and reference photometry as well as to filter and re-sort the output after magnitude transformation. The derivation of the transformation coefficients is done through the 1fit utility, which involves $mag.funct with the parameters listed in $mag.param. This example features a quadratic magnitude transformation and a linear colour-dependent correction (to cancel the effects of differential refraction). The final light curves are created by the grcollect utility, which writes the individual files into the directory $[LC]. More detailed examples are available on the web page of the project, located at http://fitsh.szofi.net. These examples include further possible applications and sample data as well (these have been or are going to be published in separate papers).

ACKNOWLEDGMENTS

The current work of AP and the recent development of this software package have been supported by the ESA grant PECS 98073 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. The author thanks Gáspár Bakos for various improvement ideas and the support from the HATNet project, including the NASA grants NNG04GN74G, NNX08AF23G and SAO IR&D grants. The author also acknowledges the useful notes of the referee, Jim Lewis. His comments helped to make the package more permeable with other existing packages. The author also thanks Brigitta Sipócz for her suggestions and bug reports. The recent ideas and assistance of György Mező and Krisztián Vida are also acknowledged. The author thanks Péter Somogyi for his help during the development of the project’s web page.

© 2012 The Author, MNRAS 421, 1825–1837
Monthly Notices of the Royal Astronomical Society © 2012 RAS

Downloaded from https://academic.oup.com/mnras/article-abstract/421/3/1825/1074470
on 27 July 2018
Fitting

The \texttt{lfit} package supports a grid (a.k.a. ‘brute-force’ minimization) in radians and the Lagrangian orbital elements and the celestial coordinates. Algorithms supported by \texttt{lfit} and their respective requirements for the model function. The first column refers to the internal and command-line identifier of the algorithms. The second column shows whether the method requires the parametric derivatives of the model function in an analytic form or not. The third column indicates whether, in the cases in which the method requires parametric derivatives, the model function should be linear in all parameters.

| Code   | derivatives | linearity | Method or algorithm |
|--------|-------------|-----------|---------------------|
| L/CLLS | yes         | yes       | Classic linear least-squares method |
| N/NULLM | yes         | no        | (Non-linear) Levenberg–Marquardt algorithm |
| U/LHID | no          | no        | Levenberg–Marquardt algorithm employing numeric parametric derivatives |
| H/MCHC | no          | no        | Classic Markov Chain Monte Carlo algorithm¹ |
| R/KNMC | yes         | no        | Extended Markov Chain Monte Carlo² |
| D/DHSS | no          | no        | Mapping the values of \(\chi^2\) on a grid (a.k.a. ‘brute-force’ minimization) |
| E/EMCE | optional³   | optional³ | Downhill simplex |
| A/FIMA | no          | yes       | Fisher Information Matrix Analysis |

¹ The implemented transition function is based on the Metropolitan–Hastings algorithm and the optional Gibbs sampler. The transition amplitudes must be specified initially. Iterative MCMC can be implemented by subsequent calls to \texttt{lfit}, involving the previous inverse statistical variances for each parameter as the transition amplitudes for the next chain.
² The program also reports the summary related to the sanity checks (such as correlation lengths, Fisher covariance, statistical covariance, transition probabilities and the best-fitting value obtained by an alternate minimization, usually the downhill simplex).
³ The downhill simplex algorithm may use the parametric derivatives to estimate the Fisher/covariance matrix for the initial conditions in order to define the control points of the initial simplex. Otherwise, if the parametric derivatives do not exist, the user should specify the ‘size’ of the initial simplex somehow during the invocation of \texttt{lfit}.
⁴ Some of the other methods (esp. CLLS, NULLM, DHSS in practice) can be used during the minimization process of the original data and the individual synthetic data sets.

Table 4. Basic functions found in the built-in astronomical extension library. These functions cover the fields of simple radial velocity analysis, some aspects of light-curve modelling and data reduction. These functions are a kind of ‘common denominator’, i.e. they do not provide the direct possibility for application but complex functions can be built on top of them for any particular usage. All of the functions below, with the exception of \texttt{hjd()} and \texttt{bjd()}, have partial derivatives that can be evaluated analytically by \texttt{lfit}.

| Function | Description |
|----------|-------------|
| \(\texttt{hjd(JD, } \alpha, \delta)\) | Function that calculates the heliocentric Julian date from the Julian day \(J\) and the celestial coordinates \(\alpha\) (right ascension) and \(\delta\) (declination). |
| \(\texttt{bjd(JD, } \alpha, \delta)\) | Function that calculates the barycentric Julian date from the Julian day \(J\) and the celestial coordinates \(\alpha\) (right ascension) and \(\delta\) (declination). |
| \(\texttt{ellipticK}(k)\) | Complete elliptic integral of the first kind. |
| \(\texttt{ellipticE}(k)\) | Complete elliptic integral of the second kind. |
| \(\texttt{ellipticPi}(k, n)\) | Complete elliptic integral of the third kind. |
| \(\texttt{eoq(}\lambda, k, h)\) | Eccentric offset function, ‘\(q\)’ component. The arguments are the mean longitude \(\lambda\) in radians and the Lagrangian orbital elements \(k = \cos \varphi, h = \sin \varphi\). |
| \(\texttt{eop(}\lambda, k, h)\) | Eccentric offset function, ‘\(p\)’ component. |
| \(\texttt{ntiu(p, z)}\) | Normalized occultation flux decrease. This function calculates the flux decrease during the eclipse of two spheres when one of the spheres has uniform flux distribution and the other one, by which the former is eclipsed, is totally dark. The bright source is assumed to have a radius of unity while the occulting disc has a radius of \(p\). The distance between the centres of the two discs is \(z\). |
| \(\texttt{ntiq(p, z, } y_1, y_2)\) | Normalized occultation flux decrease when the eclipsed sphere has a non-uniform flux distribution modelled by a quadratic limb-darkening law. The limb darkening is characterized by \(y_1\) and \(y_2\). |

REFERENCES

Alard C., 2000, A\&AS, 144, 363
Alard C., Lupton R. H., 1998, ApJ, 503, 325
Bertin E., Arnouts S., 1996, A\&AS, 117, 393
Chromey F. R., 2010, To Measure the Sky: An Introduction to Observational Astronomy. Cambridge University Press, Cambridge
Joye W. A., Mandel E., 2003, in Payne H. E., Jedrzejewski R. I., Hook R. N., eds, ASP Conf. Ser. Vol. 295, Astronomical Data Analysis Software and Systems XII. Astron. Soc. Pac., San Francisco, p. 489
Pál A., 2009, PhD thesis, Loránd Eötvös University (arXiv:0906.3486)

Pál A., 2010, MNRAS, 409, 975
Pál A., Bakos G. Á., 2006, PASP, 118, 1474
Stetson P. B., 1987, PASP, 99, 191
Stetson P. B., 1989, in Barbury B., Janot-Pacheco E., Magalhães A. M., Viegas S. M., eds, Advanced School of Astrophysics, Image and Data Processing/Interstellar Dust. Instituto Astronômico e Geofísico, São Paulo

This paper has been typeset from a \texttt{LaTeX} file prepared by the author.