F File Sizes

Your data reduction strategy will be more effective if you have an idea of how big the data and map file sizes will be on disk. Also, it will help you estimate just how many files you can backup to tape. In this appendix, we will discuss file sizes in bytes, which are 8 bits in size, rather than "blocks," which can vary in size between different computers. Inside $\mathcal{AIPS}$, the definition of "byte" is perverted, but all systems now use a 1024-byte (8192 bit) "block."

F.1 Visibility (uv) data sets

uv data files contain information about the coherence function of a wavefront at random locations and random times. Consequently, the way this information is stored on disk is different from that for images where the pixels are on a regular grid. AIPS uv data are stored on disk in a manner similar to the way that it would be organized on a FITS "random group" tape. The data are stored as logical records; each record (a "visibility") contains **all** the data taken on one baseline at a given time. Consequently, a record may contain information for several IFs, several frequencies at each of those IFs and more than one polarization combination for each frequency/IF. The first part of each logical record contains what are known as "random parameters" *e.g.*, spatial frequency coordinates and time. After the random parameters, there is a small, regular array of data.

For a multi-source data set such as might be created by `FILLM`, the random parameter group will include the following. `UU-L-SIN`, `VV-L-SIN`, and `WW-L-SIN` give the spatial frequency coordinates, computed with a sine projection in units of wavelengths at the reference frequency. `TIME1` is the label for the time in days. `BASELINE` is the baseline number ($256ant_1 + ant_2 + subarray/100.$) and `SOURCE` is the source number corresponding to an entry in the source table. If you have frequency table identifiers (which is usually the case these days), then there will be an additional random parameter, `FREQSEL`. For a compressed database, two additional random parameters will be required — `WEIGHT` to give a single data weight for all samples in the record and `SCALE` to give the gain used to compress that record.

The regular data array is similar to an image array in that the order of axes is arbitrary. However, the convention is for the first axis to be of type `COMPLEX`, having a dimension of 3 for uncompressed data (real, imaginary, weight) and a dimension of 1 for compressed data. The other axes of the regular array are `IF`, `RA`, `DEC`, `FREQ` and `STOKES`.

F.1.1 uv database sizes

The number of words in each "visibility" is given by

$$(\#\,random\ parameters) + [(dimension\ of\ \texttt{COMPLEX}\ axis) \times (\#\,polns) \times (\#\,freqs) \times (\#\,IFs)]$$

The size of the database to be loaded to disk with `FILLM` is given by

$$(word\ per\ record) \times (\#\,vis) \times 4\,\mathrm{bytes}$$

The number of visibilities is given (approximately) by

$$\frac{length\ of\ observation}{integration\ time} \times number\ of\ baselines$$

where the number of baselines is the usual $\frac{1}{2}n(n-1)$. This is equal to 351 for the VLA for the usual 27-antenna array.

For example, a 12-hour observation with 30 second integrations, one frequency and 2 IFs with RR, RL, LR and LL written in compressed format (`DOUVCOMP = TRUE` in `FILLM`) will occupy about 34 Mbytes on disk. In practice, the *uv* file will usually be a little larger due to the way the system allocates space on the disks. You must also remember to allow room for the extension tables — see § F.3. If this database had been written in uncompressed format, the *uv* data would have occupied around 62 Mbyte, but would have carried information about different data weights for different IFs.

Consider another example illustrated by the `IMHEAD` listing below:

```
Image=MULTI     (UV)          Filename=20/07/90    .L BAND.   1
Telescope=VLA                 Receiver=VLA
Observer=AFTST                User #= 1364
Observ. date=20-JUL-1990      Map date=23-JUL-1990
# visibilities    105198      Sort order  TB
Rand axes: UU-L-SIN  VV-L-SIN  WW-L-SIN  BASELINE  TIME1
           SOURCE  FREQSEL  WEIGHT  SCALE
--------------------------------------------------------------
Type    Pixels   Coord value  at Pixel    Coord incr   Rotat
COMPLEX      1   1.0000000E+00    1.00 1.0000000E+00    0.00
STOKES       4  -1.0000000E+00    1.00-1.0000000E+00    0.00
IF           2   1.0000000E+00    1.00 1.0000000E+00    0.00
FREQ         1   1.4524000E+09    1.00 2.5000000E+07    0.00
RA           1    00 00 00.000    1.00      3600.000    0.00
DEC          1    00 00 00.000    1.00      3600.000    0.00
--------------------------------------------------------------
Maximum version number of extension files of type HI is   1
Maximum version number of extension files of type AN is   1
Maximum version number of extension files of type NX is   1
Maximum version number of extension files of type SU is   1
Maximum version number of extension files of type FQ is   1
Maximum version number of extension files of type CL is   1
Maximum version number of extension files of type SN is   2
```

