Post-Correlation Review

version 3.0
21 september 2007

This document is a new guide for the creation of fits tapes following the correlation of an experiment. The FITS-creation process has two broad stages: first is the selection of subjobs you want to include in the final fits file (¶1), and second is the conversion process from correlator-data format through Measurement Set to FITS, with some flagging of the data while in the Measurement Set (¶2–4). After you've made the FITS files, there remains the important stages of archiving, cleaning up after yourself and pipelining (¶4–6).

0. for the remorselessly impatient

Below is a checklist for the process, with references to paragraphs in the main text. Items in {braces} may not need to be done for every experiment.

1) Selecting subjobs to use:

2) Making Measurement Set(s)

3) Operations on the Measurement Set(s)

The checklist on the following page can be used to guide the process of making FITS files and related activities (green means step done in glish; italic font means a step to think about before applying; brackets mean steps usually not required). Figure 1 that follows shows the general flow of data & communications within the EVN for an experiment, with the purview of this guide outlined. Figure 2 shows a flowchart for the review process itself. These can all be found on the JIVE how-to wiki (`http://juw26:8000`), as can much other supporting documentation.

The rest of this guide attempts to conform to a convention in which:

`Typewriter text font` signifies verbatim parts of a path, program input, *etc.*

Italic font signifies parts of a path, program input, *etc.* that may vary (usually experiment-, job-, or disk-related).

{braces} signify a task or parameter that may not be required.

The computers you will be using are referred to as C^3 (currently `ccsops`, for correlator control), D^3 (currently `juw26`, where raw correlated data live), and E^3 (currently PCInt, where the bulk of the post-correlation operations take place). Unless otherwise specified, everything in here assumes you have logged in as user `jops`.

There is also a web-based interface for going through these steps that has been developed under the aegis of EXPR*e*S, called EZJive. This document does not go into that, but the descriptions of the steps and the various looks under the hood at underlying principles/algorithms/problems should be directly transferable.

Post–Correlation Review Checklist

1. *select good subjobs*

2. **getdata.pl** .....

3. **j2ms2** .....

4. **Full–MS ops:**

   a) [jfilt] / [badcorr/polflag] .....

   b) check Autocorrs / SampStats .....

   c) 2bitVV .....

   d) fixfbs .....

   e) fixuvw .....

   f) plyflg (only if phscals on) .....

   g) standardplots .....

   h) other plots....

   i) [whist] / flagweight .....

5. **tConvert** .....

6. *Cover Letter*

7. **Archive: FITS** ☐ **(incl. exp.README)** .....

   Stnd.Plts/Cvr.Ltr ☐ .....

8. *FITS –-> DAT (dd) / DVD (k3b) / etc*

9. **Disk–Housekeep**

   a) DDD (juw26) ☐ *(> distrib.)* .....

   b) EEE (PCInt) ☐ .....

10. *Pipeline* .....

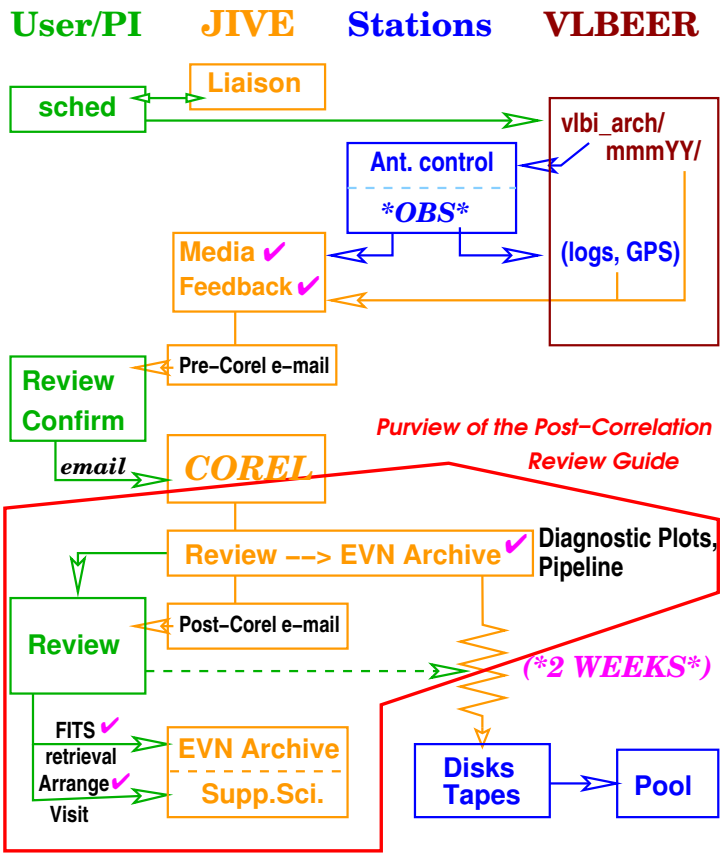

Figure 1: Operational flow of data/communications for an EVN experiment.

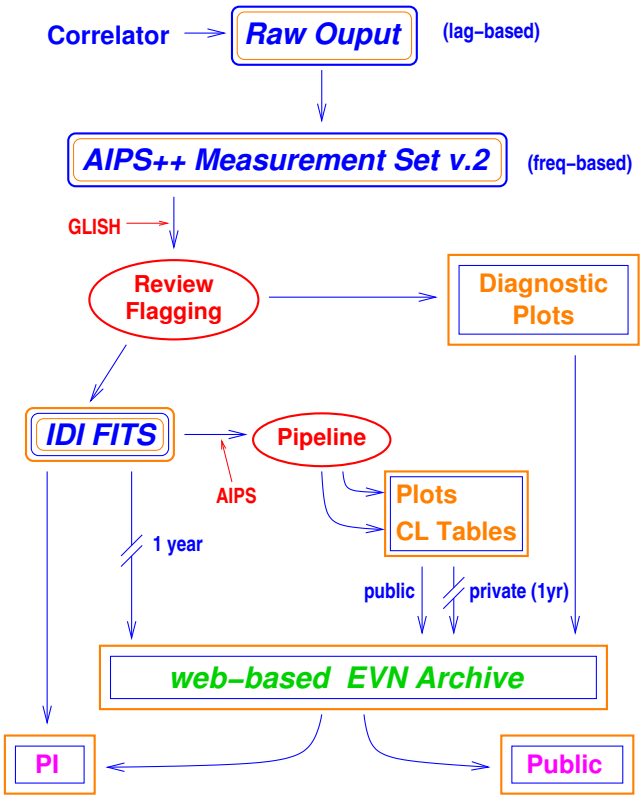

Figure 2: Flowchart for the post-correlation review process.

1. Selecting Subjobs.

a. vexsum.pl.

This should have already been run prior to correlation, with the output in the `/ccs/expr/`*EXP* directory of `ccsops`, typically with the name *exp*`.vxsm`. (Note that in this guide *EXP* denotes supplying the experiment name in upper-case, *exp* in lower-case.) If `vexsum.pl` hasn't already been run, the most usual syntax is:

`vexsum.pl -tel` *vexfile* `>` *exp*`.vxsm`

The additional parameter `-tel` provides a list of telescopes in the VEXfile for each scan. There's also a `-mode` parameter that will output the name of the mode for each scan; this can be useful at first for quickly checking whether this is a multi-mode experiment or not, but the name of the mode usually takes up so much room in each row to be not worth printing. The output of `vexsum.pl` provides an easy to read summary of the experiment — each scan gets its own output line with scan/pass number, start/stop time, start byte-number, and source. The byte-number refers to the first station in the VEXfile. If there was a gap between two scans in a given pass, there will be a "`*`" after the time-range of the scan following the gap. Below is a sample extract (from EL035A):

```
No0001 P01 159/11:00:00 - 11:02:30  1518.987970000     * J1113+1442 McOnNtUrSh
No0002 P01 159/11:02:30 - 11:10:00   713.680035424     * B1059+169  EfMcCmOnUrJbNtSh
No0003 P01 159/11:10:40 - 11:12:40* 1557.330436480     * J1113+1442 McCmNtOnShUr
No0004 P01 159/11:12:40 - 11:20:10   742.423946752     * B1059+169  EfMcCmOnUrJbNtSh
No0005 P01 159/11:20:10 - 11:22:40  1594.454220800     * J1113+1442 McCmNtOnShUrJb
No0006 P01 159/11:22:40 - 11:30:10   771.169736328     * B1059+169  EfMcCmOnUrJbWbNtSh
No0007 P01 159/11:30:50 - 11:32:50*  799.915541976     * J1113+1442 EfMcCmOnUrJbTrNtSh
No0008 P01 159/11:32:50 - 11:40:20   809.708236800     * B1059+169  EfMcCmOnUrJbTrNtWbSh
No0009 P01 159/11:40:20 - 11:42:50   838.511333376     * J1113+1442 EfMcCmOnUrJbTrNtSh
No0010 P01 159/11:42:50 - 11:50:20   848.112365568     * B1059+169  EfMcCmOnUrJbTrNtWbSh
No0011 P01 159/11:51:00 - 11:53:00*  874.740528368     * J1113+1442 EfMcCmOnUrJbTrNtWbSh
```

The output of `vexsum.pl` is a useful overview of the experiment's schedule, and as a quick look-up translation table between UTCs, scans, sources, and participating stations.

b. showlog.

`showlog` can function as the source of information about the correlated subjobs. The `vexsum.pl -tel` output provides a summary of the scans in the experiment and stations participating in them. `showlog` outputs will allow you to begin to review whether all these scans have been correlated to the best of our ability, and to pick among multiple correlations of the same scan(s).

i — Start `showlog` on `ccsops` or `juw26` by:

`showlog` *EXP*

You will see the summary of all scans come up in the window, with four buttons on the upper right. There are two kinds of output useful in the course of post-correlation review: the basic logbook file and the *exp*`.lis` file.

ii — The logbook file is made by clicking on the LogFile button. A pull-down menu will result. From here, you can restrict the Type (`PROD`, `TEST`, `CLOCK`; default = all), the Status (`GOOD`, `REVIEW`, `FAIL`, `ABORT`, `CRASH`; default = all), or Profile (*i.e.*, the `runjob.pl` profiles) displayed. you can Sort the results by `JOB` (chronologically as correlated), `SCAN`, `LAGS`, or `PROFILE`. This sorting is single-key sorting; you can't simulate a multiple-key sort by sorting first on the second key and then on the first — subsequent sorts do not retain the order from previous sorts in "breaking ties". With `FieldLayout` you can suppress some columns; the default `LogFile` output currently takes 220 characters per line. Finally, Print/Save allows you to make a hard copy of the output. ¶1.c discusses the information contained in the logbook file in more detail.

iii — When you click on the ExportFile button, the `LogFile` format summary will be replaced by a `*.lis`-file format summary (¶1.g). Clicking again on ExportFile gets a pull-down menu. Here, you can select by `profile` or Print/Save. The *exp*`.lis` file you save from here will drive the production of the FITS file. If the experiment used multiple profiles (*e.g.*, separate line/continuum passes, different phase centers, subbands correlated separately) that will go into separate Measurement Sets & FITS files, then you need to make a separate `lis`-file for each profile, saving them to distinct filenames. ¶1.f discusses the information contained in the `*.lis`-file in more detail.

