NOAO SDM Document PL017

A Guide to NEWFIRM Data Reduction with IRAF

Mark Dickinson and Francisco Valdes

National Optical Astronomy Observatory
Science Data Management

Draft version: 1 June 2009

NOAO Science Data Management Group, P.O. Box 26732, Tucson, AZ 85732

Table of Contents

Abstract

This document presents a user's guide to reducing data from the NOAO Extremely Wide-Field InfraRed Mosaic (NEWFIRM) camera within the IRAF data processing environment. The IRAF `nfextern` package provides a set of tasks specially tailored for this purpose. This guide follows step-by-step examples of NEWFIRM data reduction using `nfextern` and other IRAF tasks.

Keywords: IRAF, NEWFIRM, infrared, calibration, data processing

1 Introduction

1.1 NEWFIRM and data processing

The NOAO Extremely Wide-Field InfraRed Mosaic (NEWFIRM; Autry et al., 2003; Probst et al., 2004, 2008) is a near-infrared imaging camera that operates at the Cassegrain focus of the NOAO 4m telescopes. It is currently at Kitt Peak, and there are plans to move it to Cerro Tololo in the future. The NEWFIRM focal plane consists of a 2×2 mosaic of 2048×2048 Orion InSb array detectors, each of which is read out through 64 amplifiers. The detector scale at the 4m is approximately 0.4 arcsec per pixel, and the gaps between the detectors are approximately 35 arcsec across. The net field of view covers nearly $28' \times 28'$. The camera has two filter wheels which can stock a variety of standard and user-provided filters, including the regular J, H and K_s bands and various narrow band filters.

With its large focal plane arrays, and the fast observing cadence used for infrared broad band observing, NEWFIRM typically produces a large volume in a night. Additionally, infrared data reduction often requires many processing steps, sometimes involving multiple passes through the data in order to remove instrumental signatures to the very high level of accuracy required to detect faint sources against the bright sky background.

NOAO has developed automated pipelines for processing data taken with NEWFIRM (Swaters et al., 2009). The Quick-Reduce Pipeline (QRP) operates at the telescope, and is designed to carry out basic reductions of data soon after they are taken, in order to provide useful feedback to observers during the course of the night. The QRP takes several shortcuts when processing the data, and is not designed to produce final, science-quality data products. The NEWFIRM Science Pipeline (NSP) operates in Tucson, and processes data after the end of each observing run. The NSP carries out more elaborate and computationally intensive processing, and produces data products intended to have scientific and archival value.[1]

Although the NEWFIRM pipeline should provide data products that are suitable for many observers' needs, there will still be occasions when astronomers may wish to reduce NEWFIRM data "by hand." The pipelines operate automatically, normally without human intervention or decision-making. They should work well for data taken in standard observing modes, but may not always perform ideally for unusual observations that require non-standard processing. For example, the control of the pipeline is governed largely by the pre-defined observing sequences that are executed by the NEWFIRM Observing Control System (NOCS; (Daly et al., 2008)). Observations of crowded fields or large targets that fill much of the NEWFIRM field of view may require exposures of "blank" sky in offset fields, and the pipeline depends on observers using the correct NOCS sequences which in turn provide the pipeline with header metadata that indicate which frames include the target and which are to be used as "sky". However, some observers might take additional, separate offset sky sequences, and the pipeline has no way to know when and how to use these. As another example, the NEWFIRM pipeline is not currently fine-tuned for observations taken

[1] At the time of this writing (May 2009), the NEWFIRM Science Pipeline is undergoing science verification testing. Products are being delivered to observers on a trial basis, and are not yet being archived. Regular operations, including archive storage, should begin later in 2009.

through very narrow band filters, which pose special challenges that may require custom processing methods. Even for broad band data, an astronomer may wish to experiment with different sky subtraction strategies, or use different flat fields or other calibration data. If sky conditions were unstable during the course of an observation, one may wish to make decisions about which data to exclude and retain for the final combined images, or to experiment with various weighting schemes,[2] or with different methods rejecting artifacts. An astronomer might also want to mosaic images differently, perhaps sampling them to a different scale, adopting a different tangent point for re-projection, combining multiple observing sequences into a single image, or combining different pointings into wider-field mosaics.

The `nfextern` IRAF package provides a set of software tools designed to to facilitate "manual" reduction of infrared imaging data, and in particular NEWFIRM data via the `newfirm` sub-package. The `nfextern` package has been developed along side the NEWFIRM Science Pipeline, by the same team of people. The tasks from the `newfirm` package do not always identically replicate the functions in the Pipeline, but they follow the same general approach. This package also shares a lot of philosophical ground with other infrared data reduction packages that came before it, such as DIMSUM.[3]

Many of the `nfextern` tasks are specialized versions of more general, underlying IRAF tools for infrared data reduction and mosaic imager reduction in general. Future IRAF releases will include these more general packages and tasks, which may be adapted for use with other instruments. However, at present, we limit the scope of this document to the `nfextern` and `newfirm` packages.

The present document is a user-oriented guide to NEWFIRM data reduction with `nfextern` procedures, rather than formal documentation. It illustrates the use of the tasks for specific data reduction problems, but does not provide exhaustive descriptions of every task parameter. You should consult the help pages for each task to see a detailed description of its parameters and procedures.

This "cookbook" is a work-in-progress, and the current version of the `nfextern` package is a beta-release. This first edition of the *Data Reduction Guide* covers procedures suitable for a limited range of types of NEWFIRM observations. Others will be added in the future, ideally based on feedback from astronomers who use the `nfextern` tools to reduce their own data. Our knowledge of NEWFIRM and how to reduce its data continues to evolve, and correspondingly so do the tasks in the `nfextern` package. At present, the `nfextern` tasks have only received limited testing by a few individuals on actual NEWFIRM data, by no means spanning the full range of observations that real users have made. You can help by providing feedback, ranging from bug reports to detailed descriptions of how you have used the tasks to reduce your data. Please send your feedback by e-mail to:

[2]The NEWFIRM science pipeline weights images based on sky noise, photometric transparency and seeing before combining images into a stack, and may exclude data whose data quality parameters fall below certain thresholds, but these decisions are automated and may not be suitable for all purposes.

[3]DIMSUM is an IRAF contributed package originally written by P. Eisenhardt, M. Dickinson, S. A. Stanford, and J. Ward, with contributions from many others along the way. It is available from `http://iraf.noao.edu/contrib.html`.

`vohelp@noao.edu`

1.2 Useful resources concerning NEWFIRM and IRAF

The NEWFIRM User Information Page should be your first stop for basic information about NEWFIRM and related systems. You can find this at:

`http://www.noao.edu/ets/newfirm/`

In particular, you can find information about the NEWFIRM instrument there, as well as the NEWFIRM Observing Control System (NOCS), which is used to construct and execute sequences of observations at the telescope. Information about the NEWFIRM Pipelines can also be found by following links from the User Information site.

The NOAO Data Handbook (Shaw, ed., 2009) is another valuable resource. It includes a chapter about NEWFIRM[4] which gives basic information about the instrument, illustrates many of the characteristics of NEWFIRM data, and discusses the raw and pipeline-processed data products, including file formats, header keywords, etc. The handbook may be found at:

`http://www.archive.noao.edu/help.shtml`

This guide assumes some basic familiarity with IRAF. A basic introduction to IRAF is *A Beginner's Guide to Using IRAF* (Barnes, 1993). This guide does not describe scripting, but anything described here as typed from the command language (cl) can be included in scripts either as explicit commands (no variables or special language constructs) or in a programming context with variables, loops, etc. *An Introductory User's Guide to IRAF Scripts* (Anderson & Seaman, 1989) provides a good starting point for learning IRAF scripting. NEWFIRM and its data reduction software make extensive use of Multi-extension format (MEF) FITS files (see §2.1 below). Useful information about working with MEF files in IRAF can be found in the *IRAF FITS Kernel User's Guide* (Zarate, 1997).

2 NEWFIRM data and file formats

NEWFIRM is a "mosaic camera", in the sense that it has several focal plane array detectors that are physically and electronically disjoint, and which image separate (but nearby) fields of view on the sky. Here we provide some useful information about the NEWFIRM data files and their format.

[4]At this time (May 2009), the NEWFIRM chapter of the *NOAO Data Handbook* is being completed, and will be be released soon.

2.1 Multi-Extension Format (MEF) FITS files

One NEWFIRM exposure produces four images, one per detector, which are then packaged together into a single multi-extension FITS file that is delivered to the observer and to the NOAO Science Archive. The four extensions in NEWFIRM MEF files are designated by extension names (header keyword `EXTNAME`) `[im1]`, `[im2]`, `[im3]`, and `[im4]`.[5] There are four separate extension headers, as well as a master header for the whole MEF image, accessed in IRAF by reference to extension `[0]`.

On occasion, it may be necessary to use IRAF tasks which are not designed to work automatically with MEF files. There are several ways to do this. The `msctools` package (§3.2) includes the tasks `mscsplit` and `mscjoin` which can be used for this purpose:

```
msctools> mscsplit a.fits verbose+
a.fits[0] -> a_0
a.fits[im1] -> a_1
a.fits[im2] -> a_2
a.fits[im3] -> a_3
a.fits[im4] -> a_4

msctools> mscjoin a output=b verbose+
a_0[0] -> b
a_1 -> b[im1,1,append,inherit]
a_2 -> b[im2,2,append,inherit]
a_3 -> b[im3,3,append,inherit]
a_4 -> b[im4,4,append,inherit]
```

In general, the `append` syntax may be used (but is not always required) to add a new extension onto an existing MEF image:

```
newfirm> imhead x?.fits
x1.fits[2046,2046][real]:
x2.fits[2046,2046][real]:
x3.fits[2046,2046][real]:
x4.fits[2046,2046][real]:

ecl> imcopy x1.fits y.fits[im1,append]
x1.fits -> y.fits[im1]
ecl> imcopy x2.fits y.fits[im2,append]
x2.fits -> y.fits[im2]
ecl> imcopy x3.fits y.fits[im3,append]
x3.fits -> y.fits[im3]
ecl> imcopy x4.fits y.fits[im4,append]
x4.fits -> y.fits[im4]
```

[5]It is often possible to refer to these simply by the extension numbers, i.e., as `[1]`, `[2]`, `[3]`, and `[4]`. However, strictly speaking, the extensions need not be stored in order within the MEF file, and it is better in general to refer to them by the `extname` rather than by extension number.

2.2 Mask images

One relatively new IRAF feature employed in some `nfextern` tasks is the use of FITS to encapsulate compressed "pixel list" masks. IRAF has long used the the pixel-list (or pl) data format to store integer-value images. This is quite useful for mask images with relatively few data values and a lack of complex pixel-to-pixel structure, such as bad pixel masks. However, the pl format is not a FITS file and as such is not as transportable as other formats; many other non-IRAF software tasks cannot make use of it. Moreover, the pl format cannot have multiple extensions. IRAF now supports FITS encapsulation of pl format mask images. This feature *requires* that the mask be stored in an MEF file, even if it has only one extension. For example:

```
ecl> imcopy bpm.pl bpm_fullsize.fits ver+
bpm.pl -> bpm_fullsize.fits

ecl> imcopy bpm.pl bpm.fits[extname\=pl,type\=mask] ver+
bpm.pl -> bpm.fits[extname=pl,type=mask]

newfirm> dir bpm* long+
-b-rwr-r- med       201600 Jun  1 15:39 bpm.fits
-b-rwr-r- med       175984 Jun  1 15:35 bpm.pl
-b-rwr-r- med     16764480 Jun  1 15:40 bpm_fullsize.fits
```

The `imcopy` statement used to create the mask has two notable features. First, the `extname` must be specified explicitly, to ensure that an MEF file is created.[6] Here, we call the extension `pl`, but any name can be used. The `imcopy` statement could be shortened to:

```
ecl> imcopy bpm.pl bpm.fits[pl,type\=mask]
```

The specification `type\=mask` indicates that the image will be a FITS-encapsulated mask file. This results in the considerable size savings relative to the full-sized FITS version of the image.

2.3 NEWFIRM image filenames

If you retrieve your raw NEWFIRM data from the NOAO Science Archive, the files that are delivered to you will have different filenames than those assigned at the telescope. The NEWFIRM Observing Control System (NOCS) assigns FITS filenames for each observation that include a user-specified prefix and a 5-digit frame index number, e.g., `n03.28700.fits`. However, these filenames are not guaranteed to be unique (e.g., the index counter may be reset occasionally), and are thus not suitable for archival use. The NOAO Science Archive assigns a new, unique name to every file that it ingests, and when you retrieve data it may have a name like `NSAR3_kp698551.fits`. The `NSAR3` prefix indicates that the data were retrieved from the NOAO Science Archive (version 3), while `kp698551` is the unique identifier assigned by the iSTB archive ingestion system. There are various FITS header keywords that record both the old and new filename information:

[6]The backslashs in the strings `extname\=pl` and `type\=mask` are necessary as escape characters before the equals signs.

```
FILENAME= 'n03.28070.fits'     / Original host filename
DTACQNAM= '/home/data/n03.28070.fits'  /  file name supplied at telescope
SB_ID   = 'kp698551           '  /  unique iSTB identifier
SB_NAME = 'kp698551.fits      '  /  name assigned by iSTB
DTNSANAM= 'kp698551.fits      '  /  file name in NOAO Science Archive
```

Nothing about the `nfextern` tasks depends on the nature of the filenames, so if you retrieved data from the archive, you may leave the names as they were. However, astronomers reducing their own data might want to refer to the original filenames, e.g., for comparison to their observing logs taken at the telescope. Also, files ordered alphabetically by their archive names are *not* guaranteed to be sorted in the chronological order in which the observations were taken. NOAO provides an IRAF script, as well as a stand-alone PERL script (using WCSTOOLS), that can be used to rename files to the original names that were assigned at the telescope. These are available at:

`http://nvo.noao.edu/noaonvo/contrib.shtml`

3 The `nfextern` package and its sub-packages

You should download the latest version of the `nfextern` package from the IRAF external packages repository at:

`http://iraf.noao.edu/extern.html`

Follow the installation instructions provided with the package. You should be running IRAF version 2.14, ideally with the version 2.14.1 patch that was released on September 16, 2008.

When you load the `nfextern` package, you will find three sub-packages:

```
ecl> nfextern
      ace.        msctools.   newfirm.
```

We briefly discuss each of these in turn.

3.1 The Astronomical Cataloging Environment (`ace`)

The Astronomical Cataloging Environment, or `ace`, is a package that contains tools for generating and manipulating source catalogs within IRAF. We will not discuss it in any detail here, although several tasks in the `newfirm` package call on tasks from `ace`. General documentation for `ace` can be found in the users guide *ACE: The IRAF Astronomical Cataloging Environment* (Valdes, 2008).

3.2 `msctools`

Most of the data reduction operations required to reduce NEWFIRM data must be applied to all four arrays. It is therefore convenient to use IRAF tasks designed to work with the MEF data format. The `msctools` package provides tools for conveniently handling MEF format data independent of the the type of arrays. This package was derived from `mscred`, the first IRAF package for reducing data from the NOAO MOSAIC wide-field CCD imagers. If you have used `mscred`, much of the `msctools` package will be familiar or even identical. The concept for `msctools` was to separate the instrument–specific parts of `mscred`, those dealing with CCD data reductions and the MOSAIC instruments in particular, from the generic image format parts. Then the instrument and data reduction tasks are then provided in separate packages, such as the `newfirm` package discussed below. The tasks in `msctools` are:

```
nfextern> msctools
      aimexpr            mscedit            mscjoin            mscstack
      mimpars@           mscexamine         mscmedian          mscstat
      mkmsc              mscextensions      mscpixarea         msctoshort
      mscagetcat         mscfinder          mscpixscale        msctvmark
      mscarith           mscfindgain        mscrfits           mscwcs
      mscblkavg          mscfocus           mscselect          mscwfits
      msccmatch          mscgetcatalog      mscsetwcs          msczero
      msccmd             mscheader          mscshutcor         mskmerge
      mscctran           mscimage           mscskysub
      mscdisplay         mscimatch          mscsplit
```

Note: it is advisable *not* to load both the `msctools` and `mscred` packages at the same time! Many tasks appear in both packages, sometimes with the same names but different parameters or detailed functionality. Loading both packages can lead to collisions or confusion.