This compressed (`COMPLEX Pixels = 1`) *uv* database contains 9 random parameters, 4 polarizations, 1 frequency, and 2 IFs. for *each* of 105198 visibilities. The size of the database file itself is, therefore,

$$\Big[\{9 + [1 \times 4 \times 1 \times 2]\} \times 105198 \times 4\Big] \,\text{bytes} = 7.153\,\text{Mbytes}.$$

Note that this data set is 17/33 the size of an uncompressed data set.

F.1.2 Compressed format for *uv* data

The use of "compressed" data can make substantial savings in the amount of disk space that you require, particularly for spectral-line databases. All tasks should now be able to handle either the compressed or the uncompressed formats. Compressed data files can be identified by the dimension of 1 for the `COMPLEX` axis in the database header. (Uncompressed data will have a dimension of 3.) The savings can be close to a factor of three for spectral line observations.

This is achieved by converting all data weights into a single `WEIGHT` random parameter, by finding a single `SCALE` random parameter with which to scale all real and imaginary parts of the visibilities into 16-bit integers, and by packing the real and imaginary terms into one 32-bit location using magic-value blanking for flagged data. This is to be compared with the uncompressed format in which each of the real, imaginary and weight terms are each stored in a 32-bit floating-point location. The use of a single weight value masks *real* differences in system temperatures between polarizations and IFs, which one should retain for the lowest possible noise in imaging.

In general, data compression is a good thing and should be used, but with a little caution. With a single frequency, single IF, and single polarization, you will not save any disk space. In all other cases, there are respectable savings to be made. However, the use of a packed data word for the real and imaginary parts of the visibility function along with magic value blanking imposes a restriction on the "spectral dynamic range" of the data set of around 32000:1. Consequently, there are some situations where compressed data should *not* be used. For example, if the spectral dynamic range in the *uv* database is likely to be greater than, say, 1000:1, you must use *uncompressed* data format to avoid loss of accuracy. This situation can arise in maser spectra, for example, in which there are maser lines of 1 Jy and > 32000 Jy; in this case, you should never use compressed data. Bandpass calibration can cause large correction factors to be applied to the edge channels of a database. In the presence of noise or interference, bad channels can become very much greater in amplitude than good channels. In such cases you must either use uncompressed format or be very careful to flag bad channels or to drop them with the `BCHAN` and `ECHAN` adverbs *as* you apply the bandpass calibration. In general, continuum data sets can be loaded with data compression since these dynamic range considerations will not normally apply. The loss of weight information may not be worth the savings in disk space, however. Compressed data takes less time to read/write due to the small number of bytes, but casts in cpu time some due to the extra computation.

If there has been on-line or later flagging that depends on polarization, IF, or spectral channel (*i.e.*, RFI excision) or differences in the intrinsic weights between polarizations, IFs, or spectral channels (*i.e.*, different system temperatures for different IFs), then data compression causes a serious loss of information related to the data weight.

F.2 Image files

Since images are regular arrays, the sizes of image files are easier to calculate. The images are stored as floating-point numbers, *i.e.*, 32 bits per pixel, so the image file size in bytes is given by

$$4 \times \prod_i length(i)$$

where $length(i)$ is the number of pixels on the i^{th} axis. For example, a 128-channel cube with each plane a 256×256 image will require around 34 Mbytes of disk storage space. This may be increased a little due to the way in which the system allocates space for files.

F.3 Extension files

Subsidiary data about *uv* database and image files are written in "extension files." These include, for example, history records (`HI` files), plot instructions (`PL` files), calibration solutions (`SN` files), ad nauseum. Some extension files can become large. A history file uses 1024 bytes for every 14 history records, a small amount under most circumstances. A plot file is normally small, but the output from `GREYS` can be as large as one plane of the image and the output of `KNTR` is larger by a factor of the number of panes in the plot.