c. LogFile output.

```
JIVE Correlator Logbook    Sorted on: JOB      Selection: Type = ANY   Status = ANY   Profile = ANY                                                                  24 Sep 2002 09:57:56
Exp        JobId      SJ  Pass ScBegin ScEnd   Type   Status Lags Int  CPol Integrations          Data    validity Array                           Version  VexFile              profile          Operator
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
N02L2      206210936  1   3A   No0015  No0015  CLOCK  GOOD   256  4    Xpol (2/9/0/312/157)       100.0%  local    <><><>Wb<>JbShEfMcUr<><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# Fringes to Mc, Wb, Jb(!), Sh, and Ur(!).
# Ar and Hh were not scheduled in this scan,
# Ef started late (not present in this scan).
N02L2      206211034  1                        CLOCK  NONE   256  4    Xpol                               local    TrNt<>WbOnJb<>EfMc<><><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# Aborted. Restart correlator.
N02L2      206211058  1                        CLOCK  NONE   256  4    Xpol                               local    TrNt<>WbOnJb<>EfMc<><><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# No data coming through. Aborted.
N02L2      206211118  1                        CLOCK  NONE   256  4    Xpol                               local    TrNt<>WbOnJb<>EfMc<><><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# Correlator will be restarted.
N02L2      206211149  1                        CLOCK  ABORT  256  4    Xpol                               local    TrNt<>WbOnJb<>EfMc<><>Ar<><><><> rel3001  n0212.hs1.vix                         Zsolt
# No data from correlator.
N02L2      206211222  1                        CLOCK  FAIL   256  4    Xpol                               local    TrNt<>WbOnJb<>EfMc<><>Ar<><><><> rel3001  n0212.hs1.vix                         Zsolt
N02L2      206211247  1   5A   No0026  No0026  CLOCK  GOOD   256  4    Xpol (0/1/0/544/198)       100.0%  local    TrNt<>WbOnJb<>EfMc<><>Ar<><><><> rel3001  n0212.hs1.vix                         Zsolt
# Success! Fringes to Ar(!!!), Ef, Tr, Nt, On.
# Wb and Jb clock updates checked, OK.
N02L2      206211409  1   6A   No0027  No0027  CLOCK  GOOD   256  4    Xpol (1/1/1/544/29)        100.0%  local    TrNtHh<>On<>ShEfMcUr<><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# Hh fringes; clock is perfect.
N02L2      206211423  1   6A   No0028  No0028  CLOCK  GOOD   32   4    Xpol (0/0/0/840/54)        100.0%  local    TrNtHhWbOnJbShEfMcUr<><><><><><> rel3001  n0212.hs1.vix                         Zsolt
# All telescopes (except AR) included, nlags=32.
# It is a reverse pass except for Hh, Sh, and Ur.
# All clocks are OK. Tr produced no fringes
# in this scan (0 weights).
N02L2      206211506  1   5A   No0026  No0026  CLOCK  GOOD   256  4    Xpol (0/0/0/420/132)       100.0%  local    Tr<><>WbOnJb<>EfMc<><>Ar<><><><> rel3001  n0212.hs2.vix                         Zsolt
# 2nd headstack fringes to all telescopes
# participating in this scan (including Ar).
N02L2      206211538  1   6A   No0031  No0031  CLOCK  GOOD   64   4    Xpol (0/0/0/684/32)        100.0%  local    Tr<>HhWbOnJbShEfMcUr<><><><><><> rel3001  n0212.hs2.vix                         Zsolt
# Ur, Sh and Hh headstack 2 is also OK.
N02L2      206251911  1   1A   No0001  No0007  PROD   GOOD   32   1    Xpol 4/0/3/544/1308        100.0%  local    NtTrWbMcOnJb<><>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
N02L2      206251943  1   2A   No0008  No0014  PROD   GOOD   32   1    Xpol 13/4/18/420/1504      100.0%  local    NtTrWbMc<>Jb<><>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
N02L2      206251943  2   3A   No0015  No0018  PROD   GOOD   32   1    Xpol 4/0/3/684/1302        100.0%  local    NtTrWbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
N02L2      206252053  1   4A   No0019  No0025  PROD   CRASH  32   1    Xpol 249/3/323/684/1665    100.0%  local    NtTrWbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
# Almost everything collapsed.
N02L2      206252155  1   4A   No0019  No0025  PROD   GOOD   32   1    Xpol 485/3/561/684/1906    100.0%  local    NtTrWbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
# Dpu  2: Auto tracking failed, footage 16018 -> 466 at headpos:   96.0 (40 samples)
N02L2      206252155  2   5A   No0026  No0026  PROD   GOOD   32   1    Xpol 4/0/3/544/1302        100.0%  local    NtTrWbMcOnJbEfAr<><><><><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
N02L2      206252316  1   6A   No0027  No0034  PROD   GOOD   32   1    Xpol 4/0/3/840/1420        100.0%  local    NtTrWbMcOnJbEf<>ShHhUr<><><><><> rel1706  n0212.hs1.vix  production_hs1  Hans
N02L2      206260830  1   4A   No0019  No0025  PROD   GOOD   32   1    Xpol 4/0/3/684/1307        100.0%  local    NtTrWbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs1.vix  production_hs1  Martijn
N02L2      206261121  1   1A   No0001  No0007  PROD   GOOD   32   1    Xpol 5/2/482/420/1548      100.0%  local    <>TrWbMcOnJb<><>Sh<>Ur<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
# Dpu  1: Auto tracking failed, footage 3052 -> 6646 at headpos:  -76.9 (10 samples)
# On(4) unable to peak head - giving up (scan 8 to 14)
N02L2      206261121  2   2A   No0008  No0014  PROD   FAIL   32   1    Xpol 4/2/542/312/1507       66.7%  local    <>TrWbMc<>Jb<><>Sh<>Ur<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
# On(4) unable to peak head - giving up
N02L2      206261121  3   3A   No0015  No0016  PROD   REVIEW 32   1    Xpol 142/1/167/544/980     100.0%  local    <>TrWbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      206261338  1   3A   No0015  No0018  PROD   GOOD   32   1    Xpol 9/3/17/544/1361       100.0%  local    Tr<>WbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      206261338  2   4A   No0019  No0025  PROD   FAIL   32   1    Xpol 95/3/793/544/1569     100.0%  local    Tr<>WbMcOnJbEf<>Sh<>Ur<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      206261519  1   5A   No0026  No0026  PROD   GOOD   32   1    Xpol 5/4/408/420/1755      100.0%  local    Tr<>WbMcOnJbEfAr<><><><><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      206261519  2   6A   No0027  No0034  PROD   GOOD   32   1    Xpol 4/0/3/684/1419        100.0%  local    Tr<>WbMcOnJbEf<>ShHhUr<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      207152222  1   2A   No0008  No0014  PROD   FAIL   32   1    Xpol 4/1/337/220/1454       80.0%  local    <><><>UrSh<>McWb<><>Jb<><><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
N02L2      207152317  1   4A   No0019  No0025  PROD   GOOD   32   1    Xpol 4/0/3/544/1300        100.0%  local    <>Ef<>UrShTrMcWbOn<><>Jb<><><><> rel1706  n0212.hs2.vix  production_hs2  Martijn
# Dpu  8: Auto tracking failed, footage 13270 -> 1108 at headpos:  255.0 (31 samples)
```

Listing 1: LogFile output (from N02L2)

Listing 1 is the LogFile output for N02L2. The first line provides sorting/selection information. the output itself has one line per subjob, with some number of optional comments immediately following (beginning with "#"). Some of these comments are automatically generated by the correlator-control software, others are entered by the operator during correlation. The fields for each subjob follow the column headers at the top. The most important ones are:

`JobID, SJ` = means to identify the data

`ScBegin, ScEnd` = first & last scans in the subjob

`Type, Status` = operator-entered description of subjob purpose & quality

`Lags, Int, Cpol` = correlation parameters used for the subjob

`Integrations, Data` = summary of problems (see bullet below)

`Array` = location of stations per SU ("<>" denotes an empty SU)

`Version` = of the correlator control software

`Vexfile` = VEX file used

`profile` = `runjob.pl` profile used

The `JobID` is of the form *YMoDyHrMi*, the time corresponding to when the job began being correlated. Only the last digit of the year is included (thus there is an inherent "decadal bug" in the job-naming convention). Leading 0s are not included in the `JobID`. The subjob-ID "*SJ*" is simply an integer counting the number of subjobs done within each job. So far, `SJ` has never attained double-digits.

`ScBegin` & `ScEnd` can be compared to the `vexsum.pl` output to ensure there are

no scans missing from the correlated data (however, be alert that there are ways in which the `ScEnd` can be wrong — usually one too high — *cf.* ¶App.1.e.*i*). The `Array` allows one to trace playback problems seen on station(s) to a possible underlying cause in a specific SU/Mk5/DPU.

- The five numbers in `Integrations` are derived from the job's `data_handler.log` file (*cf.* ¶App.1.e.*i*), and denote:

5^{th}: The total number of integration periods encountered (N_{int}). The product $N_{\text{int}} \times t_{\text{int}}$ should be close to the total time spanned by the subjob (including gaps); see the `vexsum.pl` output for the start time of the first and end time of the last scans in a subjob.

4^{th}: The modal value of the number of interferometers (N_{if}) during the pass (*i.e.*, if you made a histogram of N_{if} applicable over each of the N_{int}, the number reported would be the most-occurring N_{if}). Table 1 below shows the N_{if} per subband as a function of N_{sta} and N_{pol} (this is also available on `jaw0`, in the file ~`jops/expr/nifs.out`). If the reported number corresponds to a smaller N_{sta} , you can figure out how many stations you lost; if it is some other number that doesn't correspond to any N_{sta}, you may have had serial-link or correlator-configuration problems. Remember to divide the 4^{th} number by N_{sb} before trying to find it in the table.

Nsta	1 pol	2 pol	4 pol
2	3	6	10
3	6	12	21
4	10	20	36
5	15	30	55
6	21	42	78
7	28	56	105
8	36	72	136
9	45	90	171
10	55	110	210
11	66	132	253
12	78	156	300
13	91	182	351
14	105	210	406
15	120	240	465
16	136	272	528

Table 1: N_{if} per subband as a function of N_{sta} and N_{pol}.

3^{rd}: The number of integrations that have an N_{if} different from that reported in the 4^{th} column. You don't know *a priori* whether these different N_{if}'s are greater or less (or some combination of both) than the reported N_{if} . The usual situation is that a station drops out during the subjob. Weight plots (¶1.d) can often shed light on the causes of the 3^{rd} & 4^{th} numbers.

1^{st}: The number of skipped integrations. These generally come about at the very end of the subjob where the end *BOCF* (Beginning of Correlator Frame) numbers become zero for all interferometers. A large number here will also be reflected in a correspondingly too-large N_{int}. This number of skipped integrations is currently insensitive to "`systick missed`" errors; if the 5^{th} column is

a fair amount less than the duration of the subjob (including gaps) divided by the integration time, then `grep` the job's `data handler.log` (found under the *JobID* directory — *cf.* ¶App.1.e.*i*) for "`systick missed`". If there are lots of these, you may want to consider recorrelation.

2^{nd}: The number of AAARGHHHHHs. These occur when only some number of interferometers have mismatching *BOCF* numbers at the beginning or end of an integration compared to the other interferometers. These usually mean at best a temporary loss of some fraction of the interferometers (if AAARGHHHHHs occur in pairs, the second occurrence may signal the recovery of the missing interferometers), but may also suggest a more pernicious problem.

- The Data percentage field shows the percentage of good data to expected data (based on the "modal" 4^{th} number). Percentages that can't be expressed as M/N_{sta}, where N_{sta} is the number of stations successfully participating in the subjob (*i.e.*, that haven't dropped out prior to the R stage on the status monitor) and M is an integer $\leq N_{\text{sta}}$, point to something wrong with the subjob. Most likely there has been a connection problem somewhere in the input side of the correlator, or a problem with a limited number of correlator chips. ¶App.1.e.*i* discusses how to get more detailed information out of the the job's `data_handler.log`; ¶App.1.e.*ii* discusses how to go further in tracking down where the problem may lie.

If for some reason you want to change information in a subjob (type, status are the usual things that might need changing, the others are derived automatically and should be left alone), you can edit the logfile from `showlog` *EXP*. Follow the menu-chain from the LogFile button: Edit Logfile, then Edit. The window background now becomes pink, and provides a "visual editor" environment. You can position the cursor after the type/status you want to change, use `BackSpace` to erase, and re-type your desired modification. Save the changes via LogFile, Edit Logfile, & Save.

d. plotweight.pl.

`plotweight.pl` is a perl script that plots station weights or autocorrelation amplitudes directly from the correlated data, saving you the time of having to make a Measurement Set first. Since it reads the correlated data themselves, rather than just the `data_handler.log`, it provides a more fundamental view of what's in the subjob than does `showlog`. A cron job runs every morning (05:17) to make the weight & autocorrelation-amplitude plots for all jobs run during the previous day (or more precisely, for all jobs that don't yet have corresponding plots). The treasury of plots lives on `/juw26_6/data/PltWgt/`*EXP*`/`*JobID.SJ*`.wt.{ps|png|gif}` for the weight plots and `/juw26_6/data/PltAuto/`*EXP*`/`*JobID.SJ*`.auto.{ps|png|gif}` for the autocorrelation-amplitude plots. This disk is also cross-mounted on `ccsops`. As time goes by, we may begin `gzip`'ing the postscript weight/auto-correlation plots to save on disk space. The heading of the plot shows the job/subjob, the experiment, and the range of scans covered by the plot (a partial final scan gets included). The DPU on which each station is mounted in the subjob appears in the annotation. Scan boundaries are marked by thin gray vertical lines; if there is also diagonal hatching, the time range "interior" to the hatching is a gap in the schedule (*i.e.*, no source being observed; recording media stopped). The data shown in gaps is irrelevant, since the time-range of the gap will be omitted from the Measurement Set (*cf.* ¶2.c). The subband/polarization channels are color-coded (legend at the bottom of the page).