3.3 The `newfirm` package

The `newfirm` package contains routines that have been specially tailored for reducing NEWFIRM data. Some of the tasks are scripts which in turn call more general image processing routines. The `newfirm` tasks are:

```
nfextern> newfirm
      cgroup        nfdeltasky    nfgroup       nfoproc       nftwomass
      combine       nfdproc       nflinearize   nfproc        nfwcs
      dcombine      nffocus       nflist        nfsetsky
      fcombine      nffproc       nfmask        nfskysub
```

A ‘help’ listing for the package gives these brief summaries:

```
newfirm> help
          cgroup - Group exposures
         combine - Combine exposures
        dcombine - Combine dark exposures
        fcombine - Combine flat exposures
         nfdproc - Process NEWFIRM dark exposures
      nfdeltasky - Fit and subtract residual sky background from NEWFIRM images
         nffocus - Determine best focus from NEWFIRM exposures
         nffproc - Process NEWFIRM dome flat field exposures
         nfgroup - Group NEWFIRM data in list files
     nflinearize - Linearize NEWFIRM exposures
          nflist - List NEWFIRM image parameters as derived by nfproc
          nfmask - Create data quality masks for NEWFIRM exposures
         nfoproc - Apply instrumental calibrations to NEWFIRM object exposures
          nfproc - General task for processing NEWFIRM data
        nfsetsky - Pair exposures for pairwise sky subtraction
        nfskysub - Subtract sky from NEWFIRM images
       nftwomass - Get 2MASS catalog data
           nfwcs - Derive WCS for NEWFIRM exposures
```

One of the most important tasks is `nfproc`. This is a generalized tool for carrying out many of the steps in NEWFIRM image processing, including image trimming, bad pixel interpolation, dark subtraction, linearization, generating saturation and persistence masks, flat field correction, and sky subtraction. It is analogous to `ccdproc` in the `mscred` package of MOSAIC reduction tools. `nfproc` is quite complicated, with more than 60 parameters. It can make use of expressions which can define fairly complex operations, and which are stored in database files. A general discussion of these expressions can be found by reading the help file for `procexpr`.

There are several tasks that are scripts which simplify the use of `nfproc` and other tools, setting parameters in ways that are appropriate for particular processing steps. These include `nfdproc`, `nffproc`, `nfoproc`, `nflinearize`, `nfmask`, `nfskysub` and `nfwcs`. In the discussion that follows, we will mainly use these scripts to break down the processing into several steps, and to clarify the definition of task parameters. However, once you have mastered the use of these tasks, you may wish to return to the more general `nfproc` routine, which gives greater processing flexibility and may be used to combine operations for greater execution efficiency. Future editions of this *Data Reduction Guide* may include more detailed and advanced examples using `nfproc`.

The `nfextern` package source distribution includes a directory `nfextern$newfirm/nfdat` (aliased to `nfdat$` when you load the `newfirm` package). This contains a variety of "data" files that are used by the various tasks, including the expression databases called by `nfproc` and other tasks, some parameter definition files for the `ace` source cataloging package, and some actual NEWFIRM calibration reference image files, such as a bad pixel map and linearity coefficient maps.

4 Basic instrumental characteristics of NEWFIRM

The NEWFIRM chapter of the *NOAO Data Handbook* gives a reasonably detailed description of NEWFIRM and the basic characteristics of its data. Here, we summarize certain features that are important to understand in terms of the basic data reduction.

4.1 Array layout and FITS files

Figure 1 illustrates the layout of the four NEWFIRM arrays, their orientation with respect to the sky, and their storage within multi-extension FITS files, as implemented for observing semesters

2008A and afterward.[7] In this *Data Reduction Guide*, the four arrays are indicated by their MEF image extension names (header keyword `EXTNAME`): `[im1]`, `[im2]`, `[im3]`, and `[im4]`. The raw FITS images have 2112 columns and 2048 rows in each extension. The last 64 columns (2049 to 2112) are "reference pixels" and are not illuminated (see §4.2).

4.2 Bias, dark, and reference pixels

Over most of the NEWFIRM detector arrays, the actual dark current is quite low, although there are individual pixels with higher dark count rates, and some regions of the detector where structural damage results in extra signal. Most of the structure in NEWFIRM "dark" images is actually electronic offset (or bias) which can depend in a complex way on the readout parameters such as integration time, digital averages, coadds, and Fowler sampling. The arrays are operated in correlated double sampling (CDS) mode. In CDS, the array is reset, then after a short but finite time interval, each pixel is read once, non-destructively. The amplitude of the electronic offset level can be different for each of these readouts, and may indeed be lower in the final read than it was in the initial read, leading to negative values in the CDS difference image. Therefore, you should not be alarmed if you find "darks" with negative values.

Because of the strong dependence of the electronic levels on the instrument readout timing, it is advisable to subtract dark calibration images taken with the same readout parameters (integration time, digital averages, coadds, and Fowler sampling) as those used for the science data. For some applications, dark subtraction may not be so important. E.g., when reducing IR imaging data, one frequently takes differences between science images and offset sky frames (or sky images constructed from dithered science data). In this case, the bias+dark signal is removed to first order, although some residuals may remain if the sky background intensity is varying strongly with time, requiring rescaling between the "object" and "sky" images.

The NEWFIRM arrays include several different sorts of "reference pixels" which track the electronic stability of the detectors. These are discussed in some more detail in the NEWFIRM chapter of the *NOAO Data Handbook*. In principle, these may be used to determine a reference zero level for each exposure, which can be useful if this level is unstable or temperature-dependent, as is often the case for infrared arrays. As of this writing, the use of the reference pixels is still being investigated, and we do not concern ourselves with them here, except for their location in the raw data. The first and last columns of the array (in the original native detector readout frame, which is different than the orientation in the final MEF fits file – see figure 1) are meant to represent un-illuminated and saturated pixels, and should be trimmed away from science data. There is an additional reference pixel for each of the 64 readout amplifiers that is read once per row during readout. In the MEF FITS files that are written for the raw data, the values from these reference pixels are stored in columns 2049 to 2112 (spanning all 2048 rows of the detector). It is common

[7]The arrangements and orientations of the arrays in the MEF files were different for observations taken during the commissioning and shared-risk period in semesters 2007A and 2007B.

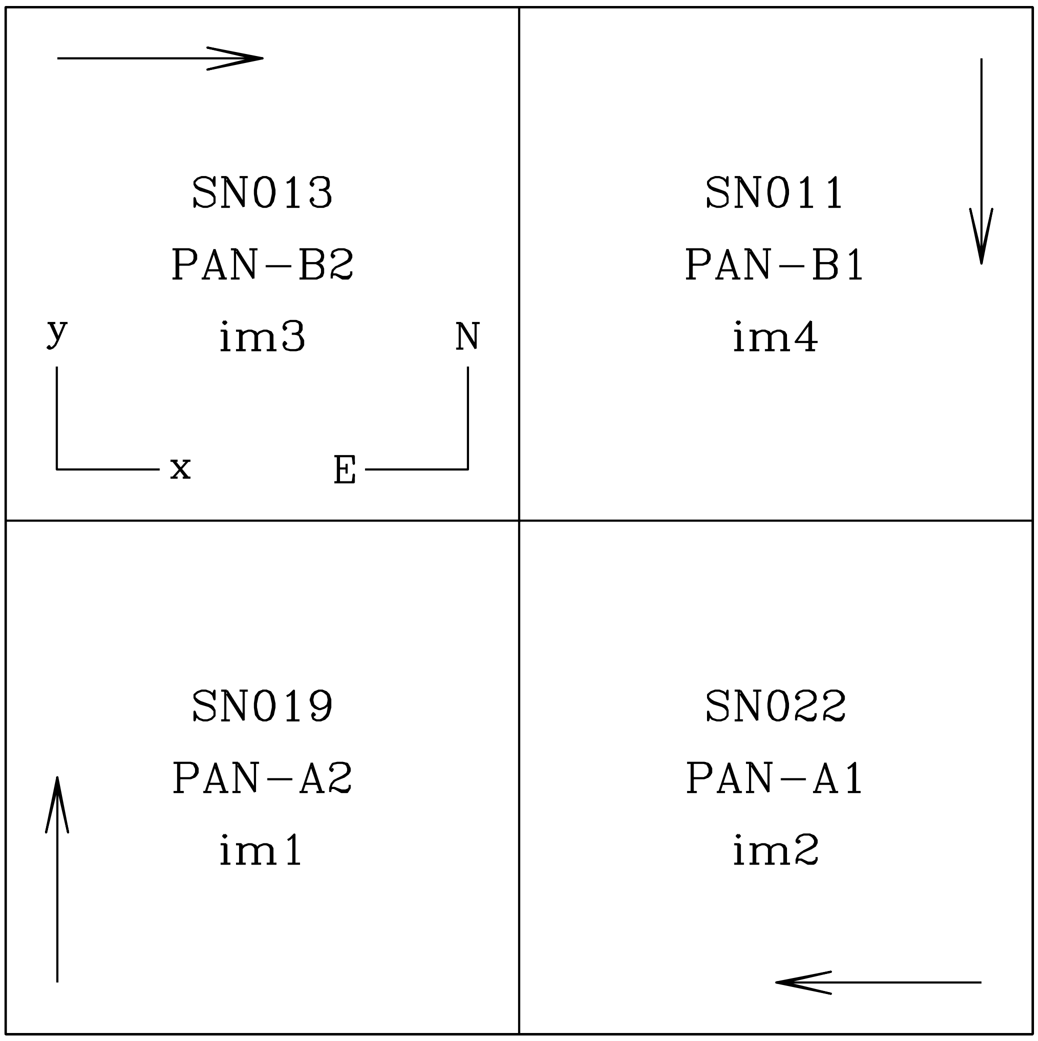

Figure 1: NEWFIRM array orientations and FITS file layout for observations from semester 2008A and after. Labels in each quadrant indicate the detector identifier (e.g., SN019), the pixel acquisition node (PAN, e.g., PAN-A2) used to read out that array, and the FITS image extension (e.g., im2) in which data from that array is recorded. Arrows in each quadrant indicate the readout direction within each amplifier. In the upper left hand quadrant, a compass rose indicates the orientation of the celestial equatorial coordinate system, while another shows the orientation of the $+x$, $+y$ pixel axes. These are the same for all four quadrants.

practice to trim the images to eliminate the reference pixels, using the image subsection `TRIMSEC = '[2:2047,2:2047]'` which is specified in the raw image headers.

4.3 Non-linearity and saturation

The NEWFIRM arrays, like most infrared detectors, are nonlinear. For an incident signal with fixed intensity, the counts recorded by the detector increase at a rate that is slower than linear. As more photoelectrons accumulate in a pixel, the rate at which additional electrons accumulate decreases. At count levels of 10000 ADU (or 80000 electrons, given the NEWFIRM gain ~ 8 electrons/ADU), the arrays are typically ~ 6 to 8% nonlinear, and they saturate at a level somewhat higher than this.

It is important to calibrate the nonlinear behavior of infrared arrays, and to remove this when reducing NEWFIRM data, for several reasons:

- to ensure accurate relative photometry between bright sources (e.g., standard stars) and faint sources (e.g., distant galaxies or faint stars of scientific interest),
- to ensure that flat field exposures (dome or sky flats) accurately map the relative pixel responsivity over the array,
- to ensure that the recorded counts from the sky vary in a linear, uniform way over the array as the sky brightness fluctuates, in order to facilitate sky subtraction by running median procedures.

Linearity calibration for NEWFIRM is complicated by the fact that the images which are recorded by the instrument do not include all of the counts that were originally collected in each pixel. The arrays are operated in correlated double sampling (CDS) mode. In CDS, the array is reset, then after a short but finite time interval, each pixel is read once, non-destructively. It takes a significant period of time to read the whole array, and therefore this "reset-to-1st-read" interval varies with pixel position. After the first read, the desired exposure time passes, and then the pixels are read again. The final value that is recorded for each pixel is difference between the values of the two readouts. However, the actual, total number of accumulated counts was larger, due to the counts collected during the r2r interval. For the most common broad band observing mode with NEWFIRM (4 digital averages and 1 Fowler sample), the r2r interval varies from 0.0346 to 1.1946 seconds, depending on pixel position on the array. For short exposures, this can be a significant fraction of the total requested exposure time, and for high count rates, the "true" number of counts collected in a pixel may be very different from the "measured" counts recorded after the CDS difference, and therefore may place that pixel on a very different part of the linearity curve. This r2r time is much longer when multiple Fowler sampling readouts are used (e.g., as a strategy to reduce the effective detector readout noise in low-background conditions such as narrow band imaging).

Figure 2 shows a schematic (but realistic) representation of NEWFIRM linearity behavior. Here, the number of counts recorded by the array departs quadratically from the true number of counts that should have been recorded if the array were linear, up to a saturation threshold at 12000 ADU. Figure 3 illustrates the number of counts that would actually be measured in a CDS

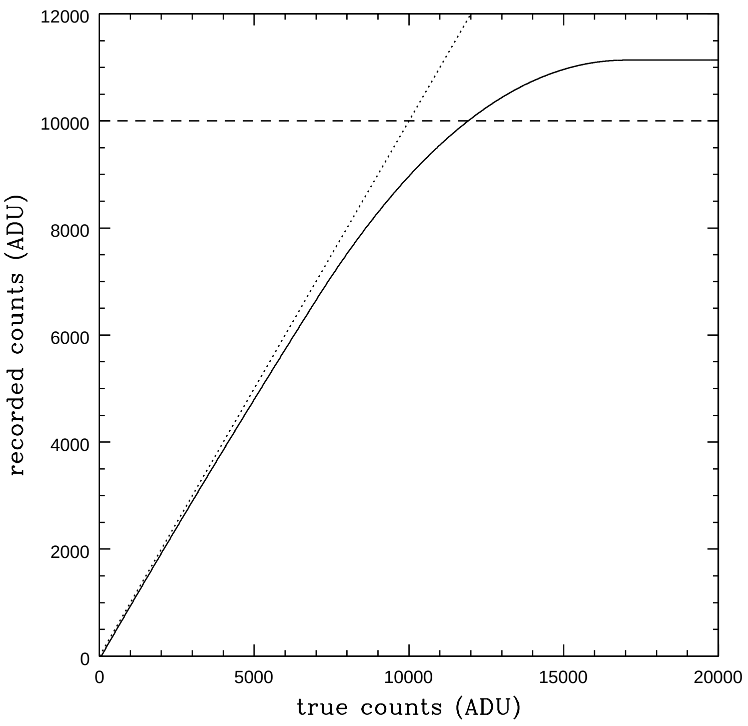

Figure 2: Schematic illustration of infrared array nonlinearity. For a true, linear count rate r_0 incident on the detector, the horizontal axis shows the "true counts" for an exposure time t, i.e., $n = r_0 \times t$, while the vertical axis shows the counts actually collected by the pixel. The dotted line shows a linear relation, while the solid line shows a function which departs quadratically in a manner similar to that of the NEWFIRM arrays, and which then saturates at a maximum value of 12000 ADU. The dashed reference line indicates 10000 ADU, where the NEWFIRM arrays depart from linear behavior by about 6%.

observation with requested exposure time t_e, taking into account the signal from the r2r interval t_r that gets subtracted away. The figure at left shows how the apparent saturation threshold decreases for shorter exposure times as the r2r time t_r becomes a significant fraction of the total exposure time $t_t = t_e + t_r$. At right, we see how the variation in the r2r time t_r over the array can lead to different linearity and saturation behavior for a fixed (short) exposure time.

The linearity behavior of NEWFIRM, and methods for correcting it, are discussed in detail in Dickinson et al. (2009). The `nfextern` package includes tasks to apply a linearity correction, as described in §5.6.

