Fpack and Funpack User's Guide

W. D. Pence, NASA/GSFC
R. Seaman, NOAO
R. L White, STScI

14 November 2013

1. Introduction

Fpack and funpack[1] are standalone programs for compressing and uncompressing images and tables that are stored in the FITS (Flexible Image Transport System) data format. They are analogous to the gzip and gunzip compression programs except that they are optimized for the types of astronomical images that are often stored in FITS format.

The compressed images that are generated by fpack are stored in a FITS binary table in a format that is defined by the FITS tiled image compression convention (see http://fits.gsfc.nasa.gov/fits_registry.html). This convention allows each image to be divided into a grid of rectangular tiles (usually row-by-row), and then each tile is individually compressed and stored in a variable-length array column in the FITS binary table. FITS binary tables are compressed and stored back into a binary table that has a slightly different structure than the original table (see http://fits.gsfc.nasa.gov/registry/tiletablecompression/tiletable.pdf)

Fpack and funpack are relatively simple wrappers that call routines in the CFITSIO library (http://heasarc.gsfc.nasa.gov/fitsio) to read the input FITS file(s), perform the compression or uncompression, and then write the output FITS files. Currently supported compression algorithms for images are: Rice, GZIP, H-compress, and the PLIO IRAF pixel list compression algorithms. Table columns can be compressed with Rice or GZIP, depending on the column data type. Other algorithms may be added in the future.

Fpack offers a number of advantages over using a general purpose program like gzip of bzip2 to compress the whole FITS file :

1. fpack is optimized for astronomical images and tables and generally achieves higher compression ratios and faster compression speeds.
2. The compressed file produced by fpack is itself a valid FITS file that can be directly manipulated by other general purpose FITS software.
3. Each Header-Data Unit (HDU) of a multi-extension FITS file is compressed separately, so each image or table can be directly accessed without having to uncompress the whole FITS file.
4. Fpack compresses only the data and not the headers in each HDU, so software can read and write the header keywords without any added computational overhead.
5. Dividing the image into tiles before compression enables faster access to small sections of the image because only those tiles covering the area of interest need be uncompressed.
6. Fpack supports lossless compression of both integer and floating point format images, and also supports a new 'lossy' compression technique for floating-point images that produces much higher compression by reducing the amount of uncompressable 'noise' in the image while still preserving the scientific information content of the image.
7. Fpack and Funpack automatically update the CHECKSUM keywords in the compressed and uncompressed files to help verify the integrity of the FITS files.
8. Many software applications (such as ds9 and programs that use CFITSIO to read and write FITS files)

1The fpack and funpack programs were originally developed by Rob Seaman. William Pence added further enhancements and maintains the image compression algorithms in the underlying CFITSIO library. Rick White wrote the Rice and Hcompress algorithms that are used by CFITSIO.

can directly read (and write) the compressed FITS image format, thus eliminating the need to create an uncompressed version of the file.

2. Obtaining and building fpack and funpack

The latest versions of the fpack and funpack C source code are included in the CFITSIO source file distributions available at http://heasarc.gsfc.nasa.gov/fitsio. Binary executables for a few common platforms are also available from http://heasarc.gsfc.nasa.gov/fitsio/fpack. This PDF users guide is available from both locations.

To build the software on linux and Mac-OS systems, first download and build the CFITSIO library by following the instructions in the included Readme file. Then enter the commands

```
make fpack
make funpack
```

in that directory to create the fpack and funpack executable files.

On Windows PCs, one can build the fpack and funpack programs using the Visual C++ compiler with the following command lines (after first building the CFITSIO library following the instructions in the README.win32 file):

```
> cl /MD fpack.c fpackutil.c cfitsio.lib /link setargv.obj
> cl /MD funpack.c fpackutil.c cfitsio.lib /link setargv.obj
```

Note: the "/link setargv.obj" argument is optional. It enables support for wildcard characters when specifying the names of the input files to fpack and funpack.

3. fpack compression options

One can specify how the input FITS images and tables are to be compressed by fpack in 2 ways:

1. The compression options may be specified on the fpack command line. These global options will be used by default when compressing all the input files.
2. 'Compression directive' keywords may be added to the header of any HDU to specify how that HDU is to be compressed. These keywords will override the global options that were specified on the fpack command line and can be used to customize the compression on an HDU by HDU basis.