Figure 3 shows an example weight plot from a correlation of a segment of scans from EK024C. You can see some of the features mentioned above (gap shading, subband/polarization color-coding). Further, there are numerous examples of "Diagonal Weights" (¶App.1.g.$i.\gamma$) and SB5/LCP seems to be dead in Jb & On (with disk-based experiments, most probably the sign of a correlator problem). You can see that the weights in gaps is irrelevant — such data won't make it through into the Measurement Set anyway. Because of the SB5/LCP problem, this whole range would have to be recorrelated. If this didn't happen, then you'd need to recorrelate some of the scans to avoid using the Diagonal Weight events: scans 27–28 or Jb & Sh, and also scans (35)–36 for Wb (¶App.1.g.$i.\gamma$ discusses how the fringes actually disappear a short time before the onset of the Diagonal Weight event in the weight plot; you'd have to look into things to see whether the recorrelation of scan 35 were absolutely necessary — but since it's so short, it might just be easier to recorrelate it anyway).

If you want to run `plotweight.pl` (say on a job from the current day, before the cron job has had a chance to run), the syntax that you would generally use is:

`plotweight.pl -dev` *output*`.ps/vcps` *JobID/SJ*

In principle, this should be run from the `juw26_`*N*`/jops/data/EXP` directory where the data live to cut down on access to network resources. The above will make a portrait color postscript file. If you have lots of stations, also including the command-line parameter `-nant` $N_{\rm ant}$ will set the maximum number of plots per

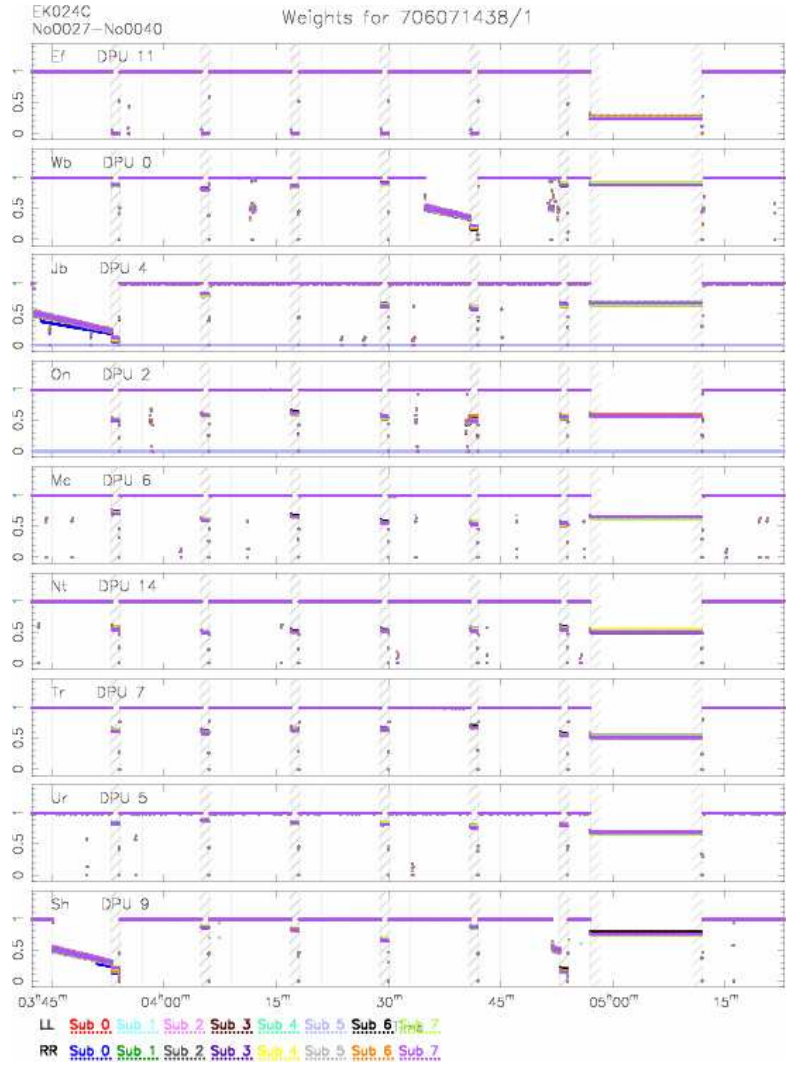

Figure 3: `plotweight.pl` weight plots for a correlation of a segment of scans (from EK024C).

page to $N_{\rm ant}$. Output can go to any recognized/supported pgplot device (the default in the absence of a `-dev` argument has the plot go to the screen, but in this case it seems you can't save or print it, and if you want to resize the `pgplot` window, you'll have to execute `plotweight.pl` a second time after setting the new size). `plotweight.pl` run without any arguments returns a list of all possible arguments. You can also generate these plots from `showlog` via the Inspect Data button on the top. Click on either screen or printer, and then double click on the subjob you want plotted. There's no way to save to a file through this route.

i — Behind the Scenes: `wgtpltr.sh`

`wgtpltr.sh` is the shell-script run in the cron job to make the `plotweight.pl` plots for the previous day's jobs. It is scheduled to run at 05:17 each day. It looks

through all jobs on all D^3 disks, and for any job that lacks a corresponding plot in the directories mentioned above, it runs `plotweight.pl` to make the plot. The early morning run time intends that there should be no jobs running, because its algorithm will be fooled by a job in progress when it runs: it will make plots based on the partially-completed job, but not go back and (re)make plots for the fully-completed job. The recent progress in getting long (> 6 hr) jobs and round-the-clock *e*-VLBI off the ground makes it harder to have this run as an automated cron job without encountering these partially-completed-subjob plots. The resolution (which you hopefully shouldn't have to worry about yourself) is to delete the partial-subjob plots (weight & autocorrelation), and either wait for the next day or run `wgtpltr.sh` manually from the command line. You could also do this, say if your experiment has just finished & you don't want to wait until the next day to look at the plots. Do this logged onto D^3 as `jops`, and execute `wgtpltr.sh`. Be careful to make sure that no job is running when you start this (or will be in the immediate future). Let me know if you think that you've caused partial-job plots to be made.

ii — Behind the Scenes: `plotweight.pl` cron job

Here's the result of `crontab -l jops` on `juw26` showing the daily automatic run of `wgtpltr.sh` to make a weight plot and an autocorrelation plot for each job (line broken into two for display purposes):

```
17 5 * * * /export/home/jops/bin/wgtpltr.sh>>
                /juw26_6/data/PltWgt/cron.log 2>&1
```

e. datasum.pl & daily review for recorrelations

`datasum.pl` forms the basis for much of the text-based information about the correlation success of individual subjobs (*e.g.*, the `ScBegin`, `ScEnd`, `Integrations`, and `Data` columns in the LogFile output. It essentially parses the `data_handler.log` associated with each job, keeping track of various statistics.

The most useful syntax is (`datasum.pl` without arguments gives help):

`datasum.pl -ignore -sort {-first` *jobID_0*`}` *vexfile jobIDs* `>` *exp*`.dtsm`

jobIDs is a list of one (or more) jobs, with wild-cards possible; optionally `datasum.pl` will ignore any data in jobs before *jobID_0*. A *jobID* is the *YMoDyHrMi* name of a job (initial "0"s are not included). The `-ignore` switch limits the output to jobs having valid data and the `-sort` switch sorts the output in lag/pass-number/scan/job order — these make the output easier to use (note that job order is the same as chronological-correlation order). The *vexfile* used should be one that has all scans. If there were more than one VEX-file & these had scans commented out, use of the wrong `vexfile` will result in loss of scan/pass information. The output is essentially a pared-down LogFile or ExportFile. Here's an example:

```
708271659/1# Pass  1  [No0001]         (0/0/0/216/222)     64 100.0% GOOD   PROD prod_cont
708230841/1# Pass  1  [No0001-No0006] (1/0/0/72/424)     2048 100.0% GOOD   PROD prod_line
708230915/1# Pass  1  [No0007-No0018] (2/0/0/72/458)     2048 100.0% GOOD   PROD prod_line
708230952/1# Pass  1  [No0019-No0060] (1/0/0/72/1726)    2048 100.0% GOOD   PROD prod_line
708270807/1# Pass  1  [No0061-No0080] (1/0/0/72/815)     2048 100.0% GOOD   PROD prod_line
708270911/1# Pass  1  [No0081-No0109] (0/0/0/72/1197)    2048 100.0% GOOD   PROD prod_line
708271322/1# Pass  1  [No0109-No0124] (0/4/107/72/639)   2048 100.0% GOOD   PROD prod_line
708271410/1# Pass  1  [No0121-No0145] (2/0/0/72/1051)    2048 100.0% GOOD   PROD prod_line
```

Listing 2: `datasum.pl` output (from EI009A).

i — For "my" experiments, I like to run a `datasum.pl` check first thing every morning for the previous day's correlation. Logged in to `ccsops` as `jops`, I would go to the experiment's data directory. The symlink `/jccs/data/` points to the current D^3 data disk to which correlator jobs are written; chances are the data from the previous day are in `/jccs/data/`*EXP*. If not, you can easily find out which disk(s) contain your data via: `ls -d /juw26_?/jops/data/`*EXP*. All the `juw26` data disks are cross-mounted on `ccsops`, and since I'm on `ccsops`, the VEX file is also readily available without having to `cp` or `ftp` it.

`datasum.pl -ignore -sort /ccsops/expr`*EXP/vexfile YMoDy*`* >` *exp*`.dtsm`*N*

where *YMoDy*`*` corresponds to jobs run on a specific (previous) day. The *N* in the output file name just increases sequentially to let each day have its own file. If the data got spread out over more than one `juw26` disk, that can be handled with wild-cards before the *YMoDy* data path-names. I then use these daily `datasum.pl` output files as crib notes for reviewing the `plotweight.pl` plots made by the `cron` job, jotting down problems that need to be recorrelated. When the experiment is completely done, I'll use these files (so far, there's never been more than 8 for a given experiment) as guides for quickly editing the *exp*`.lis` file for driving the creation of the Measurement Set (¶1.f). I find it easier to keep on top of things this

incremental way, rather than be faced with a flood of things to review at the end (although I seem to be in the minority in this view....). Also, this can give more immediate feedback to the operators about any necessary re-dos.

ii — Basic guidelines for recorrelation:

In general, we want to make sure the correlation wrings the last drop of astronomical knowledge out of the data as recorded (within reason — we have only a finite time for the correlation process per experiment). Now that stations record onto disk-packs, equipment problems at our end are generally easier to see, because the *a priori* quality of the data is much higher. However, there are still certain ways in which a station can have a deleterious effect on their recordings, some of which we might be able to overcome — as well as spurious artifacts that our system can add to the data. Much of the art of support-scientisting comes in recognizing what is and isn't worthwhile to pursue; that it remains perpetually enthralling arises from the ever-shifting state of the correlator, experiments, and the set of behaviors their interaction spawns (¶App.1.g). Some basic guide-lines:

- loss more than one station, or loss of a vital station (*e.g.*, Ef/Gb/Jb_1/Wb_{arr} for an experiment requiring sensitivity, Sh for an experiment requiring the longest baselines). Obviously, recorrelation isn't useful if, for example, a station with bad playback has other problems precluding useful data (*e.g.*, it was stowed for high winds, the *exp*`.sum` file from sched shows that sources were below the horizon).
- generic playback problems, if exactly repeatable on different playback units, are probably are intrinsic to the recorded media. Tapes in this instance were trickier than disks, since it's not out of the realm of experience for the same tape-pass to play significantly differently on two units, or even twice on the same unit. Some station units may have lower weights (on some channels) for all stations — this is usually a sign of a malfunctioning board or connection between boards and the backplane.
- sometimes entire scans may be missed, for various reasons. These need to be picked up in recorrelation (one of the prime reasons for next-day checking of the previous day's correlation, and keeping a running list of remaining scans to be correlated — much reduced chance of something inadvertently being overlooked, and easier to recorrelate while the experiment's media are still mounted).
- Diagonal weights (¶App.1.g.$i.\gamma$) can occur in experiments that use 16 MHz subbands. Typically, the fringes to the affected station actually disappear before the onset of the characteristic diagonal weight signature — make sure that DW's have been re-done, and than any necessary preceding scan(s) were included in the re-do.
- The last scan may not have been fully completed. In some cases, you may be able to intuit this problem from the weight plot (*e.g.*, an established pattern of phase-referencing scan lengths seems to be broken), but in some cases this will

not be perceptible (time missed due to premature ending is too small a fraction of the overall job length). If you have suspicions, the quickest way to check is to look at last successfully recorded integration for the sub-job in the job's `data_handler.log` (§App.1.e.i).