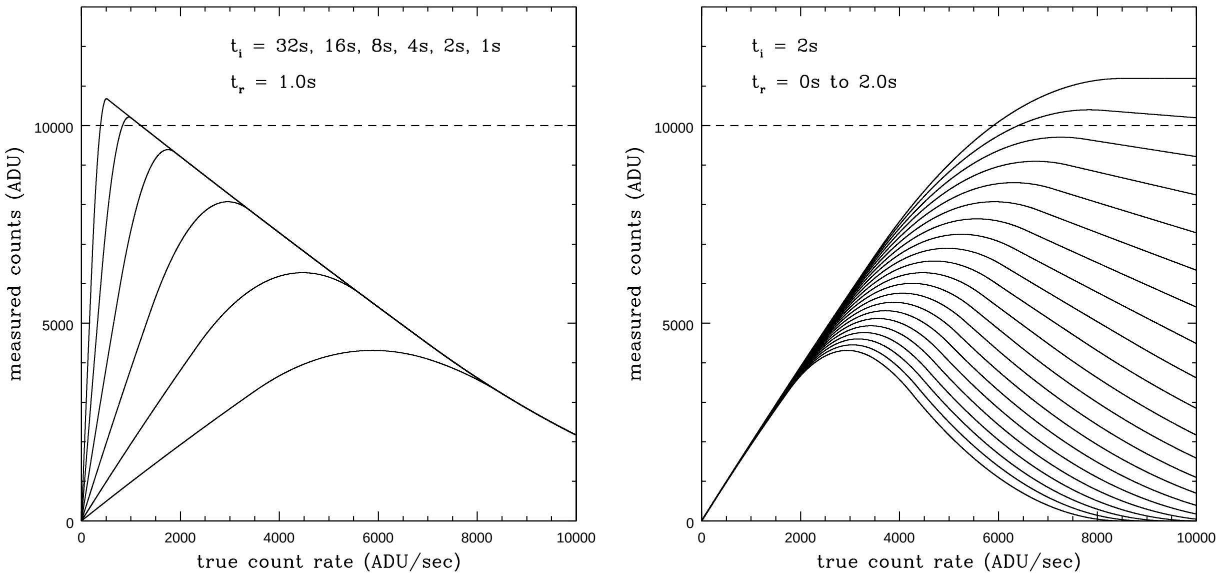

Figure 3: *Left:* Measured counts for schematic NEWFIRM array behavior, as a function of true incident count rate (r_0), for various exposure times $t_e = 1$ to 32 seconds, assuming a reset-to-1st-read interval $t_r = 1.0$ second. *Right:* Measured counts for a short (2 second) exposure as a function of incident count rate, for various reset-to-read intervals from 0 to 2 seconds.

4.4 Persistence

Infrared arrays are frequently subject to persistence effects (also sometimes called ‘latency’). Exposure to light, particularly at high bright levels, can result in a residual signal that gradually decays over time, and which can have an impact on subsequent exposures. There may be various physical detector behaviors that cause these persistent sources. One cause is an increase in the dark current. Another is a local change to the bias level. For CCD detectors, persistence may result from a failure to clear charge during array readout or clear operations. Naturally good detectors have minimal persistence which appears with low amplitude in the next exposure and decays quickly after that.

Persistence from sources in images of the sky can have a number of observable effects, some of which are complicated by the standard procedures used to reduce near-IR data. The persistent afterimages of bright sources may appear as positive “ghost images” subsequent, dithered exposures. These persistent images may then themselves impact sky subtraction. For a give image, a sky frame is commonly constructed using some sort of running average or median of several exposures taken before and after the image in question, which is then subtracted from the original image. Persistent afterimages may bias the sky frame and leave “dents” in the science data after sky subtraction. In particular, for a bright source, the persistence in subsequent exposures will lead to a systematic bias if frames “downstream” (in time) are included when determining the sky level.

There are several possible ways to handling persistence in the data calibration. Ideally, the decay of persistent signal as a function of time and/or array readout can be accurately modeled and subtracted from subsequence images. This method is often difficult to apply in practice, however, particularly for saturated sources where the initial signal level is unknown. Persistence in NEWFIRM has not has not yet been characterized well enough to enable this sort of correction. Persistent signal is found to decay quickly over the first few minutes after a pixel is exposed to a bright source, then more slowly, lasting as long as 30 minutes or more at a level of a few ADU.

Another way to treat persistence is to identify pixels affected by persistence and mask them for subsequent operations, such as estimating the background for sky subtraction, or when combining dithered images into a final stack. In §5.5 we describe how to do the latter using fairly simple assumptions.

4.5 Pixel scale, geometric distortion, and world coordinates

The NEWFIRM image scale is approximately 0.4 arcsec/pixel. However, in detail the pixel scale and geometry varies over the field of view, mainly due to optical effects in the instrument. The higher order terms of the geometric distortion, as well as the nominal pixel scales and orientations, have been calibrated by observations of astrometric reference fields. These are recorded in world coordinate system (WCS) keywords in the raw data FITS headers, including the coordinate tangent point (`CRPIX1, CRPIX2, CRVAL1, CRVAL2`), CD matrix (`CD1_1, CD1_2, CD2_1, CD2_2`), and higher order geometric terms encoded in the `WAT?_???` keywords according to the IRAF “TNX” convention.

It is believed that the NEWFIRM geometric distortion is fairly stable with time, although this has not been extensively tested by repeated astrometric calibration observations. However, low order terms may certainly vary. The exact orientation of the field may vary somewhat each time the instrument is mounted on the telescope. Although differential atmospheric refraction is a relatively small effect at infrared wavelengths, it can still lead to changes in pixel scale and field geometry as a function of telescope pointing and airmass.

The NOAO 4m telescope control system (TCS) provides information about the nominal telescope pointing at the time data are taken. This is recorded in the FITS headers of raw NEWFIRM headers in various keywords, including `RA` and `DEC`, `OBJRA` and `OBJDEC`, `TELRA` and `TELDEC`. These coordinates are not always accurate; they depend on operator and observer procedures to "zero" the pointing of the telescope on known sources, and the accuracy of the telescope pointing varies over the sky and with time.

Accurate astrometric calibration and alignment of NEWFIRM data depends on empirical measurement of source positions in the images. Fortunately, the 2MASS catalog provides a suitable resource for this purpose, with accurately calibrated astrometry for stars with a high enough surface density to provide many points of reference per NEWFIRM image, even at high galactic latitude. We describe procedures for calibrating the NEWFIRM WCS in § 5.9.

4.6 Flat fielding

The topics of flat fielding, photometric calibration, and background subtraction (discussed in §4.7) are intertwined, and to some degree are a matter of definition. The response of each pixel to illumination may be vary as a function of pixel position due to differences in the quantum efficiency of the detector and due to optical effects (filter transmission, vignetting, etc.) that may vary over the field of view. Here, we consider flat fielding to be the multiplicative operation needed to ensure that an astronomical source of fixed brightness will ultimately produce the same number of ADU in the final, reduced image, regardless of its position in the field of view.

Flat fields are often calibrated using observations of a source that nominally provides uniform illumination, such as a white spot in the telescope dome, or the sky itself. However, there are a variety of reasons why this may not fully accomplish the goal of ensuring that sources (such as stars) produce the same number of ADU in the reduced image regardless of location. For example, the "white spot" used for dome flats may not be perfectly uniformly illuminated, especially for a wide field camera like NEWFIRM. There may be color-dependent effects due to variations in the detector quantum efficiency or instrument optical transmission with wavelength that may lead to issues if the color of the flat field lamps or the sky background differs substantially from that of astronomical sources of interest. There may be scattered or stray light that can illuminate the detector in ways that are different from those of astronomical sources. The latter is particularly relevant at wavelengths longer than about 2 microns, where thermal emission from warm elements in the telescope structure or baffling can enter the light path.

For this first version of the NEWFIRM data reduction guide, we will neglect these various subtleties and address only the simple case of flat fielding using dome flats, focusing on the operational concerns of how to construct and apply master dome flat calibration images using tools available in the IRAF `nfextern` package. Several groups of observers have been investigating issues related to optimal flat fielding of NEWFIRM, and we expect to report on these in future editions of this guide.

The net efficiency of the four NEWFIRM quadrants is not the same. In particular, array SN019 (in extension `[im1]`) has an extra layer of anti-reflective coating, which leads to lower average responsivity, especially at shorter wavelengths (e.g., in the J-band). It is a matter of choice whether one tries to remove this during flat fielding, by using a MEF flat field whose normalization is different for each quadrant, or whether one normalizes the flat field in each quadrant to an average of 1, and then takes out the net zeropoint differences during the processing steps related to photometric calibration. In this flat fielding procedures described in §§5.2 and 5.6, we attempt calibrate out the different efficiencies of the arrays using the MEF flat field. However, it would be prudent to verify this by observing photometric standard stars in each quadrant, or by reference to unsaturated 2MASS stars measured in the four arrays.

Another subtle issue related to flat fielding (with any instrument, but particularly relevant to wide field imagers) is the varying angular scale of pixels over the field of view, described in the previous section. In raw images, the solid angle subtended by a pixel on the sky varies over the field of view. Flat fielding procedures that divide by a (supposedly) uniform source of illumination like a dome flat conserve *surface brightness*, not source flux: a source with fixed angular size will cover a different number of pixels in different parts of the array. In the data reduction procedures described in this guide, this will be addressed by resampling the images to a tangent plane projection that removes the nonlinear distortions. During this re-projection, the pixel resampling is done by conserving surface brightness (not pixel flux). The result is an image that is now photometrically "flat", such that a star or galaxy will have the same total number of ADU regardless of its position in the field of view.

4.7 Sky background

The ground-based near-infrared sky "background" is really a foreground, arising mainly from the atmospheric OH emission and from thermal emission (mainly at 2 microns and longer) from the atmosphere, telescope, and warm optics. This sky is quite bright, often much brighter than the astronomical sources of interest. It is therefore necessary to measure and remove it with great accuracy in order to study faint stars and galaxies. Both the intensity and the detailed 2-dimensional structure of the background can vary with position on the array and in time. The sky illumination may not be perfectly "flat" over the array, even after division by a flat field, for reasons discussed in the previous section. Moreover, the detailed response of the NEWFIRM detectors may vary with time, temperature, array readout parameters, or other conditions, leading to nonuniform background that may change over time. The *fractional* amplitude of these variations might be quite small relative to the mean sky background, perhaps a fraction of a percent, but it can still be important to account for them accurately when reducing infrared data. For example, temporal or spatial variations in the array response may be too small to matter from the point of view of source photometry. However, if the surface brightness of the sky is hundreds or even thousands of times larger than that of faint astronomical sources, then fractionally small effects on the sky background structure could easily swamp the signal from the intended science targets unless the background is removed very accurately.

Commonly, observers remove the background from infrared imaging data by taking the difference between images taken close in time to one another. The details of the procedure depend on the sort of target being observed and the observing strategy that is adopted. For large targets that fill a significant fraction of the imager field of view, or for crowded fields, an observer might occasionally offset the telescope pointing to a nearby, relatively blank field in order to obtain a sky calibration image. For observations of relatively sparse fields (e.g., deep imaging at high galactic latitudes), an observer may simply dither the telescope so that sources are moved over a number of independent, non-overlapping positions on the array. Sky background images may then be constructed from the dithered science data themselves.

The first edition of this data reduction guide will discuss only one example of sky subtraction for NEWFIRM data reduction, using tools in the IRAF `nfextern` package to carry out "running median" sky subtraction for dithered observations of relatively sparse fields. However, the `nfextern` tools can also facilitate sky subtraction for other observing modes, and it is planned that future versions of this document will discuss other examples.

5 Reducing NEWFIRM Data

The remaining sections of this guide present step-by-step examples illustrating the reduction of NEWFIRM data from raw exposures to image stacks.

In this first version of the *NEWFIRM Data Reduction Guide*, we will focus on one particular example, namely, the reduction of dithered broad band imaging data for relatively sparse fields. However, most of the steps apply equally to other sorts of observations as well. Imaging large targets that fill much of the field of view (large galaxies or galactic nebulosity, for example), or of crowded fields, may require different approaches to sky subtraction, which we hope to discuss in future versions of this document. For narrow band imaging, the sky background is lower, and additive instrumental effects such as bias and dark current may be more important and require closer attention. Narrow band integration times per image are also generally longer, leading to a slower cadence and thus perhaps to greater impact from instrumental instabilities. The sky background in narrow bands can also exhibit non-uniformities different than those in broad band data, e.g., "ring" patterns due to small shifts in the bandpass over the field of view relative to the wavelengths of OH atmospheric emission lines. Observations of relatively bright sources may require more attention to persistence and latency effects, or other artifacts not discussed here, such as internal reflections.

The basic steps of data processing that are discussed in the following sections include:

- Creating master dark calibrations (§5.1)
- Creating master dome flat calibrations (§5.2)
- Applying bad pixel masks (§5.3)
- Trimming images (§5.4)
- Linearity correction (§5.6)

- Flat fielding (§5.6)
- Sky subtraction (§5.7, §5.8)
- Astrometric calibration (§5.9)
- Re-projecting dithered images (§5.10)
- Stacking dithered images (§5.11)
- Second–pass processing: (§5.12)
  - Making object masks (§5.12.1)
  - De-projecting object masks (§5.12.2)
  - Second-pass sky subtraction (§5.12.3)
  - Stacking the second-pass images (§5.12.4)

5.1 Creating master dark calibrations

As discussed in §4.2, it is advisable to subtract dark calibration images taken with the same readout parameters (integration time, digital averages, coadds, and Fowler sampling) as those used for the science data. The actual dark current for NEWFIRM is quite low, and most of the structure in the "dark" images is actually electronic offset (or bias) whose value and structure can depend in a complex way on the readout parameters.

We will assume that you have a set of individual dark exposures in the working directory, taken in one or more NEWFIRM observing sequences, each of which may have had different instrument readout parameters. The dark sequences are most easily combined using the `dcombine` task, which is a script that runs `combine` with parameters set appropriately for darks. There are many parameters, but most can be left at their defaults. A subset of the relevant parameters are:

```
PACKAGE = newfirm
   TASK = dcombine

input   =                       List of input files to combine
output  =                 Dark_  Output rootname
(logfile=               STDOUT) Log output

(select =       obstype='dark') Selection expression
(group  =              exptime) Group expression
(seqval =               seqnum) Sequence value expression
(seqgap =                   0.) Maximum gap in sequence value

(combine=              average) Type of combine operation
(reject =               minmax) Type of rejection
(scale  =                 none) Image scaling
(zero   =                 none) Image zero point offset
(weight =                 none) Image weights
```

```
(nlow   =                     1) minmax: Number of low pixels to reject
(nhigh  =                     1) minmax: Number of high pixels to reject
(nkeep  =                     1) Minimum to keep (pos) or maximum to reject (neg)
```

The parameters `output` and `group` result in an output filename that combines the prefix `Dark_` with the image exposure time in sections, e.g., `Dark_10.fits`. The parameter `group = exptime` ensures that darks with the same exposure time are grouped together before combining. The parameters `seqval` and `seqgap` ensure that each sequence of darks taken by the observer (identified by the FITS header keyword `SEQVAL`) is combined separately. If you don't want this (e.g., you have multiple sequences of 10s darks and you want to combine them all together), set `seqval=""`. The default combine parameters are set so that the exposures are averaged without any rescaling or offsets, using `minmax` rejection, excluding the lowest and highest data values for each pixel.

You may run `dcombine` on individual lists of images (e.g., `dcombine @List.dark.10s`) that you have prepared. Or, you may simply run it for all raw images in your working directory. Setting the parameter `select` to the value `obstype = 'dark'` ensures that `dcombine` operates only on images with header keyword `OBSTYPE = DARK`, and the `group`, seqval and seqgap parameters should ensure that each dark sequence with a given exposure time are combined into a single output frame. For example, here we run `dcombine` on a directory of raw NEWFIRM data which includes four different sequences of dark exposures with various integration times, and then examine the results:

```
newfirm> dcombine n03.*.fits Dark_ group=exptime

newfirm> nflist Dark_*fits
May 13 17:11 nflist
Dark_10[im1][dark][1][][10.0][]
Dark_10[im2][dark][2][][10.0][]
Dark_10[im3][dark][3][][10.0][]
Dark_10[im4][dark][4][][10.0][]
Dark_60_1[im1][dark][1][][60.0][]
Dark_60_1[im2][dark][2][][60.0][]
Dark_60_1[im3][dark][3][][60.0][]
Dark_60_1[im4][dark][4][][60.0][]
Dark_60_2[im1][dark][1][][60.0][]
Dark_60_2[im2][dark][2][][60.0][]
Dark_60_2[im3][dark][3][][60.0][]
Dark_60_2[im4][dark][4][][60.0][]
Dark_5[im1][dark][1][][5.0][]
Dark_5[im2][dark][2][][5.0][]
Dark_5[im3][dark][3][][5.0][]
Dark_5[im4][dark][4][][5.0][]
```

Note that there were two different sequences of darks with `EXPTIME = 60`, resulting in two different combined images, `Dark_60_1` and `Dark_60_2`. In this particular case, one set of darks was taken with `EXPCOADD = 60` and `NCOADD = 1`, while the other was taken with `EXPCOADD = 15` and `NCOADD = 4`. Both have net `EXPTIME = 60`, but these darks will have different structure. Although the `newfirm` package has features that try to match calibrations with science data appropriately, e.g., by filter, exposure time, etc., the user should always be careful to keep track of these things and specify calibration files explicitly if ambiguities might otherwise result.