Each of these methods are described in the following subsections.

3.1 fpack command line syntax

The fpack command line has the following form:

```
fpack [OPTIONS]... [FILES]...
```

Various compression options may be specified prior to the list of files to be compressed. The names of the files to be compressed may contain the usual wild card characters that will be expanded by the Unix or Windows shell. The input FITS file can also be piped into fpack on the `stdin` file stream by specifying a hyphen ('-') as the file name.

A. Compression algorithm options:

-r	Rice [default], or
-h	Hcompress, or
-g or **-g1**	GZIP (per-tile), or

-g2	GZIP (per-tile) after first shuffling the bytes in the pixel values into decreasing order of significance (first byte of every pixel, followed by 2nd byte, etc.)
-p	IRAF pixel list compression algorithm. This can only be applied to images whose pixel values all lie in the range 0 to 2^{24} (16777216).
-d	tiles are not compressed (debugging mode)

B. Image tile pattern options:

When using the Rice, GZIP, or PLIO compression algorithms, the default action is to treat each row of the image as a tile (i.e., the tiles are one dimension and contain NAXIS1 pixels). The Hcompress algorithm is inherently 2-dimensional in nature, therefore the default is to use 16 rows of the image per tile, except if this would cause the last tile of the image to only contain a small number of rows, in which case the tile size is adjusted slightly so that the last tile is more nearly equal in size to the other tiles. The default tile size can be overridden with one of the following command-line options:

-w	compress the whole image as a single large tile
-t <axes>	comma separated list of tile dimensions (e. g., -t 200,100 will produce tiles that are 200 pixels wide (in the NAXIS1 dimension) by 100 pixels high

C. Floating -point image compression options:

Floating-point format FITS images (with BITPIX = -32 or -64) are usually compressed by first linearly scaling the floating-point values into quantized integer levels, and then the integer values are compressed using the Rice algorithm by default. The quantization factor is specific with the -q option:

-q <level> (default = 4.0; value of 0.0 means do not quantize and losslessly compress with GZIP)

Positive q values specify how finely the quantized levels are spaced relative to the measured R.M.S noise in the tile of the image that is being compressed. The default q value of 4.0 means that the spacing between quantized levels is equal to the RMS noise value divided by 4. Larger q values will result in more finely spaced levels (and thus the quantized values more closely match the original floating point values) but this will result in less compression of the image (as shown in Table 2). Refer to the later section on “Compression versus Noise” for a more detailed discussion on the theory and practice of image quantization.

In some cases it may be desirable to specify the exact spacing between the quantized levels (i.e., not relative to the measured noise), for example, so that all the tiles in a particular image, or all the images in a dataset, are quantized by exactly the same factor, This can be done by specifying the negative of the desired q value. In this case, smaller absolute values of q (e.g., -.001 instead of -.01), will more closely preserve the original pixel values at the expense of a lower amount of image compression. The -T option (described below) can be used to estimate the noise level in the image as an aid in estimating an appropriate absolute q value.

Quantization is inherently a 'lossy' compression process because the original pixel values are not exactly preserved. When applied judiciously, no significant scientific information is lost and much higher compression will be achieved. However, one can turn off the quantization process and losslessly compress the floating point values by specifying a q value of 0.0. One must also specify the GZIP compression algorithm in this case (with the -g1 or -g2 option) .

When the floating-point pixels are quantized, a dithering option is also usually applied. This serves to randomize the quantized values and helps to minimize any systematic biases in the measured positions or intensities of objects in the quantized image. By default, all the image pixels are dithered, but 2 other dithering options may be specified:

-qz <level> - only dither the non-zero pixels; the zero values will be exactly preserved
-q0 <level> - do not dither any of the pixel values

If dithering is turned off with -q0,, then the q level should be increased by about a factor of 4 in order to preserve the same level of scientific precision in the image as when dithering is performed.

Finally, the dithering option also requires that an initial random number seed value be defined, with an integer value ranging from 1 to 10000. By default, the initial seed value is randomly chosen based on the current system clock time. This method helps to ensure that the same dithering pattern is not applied to successive image, which would not be desirable when adding or subtracting 2 compressed images (otherwise the benefits of dithering will be lost). Note that when using this method for computing the seed value, the compressed image pixel values will be slightly different each time the image is compressed.