- Ghost-data termination (¶App.1.g.$i.\epsilon$). This can be recognized from the weight plots as an event in which more than one station has 0-weight starting (or ending) at the same time. This is caused by a correlator bug in which the lack of data from one station can knock out the data for another station. The classic GDT event occurs after one station has left the array, never to return (on the basis of scans selected for this subjob); for a while correlator will continue to output data for it (the "Ghost data" effect — ¶App.1.g.$i.\delta$), but this will eventually cease ($\sim$7-12 min). When it does, another station's weight may also go to 0 at the same time. An alternate version of the GDT problem occurs at the beginning of a subjob, when some station doesn't participate in the array from the very beginning. Here, another station's weight may be stuck at 0 until the time the "late" station come into the array, at which time its weight will also jump up. The solution is to recorrelate, being more careful in the scan-range selection to avoid subjobs whose beginning or end miss a station.

f. ExportFile output. (≡ `lis`-file)

```
EI009A ei009a.vix Prod ei009a.ms ei009a.ms.UVF

+  708271659/1    - No0001 No0001 64 4 noX 0/0/0/216/222     100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708211008/1    - No0002 No0006 64 4 noX 0/1/25/216/176    100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708201355/1    - No0007 No0018 64 4 noX 0/0/0/216/459     100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708201440/1    - No0019 No0060 64 4 noX 0/0/0/216/1727    100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708201643/1    - No0061 No0145 64 4 noX 0/0/0/216/3512    100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708210902/1    - No0121 No0126 64 4 noX 0/1/0/216/230     100.0% GOOD PROD prod_cont ei009a.cont.vix rel2806
+  708230841/1    - No0001 No0006 2048 4 noX 0/0/0/72/424      100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708230915/1    - No0007 No0018 2048 4 noX 0/0/0/72/458      100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708230952/1    - No0019 No0060 2048 4 noX 0/0/0/72/1726     100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708270807/1    - No0061 No0080 2048 4 noX 0/0/0/72/815      100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708270911/1    - No0081 No0109 2048 4 noX 0/0/0/72/1196     100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708271322/1    - No0109 No0124 2048 4 noX 0/4/107/72/639    100.0% GOOD PROD prod_line ei009a.line.vix rel2806
+  708271410/1    - No0121 No0145 2048 4 noX 0/0/0/72/1051     100.0% GOOD PROD prod_line ei009a.line.vix rel2806
```

Listing 3: ExportFile output (from EI009A).

Listing 3 is the `ExportFile` output for EI009A. The first line lists the experiment name followed by the default values for:

- VEX file (default = *exp*`.vix`) — this will be used as the top-level VEX file when running `j2ms2` from this `lis`-file (*cf.* ¶2.c).
- `runjob.pl` profile (default = Prod) — not currently significant; `j2ms2`'s use of lines in `lis`-file is controlled by the "+/−" in the first column.
- Measurement set name (default = *exp*`.ms`) — this will be the name of the MS `j2ms2` will make when run from this `lis`-file.
- FITS-file name (default = exp.ms.UVF) — not currently significant; the actual name of FITS file(s) is controlled by `tConvert` (*cf.* ¶4.a).

Usually, you would want to change the name of the VEX file and output MS to match your specific situation. Changing the name of the profile and FITS file-name isn't currently necessary, but change them anyway to achieve internal consistency.

The main body of the `lis`-file contains a subset of the columns from the LogFile output, plus a first column that is "+" or "–". This first column controls whether to include data corresponding to the subjob/scan-range in its line or not: "+"=include, "–"=skip. This feature can provide you scan-by-scan control over what goes into the MS (and hence the FITS files). By the time you've gone through the weight plots and hunted for recorrelations, you should be in a position to be able to set the "+/–" field appropriately for each subjob. There other modifications you can make to the `*.lis`-files to provide you with fine control over exactly what data goes into the MS, to be discussed presently.

The example shown in Listing 3 is for an experiment with separate continuum and line correlation passes (2 different profiles, 2 different vex files). In real life you'd want to save separate `lis`-files for each profile/VEX-file by using the `profile`-selection feature under the ExportFile pull-down menu (¶1.b.*iii*), and edit the first line of each to make the fields appropriate for specific correlation:

cont: `EI009A ei009a.cont.vix Prod ei009a.cont.ms ei009a_1_1.IDI`

line: `EI009A ei009a.line.vix Prod ei009a.line.ms ei009a_2_1.IDI`

Don't worry about FITS-file naming convention, that will be discussed in due time (¶4.a). For disk-based experiments, the column after the Job/SJ ID contains a single hyphen (this used to be place for pass number in tape experiments). The next two fields are the begin and end scan contained within the subjob. To get ahead of ourselves a bit, it is by editing these scan-range boundaries that we can gain scan-by-scan control over which subjob contributes which data to the MS. The ExportFile feature in `showlog` sorts the subjobs within each profile by scan number, facilitating the creation of pre-time-sorted measurement sets.

i — Run in the standard way, `j2ms2` limits taking data from each subjob to the range of scans specified in each row of the `lis`-file. Currently, this scan-range checking occurs though a numeric comparison of scan numbers, although this interpretation of the scan-names is a little restrictive (they are not intrinsically constrained to be in the form of "`No`*nnnn*"). To date though, this has posed no practical limitations. `j2ms2`'s scan-range checking obviates the earlier need to edit "output" VEX files (a process that will therefore not be discussed here), and provides a record of how the Measurement Set was built, allowing reliable reconstruction of it, if ever necessary later. The scan-range checking therefore makes it much more straightforward to use "partial" subjobs, to avoid duplicate data entering the MS, and to keep the resulting MS in increasing-time order. Below are some points about the process of handling initially overlapping scan-ranges in different subjobs (arising because of re-correlations).

- The usual situation is when the latter part of a subjob has to be recorrelated. In this case, both subjobs would get a "+", and you just have to change the end-scan of the first subjob to be its last good scan.
    - If there was some overlap of good scans between the first & recorrelated subjobs, then you have some more flexibility in how to pick the end-scan of the first and the start-scan of the second — in this case it would generally be better to use the 2nd scan of the recorrelated subjob as its start-scan, because the first scan of any subjob loses ∼18s of data waiting for the correlator to synch up & data to flow out.
- The situation in which some scans in the middle of a subjob get recorrelated illustrates how to handle a more general situation (most typically arising because of a diagonal weight in the middle of a subjob that gets "fixed" by a gap in which the SUs reconfigure). The two keys are to have each line contain just the scans you want to include and to maintain increasing-scan order in the `lis`-file. The trick is that you may "re-use" a subjob: first taking some scans from the original subjob (before the "bad" scans), then moving on to the scans from the re-correlated subjob, and then picking up the rest of the scans from the first subjob, as illustrated in listing 4 (original subjob `508091658` = scans 18–39; re-do subjob `508100923` = scans 24–26).

```
+  508091658/1  - No0018 No0023 32 2 noX 0/0/0/160/1455   100.0% GOOD ......
+  508100923/1  - No0024 No0026 32 2 noX 0/0/0/160/728    100.0% GOOD ......
+  508091658/1  - No0027 No0039 32 2 noX 0/0/0/160/1455   100.0% GOOD ......
```

Listing 4: `lis`-file extract (from GT006) illustrating recorrelation of scan in the middle of a subjob.

ii — The program `checklis` provides a quick way to check for overlaps or skips in a `lis`-file from the linux prompt:

`checklis` *exp*`[-`*modifier*`-].lis`

`checklis` lives in `jops`'s path on either `ccsops`, `juw26`, or PCInt. It looks through the specified `lis`-file and returns any instances of overlaps or skips in the scan-range numbering on consecutive "+" lines. If no exceptions are found, then it just prints out the first & last scan. It's possible that skips are intentional (you may be making partial MSs), but overlaps would always be wrong (either introducing duplicate data or including subjobs out of time order). Subjobs put into the `lis`-file out of time order would trigger at least one of these exceptions, so are not separately mentioned.

iii — I generally try to make/put the *exp*`*.lis`-files in the main correlation-control directory for the experiment (`/ccs/expr/`*EXP*), so that it's easy for anyone else to find them if necessary.

2. Making the Measurement Set(s).

a. Setting up the Working Disk, Necessary Files

i — If you haven't already done so during clock searching, you'll need to create a working directory for the experiment on E^3 (`PCInt`). To log onto `PCInt`, use `ssh -X -l jops eee.jivepci.nfra.nl`. The "*eee*" machine is something that will put you onto one of the PCInt nodes. The data directories have mount points of the form `/data/`*NM*`/`, where *N* is (currently) in the range $0 \rightarrow 5$ and *M* is $0 \rightarrow 1$. If you have some problem with the `eee` address, you could alway log directly onto a node, the most typical way being `core-`*N*`-m.jivepci.nfra.nl`, where *N* refers to the same 0–5 range for the data directory mount points. In the past, some post-correlation steps have gone quicker when you log onto `core-`*N*`-m`, where *N* is the same as the mount point where your data lives.

Log-in as `jops`, and create a directory for the experiment on a mount point `/data/`*Nm*`/`*your-ID*`/`*EXP*. Some mount points are being used for additional purposes, so you can run `df -h | grep data` to judge where there will be enough room for the experiment's data during the FITS-creation stage — you generally shouldn't need more than ∼2.5 times the space taken up by the output correlator jobs. *your-ID* can be your name, initials, *etc.*, which really only serves to identify who "owns" the experiments that live in subdirectories under this path, which makes the subsequent housekeeping (¶5) easier to administer. `cd` to this directory.

ii — Once the experiment has been completed, a copy of the cover-letter template (*cf.* ¶4.b) and an experiment "summary" should live on ∼`jops/piletters` as *exp*`.piletter` and *exp*`.expsum` respectively. This `expsum`-file is useful in pipelining (*cf.* ¶6) to see which sources are private, and hence whose plots and post-SPLIT FITS files receive password-protection. Complain to me if these don't exist when you need them.

iii — `scp` the *exp*`.lis` file(s) over. `j2ms2` will use one of these files in making each Measurement Set, including all subjobs whose lines begin with "+" and excluding those whose lines begin with "–". The name of the "top-level" VEX file to use (*cf.* ¶2.c.*ii*) and the name of the output Measurement Set to make are in the first line of the `lis`-file. In most cases where you need multiple `lis`-files, there should be a different top-level VEX file and a different output Measurement Set in each. There may also be reasons to make a separate Measurement Set for each mode in a multi-mode schedule (*e.g.*, a multi-source spectral-line experiment with different Doppler-shifts for each source) if the PI wants. As touched on below (¶3.h.*iii*), multi-mode Measurement Sets also currently pose problems for standard plots. If you run `j2ms2` with the name of an existing Measurement Set, it will append data to that one, which is useful in some circumstances but definitely not in others (especially if you wind up "mixing" different top-level VEX files into the same MS).

Once the *exp*.`lis` file(s) are just as you want them, it's time to move on to processing the actual data themselves.

b. getdata.pl

`getdata.pl` is a perl-script that automates the retrieval of correlator jobs. The syntax is:

`getdata.pl -proj` *EXP* `-lis` *exp*`.lis`

You will first have to supply the `jops` password when prompted (**don't include passwords in scripts** you may make to run `getdata.pl`. The screen echos "`Ignoring -` *line-text...*" for each line in the `exp`.*lis* not having a "+" as the first (non-space) character, and then the data from the "good" jobs begin to flow. The screen echos all the files that are being pulled across.

i — `getdata.pl` gets jobs rather than subjobs, so it will get all subjobs of a multiple-subjob job even if only some of them are in the `lis`-file. Other than taking up a little more disk space, this has no impact on subsequent steps. If a job already exists on the working disk, `getdata.pl` is smart enough to skip over it and not waste time getting it again, writing "`Skipping` *JobID*" to screen (thus there's no penalty for having the same subjob listed multiple times with different scan ranges).

ii — For experiments needing multiple VEX files, remember to use multiple `lis`-files and to make sure the VEX file and Measurement Set names are correct in the first line of each of your `lis`-files. `getdata.pl` will automatically pull over the VEX file mentioned in the first line of the `lis`-file. This will later be needed as the "top-level" VEX file by `j2ms2` (*cf.* ¶2.c.*ii*). The principal reasons for requiring multiple top-level VEX files involve correlation runs using different subsets of the scheduled SB/pols — continuum/line, correlation by individual SB to provide higher $N_{\rm lag}$, *etc.*

iii — Behind the Scenes: `getdata.pl`

When pulling over the specified correlator jobs, `getdata.pl` also splits a long comment line in the output VEX files (the $6^{\rm th}$ line, showing the command-line syntax for the older version of `prep_job`, *cf.* ¶2.c.*ii*). Without this operation in each output VEX file, `j2ms2` would fail, producing a "`string too long at line` ℓ" error message (but without explicitly mentioned the file in the warning). This used to be accomplished via running a separate perl-script: `fixvex.pl */*.vex`.

c. j2ms2

`j2ms2` converts correlated data into a Measurement Set. The syntax you would most likely use is:

```
j2ms2 -v exp.lis
```

This will use the specified `lis`-file to obtain the name of the top-level VEX file and resulting Measurement Set, and to control what subjobs contribute to the MS, using the scan-range checking described above (*cf.* ¶1.f.*i*). In all cases, data at times outside of scans will not go into the MS. If for some reason, you don't want the scan-range checking, replace the `-v` in front of the `lis`-file with `-V`.