Running `dcombine` on raw darks will give a combined master dark that still needs to be trimmed to the right image size. The user may also wish to `fixpix` over known bad pixels. This can be done with the `nfdproc` command, which is a script that runs `nfproc` with appropriate parameters. If this is run on the combined dark (as opposed to the individual raw frames before combining), you will want to reset the default parameters to something like this:

```
PACKAGE = newfirm
   TASK = nfdproc

input   =                       List of input NEWFIRM files
(output =                tmp_+) List of processed dark files

(trim   =                  yes) Trim?
(fixpix =                  yes) Fix bad pixels by interpolation?
(bpm    =         nfdat$nfbpm) List of masks or expression
(biascor=                   no) Bias reference pixel correction?

(list   =                   no) List only?
(logfile=       STDOUT,logfile) Log files
(mode   =                   ql)
```

This will put the trimmed image into a temporary copy which can then be renamed:

```
newfirm> nfdproc Dark_*.fits
May 13 17:20 nfdproc
Dark_10[im1][dark][1][][10.0][]
Dark_10[im1]: Trim $I
trimsec = [2:2047,2:2047]
Dark_10[im1]: Fixpix $I
$M = nfdat$nfbpm[im1]
Dark_10[im2][dark][2][][10.0][]
Dark_10[im2]: Trim $I
trimsec = [2:2047,2:2047]
```

...etc...

```
newfirm> imhead *Dark_*.fits[im1] long-
Dark_10.fits[im1][2112,2048][real]: Dark Calibration
Dark_5.fits[im1][2112,2048][real]: Dark Calibration
Dark_60_1.fits[im1][2112,2048][real]: Dark Calibration
Dark_60_2.fits[im1][2112,2048][real]: Dark Calibration
tmp_Dark_10.fits[im1][2046,2046][real]: Dark Calibration
tmp_Dark_5.fits[im1][2046,2046][real]: Dark Calibration
tmp_Dark_60_1.fits[im1][2046,2046][real]: Dark Calibration
tmp_Dark_60_2.fits[im1][2046,2046][real]: Dark Calibration

newfirm> delete Dark_*.fits ver-
newfirm> imrename tmp_Dark_*.fits %tmp_%%Dark_*.fits ver+
'tmp_Dark_10.fits' -> 'Dark_10.fits'
'tmp_Dark_5.fits' -> 'Dark_5.fits'
'tmp_Dark_60_1.fits' -> 'Dark_60_1.fits'
'tmp_Dark_60_2.fits' -> 'Dark_60_2.fits'
```

5.2 Creating master dome flat calibrations

Flat fielding for a mosaic instrument like NEWFIRM can be understood in a variety of different ways, and indeed our knowledge of the best flat fielding procedures for NEWFIRM itself is still evolving. For this version of the *Data Reduction Guide*, we will consider only the standard case using dome flats, i.e., sequence of exposures taken of an illuminated white spot inside the telescope dome. With near-infrared instruments, it is common to take exposures both with the dome flat lamps on and off, and then take the difference of the two. In the K-band in particular, thermal emission from the white spot and elements of the telescope or instrument structure may illuminate the detector in ways that are different than how the image of an astronomical object, passing through the telescope and instrument light path, would illuminate the detector. Taking the on-off difference should remove any thermal (or other scattered light) component in the dome flats.

The preferred way to take dome flats using the NEWFIRM Observing Control System (NOCS) is with the 'DomeFlatSequence' operation. This takes a series of dome flats in selected filters with specified exposure times. The "lamps off" flats are taken first, followed by "lamps on" flats. The flat field lamp control is manual, and the observer is prompted to turn on the maps at the appropriate time. The NEWFIRM instrument and NOCS do not have any computer control (or sensing) of the dome flat lamp state, so observer input is the only way to provide header metadata information about the dome flats. Calibrations taken with the 'DomeFlatSequence' should be identifiable by the header keyword `NOCTYP = DFLATS`, with the keyword `NOCLAMP` set to `On` or `Off`, reflecting the nominal state of the dome flat lamp. Alternatively, NOCS also permits the use of the 'DomeFlat' sequence, with a check-box on the NOCS GUI (NGUI) to indicate whether the flatfield lamps are on or off. In this case, the header keyword `NOCTYP` will be set to `DFLATON` or `DFLATOFF` and `NOCLAMP` will be set to `On` or `Off` accordingly. For either NOCS script, the header keyword `OBSTYPE` should have the value `FLAT`.

These dome flat observing procedures rely on the observer to turn lamps on and off and use the observing scripts and NGUI check-boxes accordingly. Thus, they are naturally subject to error. When processing NEWFIRM data you should check the statistics in your flat images to make sure that the signal levels make sense. Also, pay close attention to the possibility of saturation in the dome flat images (see §4.3 for a discussion of saturation).

There are several ways in which one might go about constructing master dome flat calibration images. Here, we will consider the full procedure for constructing dome flat master calibrations.

1. Process individual dome flats: trimming, fixpixing, dark subtraction, linearization
2. Combine processed dome flats by filter and by dome flat lamp state (on and off)

3. Take on minus off difference for each filter
4. Normalize combined flat fields to a mean value of 1.

We illustrate this full procedure below, but in practice one might take some shortcuts. If the dome flat brightness level and ambient dome illumination level are fairly constant, then all of the images in a given filter in each lamps state should have about the same count level. In that case, one could switch the order of steps 1 and 2, combining the raw dome flat exposures first (by filter and lamp state), then process the combined image through trimming, fixpixing, dark subtraction and linearization (rather than processing each individual exposure, which is computationally more expensive).

The task `nffproc` is a script that runs `nfproc` with parameters set appropriately for processing dome flats. Here we show an example of useful parameter settings:

```
PACKAGE = newfirm
   TASK = nffproc

input   =                       List of input NEWFIRM files
(output =               flat_+) List of processed dark files

(trim   =                  yes) Trim?
(fixpix =                  yes) Fix bad pixels by interpolation?
(biascor=                   no) Bias reference pixel correction?
(darkcor=                  yes) Apply dark count calibration?
(lincor =                  yes) Linearity correction?
(normali=                   no) Normalize?

(bpm    =         nfdat$nfbpm) List of masks or expression
(darks  =               Dark_*) List of dark images
(floor  =                   1.) Output minimum value (ADU)

(list   =                   no) List only?
(logfile=      STDOUT,logfile) Log files
(mode   =                   ql)
```

The parameter `output = flat+_` results in output images that attach the prefix `flat_` to the input file name. The next parameters control the various processing steps. The use of the bias reference pixels has not yet been implemented. The bad pixel mask (for fixpixing) is specified by the `bpm` keyword, and the default is the standard bad pixel mask provided with the package distribution. The `darks` parameter specifies the names of dark calibration images to be subtracted. Unless a specific filename is given, the task will use a dark with the appropriate exposure time (`EXPTIME` header keyword), or with the closest `EXPTIME` available if there is not one with the exactly correct value.

`nffproc` will run only on objects with `OBSTYPE = FLAT`. You may provide it an input list specifically identifying dome flat exposures to process, or simply run it on all images in your working directory, and it will only process the flat field exposures. Here, we run `nffproc` with `normalize = no` so that the files are not renormalized to unit mean, as we still plan to take the difference of the (unnormalized) lamps–on and lamps–off flats.

```
newfirm> nffproc n03.*.fits

May 15 13:19 nffproc
n03.27786[im1][fflat][1][H][4.0][]
n03.27786[im1]: Trim $I
trimsec = [2:2047,2:2047]
n03.27786[im1]: Fixpix $I
$M = nfdat$nfbpm[im1]
n03.27786[im1]: dark calibration = ($I-$D)
$D = Dark_5[im1]
n03.27786[im1]: linearity correction = %(lin($L))
$L = nfdat$nflincoeffs[im1]
n03.27786[im1]: replace = (max($I,1.))
```

...etc...

```
newfirm> nflist flat_n03.27*fits
May 15 13:21 nflist
flat_n03.27786[im1][fflat][1][H][4.0][TXDLR]
flat_n03.27786[im2][fflat][2][H][4.0][TXDLR]
flat_n03.27786[im3][fflat][3][H][4.0][TXDLR]
flat_n03.27786[im4][fflat][4][H][4.0][TXDLR]
```

...etc...

Next, we combine the flats using `fcombine`, which (like `dcombine`) is a front-end script that runs `combine` with parameters set in a manner that is suitable for dome flats. A subset of the relevant parameters is shown here:

```
PACKAGE = newfirm
   TASK = fcombine

input   =                       List of input files to combine
output  =                 Flat_  Output rootname

(select =       obstype='flat') Selection expression
(group  = mkid(filter,1,1)//noclamp) Group expression
(seqval =                seqnum) Sequence value expression
(seqgap =                    0.) Maximum gap in sequence value

(combine=               average) Type of combine operation
(reject =             avsigclip) Type of rejection
(scale  =                  mode) Image scaling
(zero   =                  none) Image zero point offset
(weight =                  none) Image weights
(mclip  =                   yes) Use median in sigma clipping algorithms?
(lsigma =                    3.) Lower sigma clipping factor
(hsigma =                    3.) Upper sigma clipping factor
```

The `select` parameter ensures that only images with header keyword `OBSTYPE = FLAT` are combined, while the `group` parameter divides the list of input files into groups by the header

`FILTER` parameter and by the status of the dome flat lamp (from the `NOCLAMP` header keyword). As for the case of `dcombine` illustrated above, the `seqval` and `seqgap` parameters ensure set to process each NOCS observing sequence of flats separately. These parameter values should be changed if you wish to combine flats from multiple NOCS observing sequences. We go ahead and combine the flats:

```
newfirm> fcombine flat_n03.*.fits Flat_
```

The names of the combined flats include the filter name and the lamp state:

```
newfirm> dir Flat*fits
Flat_HOff.fits   Flat_HOn.fits    Flat_JOff.fits   Flat_JOn.fits    Flat_KOff.fits   Flat_
```

Later, when we want to process object images, it is possible to use `nfproc` to flat field using the On and Off flats separately and a suitable expression to take the normalized difference "on the fly". However, it is usually simpler to just take the on–off difference and normalize the flats manually:

```
newfirm> mscarith Flat_JOn.fits - Flat_JOff.fits Flat_Jdiff.fits
```

Next, we normalize the flat field to have a typical value of 1. Here, we will normalize the mean of the whole flat (averaged over all four detector arrays) to 1, rather than normalize each extension individually. In principle, this lets us equalize the photometric zeropoints of the four NEWFIRM quadrants, using the flat field to remove net differences in system throughput from detector to detector. Here are the image statistics for the four quadrants:

```
newfirm> mscstat Flat_Jdiff.fits
#                IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
Flat_Jdiff.fits[im1]  4158523     2174.     142.7     1461.     2885.
Flat_Jdiff.fits[im2]  4139809  2629.909  121.193  2023.951  3235.806
Flat_Jdiff.fits[im3]  4119294  2802.128  182.1176  1891.54  3711.312
Flat_Jdiff.fits[im4]  4166825  2789.686  145.1734  2063.903  3514.13
```

Notice in particular that extension `[im1]` has a lower average value, which reflects the reduced efficiency due to the extra layer of anti-reflective coating that was applied to the detector array in that quadrant.

Running `mscstat` using the option `gmode+` gives a "global mode" value for the image, combining data from all four extensions. The output gives the global mode, as well as the ratio of the mode for each of the four NEWFIRM extensions to the global value:

```
newfirm> mscstat Flat_Jdiff.fits gmode+
Flat_Jdiff.fits   2574.229  0.868  1.035  1.058  1.039

newfirm> mscarith Flat_Jdiff.fits / 2574.229 Flat_Jdiffnorm.fits

newfirm> mscstat Flat_Jdiffnorm.fits gmode+
Flat_Jdiffnorm.fits      0.999  0.868  1.037  1.056  1.038
```

Finally, as a matter of tidiness, we will adjust the PROCMEAN keyword in the difference flat header. This keyword gives the mean value of the flat field image and was computed for the "on" and "off" flats separately when they were produced by fcombine. However, since we have created our difference flat "manually" with mscarith, the value of PROCMEAN is now correct. In fact this keyword is not currently used in the default flat-fielding procedures in other tasks (e.g., nfproc), but that may change in the future, so it is good practice to set PROCMEAN to 1 explicitly in our normalized difference flat:

```
newfirm> mscedit Flat_Jdiffnorm.fits PROCMEAN 1.0 show+ up+
hedit $input PROCMEAN '1.' add=NO addo=NO del=NO ver=NO show=YES upd=YES
Flat_Jdiffnorm.fits[im1],PROCMEAN: 2815.262 -> 1.
Flat_Jdiffnorm.fits[im1] updated
Flat_Jdiffnorm.fits[im2],PROCMEAN: 2815.262 -> 1.
Flat_Jdiffnorm.fits[im2] updated
Flat_Jdiffnorm.fits[im3],PROCMEAN: 2815.262 -> 1.
Flat_Jdiffnorm.fits[im3] updated
Flat_Jdiffnorm.fits[im4],PROCMEAN: 2815.262 -> 1.
Flat_Jdiffnorm.fits[im4] updated
```

5.3 Static bad pixel mask (BPM)

The NEWFIRM detector arrays have various defects which should be masked or interpolated during image reduction. Many of these are illustrated in the NEWFIRM chapter of the *NOAO Data Handbook*. They include cracks in the detector substrate, regions of sharply reduced sensitivity due to contaminants on the arrays, some portions of the detectors that have de-bonded from the readout electronics, pixels with unusually high dark current, pixels with unusual linearity response, and some regions with detector damage that "glow" quite brightly. There is no unique way to identify all bad pixels on the array, since this is a matter of degree. However, in general it pays to be conservative when masking pixels. If you have dithered your observations through many independent positions, then you can safely mask all pixels whose behavior might be suspicious. Even quite "aggressive" masking will generally only flag at most a few percent of the pixels, leading to very small reduction of net effective exposure time for a dithered, combined image, while ensuring that the bad pixels do not contaminate your final data product.

The headers of NEWFIRM raw generally already include the keyword BPM used to specify a bad pixel mask. This will have been populated by some value that refers to a mask that was used by the NEWFIRM Quick-Reduced Pipeline at the time the data were taken. However, this may not be the best bad pixel mask for your purposes. Some newfirm package tasks include a parameter that lets you specify a bad pixel mask explicitly; others rely on the header BPM keyword.

The distributed version of the nfextern package includes a bad pixel mask that is used by the NEWFIRM science pipeline. This may be satisfactory for most purposes. Here, we edit the BPM keywords of our data so that they point to this file. Note that this must be done for each image extension, pointing to the correct extension of the BPM:

```
newfirm> hedit n03.*.fits[im1] BPM nfdat$nfbpm[im1] ver- up+ show-
newfirm> hedit n03.*.fits[im2] BPM nfdat$nfbpm[im2] ver- up+ show-
newfirm> hedit n03.*.fits[im3] BPM nfdat$nfbpm[im3] ver- up+ show-
newfirm> hedit n03.*.fits[im4] BPM nfdat$nfbpm[im4] ver- up+ show-
```

5.4 Trimming, bad pixel interpolation, and dark subtraction

We will use the task `nfoproc` as our tool for basic frame-level processing of science (object) data. Like `nfdproc` and `nffproc`, the task `nfoproc` is simply a wrapper script that calls the basic task `nfproc` with parameters set to be appropriate for processing on-sky science object exposures. It will only operate on images with header keyword `OBSTYPE = OBJECT`.