There 2 also other ways to specify random number seed value:

-qt <level> - compute the seed from the checksum of the first tile of image pixels
-qN <level> where N is an integer between 1 and 10000 – use seed N (e.g., -q3010 4).

One can combine these 2 options with the 'z' dithering option by specifying '-qzt' or '-qzN'.

D. Options related to lossy compression of integer images:

-i2f This option (which stands for "integer to float") forces fpack to internally convert images with integer pixel values into floating-point pixels, which are then compressed using the quantization method that is normally used for actual floating-point FITS images. This lossy compression method may achieve higher compression, without significant loss of information, especially for long exposure images that have relatively large number of counts per pixel (and hence a large amount of Poissonian noise). This parameter will be ignored if the amount of noise in the image is less than the absolute and relative noise thresholds set by the -n3min and -n3ratio parameters described below.

-n3min <noise> (default value = 6.) This parameter is used in conjunction with the -i2f parameter to specify the minimum allowed value of the RMS background noise in the image in order for the -i2f parameter to take effect. If the image has less noise, then the -i2f parameter is ignored and the integer image is losslessly compressed.

-n3ratio <ratio> (default value = 2.0) This parameter is used in conjunction with the -i2f parameter to specify the minimum allowed ratio of the RMS background noise in the image divided by the q parameter value. If the image has less noise, then the -i2f parameter is ignored and the integer image is losslessly compressed.

-n <noise> This rarely used parameter (the -i2f parameter offers a better compression method in many cases) rescales the pixel values in a previously scaled image to improve the compression ratio by reducing the R.M.S. noise in the image. This option is intended for use with FITS images that use scaled integers to represent floating point pixel values, and in which the scaling was chosen so that the range of the scaled integer values covers the entire allowed range for that integer data type (e.g., -32768 to +32767 for 16-bit integers and -2147483648 to +2147483647 for 32-bit integers). When images are scaled in this manner, the measured R.M.S. noise in the integer images is typically so large that they cannot be effectively losslessly compressed. This -n option rescales the pixel values so that the R.M.S. noise will be equal to the specified value. Appropriate values of n will likely be in the range from 2 (for low precision and the high compression) to 16 (for the high precision and lower compression). Users should read the section on compressing floating point images, for guidelines on choosing an appropriate value for n that does not lose significant information in the image.

-s <scale> Scale factor for lossy compression when using Hcompress. The default value is 0.0 which implies lossless compression. Positive scale values are interpreted as relative to the R.M.S. noise in the image. For reference, scale values of 1.0, 4.0, and 10.0 will typically produce compression factors of about 4, 10, and 25, respectively, when applied to 16-bit integer images. In some instances it may be desirable to specify the exact scale value (not relative to the measured noise), so that all the tiles in the image, and all the images in a dataset, are compressed with the identical scale value, regardless of slight variations in the measured noise level between tiles. This is done by specifying the negative of the desired value (e.g. -30., which would be equivalent to specifying a scale value of 2.0 in an image that has RMS noise = 15.).

It is important to realize that this option achieves the high compression ratios at the expense of not exactly preserving the original pixel values in the image. Users should carefully evaluate the compressed images (e.g., by uncompressing them with funpack) to make sure that any essential information in the image has not been lost.

E. Parameters for compressing FITS binary table extensions (experimental)

-table This turns on the table compression feature in fpack (images are also compressed)

-tableonly This turns on the table compression feature in fpack and turns off image compression.

F. Parameters that affect the input and output files:

The compressed output file name is usually constructed by appending ".fz" to the input file name, and the input file is not deleted, but this behavior may be modified with the following parameters:

-F force the input file to be overwritten by the compressed file with the same name. This is only allowed when a lossless compression algorithm is used.
-D delete the input file after creating the compressed output file.

-Y suppress the prompts to confirm the -F or -D options
-S output the compressed FITS file to the STDOUT stream (to be piped to another task)

G. Other miscellaneous parameters:

-v verbose mode; list each file as it is processed
-L list all the extensions in all the input, files. No compression is performed.
-C don't update FITS checksum keywords
-H display a summary help file that describes the available fpack options
-V display the program version number
-T produce a report that compares the image compression ratio and the compression and

uncompression times for each of the main compression algorithms. The input file remains unchanged and is not compressed. When used with -BETAtable option, the report shows the compression ratio for each column as well as for the table as a whole. The input file remains unchanged and no compressed output file is produced. The format of this report is shown in the appendix.