The `CL` (calibration) table contains the total model of the interferometer at each interval. Many of these logical records are blank in the case of most interferometers but, for the VLBA, these records will contain essential information. The `CL` table contains a logical record for each antenna in the array at each CL time stamp. The CL time stamps are set by the user when loading the data. The default for VLA data is every 5 minutes. For VLBI data, it is every 1 minute. Each `CL` logical record requires

$$\{15 + [14 \times (\#\, IFs) \times (\#\, pols)]\} \times 4\,\text{bytes}$$

For a 12-hour observation with the VLA with 2 IFs and 4 polarization pairs, with entries every 5 minutes, the CL table will occupy about

$$\left[\{15 + [14 \times 2 \times 2]\} \times 4 \times \frac{(12 \times 60)}{5} \times 27\right] \text{bytes} = 1.05\,\text{Mbytes}$$

Since most other files are considerably smaller, their sizes can be ignored. That they exist and may require some disk, should not be forgotten. To look at the full AIPS disk usage on your computer in summary form:

> TASK 'DISKU' ; INP C_R	to review the inputs.
> INDISK 0 ; DETIME 0 C_R	to look at all disks and all data sets
> USERID 32000 C_R	to look at all user numbers.
> DOALL FALSE ; GO C_R	to run the task in a summary mode.

Then to look at file sizes in detail:

> INDISK n ; USERID 0 C_R	to restrict the display to your data sets and one disk.
> DOALL TRUE ; GO C_R	to run the task to list all files on disk n.

F.4 Storing data on tape

Images and *uv* databases are written to magnetic tape by `FITTP` for archival purposes and for transfer to other computers and sites. Three FITS-standard formats are available, controlled through the adverb `FORMAT`. The preferred format is 32-bit floating point (IEEE standard) format. There are no dynamic range limitations in this format and, on many modern computers, no bit manipulation is required since they use IEEE floating internally.

Of the two integer formats, there is little reason to use the 32-bit integer since it poses dynamic range, re-scaling, and other problems with no saving in space. The 16-bit integer format uses 16-bit signed 2's complement integers to represent the data. Such numbers are limited to the range -32768 to 32767. `FITTP` has to find the maximum and minimum in the image and then scale the data to fit in this numeric range. For images of limited dynamic range, this format is perfectly adequate. In fact, `FITAB` offers the option to reduce the dynamic range even further with the `QUANTIZE` adverb. For images written to FITS disk files, this allows for better compression before the files are transmitted over the Internet. For high-dynamic range images, the 16-bit format may not be adequate. (The integer formats are no longer allowed for *uv* data. More than one user has reduced all his "good" spectral channels to pure 0 by scaling all the *uv* data to include one really horrendously bad sample.) A less important benefit of the floating point format is that the numbers representing your data are recorded exactly on tape as they are stored on disk; there are no "quantization errors". This may be important for software development.

The preceding paragraphs do not tell the full story, however. The portion of the FITS standard used by `FITTP` does not allow for *uv* data on tape in a compressed format. Instead, `FITTP` expands the data into the uncompressed form and then writes the data on tape. In the conversion, the real and imaginary values that were stored in one packed number are expanded into three real values — one each for real, imaginary and weight terms — and the weight and scale random parameters are removed since they are no longer required. Consequently, the compressed data are expanded to

$$\frac{\{(\#\, random\ parameters - 2) + [(\#\, pol) \times (\#\, IFs) \times (\#\, frequencies) \times 3]\}}{\{(\#\, random\ parameters) + [(\#\, pol) \times (\#\, IFs) \times (\#\, frequencies)]\}}$$

the original size (where $\#\ random\ parameters$ is the original number in the compressed database).

As an example, let us consider a multi-source spectral-line database stored on disk in compressed format. The data set has seven channels each at 2 IFs with 2 polarizations. There are nine random parameters and 834031 visibilities. From § F.1, we can calculate the size of the *uv* file to be 123 Mbytes. (Remember, this doesn't include any of the extension files, some of which might be several Mbytes in size.) Before the file is written to tape in 32-bit floating format, it is first expanded by a factor of

$$\frac{\{(9-2)+[2\times 2\times 7\times 3]\}}{\{9+[2\times 2\times 7]\}} = 2.333\,.$$

Consequently, the data will occupy

$$123 \times 2.333\,\mathrm{Mbytes} = 287\,\mathrm{Mbytes}$$

on tape. In other words, this database and all the associated extension files will not fit on a standard, 6250 bpi tape even using `BLOCKING = 10`, but modern tape technology solves this problem.

Note that `FITTP` writes history file data into the FITS header and writes table extension files as extensions after the main image or data set within the same tape file. Plot (`PL`) and slice (`SL`) files are not saved to tape.