First, we set the `noproc` parameters to trim the image, interpolate over bad pixels specified by the bad pixel mask given in the `bpm` parameter, and to subtract a dark frame specified by the `darks` parameter. If a list of dark frames is given (as in the example below), the task will choose the one that has the closest exposure time (from header keyword `EXPTIME`) to that of the object data being processed. However, be careful about this, as the built-in intelligence is limited: if you have used various combinations of exposure times, digital averages, coadds, and Fowler sampling when taking data, the task may not automatically select the right dark. When in doubt, specify the dark frames explicitly.

```
PACKAGE = newfirm
   TASK = nfoproc

input   =            n03.*fits  List of input NEWFIRM exposures
(output =               obj_+) List of processed object exposures

(trim   =                 yes) Trim?
(fixpix =                 yes) Fix bad pixels by interpolation?
(biascor=                  no) Bias reference pixel correction?
(darkcor=                 yes) Apply dark count calibration?
(lincor =                  no) Linearity correction?
(flatcor=                  no) Apply flat field calibration?
(bpm    =         nfdat$nfbpm) List of masks or expression
(darks  =              Dark_*) List of dark calibrations
(flats  =              Flat_*) List of flat calibrations
(flattyp=                  on) Type of flat field (on|off|diff)

(list   =                  no) List only?
(logfile=      STDOUT,logfile) Log files
(mode   =                  ql)
```

Running `nfoproc`, we see this log output:

```
May 19 12:34 nfoproc
n03.28057[im1][object][1][J][60.0][]
n03.28057[im1]: Trim $I
trimsec = [2:2047,2:2047]
n03.28057[im1]: Fixpix $I
$M = nfdat$nfbpm[im1]
n03.28057[im1]: dark calibration = ($I-$D)
$D = Dark_60_2[im1]
```

A listing of the partially processed images shows the T, X and D flags that indicate trimming, fixpixing, and dark subtraction:

```
newfirm> nflist obj_n03.*.fits
May 19 16:59 nflist
obj_n03.28057[im1][object][1][J][60.0][TXD]
obj_n03.28057[im2][object][2][J][60.0][TXD]
obj_n03.28057[im3][object][3][J][60.0][TXD]
obj_n03.28057[im4][object][4][J][60.0][TXD]
```

5.5 Saturation and persistence masks

In §§4.3 and 4.4 we discussed the nature of saturation and persistence in the NEWFIRM arrays. Here, we will use the `nfmask` task to generate a mask for each object image that flags pixels according to various conditions, including saturation and the possible presence of persistence. For good measure, we will also include the ordinary static bad pixels in this mask, so that it can serve many purposes in later processing. The default parameters for `nfmask` are:

```
PACKAGE = newfirm
   TASK = nfmask

input   =                          List of input NEWFIRM files
(output =                   +_msk) List of output mask files
(maskkey=                    MASK) Keyword to record mask in input (optional)

(bpmvalu=                       1) Mask value for BPM
(obmvalu=                       2) Mask value for OBM
(satvalu=                       3) Mask value for saturation
(pervalu=                       4) Mask value for persistence

(lcoeffs=               parameter) Lin coeffs (parameter|exprdb|keyword|image)
(lin1   =  -6.1230000000000E-6) Linearity coefficient for im1
(lin2   =  -7.0370000000000E-6) Linearity coefficient for im2
(lin3   =  -5.4040000000000E-6) Linearity coefficient for im3
(lin4   =  -5.9520000000000E-6) Linearity coefficient for im4
(linimag=     nfdat$nflincoeffs) List of linearity coefficient images

(scoeffs=               parameter) Sat coeffs (parameter|exprdb|keyword|image)
(sat1   =                   10000) Saturation threshold for im1
(sat2   =                   10000) Saturation threshold for im2
(sat3   =                   10000) Saturation threshold for im3
(sat4   =                   10000) Saturation threshold for im4
(satimag=                        ) List of saturation images

(perwind=                       5) Persistence window
(perthre=                     0.8) Persistence threshold as fraction of saturation
(bpm    =            nfdat$nfbpm) List of masks or expression
(obm    =              (objmask)) List of object masks or expression
(exprdb =      nfdat$exprdb.dat) Expression database
```

```
(list   =                   no) List only and show full expressions?
(logfile=       STDOUT,logfile) Log files
(mode   =                   ql)
```

As described in §4.3, saturation occurs at different ADU levels depending on the count rate of the source and the array readout parameters such as exposure time, digital averages, coadds, and Fowler sampling. The task `nfmask` keeps track of this and should flag saturated pixels appropriately. The use of a fixed "yes/no" saturation threshold is an artificial but useful simplification. `nfmask` can use a single, constant threshold (set with the `sat` keyword) or an image of saturation values (specified by the `satimage` keyword).

To flag pixels that may be affected by persistence, `nfmask` operates at a time series of exposures. No persistence masking will be done if the task is run on a single image. The images are sorted chronologically by the time of observation. For each pixel in an image, `nfmask` uses an efficient "running maximum" algorithm to examine data values in several preceding images, the number of which is specified by the parameter `perwindow` (default value = 5). The task looks for pixels whose value exceeded some ADU threshold appropriate for triggering persistence. This threshold is set to the saturation value (keyword `sat`) multiplied by a constant specified by the keyword `perthresh` (with a default value of 0.8). Effectively, when a pixel exceeds this threshold in one image, that pixel is flagged for `perwindow` images temporally "downstream".

Users may wish to adjust the values of `perthresh` and `perwindow` depending on the nature of their observations. For example, data taken with high background levels (e.g., broad band K imaging) will also have high per-pixel noise per exposure, and persistence may quickly fade below the level where it exceeds the noise level. On the other hand, persistence may be visible for longer times, at lower ADU levels, in low-background imaging (e.g., in the J-band or narrow bands). `perwindow` might also be adjusted depending on the cadence of the data-taking: if many short exposures are taken rapidly, it may need to be increased, or decreased if long exposure times are used.

When processing a given sequence of NEWFIRM observations, be aware that images from *preceding* sequences may also cause detectable persistence. E.g., a bright star used to calibrate the telescope pointing, a standard star field, or a focus sequence may all cause persistence which can then affect subsequent science observations. Ideally, you should run the `nfmask` on a series of all images taken during a given night (or at least any series of exposures taken without long time gaps), even if some of those data (e.g., focus sequences or pointing tests) may not be reduced for their own value.

We run `nfmask` on our partially-processed object images and see this output:

```
newfirm> nfmask obj_n03.*.fits bpmval=1 obmval=0 satval=3 perval=4
May 19 17:01 nfmask
obj_n03.28057[im1][object][1][J][60.0][TXD]
obj_n03.28057[im1]: saturation = %(allmask(nfmask.lin\I,nfmask.sat,0.8,1,2,3,4))
$M = nfdat$nfbpm[im1]
$O = EMPTY
$P = persistence
obj_n03.28058[im1][object][1][J][60.0][TXD]
obj_n03.28058[im1]: saturation = %(allmask(nfmask.lin\I,nfmask.sat,0.8,1,2,3,4))
```

```
$M = nfdat$nfbpm[im1]
$O = EMPTY
$P = persistence
```

...etc...

If we now use `nlist` for our object images, we see that the processing flags have changed:

```
newfirm> nflist obj_n03.?????.fits
May 19 19:51 nfmask
obj_n03.28057[im1][object][1][J][60.0][TXD,Z]
obj_n03.28057[im2][object][2][J][60.0][TXD,Z]
obj_n03.28057[im3][object][3][J][60.0][TXD,Z]
obj_n03.28057[im4][object][4][J][60.0][TXD,Z]
```

The mask names have been added to the extension headers with the `MASK` keyword:

```
newfirm> mscselect obj_n03.?????.fits $I,mask
obj_n03.28057.fits[im1] obj_n03.28057_msk[im1]
obj_n03.28057.fits[im2] obj_n03.28057_msk[im2]
obj_n03.28057.fits[im3] obj_n03.28057_msk[im3]
obj_n03.28057.fits[im4] obj_n03.28057_msk[im4]
obj_n03.28058.fits[im1] obj_n03.28058_msk[im1]
obj_n03.28058.fits[im2] obj_n03.28058_msk[im2]
obj_n03.28058.fits[im3] obj_n03.28058_msk[im3]
obj_n03.28058.fits[im4] obj_n03.28058_msk[im4]
```

...etc....

If for some reason you wish to regenerate the masks (e.g., changing the thresholds, or setting different mask values), you must first delete the `MASK` header keywords and reset the processing flags (`PROCDONE` keyword) to `TXD`, otherwise you will not be able to rerun `nfmask` on the same images:

```
newfirm> mscedit obj_n03.?????.fits MASK delete+ ver- up+ show-
newfirm> mscedit obj_n03.?????.fits PROCDONE "TXD" ver- up+ show-
```

5.6 Linearity correction and flat fielding

The nature of array nonlinearity in NEWFIRM was discussed in §4.3. This correction consists of evaluating a function of the observed counts, exposure time, and readout parameters (number of Fowler reads, digital averages, coadds, and exposure time) as a function of position. The function also depends on controller and array coefficients. For generality the linearity correction is evaluated using a general expression evaluator. The complex expression is defined in an text file called the expression database. The expression evaluator is part of the the general `nfproc` task. A simpler interface is provided by the task `nflinearize`. The expression database provided with the software is indicated by the default `exprdb` parameter for these tasks. The file is distributed with the software, and is located in the `nfdat$` directory as `nfdat$exprdb.dat`

To allow user tuning and updates the linearity coefficients may be specified in various ways. The `coeffs` parameter allows global coefficients for each array to be specified by task parameters, in the expression database, and in header keywords. Rather than a global value the coefficients may also be individually specified for each pixel using a coefficient image. The recommended values for the global coefficients as well as coefficient images are provided with the package as of the time of the package release. Updates to these will be posted to the NEWFIRM web sites.

```
newfirm> nfoproc @List.n03.J1 output="f+" trim- fixpix- biascor- darkcor- lincor+ flatcor+
May 19 18:03 nfoproc
obj_n03.28057[im1][object][1][J][60.0][TXD,Z]
obj_n03.28057[im1]: linearity correction = %(lin($L))
$L = nfdat$nflincoeffs[im1]
obj_n03.28057[im1]: flat calibration = $I/max(0.1,$F)
$F = Flat_Jdiffnorm[im1]
obj_n03.28057[im2][object][2][J][60.0][TXD,Z]
obj_n03.28057[im2]: linearity correction = %(lin($L))
$L = nfdat$nflincoeffs[im2]
obj_n03.28057[im2]: flat calibration = $I/max(0.1,$F)
$F = Flat_Jdiffnorm[im2]
obj_n03.28057[im3][object][3][J][60.0][TXD,Z]
obj_n03.28057[im3]: linearity correction = %(lin($L))
$L = nfdat$nflincoeffs[im3]
obj_n03.28057[im3]: flat calibration = $I/max(0.1,$F)
$F = Flat_Jdiffnorm[im3]
obj_n03.28057[im4][object][4][J][60.0][TXD,Z]
obj_n03.28057[im4]: linearity correction = %(lin($L))
$L = nfdat$nflincoeffs[im4]
obj_n03.28057[im4]: flat calibration = $I/max(0.1,$F)
$F = Flat_Jdiffnorm[im4]
```

...etc....

We can now see that linearization (`L`) and flat fielding (`F`) have been added to the processing flags for our data:

```
newfirm> nflist fobj_n03.28*fits
May 20 11:10 nflist
fobj_n03.28057[im1][object][1][J][60.0][TXD,Z,LF]
fobj_n03.28057[im2][object][2][J][60.0][TXD,Z,LF]
fobj_n03.28057[im3][object][3][J][60.0][TXD,Z,LF]
fobj_n03.28057[im4][object][4][J][60.0][TXD,Z,LF]
fobj_n03.28058[im1][object][1][J][60.0][TXD,Z,LF]
fobj_n03.28058[im2][object][2][J][60.0][TXD,Z,LF]
fobj_n03.28058[im3][object][3][J][60.0][TXD,Z,LF]
fobj_n03.28058[im4][object][4][J][60.0][TXD,Z,LF]
```

...etc...

5.7 Sky Subtraction

There are many different approaches to sky subtraction for near-infrared data. As discussed in §4.7, the choice may depend on the nature of the target being observed, as well as the observing strategy that was adopted. Observations of large, extended targets that fill a significant portion of the field of view, or of crowded fields, may require chopping to “offset” fields to measure and remove the sky background. When imaging relatively sparse fields, it is often possible to construct sky images from the dithered science data themselves.

Here, we discuss only one example of NEWFIRM sky subtraction, namely that of “running median” sky subtraction for dithered imaging in sparse fields. However, the `nfskysub` task (and the underlying `nfproc` tool) enable considerable flexibility in defining sky subtraction operations for other types of data sets. Future versions of this *Data Reduction Guide* will discuss other examples in more detail.

Once again, `nfskysub` is a script that calls `nfproc` with a restricted set of parameters related to sky subtraction. For our example of “running median” sky subtraction, we might set the parameters as follows:

```
PACKAGE = newfirm
   TASK = nfskysub

input   =           @List.fobj  List of input images
(output =                 +_ss) List of output images
(outtype=                image) Output type (image|list|<keyword>)
(skies  =                     ) List of skies (empty to use input)
(skymatc=                     ) Match boolean expression
(skymode=         median 9 1) Sky subtraction mode
(stype  =                     ) Sky selection expression (boolean)
(obm    =                     ) Input mask or keyword reference
(logfile=      STDOUT,logfile) List of output logfiles
(fd     =                     )
(mode   =                   ql)
```

The default syntax for the `output` parameter appends the suffix `_ss` to the input filename. The parameter `outtype = image` tells the task to actually produce sky subtracted output images, as opposed to just a listing of the operations, or (optionally) to encode the sky subtraction operation syntax in a header keyword.

The `skies` parameter can be used to specify a list of images that will be used to construct the sky background for the `input` images. If `skies` is left blank, then the `input` list itself is used, as for our example here of sky subtraction for a dithered sparse field. If `skymatch` is specified, it can be a logical expression that is used to evaluate which images from the `skies` list are used to construct the background correction for each `input` image. For example, one might define functions that set a minimum (or maximum) pointing separation between the `input` image and the images from the `skies` list that will be used to construct the background. Or, `skies` may be used to test header keywords that indicate which images are “on target” and which are “offset sky” observations. For the present example, we will not use the `skies` or `skymatch` parameters.

When the sky images are constructed from the `input` list instead, the `stype` parameter plays a similar role to that of `skymatch` for the `skies` list. Here, however, we leave it blank, meaning that all images from the `input` list are potentially available to use for sky subtraction. Note that in all circumstances, the particular `input` image being processed at any given time is automatically *excluded* from whatever set of images is used to construct the sky background. This avoids any issues with auto-subtraction of the science data.

The `skymode` parameter defines the algorithm that will be used to construct the sky from the list of potential images (in this example, the `input` list). The list of images is sorted according to a parameter specified by the `nfproc.sortval` parameter, which defaults to the time of the observation (`MJD-OBS` header keyword). For `skymode = before`, `after`, or `nearest`, the task performs a simple pair-wise difference, subtracting the image before, after, or closest in `MJD-OBS` to the `input` image.

Instead, `skymode = median` constructs the sky from the median of images from the `input` (or `skies`) lists. If no other parameters are specified, then all images are used (always excluding the image being processed, if the the sky images are taken from the `input` list). There are two optional parameters: `median [<N> [<AVG>]]` (e.g., `skymode = "median 9 1"` in our example above). `N` gives the number of images in the "running median" window. Literally, for a given image from the `input` list, the chronologically sorted list of potential sky images is considered. The first image that is used is `int(N/2)` before the image being processed, followed by `N-1` more images. The `input` image being processed is always excluded, so that if the sky is being constructed from the `input` list, then the median is actually computed from `N-1` images. E.g., for `skymode = "median 9"`, the sky is the median of ±4 images images before and after the observation being processed, excluding that image itself. Near the start or end of the `input` list, the running median window necessarily becomes asymmetric around the image being processed in order to keep the window size equal to `N`. If there are fewer than `N` images, all are used (again always excluding the image being processed). The second optional parameter `AVG` indicates the number of data values around the median to be averaged when computing the sky value for each pixel. If `AVG = 1`, then this is a simple median; for larger `AVG`, more values are averaged, up to `AVG = N`, where all data values are used and averaged. If the actual number of images used for the running median is even (as in our example, where we construct the sky from the `input` list with `skymode = "median 9 1"`, such that 8 images are actually used to compute the median), then the value of `AVG` is rounded up to an even number to keep the median calculation symmetric - i.e., in this case, the central 2 values from the sorted list of 8 will be averaged.