A single experiment that requires more than one MS will need a separate `j2ms2` run for each `lis`-file, which will result in separate MSs if the `lis`-files have different output-MS names in their first lines. All of the subsequent operations in ¶3–6 should also be performed for each MS and resulting FITS files.

```
j2ms2: Version of Tuesday 29 May 2007/11:50:50 begins

JIVEMSFiller: Adding GhostBusterFilter []
VEXperiment - Using VEXfile=eb032c.geod.vix
************ Experiment EB032C ************
Output is placed in: tst.ms
Data is written in the MS in the frequency domain

readCorrelatorSetup: '/data/31/rmc/EB032C/707121059/EB032C_707121059.vex'
readCorrelatorSetup: Skipping unknown  validity_mode = local
readCorrelatorSetup: Skipping unknown  feature_bits =

===> Find: Scan No0002 (#2): 1727+453/geodetic ** 17-Jun-2007/07:03:36
     Corr: Nf 32/Ti 2s/P [rr,ll,]/ABsw v. 2.0/BOCF 32Hz

FrequencyConfig: 'geodetic' (#0) recorded [rl]
     [#0: 8 MHz, from 6.61299 GHz, LSB (VEX channels: <0:r><1:l>)]
     [#1: 8 MHz, from 6.62099 GHz, USB (VEX channels: <2:r><3:l>)]
     [#2: 8 MHz, from 6.63299 GHz, LSB (VEX channels: <4:r><5:l>)]
     [#3: 8 MHz, from 6.64099 GHz, USB (VEX channels: <6:r><7:l>)]
     [#4: 8 MHz, from 6.65699 GHz, LSB (VEX channels: <8:r><9:l>)]
     [#5: 8 MHz, from 6.66499 GHz, USB (VEX channels: <10:r><11:l>)]
     [#6: 8 MHz, from 6.68299 GHz, LSB (VEX channels: <12:r><13:l>)]
     [#7: 8 MHz, from 6.69099 GHz, USB (VEX channels: <14:r><15:l>)]

Tue Aug 28 12:58:16 2007      WARN MeasIERS::fillMeas(MeasIERS::Files, Double) (file /data/00/casa-devel/casa/code/measures/implemen
t/Measures/MeasIERS.cc, line 94):
Requested JD54268.3 is outside the IERS table data range
Calculations will proceed with less precision

Tue Aug 28 12:58:16 2007      WARN MeasTable::dUT1(Double) (file /data/00/casa-devel/casa/code/measures/implement/Measures/MeasTable
.cc, line 6437):
No requested dUT1 data available from IERS tables.
Proceeding with probably less precision.


===> Finding visibilities without a Scan [not writing 'm]

===> Find: Scan No0003 (#3): 1740+521/geodetic ** 17-Jun-2007/07:05:24
     Corr: Nf 32/Ti 2s/P [rr,ll,]/ABsw v. 2.0/BOCF 32Hz

FrequencyConfig: 'geodetic' (#0) recorded [rl]
     [#0: 8 MHz, from 6.61299 GHz, LSB (VEX channels: <0:r><1:l>)]
     [#1: 8 MHz, from 6.62099 GHz, USB (VEX channels: <2:r><3:l>)]
     [#2: 8 MHz, from 6.63299 GHz, LSB (VEX channels: <4:r><5:l>)]
.
.
.
.
===> Find: Scan No0005 (#5): 1800+782/geodetic ** 17-Jun-2007/07:09:24
     Corr: Nf 32/Ti 2s/P [rr,ll,]/ABsw v. 2.0/BOCF 32Hz

FrequencyConfig: 'geodetic' (#0) recorded [rl]
     [#0: 8 MHz, from 6.61299 GHz, LSB (VEX channels: <0:r><1:l>)]
     [#1: 8 MHz, from 6.62099 GHz, USB (VEX channels: <2:r><3:l>)]
     [#2: 8 MHz, from 6.63299 GHz, LSB (VEX channels: <4:r><5:l>)]
     [#3: 8 MHz, from 6.64099 GHz, USB (VEX channels: <6:r><7:l>)]
     [#4: 8 MHz, from 6.65699 GHz, LSB (VEX channels: <8:r><9:l>)]
     [#5: 8 MHz, from 6.66499 GHz, USB (VEX channels: <10:r><11:l>)]
     [#6: 8 MHz, from 6.68299 GHz, LSB (VEX channels: <12:r><13:l>)]
     [#7: 8 MHz, from 6.69099 GHz, USB (VEX channels: <14:r><15:l>)]


Translated @ 1.71056 Mbytes/s

---------====== Summary ==========
ToMS:            12600
Filt:                0
NoSCN:           14616
OOSubjobRNG:      4872
```

Listing 5: `j2ms2` output (from EB032C).

Listing 5 shows an example of what you'd see on the screen as `j2ms2` runs. It begins by mentioning the `j2ms2` version and the top-level VEX file it uses, and and

gives the name and domain (*i.e.*, frequency or time) of the output Measurement Set. `j2ms2` then starts in on converting the data in the correlator output, scan-by-scan. During the process of data conversion, it lists the scan that it's currently working on, together with the relevant source, mode, start-time, and some basic frequency set-up information derived from the VEX file. Together with the vexsum.pl output, this provides a way of tracking progress as it runs. When run in the standard fashion as above, the scan number reported by `j2ms2` matches the scan-name's "number" from the VEX file. The description of the correlator parameters ($N_{\rm frq}$, $t_{\rm int}$, polarizations) lets you check whether there was some problem in the correlation (which of course, you would have most likely already found from `showlog`). More information about the frequency/polarization set up of the scan follows; this provides the means to check that you're using the proper top-level VEX file — if you've used the wrong VEX file in the first line of the `exp.lis` you will see unexpected information here (and get wrong frequency information in the MS). It's also possible that `j2ms2` will refuse to run, should $N_{\rm sb}$ in the top-level VEX file be inconsistent with what it finds in the correlator output. You're provided a running total of the number of rows created in the Measurement Set (not shown in the hard copy in Listing 5). When a new scan begins, you see the information for it. If there is a gap in the experiment or `j2ms2` encounters scans which you ruled out via the scan-range checking in the `lis`-file, you will see the "`Finding visibilities without a Scan`" message. When a subjob is completed, you see a summary of the number of visibilities written to the MS, filtered (mostly ghost data, *cf.* ¶App.1.g.*i.*δ), outside of scans (*i.e.,* in gaps), and outside of the `lis`-file specified scan ranges. Since the *exp*`.lis` is (or should have been...) in scan order, the resulting Measurement Set will be in time order.

i — A more manual way of running `j2ms2` is:

`j2ms2` {`-d time`} `-o` *MSname* *jobID/subjobID*

Here, the "`-d time`" is included to make a lag-based MS, otherwise the default frequency-based MS results. The "`-o`" parameter controls the name of the MS produced. Multiple `j2ms2` runs using this format with the same MS name will continue building up the MS by appending the data for the currently-specified subjob at the end. In this usage, the top-level VEX file must have a name *dirname*`.vix`, where "*dirname*" is the name of the working directory (*i.e.*, easiest if the correlator jobs live in, the MS goes to, and `j2ms2` is run from the same directory) — usually *dirname* = *EXP*. Failure to have this name match results in a cryptic "`Aargh - ... [StdException: Experiment is not COF.] (j2ms2.cc:804)`" error right at the beginning of the `j2ms2` run. If you have to juggle multiple top-level VEX files in this approach, it's best to `ftp` them all to your working directory on E^3, and `cp` the appropriate one to *dirname*`.vix` just before running `j2ms2` (especially useful if you have a shell-script of several of these single-subjob `j2ms2` runs). **Of course, the intent of driving production via the `lis`-file shields you from all this complication.**

3. Operations on the Measurement Set(s).

Measurement Set ops usually occur in the aips++ environment. Run

```
glish {-l logger.g}
```

and wait for the "-" prompt. The "`-l logger.g`" option starts a GUI dialogue-box with logging information. Without the "`-l logger.g`", some of the logging information would go directly to the terminal. Depending on what you're doing, either may be preferable.

There are a couple tricks that may make your experience more enjoyable:

- make your xterm wider — glish function calls can go from line to line, but once you're on the second line you can't get back to the first (say to backspace over a mistake), and even on the second line backspacing is annoying ($2^{\rm nd}$ line reprinted for each backspace).
- before starting glish, check the `tty` in the xterm you're working in, in case you'll need to safely kill it (& not someone else's) from another xterm. On `PCInt`, this really hasn't been a problem anymore.
- `Ctrl-a` & `Ctrl-e` bring you back to the beginning and end of the line respectively. `Up-arrow` retrieves the history of commands; the well-planned session will involve mostly `Up-arrows` and minor editing of previous commands.

Basic Measurement Set architecture to keep in mind:

- the main table comprises a set of rows. Each row contains 2 stations (autocorrelation: both stations the same), a time, a data array with dimensions $N_{\rm pol} \times N_\nu$ (or $N_{\rm lag}$ for a time-based MS), a weight vector with dimensions of $N_{\rm pol}$, and various pointers to sub-tables that allow one to figure out:
  - what frequencies correspond to the different N_ν frequency points, what the net sideband is, *etc.* (the `DATA_DESCRIPTION` & `SPECTRAL_WINDOW` tables)
  - what the polarization mapping is for the $N_{\rm pol}$ entries in the `DATA` column (the `DATA_DESCRIPTION` & `POLARIZATION` tables)
  - what source is being observed (the `FIELD` table)
  - information about the stations (the `ANTENNA` table)
  - information about the job from which the row came (the `PROCESSOR` table).
- any column from the main or sub-tables can be read into variables in glish, an interactive/scripting language that has some of the features of FORTRAN, C, IDL, and pascal, including support for pgplot.

When you run a "program" in glish, you're actually calling functions by passing a command-line of parameters. Most much-used functions are automatically loaded whenever you start glish (as user `jops`). Other functions have to be explicitly loaded via the `include '`*FUNCTname*`.g'` command. The expected syntax and default

values can be seen by simply typing the function name without any arguments (if you get just `F` back, then the function isn't loaded). You can pass parameters to the function according to the syntax returned; if you follow the parameter order, then you don't have to supply the parameter names. Unsupplied parameters revert to their default values. You can omit some parameters, but then the first one (and all subsequent ones) you do supply after the omission requires a name. The same applies if you include a parameter out of order. Some examples for a fictitious function (here, $\surd$ means good syntax, $\times$ means bad syntax):

full syntax: `funct(MSname, weight=0.2, outfile=' ', live=T)`

$\surd$ `funct('MyMS.ms', 0.4, 'file.out', F)`

$\surd$ `funct('MyMS.ms', 0.4, 'file.out')` — live stays `T`

$\times$ `funct('MyMS.ms', 'file.out')` — `weight` omitted, `outfile` parameter name not specified — will try to assign "*file.out*" to the parameter `weight`, which may have adverse consequences depending on how carefully the function checks for expected variable type, *etc.*

$\surd$ `funct('MyMS.ms', outfile='file.out')`

$\times$ `funct('MyMS.ms', outfile='file.out', F)` — once `outfile` specified, all subsequent parameter names also need to be given

$\surd$ `funct('MyMS.ms', outfile='file.out', live=F)`

$\surd$ `funct(live=T, weight=0.2, MSname='MyMS.ms', outfile='file.out')` — order doesn't matter if all names given

Generally, it's much less typing if you get in the habit of entering the parameters in order. Most functions have been tailored to have at least a fairly reasonable order.

Functions return a single "thing" (integer, string, record, *etc.*). If the function isn't assigned to a variable (using *a* `:=` *function(parameters)*). then the return goes to the screen (which may or may not be desired). glish doesn't remember internal variables in functions once the function returns (unlike, say, IDL does).

A quick way to get a summary of what's in your MS (stations, sources, frequency set-up) is with the `mssum` function: `mssum('`*MSname*`')`. Listing 6 shows the result for EZ015 (a single-mode experiment).

```
- mssum('ez015.ms')
 Antenna ID 0 is Ef EFLSBERG         Source ID 0 is 487
 Antenna ID 1 is Mc MEDICINA         Source ID 1 is 515
 Antenna ID 2 is Wb WSTRBORK         Source ID 2 is 947
 Antenna ID 3 is On ONSALA85         Source ID 3 is 0234+285
 Antenna ID 4 is Tr TORUN            Source ID 4 is J0801+4401
 Antenna ID 5 is Nt NOTO             Source ID 5 is J0808+4950
 Antenna ID 6 is Jb JODRELL1         Source ID 6 is 4C39.25
                                     Source ID 7 is J0932+5306

 Subband 0:  8.00 MHz @ 1.63061 GHz / 64 fp (LSB) [RR LL RL LR]
 Subband 1:  8.00 MHz @ 1.63849 GHz / 64 fp (USB) [RR LL RL LR]
 Subband 2:  8.00 MHz @ 1.64661 GHz / 64 fp (LSB) [RR LL RL LR]
 Subband 3:  8.00 MHz @ 1.65449 GHz / 64 fp (USB) [RR LL RL LR]
 Subband 4:  8.00 MHz @ 1.66261 GHz / 64 fp (LSB) [RR LL RL LR]
 Subband 5:  8.00 MHz @ 1.67049 GHz / 64 fp (USB) [RR LL RL LR]
 Subband 6:  8.00 MHz @ 1.67861 GHz / 64 fp (LSB) [RR LL RL LR]
 Subband 7:  8.00 MHz @ 1.68649 GHz / 64 fp (USB) [RR LL RL LR]
T
```

Listing 6: `mssum` output (from `ez015.ms`).