-R <filename> save the comparison test report (produced by the -T option) to the named ASCII file.

3.2. funpack command-line parameters

Funpack shares many of the same parameters as fpack as shown below:

A. Parameters that affect the input and output files:

-F	force the input file to be overwritten by the uncompressed file with the same name. This is only allowed when a lossless compression algorithm is used.
-D	delete the input file after creating the compressed output file.
-P <pre>	name of the uncompressed output file is constructed by prepending the <pre> string to the name of the input file
-O <name>	used to specify the full name of the uncompressed output file.
-S	output the uncompressed FITS file to the STDOUT stream (to be piped to another task)
-Z	recompress the unpacked output file with the host gzip program

B. Other miscellaneous funpack parameters:

-v	verbose mode; list each file as it is processed
-L	list all the extensions in all the input, files. No uncompression is performed.
-C	don't update FITS checksum keywords
-H	display a summary help file that describes the available funpack options
-V	display the program version number

3.3 Fpack compression directive keywords

One can specify how each HDU in a FITS file is to be compress by adding "compression directive" keywords to the header of that HDU. The value of these keywords will take precedence over whatever compression parameters where specified on the command line when fpack was executed. Data providers can use these keywords to control the way in which each individual HDU in the file is compressed. The following table lists the allowed values for all the compression directive keywords in image HDUs:

Keyword	Description	Allowed Values
FZALGOR	Compression Algorithm	'RICE_1' (default) 'GZIP_1' 'GZIP_2' - bytes are shuffled in order of decreasing significance before being compressed 'HCOMPRESS_1' 'PLIO_1' 'NONE' – the HDU remains uncompressed
FZTILE	Tiling pattern	'ROW' – row-by-row tile pattern (default) 'WHOLE' - treat entire image as a single tile '(n,m)' – tile dimensions, for example, '(250,100)'
FZQVALUE	Quantization Factor	Float value - default = 4; a value of 0 means do not quantize, and instead losslessly compress the floating-point image (must use GZIP)
FZQMETHD	Quantization Method	'SUBTRACTIVE_DITHER_1' (default) 'SUBTRACTIVE_DITHER_2' – zero-valued pixels are not dithered 'NO_DITHER' – turns off dithering completely
FZDTHRSD	Dithering Seed Value	'CLOCK' – seed is randomly chosen based on the system clock time 'CHECKSUM' – seed is calculated from checksum of the first tile '1' through '10000' – specifies which seed value to use
FZI2F	Convert ints to floats?	T, F – convert integer images into floats and then quantize?
FZHSCALE	Hcompress scale factor	Float value (default = 0.0 = lossless compression)

Compression directive keywords allowed in FITS binary tables:

Keyword	Description	Allowed Values
FZALGOR	Default Compression Algorithm to be applied to every column, if possible	'RICE_1' 'GZIP_1' 'GZIP_2' - bytes are shuffled in order of decreasing significance before being compressed 'NONE' – the HDU remains uncompressed
FZALGn	Compression algorithm for column n of the table. This overrides the FZALGOR value	RICE_1' , 'GZIP_1', or 'GZIP_2' If the data type of the column is incompatible with the specified algorithm, then a suitable default algorithm will be used instead.
FZTILELN	Rows per Tile	Integer number of rows to be compressed as a group

4. Theory and Practice of Image Compression

This section provides background information on the theory and practice of image compression. Most of this material comes from our 2 papers on image compression:

Paper I: Pence, Seaman, and White, 2009, PASP 121,414; preprint: http://arxiv.org/abs/0903.21401

Paper II : Pence, White, & Seaman, PASP 122, 1096 (2010); preprint: http://arxiv.org/abs/1007.1179)