The `obm` parameter is used to provide a list of masks (or a pointer to mask file names) that may be used to exclude pixels when constructing the sky images. For example, the user may wish to mask objects in the images explicitly, or latent images from previous exposures, so that they do not bias values of the sky image. For our current example, we will leave the `obm` parameter blank for now, but return to it later when we discuss "second pass" sky subtraction, where we will include object and persistence masking.

5.8 Further adjustment to the sky subtraction

Experience has shown that "running median" sky subtraction removes the background from well-dithered broad band data quite accurately. However, small offsets and gradients often remain, and are typically different from quadrant to quadrant. In one test, `nfskysub` was run on a long series of 60 second J-band exposures taken in photometric conditions. Low-order polynomial surfaces were fit to the sky residuals in each quadrant. The median peak-to-peak amplitude of these gradients was found to be about 0.3%. However, the offsets and slight gradients were distinctly noticeable over the wide NEWFIRM field of view, and it is important to remove these before re-projecting and stacking the images.

The task `nfdeltasky` is designed to measure and remove residual sky offsets and gradients. It is a script that in turn calls the `msctools` task `mscskysub`, which may be familiar to MOSAIC observers from the `mscred` package. `mscskysub` fits a polynomial surface to an image, typically after using a large-scale spatial median filtering to minimize the effects of objects on the fitting. The parameters are:

```
newfirm> lpar nfdeltasky
       inlist =                   Input image(s)
      outlist =                   Output image(s)
      (xorder = 2)                Order of mscskysub function in x
      (yorder = 2)                Order of mscskysub function in x
    (function = "legendre")       Function to be fit (legendre,chebyshev,spline3)
 (cross_terms = yes)              Include cross-terms for polynomials?
     (xmedian = 100)              X length of median box
     (ymedian = 100)              Y length of median box
(median_perce = 50)               Minimum fraction of pixels in median box
       (lower = INDEF)            Lower limit for residuals
       (upper = INDEF)            Upper limit for residuals
       (ngrow = 0)                Radius of region growing circle
       (niter = 0)                Maximum number of rejection cycles
     (regions = "mask")           Good regions (all,rows,columns,border,sections,circle,invc
        (rows = "*")              Rows to be fit
     (columns = "*")              Columns to be fit
      (border = 50)               Width of border to be fit
    (sections = "")               File name for sections list
      (circle = "")               Circle specifications
        (mask = "BPM")            Mask
     (div_min = INDEF)            Division minimum for response output
   (inimglist = "")
  (outimglist = "")
        (mode = "ql")
```

The parameter `regions` can be used to define regions where pixels are used or excluded from the surface fitting. Among the various options, `regions = mask` is particularly useful, as it allows the user to specify a mask (with the `mask` parameter) that defines which pixels are fit. The default `mask = BPM` reads the name of the mask from the header keyword `BPM`. This makes it easy to exclude regular, static bad pixel regions from the fits. However, more elaborate masks could be created and used to exclude pixels associated with objects from the fitting. For sparse–field imaging, you can generally leave `mask = BPM` and rely on the median filtering (`xmedian`, `ymedian`, and `median_percent` parameters) to minimize the impact of objects on the surface fitting.

Experience so far suggests that `xorder = yorder = 2` or `3` generally does a quite adequate job of removing residual backgrounds in broad-band data. Constant offsets (i.e., `xorder`

`= yorder = 1`) are generally not sufficient.

`nfdeltasky` does not (yet) use the new prefix and suffix syntax for output file names that many other `newfirm` package tasks can use, so you must specify the input and output file lists explicitly or with IRAF filename substring substitution, e.g.:

```
newfirm> nfdeltasky fobj*_ss.fits fobj*_%ss%ss1%.fits verbose+
nfdeltasky fobj_n03.28060_ss.fits[im1] fobj_n03.28060_ss1.fits[im1]
nfdeltasky fobj_n03.28060_ss.fits[im2] fobj_n03.28060_ss1.fits[im2]
nfdeltasky fobj_n03.28060_ss.fits[im3] fobj_n03.28060_ss1.fits[im3]
nfdeltasky fobj_n03.28060_ss.fits[im4] fobj_n03.28060_ss1.fits[im4]
nfdeltasky fobj_n03.28061_ss.fits[im1] fobj_n03.28061_ss1.fits[im1]
nfdeltasky fobj_n03.28061_ss.fits[im2] fobj_n03.28061_ss1.fits[im2]
nfdeltasky fobj_n03.28061_ss.fits[im3] fobj_n03.28061_ss1.fits[im3]
nfdeltasky fobj_n03.28061_ss.fits[im4] fobj_n03.28061_ss1.fits[im4]
```

...etc...

5.9 Astrometric Calibration

The next several steps (astrometric calibration, re-projection, and stacking) are common to the reduction of imaging data from any mosaic camera. The procedures for NEWFIRM are essentially the same as those for MOSAIC, and which will be familiar to MOSAIC observers who have used the `mscred` package.

Astrometric calibration consists of matching sources in the exposure to a reference catalog and fitting a world coordinate system (WCS) transformation function between the pixel coordinates in the images and the reference celestial coordinates. The WCS is stored in the extensions headers, one for each array.

NEWFIRM data has an initial WCS provided by the data acquisition system. This is an important component of the raw data because it simplifies identifying the reference sources. The WCS is based on a calibration made at an earlier time in a good astrometric field. At the telescope this prior calibration is added to the headers with the tangent point set to the coordinates provided by the telescope control system.

For the purpose of an initial WCS it is assumed that the distortions in and the relative orientation of the arrays do not change with position on the sky relative to fiducial point in the field, called the tangent point. What does change is celestial coordinate of the tangent point (defined by the telescope control system or TCS), a rotation of the camera, a small scale change when the initial WCS was derived from a different wavelength and/or filter, and refraction. NEWFIRM is used with a fixed orientation on the sky so rotation is only a small amount from the remounting of the instrument. As an infra-red camera refraction effects are also small. The telescope pointing error is the main effect that has to be determined by the calibration algorithm. Once this is determined the source matching is straightforward and the other effects are calibrated by fitting an improved WCS.

That task used to perform the astrometric calibration is `nfwcs`. The algorithms are described in the help page for that task and in the supporting task `acematch`. The task generates catalogs of image sources and reference sources, matches the former to the latter, and uses this information to derive a new WCS solution. The catalogs may be run in a transient fashion or saved for use as overlays or for other purposes.

The default parameters for `nfwcs` are:

```
PACKAGE = newfirm
   TASK = nfwcs

input   =                       List of NEWFIRM exposures
(sky    =                     ) Optional sky exposure
(outroot=                     ) Output rootname (+=input rootname)
(coords =     !nftwomass $I $C) Astrometric reference catalog
(cdef   =             c1,c2,c3) Ref fields for ra, dec, and mag
(search =                 300.) Search radius (arcsec)
(raoffse=                   0.) RA offset (arcsec)
(decoffs=                   0.) DEC offset (arcsec)
(ngrid  =                  100) Refit grid
(reset  =                   no) Reset offsets between input?
(logfile=                     ) Logfile
(verbose=                  yes) Verbose?
(fd     =                     )
(mode   =                   ql)
```

In our example here, we will run this on images that have already been sky-subtracted, but one might also choose to subtract a sky image specified by the `sky` parameter. The task uses the cataloging package `ace` to generate a list of sources detected in the NEWFIRM image.

The astrometric reference source catalog can be produced in various ways, have several formats, and may contain more fields than are used. The simplest and default reference catalog is a text file with right ascension (RA) in hours, declination (DEC) in degrees and one or more magnitudes (MAG). The `nfwcs` parameter `coords` defines the reference catalog. It can specify a particular file, or a task that is run to produce the reference catalog for each exposure based on its initial WCS. Typically this automated method is used, at least for the first time the task is run. For NEWFIRM, the default is to run the task `nftwomass`, which is a script that in turn calls the IRAF task `getcatalog` to retrieve coordinates and magnitudes from the 2MASS catalog using a web service hosted at NOAO.

The task starts with a small matching radius between the NEWFIRM and 2MASS catalogs, and increases this up to a maximum specified by the `search` parameter (300 arcsec by default). Typically, the offset between the TCS coordinates and the actual pointing will be similar from one exposure to the next over a significant stretch of time, e.g., within a given dither sequence. When `nfwcs` is run on an image, it records the offsets in the parameters `raoffset` and `decoffset`. If these are not reset (`reset = no`), then they are applied the next time the task is run on another image, which can speed the searching/matching process.

Magnitude information from the 2MASS catalog may also be used to derive an approximate photometric zeropoint for an image. This is done quite simplistically by comparing the NEWFIRM instrumental magnitudes for sources from the `ace` catalog to one of the magnitudes from the `2MASS` catalog, without applying any color terms or other transformations. The third column specified in the `cdef` keyword tells which `2MASS` magnitude to use. You should set this to `c3` for J, `c4` for H, or `c5` for K_s. The zeropoint derived for each image extension is written into the `MAGZERO` extension header keyword.

Here, we show a portion of the output that is generated when running `nfwcs` with its default parameters on an image, illustrating the way in which it matches 2MASS sources to objects detected in the NEWFIRM image, updates the world coordinate solution, and computes an approximate magnitude zeropoint:

```
newfirm> nfwcs fobj_n03.28070.fits
nftwomass n03.28070 nfwcs7791bc_coords.txt search=300.
fobj_n03.28070_ss1[im1]: nfwcs7791bc_cat.fits[im1]
fobj_n03.28070_ss1[im2]: nfwcs7791bc_cat.fits[im2]
fobj_n03.28070_ss1[im3]: nfwcs7791bc_cat.fits[im3]
fobj_n03.28070_ss1[im4]: nfwcs7791bc_cat.fits[im4]
ACEMATCH: NOAO/IRAF V2.14EXPORT med@dappcj43.extra.cea.fr Mon 14:32:04 25-May-2009
  match nfwcs7791bc_cat.fits[im1] and nfwcs7791bc_coords.txt
  match nfwcs7791bc_cat.fits[im2] and nfwcs7791bc_coords.txt
  match nfwcs7791bc_cat.fits[im3] and nfwcs7791bc_coords.txt
  match nfwcs7791bc_cat.fits[im4] and nfwcs7791bc_coords.txt
  Search 15.814 arcsec ...
  Search 31.628 arcsec ...
  Search 47.442 arcsec ...
    matched 54 out of 357 detections and 54 references
    matched 65 out of 145 detections and 66 references
    matched 46 out of 136 detections and 46 references
    matched 65 out of 167 detections and 65 references
    total matched 230 out of 805 detections and 231 references
    ra shift is -36.11 arcsec
    dec shift is 15.91 arcsec
    rotation is 0.00 deg
    tangent point is 14:20:15.89 53:00:17.6
  find magnitude zeropoint
      nfwcs7791bc_cat.fits[im1]: magzero = 26.640 +- 0.050
      nfwcs7791bc_cat.fits[im2]: magzero = 26.817 +- 0.021
      nfwcs7791bc_cat.fits[im3]: magzero = 26.802 +- 0.017
      nfwcs7791bc_cat.fits[im4]: magzero = 26.836 +- 0.017
    magzero = 26.789 +- 0.011
  write nfwcs7791bc_mcat.fits[im1]
  write nfwcs7791bc_mcat.fits[im2]
  write nfwcs7791bc_mcat.fits[im3]
  write nfwcs7791bc_mcat.fits[im4]
```

When using `nfwcs` to derive a new astrometric solution, the main failure mode is not being able to determine the TCS offset because it is too large. The first thing you might try is using a larger search radius. This will make the task work harder and slower but this may only be needed for one exposure after which the offsets will apply to subsequent exposures.

If this fails you can use the reference coordinate file, saved if you specified an `outroot` parameter value, with the task `msczero` to interactively determine the offset. Be sure to not apply the offset when quiting the task. Instead put the offsets into the parameters `nfwcs.raoffset` and `nfwcs.decoffset`.

Another common failure occurs when an exposure has few or no detected sources because of clouds. The only sensible solution in this case is to exclude these exposures as unusable. But identifying these and excluding may be more trouble than simply letting the astrometric calibration fail. This is fine though you would likely want to display those which failed to check that this is why the calibration failed.

5.10 Re-projection

With an accurate WCS now encoded in our images, and the background accurately subtracted, we now project each MEF image onto an undistorted celestial tangent plane, combining the four extensions into a single image. For all images that we intend to stack together, We wish to project all overlapping images onto a common pixel grid, with the same tangent point, pixel scale and orientation, in order to make it simple to coadd them into a combined stack.

We use the `msctools` task `mscimage`, which is essentially identical to that which may be familiar to MOSAIC observers from the `mscred` package. There are many parameters, and you should consult the `mscimage` help pages for a detailed explanation. Here is one example of `mscimage` parameters that we will use for our example:

```
PACKAGE = msctools
   TASK = mscimage

input   =                       List of input mosaic exposures
output  =                       List of output images
(format =                image) Output format (image|mef)
(pixmask=                  yes) Create pixel mask?
(verbose=                  yes) Verbose output?

                                # Output WCS parameters
(wcssour=           parameters) Output WCS source (image|parameters|match)
(referen=                     ) Reference image
(ra     =                14.34) RA of tangent point (hours)
(dec    =                53.00) DEC of tangent point (degrees)
(scale  =                  0.4) Scale (arcsec/pixel)
(rotatio=                  0.0) Rotation of DEC from N to E (degrees)

                                # Resampling parmeters
(blank  =                   0.) Blank value
(interpo=               linear) Interpolant for data
(minterp=               linear) Interpolant for mask
(boundar=              reflect) Boundary extension
(constan=                   0.) Constant boundary extension value
(fluxcon=                   no) Preserve flux per unit area?
(ntrim  =                    0) Edge trim in each extension
(nxblock=                INDEF) X dimension of working block size in pixels
(nyblock=                INDEF) Y dimension of working block size in pixels
```

```
                                # Geometric mapping parameters
(interac=                    no) Fit mapping interactively?
(nx     =                    10) Number of x grid points
(ny     =                    20) Number of y grid points
(fitgeom=               general) Fitting geometry
(xxorder=                     4) Order of x fit in x
(xyorder=                     4) Order of x fit in y
(xxterms=                  half) X fit cross terms type
(yxorder=                     4) Order of y fit in x
(yyorder=                     4) Order of y fit in y
(yxterms=                  half) Y fit cross terms type
```

The parameter `format = image` specifies that we will stitch the four NEWFIRM extensions together into a single (non-MEF) image. By setting `pixmask = yes`, we re-project not only the science images, but also their bad pixel masks, specified by the header `BPM` keyword. The output masks have the root names of the input images, with the suffix `_bpm` appended, and will be in `.pl` format.[8] The output masks will be used to identify bad pixels from the input BPMs, as well as the gaps between the detectors and any regions beyond the detector boundaries. The parameter `ntrim` may be used to trim extra pixels around the image boundaries. Since we have sky-subtracted our data, and because the bad pixel masks should include any bad regions near the detector edges, then we may probably leave `ntrim = 0`.

We may specify the output WCS parameters (specifically, the tangent point, pixel scale and image orientation) in a number of ways, depending on the `wcssource` parameter. For `wcssource = image`, an image (specified by the `reference` parameter) is used to define the output WCS. If `reference` is left blank, the first image from the `input` list is used. The tangent point, scale and orientation may also be specified explicitly with the parameters `ra`, `dec`, `scale` and `rotation`.

The next group of parameters controls the interpolation. As with most IRAF tasks, there are many choices for interpolation functions. Higher order functions like `spline3` or `sinc` will better preserve the image noise properties, but may be subject to ringing from sharp features such as bad pixel artifacts. This is one important reason why we typically interpolate or "fixpix" over bad pixels early in the processing, even if we later plan to use the bad pixel masks to exclude them in the stacking. The help pages for `mscimage` give a detailed discussion about the interpolation options and how to choose them.

The parameter `fluxconserve` should be set to `no` here, for reasons discussed in §4.6. Our images have been flat fielded to constant surface brightness. When we re-project, we wish to preserve this, and the reprojected images will now have constant pixel solid angle and thus a uniform flux scaling over the whole field of view.