Task `FITAB` uses "binary tables" to represent visibility data rather than the old, mildly deprecated "random groups" form of the FITS format. This has several advantages for *uv* data. It allows compressed data to be recorded in that form exactly (except for byte-order questions which should not concern the reader). It also allows the data and attached tables to be divided into pieces which will reduce the size of files to be copied over the Internet, making copying and tape storage somewhat more reliable. The principal disadvantage of `FITAB` *uv* data output is that only two packages can read it so far. The one "outside" package that reads the format is `obit` available from Bill Cotton at NRAO Charlottesville.

F.4.1 DAT and Exabyte tapes

The arrival of modern tape technologies has hastened the demise of 9-track tapes. First Exabyte (8mm) and then DAT (4mm) have provided much higher storage capacities than the 9-track tapes and have also provided faster seeks between file marks and greater data reliability. The new technologies are very much cheaper as well, in part because they have been adopted by the PC market. They are both technically quite complex internally. The DAT tape has a "system log" area at the beginning which allows for the fast seeks. It is a bit fragile, however, since it is updated when the tape is unloaded and hence can be incorrect if there is an unfortunate power failure. Both technologies are still evolving and both now offer various data encoding/compression options. Unfortunately, the data compression techniques vary considerably with tape model and manufacturer and hence should not be used to archive or transport data. The data are blocked on the tapes by means known only to the manufacturers and are not significantly under user control. It is still probably good to use a large `BLOCKING`, but only for I/O transfer reasons. The EOF marks can be expensive on these tape devices.

Exabytes at low density have a capacity of about 2.2 Gbytes on a 112m tape and use about 1 Mbyte (or maybe even 4 Mbytes) for each EOF mark. The large size of the EOF limits the number of files you can write rather significantly. The EOFs are also slow to process mechanically. Exabytes at high density have a capacity of 4.5 Gbytes on a 112m tape and use 48 Kbytes per EOF mark. DATs have a capacity of 2.0 Gbytes on a 90m tape, but also come in 60m and 120m sizes. The EOF mark size is not readily available, but is probably no more than 48 Kbytes. The early warning of the end-of-medium is 40 Mbytes before the actual end of tape.

F.5 Very large data sets

FITTP and FITAB cannot write multi-volume tapes. Some spectral-line and VLBA databases (and perhaps some continuum databases) may be so large that the file cannot fit on one modern tape even with BLOCKING = 10. What can you do to backup your data? The simplest solution is to use FITAB. This task can write *uv* data in compressed form and can break up a data set into "pieces." You write as many pieces as will fit on the first tape, noting what the piece number is when the end-of-tape is reached. You can then tell FITAB to begin with that piece number on the next tape (BDROP = n-1 where n is the number of the piece that encountered the end-of-tape). Unfortunately, the FITS format used by FITAB, while perfectly legitimate, is only understood by AIPS versions beginning with 15APR99. Furthermore, FITAB output is only understood by AIPS and OBIT. If the data set is so large that it will not fit on the required tape device using FITTP or the data must be taken to a system that does not understand FITAB's format, there are several approaches that you can adopt.

First, you could SPLIT out the database into *single-source* databases and back each of these up individually. Alternatively, you could subdivide the large database in several smaller databases with UVCOP by specifying a different time range for each of the smaller databases and then back these up individually. Another way to solve the problem is to realize that the calibration and flagging information that you have carefully generated during the calibration is contained in the extension tables — the raw data that you loaded is not modified until you finally SPLIT out the individual sources. Consequently, you can write create a dummy *uv* database to which all the extension tables are attached with the task TASAV, then save this "database" on tape with FITTP. The raw visibilities can be saved in the form of copies of the archive tapes.

F.6 Additional recipes

F.6.1 Banana stuffing

1. Pare and rub 4 **bananas** through a sieve into bowl.
2. Add 1/2 grated **onion**, 1 **green pepper** chopped fine, 3 tablespoons finely chopped **parsley**, 4 slices cooked **bacon** chopped fine, 1 1/4 cups **bread crumbs**, pinch of **thyme**, 1 teaspoon **salt**, and 1 **egg**.
3. Mix thoroughly, fill 1 **chicken**, and roast in the usual manner.

F.6.2 Banana nut bread

1. Cream 1 cup **sugar** and 1/2 cup **margarine** together.
2. Add 2 **eggs**, 2 cups **flour**, 1/2 teaspoon **salt**, and 1 teaspoon **baking soda** and mix thoroughly.
3. Add 1 cup chopped **nuts** (walnuts or pecans), 3/4 cup mashed **bananas**, and, lastly, 4 teaspoons **sour milk** and mix well.
4. Put in greased loaf pan.
5. Bake in 350° F oven for 1 hour.