4.1 Compression versus noise

When losslessly compressing typical astronomical images, the amount of compression that can be achieved depends almost completely on one simple factor: the amount of the noise (or more precisely, the entropy) that is present in the pixel values. The noise, by definition, cannot be compressed, so the compression ratio of an image will be inversely proportional to the total number of noise bits in the image. As is discussed in greater detail in Paper I, the amount of noise in an image can be estimated from the measured standard deviation (sigma) of the pixels in the "background" areas of the image (e.g., excluding bright stars of other objects in the image) which typically closely approximates a Gaussian distribution. It can be shown that the average number of noise bits per pixel is given by:

$$\text{Nbits} = \log_2(\text{sigma}) + 1.792$$

Since the noise bits cannot be compressed, the upper limit on the compression ratio, in the ideal case where all the remaining bits are compressed to zero size, is simply given by the ratio BITPIX / Nbits (where BITPIX is the number of bits in each pixel value and is either 8, 16, or 32 in FITS images). Since no actual compression algorithm can achieve this theoretical limit, in practice the compression ratio can be expressed as

$$R = \text{BITPIX} / (\text{Nbits} + K)$$

where K is an empirical measure of the efficiency of the particular compression algorithm. For the Rice algorithm, K has a value of about 1.2, and for Hcompress it is about 0.9. The k value for GZIP is more variable and depends on the number and distribution of different bit patterns in the data, but, typically has a value of about 4 or 5 in typical astronomical images.

The amount of Poissonian noise in astronomical image typically scales with the square root of the number of detected photons. (There may be other sources of noise as well). The practical implication of this fact is that the different types of exposures taken during a astronomical observing session (e.g., bias frames, short calibration exposures, deep sky exposures, and flat field images) all have distinctly different amounts of noise and hence will compress by differing amounts. For example, the bias frames, with very low number of counts, will compress much better than the flat field images or long exposures of the night sky that contain much more noise.

4.2. Lossless compression of integer FITS images.

In Paper I, we examined a large set of direct imaging CCD exposures from NOAO taken of star fields in the night sky, plus the associated calibration exposures, to compare the compression speeds and file compression ratios for the 3 different general purpose compression algorithms that are currently supported by fpack, namely, Rice, GZIP, and Hcompress. We also compared these to the widely-used method of compressing the entire FITS file with the host-level GZIP file compression program.

The mean file compression ratios and the relative compression and uncompression elapsed CPU times for these 4 different compression methods are shown in Table 1. These values are the mean for all 1632 16-bit integer images in the sample data set; the CPU times in each case are relative to those when using the Rice algorithm.

Table 1. Compression Statistics for 16-bit Integer Images

	Rice	**Hcompress**	**GZIP**	**Host GZIP**
Compression Ratio	2.11	2.18	1.53	1.6
Relative compression CPU time	1.0	2.8	5.6	2.6
Relative uncompression CPU time	1.0	3.1	1.9	0.9

As shown in the first row, the Rice and Hcompress methods achieve significantly larger compression of these astronomical images than GZIP.   The GZIP compressed files are on average about 1.4 times larger than the Rice or Hcompressed files. This depends slightly on the amount of noise in the image: the ratio is about 1.3 for the images with the most amount of noise and about 1.5 for the least noisy images. Hcompress produces slightly better compression than Rice (about 3% smaller), but for most applications this small gain is not worth the much greater CPU times required to compress and uncompress the images with Hcompress.

The second and third rows of the table show that the Rice compression algorithm is generally considerably faster than Hcompress or GZIP.   Note that the factor of approximately 2 timing difference between using the host-level GZIP program to compress an image and using the tiled-image implementation of the same GZIP algorithm within fpack/CFITSIO is due to the fact that the host-level GZIP program can read and write the files more efficiently as simple continuous streams, whereas the fpack implementation requires that the input and output files be copied to and from intermediate storage buffers in memory. As a benchmark point of reference, a Linux machine with a 2.4 GHz AMD Opteron 250 dual core processor can compress or uncompress a 50 MB 16-bit integer image in 1 second of CPU time when using the Rice compression algorithm.

Similar trends are seen when compressing 32-bit integer images, only the compression factors that are achieved are typically twice that of a 16-bit image, given the same noise level.   See Paper I for more details.