With the task parameters set as shown above, we run `mscimage`:

```
newfirm> mscimage @List.fobj_ss1 r//@List.fobj_ss1
WCS reference image is fobj_n03.28060_ss1.fits[im1]
```

[8]The `mscimage` task has not yet been written to take advantage of the FITS-encapsulated compressed mask format used by some other `newfirm` tasks.

```
Output WCS parameters:
    RA=14:20:24.00, DEC=53:00:00.0, SCALE=0.4 arcsec/pixel, ROTATION=0. degrees
Resampling fobj_n03.28060_ss1.fits[im1] ...
Resampling fobj_n03.28060_ss1.fits[im2] ...
Resampling fobj_n03.28060_ss1.fits[im3] ...
Resampling fobj_n03.28060_ss1.fits[im4] ...
Creating image rfobj_n03.28060_ss1.fits ...
Resampling fobj_n03.28061_ss1.fits[im1] ...
Resampling fobj_n03.28061_ss1.fits[im2] ...
Resampling fobj_n03.28061_ss1.fits[im3] ...
Resampling fobj_n03.28061_ss1.fits[im4] ...
Creating image rfobj_n03.28061_ss1.fits ...
```

...etc...
This will create the re-sampled images as well as the re-sampled bad pixel maps.

```
newfirm> files rfobj_*ss1*
rfobj_n03.28060_ss1.fits
rfobj_n03.28060_ss1_bpm.pl
rfobj_n03.28061_ss1.fits
rfobj_n03.28061_ss1_bpm.pl
rfobj_n03.28062_ss1.fits
rfobj_n03.28062_ss1_bpm.pl
```

The new bad pixel map filenames are stored in the `BPM` header keyword of the re-sampled images.

5.11 Stacking

We now combine our reprojected images into a stack, using the ordinary IRAF `imcombine` task. Most `imcombine` parameters are familiar to habitual IRAF users, and you may consult the help pages for details.

There are several important topics which are common to the reduction of any imaging data and which we will *not* discuss in detail in this edition of the *NEWFIRM Data Reduction Guide*. These include:

- Identification and exclusion of bad images from the stack
- Rejection of pixel outliers, such as cosmic rays, unmasked pixel defects, satellite and asteroid trails, etc.
- Intensity rescaling, e.g., for photometric transparency variations
- Weighting based on data quality parameters to optimize the signal-to-noise of the stacked images

Identification and exclusion of bad images from the stack: Some images are better than others. In particular, data taken in poor weather conditions can have highly variable quality as the sky brightness, transparency, and seeing vary. Infrared data in particular tend to suffer badly under moderately cloudy conditions: the sky subtraction can become quite poor and non-uniform, and artifacts may become prominent. You may want to screen your data carefully and eliminate bad data from the stacks.

Rejection of pixel outliers: If your bad pixel mask is good, you will exclude most of the static detector defects, but images are also subject to transient artifacts such as cosmic rays, satellite and asteroid trails, etc. There are many different strategies for removing these, including sigma-clipping or other rejection schemes during image combination, or spatial filtering (e.g., to reject sharp features such as cosmic rays). We will not discuss these here; your choice may depend on the nature of your targets, your science goals, etc. The NEWFIRM Science Pipeline uses a combination of several different strategies for outlier rejection in a multi-stage process. We hope to describe this process in some detail in a future version of this *Guide*, and show how it may be implemented using IRAF tasks.

Intensity rescaling for transparency variations: If observing conditions were not photometric, then you may need to multiplicatively adjust the flux scaling of images before combining them into a stack, in order to ensure photometric uniformity over the image mosaic. In addition, you may wish to adjust the photometric zeropoint scaling of each of the four NEWFIRM detectors. In principle, the flat-fielding procedure described in §5.2 should have taken out the mean response differences between the different arrays, putting them onto the same photometric scale. In practice, however, small differences may remain. The NEWFIRM Science Pipeline measures photometry for 2MASS stars in each image to set the photometric zeropoint, and thus could be done for each quadrant as well. Care must be taken to avoid using stars that are saturated or where photometry may be contaminated by bad pixels or other defects. The saturation and bad pixel masks discussed in §5.5 can be useful for this purpose. The `nfwcs` task (see §5.9) makes a rough estimate of `MAGZERO` for each image extension based on 2MASS stars, but you may wish to do this more carefully before combining your data. If you measure variations in the zeropoints from image to image, these can then be applied using the `imcombine` parameter `scale`.

Weighting during image combination: You may also wish to weight your images based on the data quality in order to optimize the signal-to-noise ratio of the stacked image. This can be particularly important if observing conditions (such as transparency, sky brightness or seeing) were highly variable. There is no unique recipe for optimal weighting images, although different algorithms may be defined based on the science goals of the program. The `imcombine` parameter `weight` provides a straightforward way to apply weights to images when combining them.

The images that we re-sampled in §5.10 should all have been projected onto the same world coordinate grid so that their pixels should line up modulo integer shifts.[9] that specify the axis orientations and pixel scales. The image sizes `NAXIS1, NAXIS2` may be different, and the pixel position of the tangent point, `CRPIX1, CRPIX2`, will generally be different for each image, although they should all be offset from one another by integer amounts. This can be accomplished easily in `imcombine` using the parameter `offsets = wcs`. Here is an example of `imcombine` parameters set for making a simple median of a list of reprojected images, using their associated bad pixel maps with no other rejection, scaling or weighting:

[9]I.e., the reprojected images have the same tangent point `CRVAL1,2` and the same CD-matrix header keywords `CD1_1`, CD1_2, CD2_1, CD2_2

```
PACKAGE = immatch
   TASK = imcombine

input   =        @List.rfobj.29  List of images to combine
output  =                Stack1  List of output images
(headers=                      ) List of header files (optional)
(bpmasks=                      ) List of bad pixel masks (optional)
(rejmask=                      ) List of rejection masks (optional)
(nrejmas=                      ) List of number rejected masks (optional)
(expmask=            Stack1_exp) List of exposure masks (optional)
(sigmas =                      ) List of sigma images (optional)
(imcmb  =                    $I) Keyword for IMCMB keywords
(logfile=                STDOUT) Log file

(combine=                median) Type of combine operation
(reject =                  none) Type of rejection
(project=                    no) Project highest dimension of input images?
(outtype=                  real) Output image pixel datatype
(outlimi=                      ) Output limits (x1 x2 y1 y2 ...)
(offsets=                   wcs) Input image offsets
(masktyp=             goodvalue) Mask type
(maskval=                     0) Mask value
(blank  =                    0.) Value if there are no pixels

(scale  =                  none) Image scaling
(zero   =                  none) Image zero point offset
(weight =                  none) Image weights
(statsec=                      ) Image section for computing statistics
(expname=                      ) Image header exposure time keyword

(lthresh=                 INDEF) Lower threshold
(hthresh=                 INDEF) Upper threshold
(nlow   =                     1) minmax: Number of low pixels to reject
(nhigh  =                     1) minmax: Number of high pixels to reject
(nkeep  =                     1) Minimum to keep (pos) or maximum to reject (neg)
(mclip  =                   yes) Use median in sigma clipping algorithms?
(lsigma =                    3.) Lower sigma clipping factor
(hsigma =                    3.) Upper sigma clipping factor
(rdnoise=                    0.) ccdclip: CCD readout noise (electrons)
(gain   =                    1.) ccdclip: CCD gain (electrons/DN)
(snoise =                    0.) ccdclip: Sensitivity noise (fraction)
(sigscal=                   0.1) Tolerance for sigma clipping scaling corrections
(pclip  =                  -0.5) pclip: Percentile clipping parameter
(grow   =                    0.) Radius (pixels) for neighbor rejection
(mode   =                    ql)
```

The parameter `masktype = goodvalue` and `maskvalue = 0` tells `imcombine` to use the reprojected bad pixel masks that are identified by the `BPM` header keywords in the images from the `input` list, and to only combine data from pixels with BPM mask values of 0. Other options (e.g., masking based on bit values) are described in the `imcombine` help pages. If you have a different mask that you want to use instead, you can put its name into a different header keyword, and then specify that keyword name explicitly as `masktype = !<keyword> [goodvalue|badvalue|novalue|goodbits|badbits]`.

In addition to our `output` image, we also create an exposure map specified by the `expmask` parameter. This will be a `pl` format mask with integer values that represent the exposure time in seconds for each pixel in the stacked image, based on the number of (unmasked) images that contribute to each pixel and their summed exposure times (from the `EXPTIME` header keywords of the `input` images).

Running `imcombine` with verbose output, we see the list of images and masks that are used, and the integer pixel offsets that are applied:

```
ecl> imcombine @List.rfobj.29 Stack1 expmask=Stack1_exp

Jun  1  9:24: IMCOMBINE
  combine = median, scale = none, zero = none, weight = none
  blank = 0.
  masktype = goodval, maskval = 0
                Images    Offsets Maskfile
  rfobj_n03.28058_ss3.fits   70  319 rfobj_n03.28058_ss3_bpm
  rfobj_n03.28059_ss3.fits   42   15 rfobj_n03.28059_ss3_bpm
  rfobj_n03.28060_ss3.fits  367  314 rfobj_n03.28060_ss3_bpm
  rfobj_n03.28061_ss3.fits  217  216 rfobj_n03.28061_ss3_bpm
  rfobj_n03.28062_ss3.fits  131  125 rfobj_n03.28062_ss3_bpm
  rfobj_n03.28063_ss3.fits  270  360 rfobj_n03.28063_ss3_bpm
  rfobj_n03.28064_ss3.fits  194   19 rfobj_n03.28064_ss3_bpm
  rfobj_n03.28065_ss3.fits    0  121 rfobj_n03.28065_ss3_bpm
  rfobj_n03.28066_ss3.fits   26  254 rfobj_n03.28066_ss3_bpm
  rfobj_n03.28067_ss3.fits  116  361 rfobj_n03.28067_ss3_bpm
  rfobj_n03.28068_ss3.fits  369  110 rfobj_n03.28068_ss3_bpm
  rfobj_n03.28069_ss3.fits  333  268 rfobj_n03.28069_ss3_bpm
  rfobj_n03.28070_ss3.fits  230  194 rfobj_n03.28070_ss3_bpm
  rfobj_n03.28071_ss3.fits  140  262 rfobj_n03.28071_ss3_bpm
  rfobj_n03.28072_ss3.fits  292  344 rfobj_n03.28072_ss3_bpm
  rfobj_n03.28073_ss3.fits  277  120 rfobj_n03.28073_ss3_bpm
  rfobj_n03.28074_ss3.fits  280  223 rfobj_n03.28074_ss3_bpm
  rfobj_n03.28075_ss3.fits  230    4 rfobj_n03.28075_ss3_bpm
  rfobj_n03.28076_ss3.fits    3   68 rfobj_n03.28076_ss3_bpm
  rfobj_n03.28077_ss3.fits    2  232 rfobj_n03.28077_ss3_bpm
  rfobj_n03.28078_ss3.fits  101  367 rfobj_n03.28078_ss3_bpm
  rfobj_n03.28079_ss3.fits  151  121 rfobj_n03.28079_ss3_bpm
  rfobj_n03.28080_ss3.fits   99    0 rfobj_n03.28080_ss3_bpm
  rfobj_n03.28081_ss3.fits   98  236 rfobj_n03.28081_ss3_bpm
  rfobj_n03.28082_ss3.fits  326   81 rfobj_n03.28082_ss3_bpm
  rfobj_n03.28083_ss3.fits  234  211 rfobj_n03.28083_ss3_bpm
  rfobj_n03.28084_ss3.fits  263  325 rfobj_n03.28084_ss3_bpm
  rfobj_n03.28085_ss3.fits  168  267 rfobj_n03.28085_ss3_bpm
  rfobj_n03.28086_ss3.fits   18  322 rfobj_n03.28086_ss3_bpm

  Output image = Stack1, ncombine = 116
  Exposure mask = Stack1_exp.pl
```

Note that the output log indicates "`ncombine = 116`" (and this will also be recorded in the header keyword `NCOMBINE` in the stacked image), even though in this example 29 images were actually provided in the input list. This is because each reprojected image produced by `mscimage` had `NCOMBINE = 4`, reflecting the fact that the four image extensions (i.e., the four NEWFIRM arrays) were combined into a single output image, leading to $4 \times 29 = 116$ for the final `NCOMBINE` value. This is generally harmless, but we note it here to avoid possible confusion.

The exposure map that is produced by `imcombine` is in the pixel list mask format (`Stack1_exp.pl`). Although not necessary, it may be convenient later to have mask this encapsulated as a FITS file (e.g., making it possible to display it from the command line in ds9):

```
ecl> imcopy Stack1_exp.pl Stack1_exp.fits[extname\=pl,type\=mask] ver+
Stack1_exp.pl -> Stack1_exp.fits[extname=pl,type=mask]
```

If you are not rescaling images (`scale` parameter), and if all images from the `input` list had the same `EXPTIME` (as is typical for NEWFIRM observing sequences), then from the point of view of the photometric scaling, the effective exposure time of the stacked image will be the same as to that of the input images, although the *actual* exposure time varies from place to place over the dithered mosaic stack, as reflected by the `expmask`. The `EXPTIME` in the stacked image header should be the same as that from the images in the `input` list. You may wish to rescale your stacked image to some fiducial exposure time like 1 second by dividing by the original per-image `EXPTIME`. NEWFIRM Science Pipeline data products are rescaled to data values of ADU per second.

5.12 Second–pass processing

The "running median" approach to sky subtraction that we used in §5.7 may still be affected by the presence of objects in the science images that were used to construct the sky frames. Although the median is more robust against outlying pixel values than is, say, the average, the presence of objects can still bias the median values in the sky images. This can lead to "mottling" in the background of sky-subtracted images, and sometimes to subtle negative "shadows" around objects, depending on the nature of the dithering pattern that was used for the observations.

For careful data reduction, we wish to mask objects from images when constructing the sky frames. This can be done in a number of ways. Relatively bright objects can be detected in individual exposures, and we could generate masks and exclude them during the first pass of sky subtraction. Here, we take advantage of the fact that we have now created a deeper first-pass mosaic (§5.11) from our dithered images, and we use this to detect objects to fainter isophotal levels than would be possible in the individual exposures. Our first-pass mosaic should also be quite clean of image artifacts, thanks to the bad pixel masking and the use of median combination in the first-pass stacking.

Our second–pass processing will consist of several steps:

1. Make an object mask from the first-pass stack

2. De-project the object mask to the pixel frame of the original images
3. Rerun sky subtraction, masking sources when constructing sky frames
4. Rerun the WCS calibration, re-projection and stacking.

5.12.1 Making an object mask

There are many ways in which one might create on object mask from the first-pass stack. and we describe only one here, using the task `acesegment` from the `ace` source cataloging package. We wish to detect objects and mask that exceed the background noise by some amount, setting this threshold low enough to detect as many real objects as possible without contamination from image noise. The task `acesegment` carries out a series of steps needed to detect sources in an image, producing a “segmentation map” that identifies pixels associated with objects. The processing steps include measurement and subtraction of a local background (arguably unnecessary here, since our stacked image should have been accurately sky subtracted), convolution by a detection kernel to reduce local pixel-to-pixel noise and enhance the detectability of extended objects, thresholding at a specified noise level, splitting of potentially blended sources (not strictly necessary for the simple goal of creating an object mask, but important for scientific catalogs), and finally growing the area of each object in the segmentation mask in order to include additional light in the “wings” that may fall below the detection threshold.