−Ω. {scanfix.g} `scanfix.g` has a long history as a utility to fix up various things in the MS. By now, developments in `j2ms2` have rendered these corrections moot in normal circumstances. We've recently come across an instance in which data in gaps were not properly excised in an *e*-VLBI experiment. Some background in this:

- `j2ms2` omits data coming off the correlator for stations at times in which the station isn't participating in a scan (which include all stations in gaps). However, *e*-VLBI needs to see no gaps in the incoming data. One way to handle this is to make sure the VEXfile used to control the correlation has its scans extended to run up to the start time of the following scan. The stations observe based on the original schedule with gaps.
- The apparent gap-less `$SCHED` section of the VEXfile propagates into the output vexfiles, thus `j2ms2` has no way of seeing that there are indeed gaps from the stations' point of view. Thus all the periods when the stations are off source make it through into the MS, and hence the FITS file.
- `scanfix.g`, given the original *exp*`.skd` file, could flag the data in the MS corresponding to times in gaps. Subsequent processing steps wouldn't use them (except currently `standardplots`). The data would go into the FITS files, but with negative weights.

So until a fix at the correlation-control stage is enacted, the following step will excise the data in gaps manually for *e*-VLBI observations:

```
include 'scanfix.g'
scanfix('MSname', 'SKDname', evlbi=T)
```

where the *SKDname* refers to the original `skd`-file. You can `scp` this from
`/ccs/var/log2vex/logexp_date/`*EXP_yymmdd*`/`*exp*`.skd`
or from `vlbeer` as user `evn` at `vlbi_arch/`*monYY*`/.latest/`*exp*`.skd`
(The *yymmdd* refers to the observing date — the initial day for experiments that cross day boundaries; *cf.* ¶4.c for where to get this date.)

a. {jfilt}

`jfilt` (run at the linux prompt outside glish) combines multiple correlation passes. The principal application is in experiments that have more than 16 stations at any given time, which require (at least) three correlator passes to complete, with each pass involving a different subset of stations. There are a couple MS-conditioning tricks that we've picked up in the two experiments that have required `jfilt`, but I've never gotten around to documenting these (they relate to ensuring the integration epochs in the different pre-`jfilt` MSs are compatible & adjusting them back afterwards, and to ensuring the appropriate data gets through `jfilt`'s selection criteria, *e.g.*, excising diagonal weights that fall in periods also having synch loss). We haven't had a > 16-station experiment since the Nov'03 session, so I'll refrain from further discussion here.

b. {badcorr.g}

`badcorr.g` flags visibilities that have good weights (above a weight cut-off, default $W_{\min} = 0.2$), but also have amplitude in their DC frequency point (first one for upper sideband channels; last one for lower sideband channels) greater than some unreasonable value (default = $\mathcal{A}_{\max} = 10.0$). This was created to handle a very specific correlator bug that seems not to have recurred recently. A fuller description of `badcorr.g` is in the Appendix (¶App.3.b).

c. {polflag.g}

`polflag.g` takes care of cases where a station records a single polarization into both pols in a dual-pol experiment — most typically On & Ur having only LCP at K-band (and in the past, Sh having only LCP at L-band). The syntax is:

`polflag('`*MSname*`',` *antenna*`,` *subband*`,` *pol*`,` *scan*`)`

where *antenna* is the affected station (passed as an antenna-ID integer), and *pol* can be `'L'`, `'R'`, or `'C'` (this is to specify the unrecorded polarization). `polflag.g` can handle only one antenna at a time (advertized as a safety feature, but really to make the coding more straightforward in handling the effects of baseline order for cross-pols). New selection criteria include *subband* and *scan*; the need for this extra filtering has come up in specific experiments. Currently, *scan* can be only a single scan, passed by scan-number.

`polflag.g` will flag both the parallel- and cross-hand Stokes visibilities associated with *pol* on all baselines & autocorrelations involving *antenna*, taking into account the "baseline order" for the cross-pols, if there are any. Therefore, if a station recorded only LCP, then the *pol* parameter should be `'R'`. There's also a special possibility `pol='C'`, which will flag both cross-pols but neither parallel-hand pols for baselines involving *antenna*. The rationale behind this option was handling cases in which *antenna* recorded linear polarizations (while everyone else had circular pols).

d. 2-bit van Vleck Correction

The correlator always works in a 2-bit mode (for 1-bit data, the "magnitude" bit in each sample is set to 1). The correlator now outputs (lag-based) correlation functions for a baseline/autocorrelation without any correction for the participating stations' "sampler statistics": the distribution of magnitude (high/low) and sign (+/−) bits. The distribution of the magnitude bits in particular can make a noticeable difference to the magnitude of the correlation function. With the correlator's current output, the amplitude of the central lag of an autocorrelation is directly proportional to the fraction of high-bits (hereafter f_h) — unity autocorrelation amplitude corresponds to $f_h = 0.364$. Of course, for $f_h \neq 0.364$, the raw correlator output has the intrinsically nonsensical result of non-unity autocorrelation peaks. The amplitude on baselines can likewise be affected as a function of the f_h of the two participating stations' data streams. The program `2bitVV` computes the appropriate 2-bit van Vleck correction for each visibility in the MS as described below.

i — First, however, we can use the linear relationship between the autocorrelation peak amplitude and f_h to provide feedback to the stations. `2bitVV` can correct the amplitudes of the correlation functions in the MS, but if f_h is significantly different from our ideal 0.364, there will be an irrecoverable loss of sensitivity; in the extreme case, $f_h = 1$ would be the same as 1-bit recording. Near-zero f_h are potentially even worse in terms of SNR. So it's in the EVN's advantage if the stations could set the BBC attenuators on their VCs/BBCs such that $f_h \simeq 0.364$.

The autocorrelation-amplitude plots generated by the daily cron-job (*cf.* ¶1.d) provide this information, without the scaling to f_h. There is a type of plot in `tplot` that directly shows f_h (plotted as %, with a fixed y-axis range of $-1\% \leq f_h \leq 101\%$). The most direct way to obtain this is:

$a :=$ `readms('`*MSname*`', auto=T, firsttime='`*yyyy/doy/hh:mm:ss*`', dur=`Δt`)`

`tplot(`a`, type='samp', yplot=`$N_{\rm sta}$`,xplot=`$N_{\rm sb}$`,`
`  multisub=F, outfile='`*PLOTfilename*`')`

In `readms`, the parameters `firsttime` & `dur` can be used to read in only part of the MS by time (Δt minutes starting from `firsttime` — *cf.* ¶3.j.*i*). Especially for experiments with short $t_{\rm int}$, it would be faster to plot a few instances of a few minutes than the whole experiment. The parameter `auto=T` limits the MS-read to only autocorrelations. In the `tplot` call, only the parallel-hand polarizations are used (cross-pol correlations are treated as ordinary baselines in the correlator). Parameters in `tplot` not previously discussed include:

- `multisub=F` makes separate plots for each SB; this makes it easier to see which (if any) SB/pol has problematic f_h without having to worry about the color-scheme (in these plots, red=LCP, blue=RCP).
- Depending on which of $N_{\rm sta}$ or $N_{\rm sb}$ is larger, you may want to use `landscape=T` (default is to make a portrait postscript file). Typically, `xplot=8`

doesn't look good in portrait. You can always put `yplot`$< N_{\rm sta}$, and the output file will span >1 page.

The mapping between SB/pol on the plots and BBC at the stations can be figured out from the `$FREQ` section in the top-level VEX file. This may differ from station to station, depending on the type of recorder used. The `'samp'` plots do not do the reverse-FFT back to lag space to calculate f_h, but rather make use of the property that the integral of a function (the real freq-space autocorrelation bandpass) over all ω is equal to the value of its transform at the origin (the lag-space autocorrelation peak). It sums the `real` over all frequency points to approximate the integral over all ω.

ii — `2bitVV`

`2bitVV` (run at the linux prompt outside glish) applies the 2-bit van Vleck correction to the MS in-place. Unlike other programs in the section that flag data, `2bitVV` actually changes the visibility data. If for some reason it fails in the middle of a run, don't just restart it on the same (partially-corrected) MS. The choices are to remake the MS or to revert to a previously saved pristine copy of the MS via `cp -r` *MSname MSname*`ORIG`). Now that `2bitVV` is done outside glish, the chances of encountering a problem during running is low enough it's probably not worth the time/effort to save a pristine copy of the MS prior to running `2bitVV`.

The syntax is:

`2bitVV` {`-H` $f_{h_{\rm min}}$} {`-W` $w_{\rm min}$} *MSname*

Here, the optional parameters `-H` & `-W` provide the means to flag data for which the process of computing the 2-bit van Vleck correction would likely be suspect: defaults are $f_{h_{\rm min}} = 0.01$ and $w_{\rm min} = 0.1$ (below the weight cut-off used for most other reasons). These parameters control "*targeting*"" individual integrations in an autocorrelation BBC (SB/parallel-hand pol). Any integration for a baseline SB/pol in which a *targeted* autocorrelation BBC participates is in turn *targeted*. Any such *targeted* visibility has it weight set negative in the MS, which will follow into the FITS file. You might be able to use insights gained through plots from ¶3.d.*i* to adjust $f_{h_{\rm min}}$ if necessary.

e. fixfbs

`fixfbs` (run at the linux prompt outside glish) applies a post-correlation correction for the phase-slope error across the band resulting from the fractional bit-shift, which is most noticeable near times of zero delay-rate. Here, the slowly changing residual delay error, caused by the difference between the (discontinuous quantized) integral-lag delay by which the two bit-streams can be adjusted with respect to each other and the actual continuous $\tau(t)$ geometric model, results in the phase slope across the band pivoting about the central frequency point. This could affect es-

timation of delays in fringe-fitting and create sawtoothy plots of vector-averaged amp(t). As long as the vector-averaging takes place over a range of frequency points symmetric about the center, the phase(t) plot should remain less affected since the pivoting takes place about the central frequency point. However, for oversampled data, the pivoting appears to take place $1/(2os)$ into the band, where os is the oversampling factor. Figure 4 shows before/after amp(t) plots for a subset of N06M2.