4.3. Compression of floating-point FITS images

It is generally not efficient (nor necessary) to losslessly compress floating-point format FITS images (which have BITPIX = -32 or -64).   This is because a large fraction of the bits in the mantissa of the image pixel values are often filled with uncompressible noise, which severely reduces the maximum possible file compression ratios. Note that a 32-bit floating-point value can represent 6 – 7 decimal places of precision, which often far exceeds the necessary precision needed to represent individual image pixel values. For this reason, fpack usually quantizes the pixel values into 32-bit integers using a linear scaling function:

integer_value = (floating_point_value - ZERO_POINT) / SCALE_FACTOR

This array of scaled integers is then compressed using one of the supported compression algorithms (the default algorithm is Rice). When the image is subsequently uncompressed, the integer values are inverse scaled to closely (but not exactly) reproduce the original floating point pixel values.   Separate scale and zero point values are computed for each tile of the image. These and other issues related to compressing floating-point images are discussed in greater detail in a Paper II.

The value of SCALE_FACTOR in the above scaling function controls how closely the inverse scaled values approximate the original floating point values: decreasing SCALE_FACTOR reduces the spacing between the quantized levels in the inverse-scaled values and thus more closely reproduces the original pixel values. However, this also magnifies the dynamic range and the noise level in the integer array that is to be compressed, which adversely affects the amount of compression that is achieved. Thus, there is a direct trade-off between providing more precision or achieving greater compression.

One refinement to the quantization procedure described above is to “dither” the pixel values by adding a small amount of random noise to the floating point value before scaling it to an integer; that same random value is then subtracted when converting back to the floating point value. This “subtractive dithering” technique helps preserve low amplitude signals in the quantized image through an effect known as “stochastic resonance”. This is especially important for preserving the mean value of the background sky level when measuring the flux of faint sources in the image.

It is not easy to directly calculate an appropriate SCALE_FACTOR value to use with a given image, therefore fpack provides instead a quantization parameter called "q" for specifying how closely the inverse-scaled integer pixel values must approximate the original floating point pixel values, relative to the measured noise in background areas in the image. The image pixel values will be quantized so that the spacing between the adjacent discrete levels is equal to the measured sigma of the R.M.S. noise in the background regions of the image divided by q. The maximum deviation between the pixel values in the compressed image and in the original image will be half this value. Formally, the number of noise bits that are preserved in each pixel value is given by $\log_2(q) + 1.792$. For example, if q = 4, then the quantized levels are spaced at intervals of sigma/4 and about 3.8 bits of noise are preserved in each pixel value. Increasing the value of q will produce compressed images that more closely approximate the pixel values in the original floating point image, but will also increase the size of the compressed image file, as shown in Table 2. The third column in this table shows how much the background noise will increase in the quantized images, which is given by $\mathrm{sqrt}(1 + 1/(12q^2))$. The numerical experiments described in Paper II demonstrate that the statistical noise on the measured magnitudes and positions of faint stars in the quantized images can be expected to increase by about the same percentage amount. Unless otherwise specified, fpack currently uses a default q value = 4 when subtractive dithering is also perfomed, or q = 16 when dithering is not performed.

Table 2. Compression of Floating Point Images

q	**Compression Ratio**	**Noise increase %**
1	9.5	4.1
2	8	1
4	6.5	0.26
8	5.3	0.07

Users of fpack are urged to perform quantitative tests on there own floating-point image data sets using different values of q to determine the appropriate value for their particular application.

In some situations, or for test purposes, it may be desirable to losslessly compress floating point FITS images. This can be accomplished by specifying the GZIP compression algorithm with -g, -g1, or -g2 (because the other algorithms can only compress integers) and a q value of 0 (e. g., “fpack -g2 -q 0”). This will exactly preserve every bit in the floating-point image at the expense of much lower compression ratios than given by the quantization technique. Preliminary tests indicate that the -g2 variant (where h the bytes in the pixel values are shuffled so that all the most-significant bytes occur first, followed by the next most significant bytes, and so on before the entire byte stream is compressed with the GZIP algorithm) often produces better compression of floating-point images than the plain -g1 compression method.

4.4. Lossy compression of integer FITS images

Some integer FITS images contain too much noise to be losslessly compressed very effectively. In our Paper I, for example, Figure 3 shows a case of deep CCD exposures which contain 7 – 8 bits of noise per pixel that can only be losslessly compressed by less than a factor of 2. Just as with typical floating-point images, removing some of the noise from these integer images can increase the image compression ratio without significantly degrading the scientific accuracy of measurements in the image.