The default parameters for `acesegment` are shown here:

```
PACKAGE = ace
   TASK = acesegment

images  =                       List of input images
(masks  =                 !BPM) List of bad pixel masks
(exps   =                     ) List of exposure maps
(extname=                     ) Extension names
(logfile=                     ) List of log files
(verbose=                    2) Verbose level

                                # Object masks
(objmask=                +_obm) List of object masks
(omtype =                  all) Object mask type

                                # Sky
(skyotyp=                  sky) Output sky type (sky|subsky)
(skyimag=                +_sky) List of output sky images
(sigimag=                     ) List of output sigma images
(skies  =             +_skymap) List of input/output sky maps
(sigmas =                     ) List of input/output sigma maps
(fitstep=                  100) Line step for sky sampling
(fitblk1=                   10) Block average for line fitting
(fithcli=                   2.) High sky clipping during 1D sky estimation
(fitlcli=                   3.) Low sky clippling during 1D sky estimation
(fitxord=                    1) Sky fitting x order
```

```
(fityord=                    1) Sky fitting y order
(fitxter=                 half) Sky fitting cross terms
(blkstep=                    1) Line step for sky sampling
(blksize=                  -10) Block size (+=pixels, -=blocks)
(blknsub=                    2) Number of subblocks per axis

                                # Detection
(hdetect=                  yes) Detect objects above sky?
(ldetect=                   no) Detect objects below sky?
(updates=                  yes) Update sky after detection, splitting, and growing?
(bpdetec=                1-100) Input bad pixel values in image
(bpflag =                1-100) Input bad pixel values to flag in sources
(convolv=         bilinear 3 3) Convolution kernel
(hsigma =                   3.) Sigma threshold above sky
(lsigma =                  10.) Sigma threshold below sky
(neighbo=                    8) Neighbor type
(minpix =                    6) Minimum number of pixels in detected objects
(sigavg =                   4.) Sigma of mean flux cutoff
(sigmax =                   4.) Sigma of maximum pixel
(bpval  =                INDEF) Output bad pixel value

                                # Splitting
(splitst=                  0.4) Splitting steps in convolved sigma
(splitma=                INDEF) Maximum sigma above sky for splitting
(splitth=                   5.) Splitting threshold in sigma
(sminpix=                    8) Minimum number of pixels in split objects
(ssigavg=                  10.) Sigma of mean flux cutoff
(ssigmax=                   5.) Sigma of maximum pixel

                                # Growing
(ngrow  =                    2) Number of grow rings
(agrow  =                   2.) Area grow factor
(mode   =                   ql)
```

We will not discuss the task or all of its parameters in detail here, since we are not aiming to make a catalog with scientific value, but simply a mask that can be used to identify pixels associated with objects. More details about this task and others in the `ace` package may be found in *ACE: The IRAF Astronomical Cataloging Environment* (Valdes, 2008), a users guide to the software. For the moment, we will leave most parameters at their default values, with a few exceptions.

The name of the output object mask is specified by the parameter `objmask = +_obm`, whose default value indicates that the suffix `_obm` will be appended to the filename of the input image.

The exposure time and hence the pixel noise level varies over stacked mosaic image because of the dithering, masking, etc. To first order, the variance of the local pixel noise in the mosaics should vary as the reciprocal of the local exposure time, so we use the parameter `exps` to specify the mosaic exposure map.

Several parameters control the detection of objects and determine which pixels will become part of the object segmentation mask. The image will be convolved by a smoothing kernel in order to suppress pixel-to-pixel noise and enhance the detectability of objects. This is specified by the parameter `convolve`, whose default is a 3×3 bilinear kernel (see output example below). Larger kernels will lead to greater noise suppression, but may also blur compact objects reducing their contrast. The parameter `hsigma` specifies the positive detection threshold in units of the sky RMS. Negative thresholds can be specified as well with `lsigma`, but here we leave `ldetect = no` to disable detection of negative sources. The value of `minpix` sets the minimum size (in connected pixels) of detected objects, which helps to reduce spurious detection of noise spikes or pixel defects. The parameters `ngrow` and `agrow` are used to extend the masks to cover fainter emission from the wings of objects. Masks may be grown linearly (by adding `ngrow` rings of pixels around them) or in area (increasing their area by the factor `agrow`), or both.

Running `acesegment` we see this output:

```
ace> acesegment Stack1.fits exps=Stack1_exp.fits[pl]
ACE: NOAO/IRAF V2.14.1 med@Django.local Mon 09:45:48 01-Jun-2009
  Image: Stack1.fits - EGS FIELD3 Jband 60
Warning: Can’t open map (Stack1_exp.fits)
ace> acesegment Stack1.fits exps=Stack1_exp.fits[pl]
ACE: NOAO/IRAF V2.14.1 med@Django.local Mon 09:45:51 01-Jun-2009
  Image: Stack1.fits - EGS FIELD3 Jband 60
      Determine sky and sigma by surface fits:
      start line = 51, end line = 4468, step = 100.4
      xorder = 1, yorder = 1, xterms = half
      hclip = 2., lclip = 3.
    Determine sky and sigma by block statistics:
      Number of blocks: 10 10
      Number of pixels per block: 450 450
      Number of subblocks: 20 20
      Number of pixels per subblock: 225 225
  Detect objects:
    Convolution:
         1.      2.      1.
         2.      4.      2.
         1.      2.      1.
    13315 objects detected
  Split objects: sminpix = 8
  Grow objects: ngrow = 2, agrow = 2.
      Determine sky and sigma by surface fits:
      start line = 51, end line = 4468, step = 100.4
      xorder = 1, yorder = 1, xterms = half
      hclip = 2., lclip = 3.
    Determine sky and sigma by block statistics:
      Number of blocks: 10 10
      Number of pixels per block: 450 450
      Number of subblocks: 20 20
      Number of pixels per subblock: 225 225
    Write sky map: Stack1_skymap
  Write object mask: Stack1_obm[pl]
  Output sky images: sky = Stack1_sky
```

The resulting object mask is `Stack1_obm.fits`, which is a FITS-encapsulated mask file (like the FITS-encapsulated version of the exposure mask that we created in §5.11):

```
ace> imhead Stack1.fits
Stack1.fits[4518,4518][real]: EGS FIELD3 Jband 60
ace> imhead Stack1_obm.fits[pl]
Stack1_obm.fits[pl][4518,4518][int]: EGS FIELD3 Jband 60
```

You should examine the object mask that is generated by `acesegment` and compare it to the stacked image itself, to make sure that the mask is reasonable does not contain excessive numbers of spurious sources:

```
newfirm> display Stack1.fits 1
z1=-23.186 z2=30.62507
newfirm> display Stack1_obm.fits[pl] 2 zs- zr+
z1=0. z2=1323587.
newfirm> display Stack1_obm.fits[pl] 3 zs- zr- z1=0 z2=1
z1=0. z2=1.
newfirm> display Stack1.fits 4 overlay=Stack1_obm.fits[pl]
z1=-23.186 z2=30.62507
```

The segmentation map includes very high values that outline the boundaries of objects, which can be viewed with the second `display` command above. The "interior" pixels of each object are labeled by the object catalog number, which does not concern us here; we can view all pixels flagged as part of objects with either the third or fourth `display` commands above. The fourth uses the mask `overlay` feature from `display`.

If the mask looks reasonable, then you may proceed. If it seems to have many spurious sources, you can experiment with changing various `acesegment` parameters, e.g., by raising the detection threshold `hsigma`.

5.12.2 De-projecting the object mask

Next, we need to "de-project" the object mask from the coordinate frame of the first-pass stack back to the pixel coordinates of the original NEWFIRM images. We use `mscimage` again for this purpose, using world coordinate transformations to map from the rectified tangent plane projection of the first-pass stack back to the pixel planes of the individual, geometrically distorted NEWFIRM images. Unfortunately, in this case, the current design of `mscimage` does not let us carry out this procedure with a single command, as was the case for the "forward" projection described in §5.10. Instead, we must make some file lists and execute a series of commands.

First, we prepare the object mask. The object mask from `acesegment` has value 0 where there are no objects, and non-zero where there are objects. The process of interpolation when re-projecting with `mscimage` will change the mask pixel values, which themselves are not important, but we wish to have some control over this process. We make a copy of the object mask, and use `imreplace` to set its pixels to values of 0 and 1000:

```
msctools> copy Stack1_obm.fits Stack1_obm1000.fits
msctools> imreplace Stack1_obm1000.fits[pl] 1000 low=1 up=INDEF
```

Next, we need to prepare some file lists. In our working example here, the WCS-calibrated, sky subtracted images produced in §5.9 had names like `fobj_n03.28070_ss1.fits`. We need the following lists:

1. object mask MEF FITS files for each NEWFIRM image:
   `obm_n03.28070.fits`
   `obm_n03.28071.fits`
   etc.
2. all extensions from the sky subtracted files:
   `fobj_n03.28070_ss1.fits[im1]`
   `fobj_n03.28070_ss1.fits[im2]`
   etc.
3. all extensions from the mask files, including the flags needed to create MEF masks:
   `obm_n03.28070.fits[im1,append,type=mask]`
   `obm_n03.28070.fits[im2,append,type=mask]`
   etc.[10]
4. temporary FITS masks for each extension:
   `obm_n03.28070_1.fits`
   `obm_n03.28070_2.fits`
   etc.

You can create these lists with a text editor or however you wish. Here, we make them using the Unix `sed` commands, run within IRAF by using ! to escape to the Unix shell:

```
msctools> files fobj_n03.*_ss1.fits > List.fobj_ss1

msctools> !sed 's/_ss1//' List.fobj_ss1 | sed 's/fobj/obm/' > List.obm

msctools> !sed 's/fits/fits[im1]/' List.fobj_ss1 >  List.fobj_ss1_4ex
msctools> !sed 's/fits/fits[im2]/' List.fobj_ss1 >> List.fobj_ss1_4ex
msctools> !sed 's/fits/fits[im3]/' List.fobj_ss1 >> List.fobj_ss1_4ex
msctools> !sed 's/fits/fits[im4]/' List.fobj_ss1 >> List.fobj_ss1_4ex

msctools> !sed 's/fits/fits[im1,append,type=mask]/' List.obm >  List.obm_4ex
msctools> !sed 's/fits/fits[im2,append,type=mask]/' List.obm >> List.obm_4ex
msctools> !sed 's/fits/fits[im3,append,type=mask]/' List.obm >> List.obm_4ex
msctools> !sed 's/fits/fits[im4,append,type=mask]/' List.obm >> List.obm_4ex

msctools> !sed 's/.fits/_1.fits/' List.obm >  List.obm_tmp
msctools> !sed 's/.fits/_2.fits/' List.obm >> List.obm_tmp
msctools> !sed 's/.fits/_3.fits/' List.obm >> List.obm_tmp
msctools> !sed 's/.fits/_4.fits/' List.obm >> List.obm_tmp
```

[10]Because we will be using these filenames as input to an IRAF task from an @list, we do not need a backslash before the = in `type=mask`. Go figure.

Next, we set a few key parameters of `mscimage`:

```
PACKAGE = msctools
   TASK = mscimage

(format =                image) Output format (image|mef)
(pixmask=                   no) Create pixel mask?
(verbose=                  yes) Verbose output?

                                # Output WCS parameters
(wcssour=                match) Output WCS source (image|parameters|match)

                                # Resampling parmeters
(blank  =                   0.) Blank value
(interpo=               linear) Interpolant for data
(boundar=             constant) Boundary extension
(constan=                   0.) Constant boundary extension value
(fluxcon=                   no) Preserve flux per unit area?
(ntrim  =                    0) Edge trim in each extension
```

The parameter `wcssource = match` forces the `output` image to have the same pixel dimensions (as well as the same WCS) as the `reference` image. We use linear interpolation and no flux conservation. We now must write a script that executes `mscimage` to transform the first-pass stack mask (`input`) to the coordinate frame of each extension of the WCS-calibrated NEWFIRM images (`reference`), writing out the result as a temporary mask image (`output`). I.e., the script should look something like this:

```
mscimage Stack1_obm1000.fits[pl] obm_n03.28058_1.fits reference=fobj_n03.28058_ss1.fits[im1]
mscimage Stack1_obm1000.fits[pl] obm_n03.28059_1.fits reference=fobj_n03.28059_ss1.fits[im1]
...etc...
mscimage Stack1_obm1000.fits[pl] obm_n03.28058_2.fits reference=fobj_n03.28058_ss1.fits[im2]
mscimage Stack1_obm1000.fits[pl] obm_n03.28059_2.fits reference=fobj_n03.28059_ss1.fits[im2]
...etc...
```

We may use Unix commands to write such a script as follows:

```
msctools> !paste List.obm_tmp List.fobj_ss1_4ex > _tmp1
msctools> !awk '{print "mscimage Stack1_obm1000.fits[pl]",$1,"reference="$2}' _tmp1 > do_mscimage_obm.cl
msctools> delete _tmp1 ver-
```

We are now ready to run this script to do the `mscimage` re-projections:

```
msctools> cl < do_mscimage_obm.cl
```

We copy the temporary images into a more permanent home in a FITS-encapsulated mask file:

```
msctools> imcopy @List.obm_tmp @List.obm_4ex ver+
obm_n03.28058_1.fits -> obm_n03.28058.fits[im1,append]
obm_n03.28059_1.fits -> obm_n03.28059.fits[im1,append]
obm_n03.28060_1.fits -> obm_n03.28060.fits[im1,append]
```

...etc...
Finally, we use imreplace to convert the resulting masks to values of 0 or 1. Our original mask had values of 0 and 1000; we will keep any pixel in the resampled masks that has value greater than 100.

```
msctools> msccmd "imreplace $input 0 low=INDEF up=100" @List.obm
msctools> msccmd "imreplace $input 1 low=100 up=INDEF" @List.obm
```

At this point we are done, and we delete the temporary (uncompressed) FITS versions of the masks for each extension, to save some disk space:

```
msctools> delete @List.obm_tmp ver-
```

5.12.3 Second-pass sky subtraction

We now wish to redo the sky subtraction described in §5.7, but this time use our masks to exclude pixels associated with objects when constructing the sky frames. We revisit the parameters for `nfskysub`, now setting the `obm` parameter to point to our list of object masks:

```
PACKAGE = newfirm
   TASK = nfskysub

input   =          @List.fobj  List of input images
(output =               +_sss) List of output images
(outtype=               image) Output type (image|list|<keyword>)
(skies  =                    ) List of skies (empty to use input)
(skymatc=                    ) Match boolean expression
(skymode=          median 9 5) Sky subtraction mode
(stype  =                    ) Sky selection expression (boolean)
(obm    =          @List.obm) Input mask or keyword reference
(logfile=      STDOUT,logfile) List of output logfiles
(fd     =                    )
(mode   =                  ql)
```

Here we use the output prefix `_sss` to distinguish this second pass from our earlier first pass. Now that objects are being explicitly masked, the sky calculation should be more robust against outlying values (most of which come from object pixels that have been masked). We therefore increase the second optional parameter to the `skymode = median` function, which controls the number of data values that are averaged. This is now operating more like `minmax` rejection in `imcombine` with small values for `nlow` and `nhigh` (in this case, 2).

We rerun `nfskysub` on the linearized, flatfielded data (i.e., the `fobj_*` images produced at the end of §5.6), followed by `nfdeltasky` as in §5.8.

5.12.4 Re-stacking the second-pass images

To complete the second pass, first we must redo the world coordinate system recalibration described in §5.9. The WCS will not have changed, but in redoing the sky subtraction we went back to the images from a step before the WCS recalibration, so our newly re-sky-subtracted data do not have the updated WCS. You could copy the relevant image header parameters from the old `fobj_*_ss1` images (§5.8) to the new versions, but it may be simpler and not very time consuming to simply rerun `nfwcs` again to recalibrate the new images.

Finally, we re-project the images using `mscimage` (§5.10), and re-stack (§5.11). In some cases, we may wish to set parameters somewhat differently to generate these "final" data products. For example, when re-projecting, we may wish to set the interpolation algorithm in `mscimage` to some higher-order function like `sinc`. When combining the images with `imcombine`, we may wish to average more data values (`combine = average` rather than `combine = median`), using the various `imcombine` clipping algorithms to reject outliers.

Acknowledgements

The authors thank the other members of the Pipeline Group, a joint effort of NOAO and the University of Maryland, including Rob Swaters and Tracy Huard. We also thank Ron Probst and Mike Merrill for providing extensive information and hands-on guidance concerning NEWFIRM and infrared imaging in general.

References

Autry, R. G., et al. 2003, SPIE, 4841, 525

A Beginner's Guide to Using IRAF. Barnes, J., 1993 (Tucson: NOAO)

Daly, P. N., et al. 2008, SPIE, 7019,

NEWFIRM Linearity Calibration and Correction. Dickinson, M., et al. 2009 (Tucson: NOAO)

Probst, R. G., et al. 2004, SPIE, 5492, 1716

Probst, R. G., George, J. R., Daly, P. N., Don, K., & Ellis, M. 2008, SPIE, 7014,

An Introductory User's Guide to IRAF Scripts. Anderson, E., rev. by Seaman, R., 1989 (Tucson: NOAO)

The NOAO Data Handbook, version 1.1, 2009, Shaw, R. A., editor (Tucson: NOAO)

Swaters, R. A., Valdes, F., & Dickinson, M., 2009, in ADASS XVIII in press (arXiv:0902.1458)

ACE: The IRAF Astronomical Cataloging Environment. Valdes, F., 2008 (Tucson: NOAO)

IRAF FITS Kernel User's Guide. Zarate, N., 1997 (Tucson: NOAO)