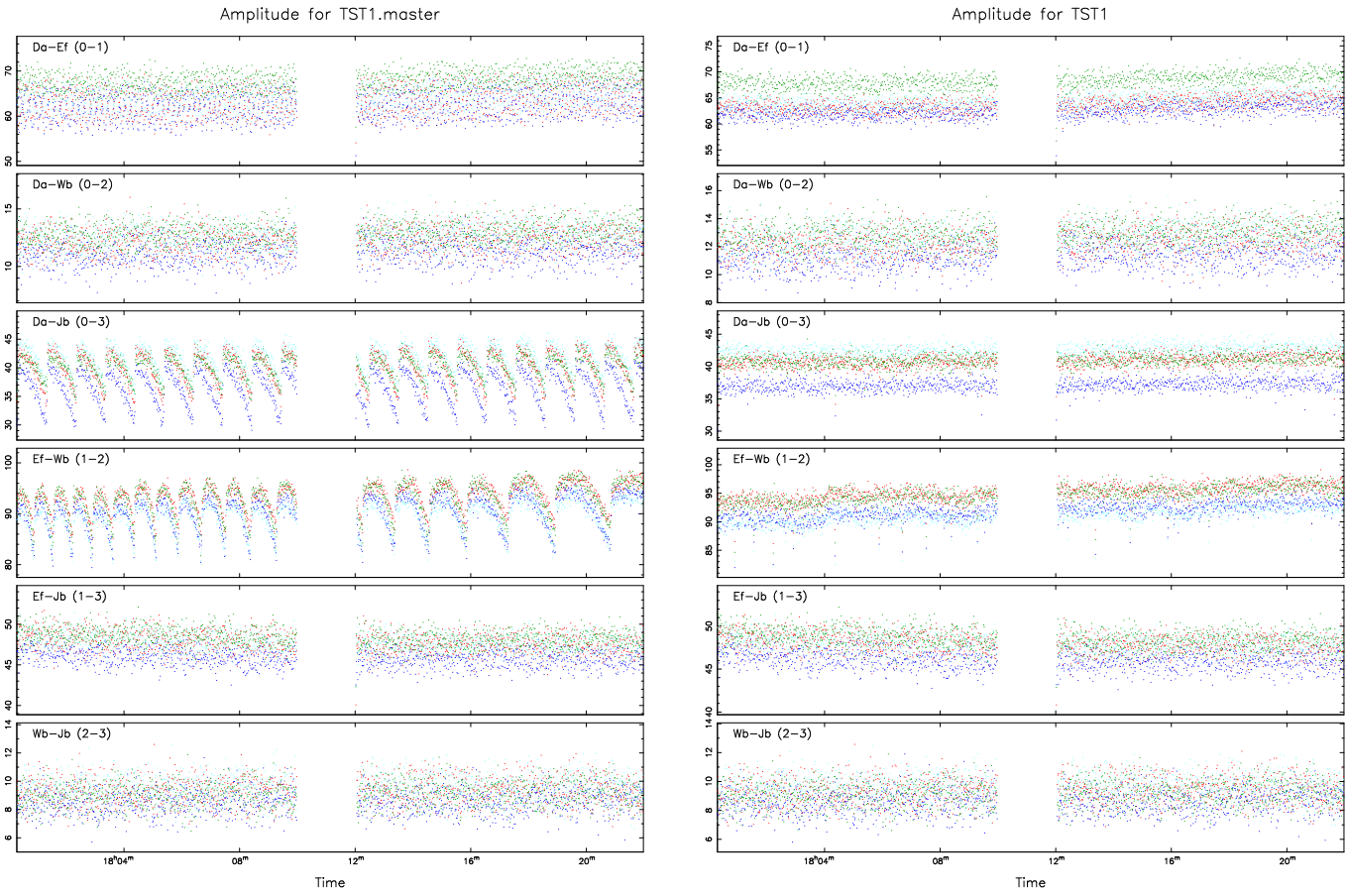

Figure 4: amp(t) plots before (left) and after (right) `fixfbs` (from N06M2).

i — For safety's sake, it might be prudent to make a copy of your MS at this point (via `cp -r` *MSname MSname*`0`). `fixfbs` has been tested relatively thoroughly, but there are some kinds of experiments it hasn't seen. Like `2bitVV`, this is the kind of program that would not recover well from stopping part-way through. It's also important to remember to run `fixfbs` only once per MS; the modifications it makes in successive runs would be cumulative — the first time fixes the FBS, a second run would add back in the FBS error with the opposite sign, a third run would double that, *etc.*

ii — The syntax for `fixfbs` is straightforward, with a slight complication (currently) in the formatting of the MS name:

`fixfbs` *./MSname*

The "`./`" is currently required, and there should be no trailing "`/`" at the end of the MS name (as would occur should you use tab-completion half-way through *MSname*). Like `fixuvw.g` & `plyflg.g`, `fixfbs` currently requires the jobs to be on disk, since it finds the *a priori* model polys files there.

iii — If you have saved the pre-corrected MS, you can use `movie.g` to compare the before/after phase across the band in animations. The syntax (without deeper explanation) is:

`a := readdata('`*MSname*`', `*Ant1*`, `*Ant2*`, firsttime='`*yyyy/doy/hh:mm:ss*`', dur=`Δt`)`

`movie(a, `*N_sb*`, `*M_pol*`, 'phas', weight=0.2, wait=30, plotline=F)`

Here, *Ant1* & *Ant2* are the index numbers for the two stations (*e.g.,* 0,3 or 1,2 for the affected baselines in figure 4). The time-range is most quickly specified with the `firsttime` and `dur` parameters (Δt in units of minutes — *cf.* ¶3.j.*i*). `movie.g` will open a pgplot widget, and show the phase across the band for each integration of the selected subband/polarization, specified by the subband & pol indices. You can click on the `done` button when the animation is over. A subsequent run will open another pgplot widget, whether you've dismissed the previous one with `done` or not. The `wait` parameter controls how quickly the animation-refresh occurs; larger numbers lead to slower animations.

f. fixuvw.g

`fixuvw.g` corrects values for the UVW coordinates in the MS to be consistent with the *a priori* model used during correlation. To do this, it uses UVW polynomials written to the polys files in a way comparable to the delay and phase model polynomials. These UVW overwrite the existing ones, so there's no danger to multiple runs. This is run in glish:

`fixuvw('`*MSname*`')`

Like `fixfbs` & `plyflg.g`, `fixuvw.g` currently requires the jobs to be on disk, since that is where the polys files live.

g. {plyflg.g}

`plyflg.g` flags time ranges where the *a priori* correlator models lead to known problems for the output data. Currently (correlating in local validity), the only applicable situation is when an *a priori* delay-rate on a baseline is close enough to zero that the phase-cal tones at the two stations correlate with each other, causing brief spikes in the bandpass corresponding to frequency points where the phase-cal tones lie. This only needs to be run for stations that have phase-cal on, which can be checked from plots of the autocorrelation real(ν). If the experiment isn't spectral line, you may not have enough resolution to tell for sure; there may be higher-lag correlations from the clock-searching (plots should be in the yellow experiment folder).

The first sub-paragraph discusses `plyrecon.g`, which can be used to get a feel for where such model-based flaggable events may lie — this is fast because it doesn't read the MS to see what data actually exists. The second sub-paragraph gives some basic functions you can use to investigate the time-range found by `plyrecon.g`. The third sub-paragraph discusses flagging the MS for real with `plyflg.g`.

i — As a first step, you can run `plyrecon.g` to get an easier-to-read summary of time-ranges that "should" have events to flag. Once back in glish after completing `2bitVV`, run `plyrecon.g` for standard $DR{\sim}0$ flagging:

`plyrecon('`*MSname*`', '`*OUTPUTfilename*`')`

Here, *OUTPUTfilename* will hold the summary of events it found to flag — a list of time-ranges per baseline, sorted in roughly time-order (actual sort = by job). If the experiment used Wb single dish (not fed through the adding box), also include the parameter `WBarr=F`. If you have a subset of stations which you know did not have phase-cal on, you can also include `noDR=['`Sta_1`', '`Sta_2`',....]`, where Sta_i are the two-letter station abbreviations of the stations without phase-cals. An extract from output for GP042 looks like:

```
** From job 505102134 **
Br-Kp
   Rate event #01:  063/04:49:01.000 - 063/04:50:36.000

** From job 505102204 **

** From job 505102346 **
Cm-Ef
   Rate event #01:  063/05:54:12.000 - 063/05:57:00.000
   Rate event #02:  063/05:58:08.000 - 063/06:07:17.000

Jb-Sc
   Rate event #01:  063/05:59:10.000 - 063/05:59:51.500
```

Listing 7: `plyrecon.g` output (from GP042).

There is a further parameter (`minwrap`) to control the extent of flagging per $DR{\sim}0$ event. It might be useful to reduce this from its default of 4 for experiments with low $t_{\rm int}$. The subparagraph below describes how to check whether the time-ranges

found by `plyrecon.g` are appropriate. There are even further capabilities to flag fringe-rate harmonics (so far only needed twice for Jb_1) and/or delay-based event where tape(disk)-frame headers correlate with each other (occurring for Mk4–Mk4 baselines under global-validity, which use currently don't use). We won't go into these here; see the *Operational Impact of Delay/Rate Problems* guide on the JIVE how-to wiki.

Since every advantage has its disadvantage, the downside of the speed gained by not reading the full MS data is that `plyrecon.g` depends entirely on the polynomial models. These are computed during the job-preparation stage and last over the entire time-range selected for correlation. If the job stopped earlier than this, the polynomial models would last longer than do the data from that job. If your experiment has lots of jobs that were truncated before finishing or otherwise had part of their initial duration re-done, then there may be some duplication in the `plyrecon.g` output. As an example, if the first job was run trying the entire experiment, but then a second job had to be re-correlated over the 2nd half of the experiment, then any event found by `plyrecon.g` in the 2nd half of the experiment will be listed twice (once in each of the two jobs). If you prepared the `lis`-file correctly, there would not be such duplication in the data themselves.

ii — `plyrecon.g` only uses the *a priori* model polynomial files associated with each job/station; it makes no attempt to check whether there are actually any data in the MS corresponding to the full time ranges found. If you want, you can alway plot the time-ranges corresponding to these events (say, if you're unsure whether some stations really do have phase-cal on or not, or if you want to investigate fine-tuning the `minwrap` parameter). The steps for doing this include:

`mssum('`*MSname*`')` — check the antenna numbers for your stations

a `:= readms('`*MSname*`', F,`*Ant*`,`*Bsln*`, firsttime='`*yyyy/doy/hh:mm:ss*`', dur=`*Deltat*`)`

`tplot(`*a*`, F,F,F,F,[1,2], 'amp', yplot=2,xplot=1)`

For more syntax about these glish functions used here, see the discussion in ¶3.j. Note that you don't have to "`include`" these functions, as they are automatically loaded upon starting glish (if logged in as user `jops`). Briefly, in `readms`, *Ant* & *Bsln*, here integral antenna-numbers matching the return of `mssum` (2-letter station abbreviations are also allowed), function as `ANTENNA` & `BASELINE` in classic AIPS, respectively (the first `F` would be used for passing an array of stations, from which all possible baselines/autocorrelations would be formed — this is usually the way `readms` is called). They can be single stations or a vector of stations, passed by antenna number. The `firsttime` & `dur` parameters control the time-range read in from the MS; these can be set to a few minutes before the beginning and a few minutes after the end of the `plyrecon.g` time-range to investigate (units of Δt are minutes; *cf.* ¶3.j.*i*). The combination of these two is to read in `dur` minutes of

data following the time in `firsttime`. In `tplot`, the 4 `F`'s correspond to `antenna, baseline, subband`, and `fg` (frequency group). Their being `F`'s denote use all data (the selection was done in reading the data into the variable with `readms`. The next parameter is `pol` — here only the two parallel-hands (if not a cross-pol experiment, this can also be `F`). `yplot` and `xplot` control the layout of plots on a page; if plotting more than 1 or 2 baselines, you'll want to supply appropriate numbers here. The default weight cut-off for such amp(t) plots is 0.2, which suffice in almost all cases (otherwise you could also include a `weight=`w parameter. By default, all SBs/pols will be combined into a single plot per baseline, color coded following the scheme in ¶3.j.*i* (Table 2).

iii — Once you're happy with any `plyrecon.g` investigations, you can run `plyflg.g` for standard $DR{\sim}0$ flagging:

`plyflg('`*MSname*`', '`*OUTPUTfilename*`')`

Here, *OUTPUTfilename* will hold the list of MS-rows flagged, with each visibility on a separate line (including the baseline, integration epoch, and value of the DR). There is an `undo` option, if necessary (by default, `undo=F`, to undo the `plyflg` flagging, include `undo=T` in the parameter list). There are additional parameters similar to `plyrecon.g`: `noDR`, `minwrap`, parameters for flagging phase-rate harmonics, and parameters for flagging Mk4–Mk4 delay events for global-validity correlations. Like `fixfbs` & `fixuvw.g`, `plyrecon.g` and `plyflg.g` currently require the jobs to be on disk, since that is where the polys files live.

h. standardplots.g

`standardplots.g` makes a set of CBD-approved plots, whose existence officially permits us have our policy of automatic tape/disk release a couple weeks after distribution of the experiment to the PI. In other words, the plots provide PIs sufficient overview of the correlation that they don't need to analyze their data through AIPS themselves before they're confident that the correlation went okay. Looking from the other point of view, the plots will let the PIs know if there was a problem within the 2-week period before tapes/disks become eligible for release to get back in touch with us if they think re-correlation is necessary.

i — The syntax for standard plots is straightforward (and provided on-screen each time you `include` the program). Within glish, enter:

```
include 'standardplots.g'
standardplots('MSname', 'Sta', 'Src')
```

You can also use two sources, in which case you need to use double- quotes (empirically, double-quotes also seem to work in the single-source case as well):

```
standardplots('MSname', 'Sta', "Src1 Src2")
```

Here, *Sta* is the 2-letter station abbreviation you want to use as the reference station in the plots (baselines will be *Sta*–*). Frequency-based plots show 1-minute of data taken from the middle of automatically-selected scans; *Src* (either one or two) will control the selection of these scans. If one source is given, then the first & last scans of that source will be used (if only one scan of the source exists, then only the middle 1-min interval of that single scan is used). If two sources are given, then a 1-min interval from the first and the last scan from the combined set of scans of the two sources are used. There is a consequence of this for typical phase-referencing experiments: if the fringe-finder(s) don't occur at the very beginning or end of the experiment and you specify a fringe-finder & the phase-reference source in `standardplots.g`, you will not get plots from a fringe-finder scan. The weight cut-off for the plots (other than the weight plot) is 0.7. You can change the weight cut-off with the `wthresh=`*w* parameter, and can skip the weight plot altogether with `NoWgt=T`. Since the weight plot is made for the entire experiment, there is no need to remake it if you want to re-run `standardplots`, say to try a different reference station or different reference source(s). The weight plot can often take the longest to make, so skipping it in subsequent runs could save noticeable time.

ii — `standardplots.g` makes four types of plots:

- Weight *vs* time plot (plot filename: *MSroot*-`weight.ps`, where *MSroot* is the name of the MS, not including the "*.ms*") — Shows the station (autocorrelation) weight for each station in the MS. The plot covers the whole extent of the experiment (or at least of the MS, which should be the same), and each

integration is plotted as a separate "." (plot can get big). Only the first four SB/pol are plotted: either SB 0&1 for dual-pol or SB 0—3 for single -pol.