Fpack has an option (specified with the -i2f command line parameter) to internally convert integer FITS images into floating-point format, and then apply the same quantization procedure that it uses for actual floating-point FITS images. The compression ratio will depend primarily on the specified q quantization parameter, as shown in Table 2, except that the compression ratio will only be ½ of the amount shown in the table, since the original 16-bit integer pixels are ½ the size of the 32-bit floating point-pixels. Note that if these compressed images are subsequently uncompressed, they will have a floating-point, not integer, data type.

If an integer image contains too little noise, then it does not make sense to use this lossy compression option because the compressed file will actually be larger than if the standard lossless compression method was used. To prevent this, there are 2 other fpack parameters that specify the minimum threshold for the amount of noise in an image for this lossy compression method to be applied. If these thresholds are not met, then the -i2f option is ignored and the integer image is losslessly compressed. For further information, see the description of the -n3ratio and -n3min fpack parameters in the following section.

5. Compression of FITS binary tables (Experimental feature)

An experimental capability to losslessly compress FITS binary tables is available in fpack. This uses a prototype FITS convention (http://fits.gsfc.nasa.gov/tiletable.pdf) for compressing each column of the table and storing the compressed bytes in a variable-length array column in the compressed table.

This table compression method is invoked by adding the -table or -tableonly option on the command line. Funpack will automatically uncompress these tables (no command line switch is required).

Note that while CFITSIO can directly read and write FITS images in their compressed form, this is currently not possible for FITS binary tables. One must first create the uncompressed FITS binary table file, then compress it using fpack. Similarly, one must uncompress the table with funpack before CFITSIO can read the contents of the table.

Appendix I: Example of the fpack -T test report

```
File: ct655046_13.fits
Ext BITPIX Dimens. Nulls    Min    Max     Mean   Sigma  Noise3 Nbits  MaxR
  0  16  (1112,4096)   0 -31503  25967 -26679.3 2.5e+03    56.8   7.6  2.10

  Type   Ratio       Size (MB)     Pk (Sec) UnPk Exact ElpN CPUN  Elp1  CPU1
  Native                                             0.024 0.016 0.013 0.010
  RICE    1.83    9.11 ->   4.98    0.57    0.55 Yes 0.053 0.047 0.045 0.040
  HCOMP   1.85    9.11 ->   4.92    1.91    1.56 Yes 0.175 0.159 0.179 0.162
  GZIP    1.35    9.11 ->   6.73    3.07    1.09 Yes 0.114 0.106 0.108 0.101
  NONE    0.99    9.11 ->   9.18    0.35    0.31 Yes 0.022 0.021 0.015 0.013
```

The first line of the report gives the name of the FITS file; the 3rd line gives the following parameters:

Ext – extension number within the file (zero based)
BITPIX – FITS datatype of the image (8, 16, 32, -32 or -64)
Dimens – image dimensions
Nulls – number of undefined or null pixels in the image
Min, Max – the minimum and maximum values in the image
Mean – mean value of all the non-null pixels
Sigma – standard deviation of all the non-null pixels
Noise3 – a measure of the noise in the background regions of the image
Nbits – number of noise bits per pixel = $\log_2(\text{noise3}) + 1.792$
MaxR – theoretical maximum possible compression ratio = BITPIX / Nbits

This is followed by a table with the following columns:

Type – name of compression method, if any
Ratio - file compression ratio
Size – uncompressed and compressed sizes of the files, in MB
Pk – the CPU time in seconds to compress the image with fpack
UnPk – the CPU time in seconds to uncompress the image with funpack
Exact – is the compression lossless (i.e., does it exactly preserve the pixel values)?

The following 4 parameters give the measured image read rates, in units of seconds/MB

ElpN – elapsed time to read the entire image with a single subroutine call
CPUN – CPU time to read the entire image with a single subroutine call
Elp1 – elapsed time to read the whole image, one row at a time
CPU1 – CPU time to read the whole image, one row at a time

The rows in this table correspond to the following cases:

Native – this just gives the read speed of the input uncompressed image
Rice – when using the Rice compression algorithm
Hcomp – when using the Hcompress algorithm
GZIP – when using the gzip algorithm (within the FITS tiled image compression format)
None – the image is simply tiled and packed into the FITS tiled image format, without performing any compression on the tiles.