- Autocorrelation amplitude *vs* frequency plot (plot filename(s): *MSroot*-`auto-`*N*`.ps`) — One plot for each of the 1-min intervals from the 1 or 2 selected scans (the *N* in the file name refers to these two scans, taking the value 1 or 2). Shows all pols of each station, with each SB in separate plots (thus $N_{\rm sta} \cdot N_{\rm sb}$ individual plots). The y-axis ranges of the individual plots scale separately, not necessarily going down to 0.
- Baseline amplitude/phase *vs* frequency plot (plot filename(s): *MSroot*-`cross-`*N*`.ps`) — One plot for each of the 1-min intervals from the 1 or 2 selected scans. Shows all pols for each baseline to the reference station, with the SBs shown in separate plots. Amplitudes are on the bottom of each individual plot; phases are on the top.
- Baseline amplitude/phase *vs* time plot (plot filename: *MSroot*-`ampphase.ps`) — Shows the the amp and phase on baselines to the reference station for a time-range starting ∼30min before and ending ∼60min after the first of the scans used for the frequency-based plots. Only one pol from the first two SBs (or from just SB0, if only 1 subband) are plotted. Each integration is plotted as a separate ".", vector-averaged over the middle ∼80% of the band. All sources in that fall in the time-range shown are plotted, and color-coding marks the source changes.

Under the current printer-server system in the building, you need to take an extra step when printing out multi-page postscript level-1 plots (as are the standard plots, and other plots made via pgplot in glish). Use the syntax:

`lprps1 -p '-Pxrxict'` *filename*

Where here, the color printer is being used.

iii — Limitations. The *raison d'être* of standard plots is to allow the PI to have confidence in the correlation such that the original recording media may be released prior to the experiment being analyzed in any real depth. The CBD has approved standard plots to serve in this capacity, so in terms of operations, all is well. However, there are specific characteristics of the standard plots that leave them a bit short of this of this goal in reality. First, they show only a small part of the data, especially for experiments with many subbands (*e.g.*, 1 Gb/s observations) — the weight plot shows only the first 4 "channels", avoiding the higher BBCs which are usually more troublesome than the lower ones, and the amp/phs(t) plot shows just ∼1.5 hr of data on 2 channels to baselines to just one station. Besides the sparse channel-sampling, this latter plot is quite insensitive to correlator-based problems, which can be baseline- rather than station-based. Second, there is not a great deal of leverage in terms of picking the scans to use for the frequency-based plots, especially important in globals where it's quite common for there to be no single scan

observable by all stations. For sources observed repeatedly in the schedule, there is no means to steer the standard plots to specific scans, short of making (& processing through the steps up to here) a sub-MS specifically crafted to force standard plots to pick the desired scan(s). Even in this case, care has to be exercised with the resulting file-names to avoid overwriting. Third, a similar lack of leverage exists in terms of picking the 1-minute interval used by the frequency-based plots. The most common problem is that the middle of the scan comes before some of the slower stations have slewed on-source (especially in the baseline plots, since the chances are the reference station is going to be big, and hence more likely to be slow). Occasionally, the 1-min interval can be too late, in the case of an undetected diagonal weight (required conditions: the scan was surrounded by gaps, and long enough to have lost fringes but too short to have displayed the characteristic diagonal-weight signature that seems to follow a couple minutes after loss of fringes — thus nothing for the operators to catch during correlation). Also, in a multi-mode MS, standard plots just "see" only the first mode (not applicable in a multi-mode experiment that has individual-mode Measurement Sets, *cf.* ¶2.a.*iii*).

All the items in this sub-section are on a list being looked at by the responsible people.

j. Other Investigative/Diagnostic Plots

There are fairly painless ways of overcoming some of the standard-plot shortcomings discussed above (¶3.h.*iii*) via plots you can make interactively in glish. These "old-style" plots let you check out the experiment more exhaustively (more SB/pol, more baselines, different time ranges, *etc.*), and clever use of the data-reading and plotting functions can make these more enjoyable (minimize run-time, typing, *etc.*).

The principal data-reading function is `readms`, and the two principal plotting engines are `tplot` & `fplot`. Viewing each baseline/SB/pol in the MS as a data-cube abscissae time and frequency, `tplot` produces slices in time, averaging over frequency and `fplot` produces slices in frequency, averaging over time.

The easiest way to proceed is usually to read (portions of) the MS into a variable, then carry out all plotting from the variable. If you haven't pre-read the MS into a variable, putting the name of the MS as the first parameter will essentially invoke a call to `readms` to access the data, but without as much fine control as you have in running `readms` yourself. If you want to make N plots from the same data, pre-reading the MS into a variable will save you $(N-1)$ iterations of reading the MS from disk. `readms` returns a record containing the specified data from the MS, along with the useful ancillary data to allow characterization of the data in physical terms (baseline names, frequencies per subband, polarizations, times, weights, *etc.*).

There are lots of possible input parameters to `readms`, but they basically fall into functional categories to:

- control the stations selected
- control the subbands selected
- control the time-range selected
- limit what sort of data to select

The syntax for `readms` is:

`readms('`*MSname*`',` *Ants*`,` *Ant*`,` *Bsln*`,`
`continue with 2nd+ lines....`

As `readms` begins it figures out how many visibilities it will need to read, and as it runs it will keep you updated on what visibility range it's currently processing.

Once you've read in the data you want into a variable (record), the basic calling syntax for the plotting functions `tplot` & `fplot` is:

`tplot('`*vrble*`',` *Ant*`,` *Bsln*`,` *SB*`,` *FrqGrp*`,` *Pols*`, '`*type*`',`
various control parameters`)`

The five parameters following the variable to plot further control what data gets plotted; by default they are all `F`, which means plot everything (in the MS, or in the variable read in from a MS, as appropriate). If specified, each of these parameters must be a vector of integers (a single integer counts as a 1-length vector, naturally); use `mssum('`*MSname*`')` to check the integer-coding for the various stations/subbands/*etc.* As above (¶3.g.*ii*), *Ant* & *Bsln* function as the `ANTENNA` & `BASELINE` adverbs in AIPS, and are passed as integral antenna numbers. *SB*, *FrqGrp*, and *Pols* are the subband (0-based), frequency-group, and polarizations (1-based) to plot. You'll almost never use *FrqGrp*. Currently on PCInt, polarization-coding is typically 1=RR, 2=LL, 3=RL, 4=LR up to the number of polarizations actually correlated — the exception would be a 1-pol LCP experiment, in which case 1=LL (rigorously, there's no explicit requirement for this mapping, and the proper way to determine what polarizations are in the MS is via the `DATA_DESCRIPTION` & `POLARIZATION` tables). The *type* parameter is the quantity to plot on the y-axis: `weight`, `real`, `amp`, `phas` are the ones you're likely to use (we've also seen `samp` earlier in ¶3.d.*i*). There are many other control parameters whose default values you're likely to be happy with. The ones you'll use the most are `yplot` & `xplot` to control how many plots go on a page (default = 8×1 for `tplot`, 8×2 for `fplot`), and `outfile` to specify the output postscript file if desired (default = goes to screen). The default weight cut-off is -1 for weight plots and 0.2 for other types of plots. The subparagraphs below give some basic recipes for using these functions to expand upon standard plots.

i — Weight plots

If you have a large $N_{\rm sb}$ experiment, the standard plots won't show all the SB/pol (plots $\leq$4 channels). Here's how to generate an "old-style" weight plot with all channels:

a `:= readms('`*MSname*`', auto=T, wonly=T, [firsttime=`$Tstr_1$`,`
`lasttime=`$Tstr_2$`), dur=`Δt`],`

`tplot(`*a*`,F,F,F,F,[1,2],'weight',T, yplot=`N_y`,xplot=`N_x`, {landscape=T,}`
`outfile='`*PLOTfilename*`')`

a `:= F` (once you've finished with the variable *a*)

Note that you don't have to "`include`" these functions; they are loaded automatically upon starting glish (if logged in as user `jops`). The `readms` function as called above reads only the autocorrelation weights into the variable `a`. This is much faster and uses much less memory than reading in actual data (weights = $N_{\rm pol}$ per visibility instead of $N_{\rm pol} \times N_\nu$; $N_{\rm auto} \sim \sqrt{N_{\rm bsln}}$). Weight plots need only the parallel-hand polarizations — if there are no cross-pols in the MS, then the "`[1,2]`" can be replaced by an `F`. The "`T`" after the *type* means plot only autocorrelations (needed even the data you're plotting is only autocorrelations). The combination of `xplot=1, landscape=T` lets you get the maximum physical length for the x-axis; `yplot=8` produces pretty legible results without using up too much paper.

- If you want to plot subsets of the MS in time, you have two options: use at most two of the three parameters: `firsttime=` or `lasttime=` or `dur=`. The syntax for the absolute times in the first two parameters is either *yyyy/doy/hh:mm:ss* (day-of-year) or *yyyy/mo/dy/hh:mm:ss* (month/day). The units of `dur=` are minutes). If you use `dur` with either of the other two, you will get the Δt minutes following `firsttime` or preceding `lasttime`. The default `firsttime` is the beginning of the data, and the default `lasttime` is the end. You also have the opportunity to use the `initvis=` (default = first) and/or `ntimes=` (default = all starting from `initvis`) parameters in `tplot`. Here, the units are visibility number of the data read into the variable *a*. The `readms` time-specification parameters have more user-friendly units (albeit with more typing), and read in only the time-range you want (faster, less memory use), but would require another `readms` if you want another time-range.
- Each baseline/autocorrelation in `tplot/fplot` plots will always start a new plot; color-coding is used to for multiple SB/pols in a single plot. The color-coding scheme matches that used by `plotweight.pl` (¶1.d), with 16 colors, enough for 8SB/2pol (*e.g.*, a 1 Gb/s experiment without cross-pols). In assigning colors, the polarizations form the inner loop, and subbands the outer loop (*i.e.*, SB0/RR, SB0/LL, SB0/RL, SB0/LR, SB1/RR, and so on). The colors get assigned based on the data that is present, *i.e.*, there is no fixed SB/pol–color identification — if there are cross-pols, SB0/RL gets the "third" color (cyan; *cf.* Table 2 below); if there are only dual-pols, SB1/RR gets cyan; if only single pol, SB2 gets cyan

(the last two statements apply if there are indeed that many subbands). But the color-coding in a single output file is indeed global for all plots in the file (*i.e.*, if a station/baseline is missing some SB/pol, the remaining colors in that baseline's plot mean the same thing as in all the other baselines' plots). Table 2 shows the order of the color-coding.

lines 1–4	red	blue	cyan	green
lines 5–8	pink	med.gray	brown	purple
lines 9–12	teal	yellow	steel blue	lt.gray
lines 13–16	black	orange	lt.green	lt.purple

Table 2: Color-coding for `tplot` & `fplot` output screen plots or postscript files.

ii — Autocorrelation plots

These plot real(ν) rather than amp(ν) because the autocorrelations are intrinsically even functions (thus imag=0 in the Fourier transform). Advantages are speed (no need to compute the quadrature sum) and direct sensitivity to negative autocorrelations (although no longer an issue with the correlator's original 2-bit correction turned off, and certainly not once `2bitVV` has been run). Here's how to generate "old-style" autocorrelation plots:

a `:= readms('`*MSname*`', auto=T, firsttime=`*Tstr*$_1$`, lasttime=`*Tstr*$_2$ `dur=`Δt`)`

`fplot(`*a*`,F,F,F,F,[1,2],'real',T, yplot=`N_y`,xplot=`N_x`,` {`multisub=F,`}` `outfile='`*PLOTfilename*`')`

a `:= F` (once you've finished with the variable *a*)

The `firsttime`, `lasttime`, and `dur` parameters to `readms` control the time-range read in from the MS (use no more than two of these three at once; *cf.* ¶3.j.*i*). By default, `fplot` will vector average over the time-range in the data, which of course for `real` is numerically a moot point, but the design of the time-squashing code is such that vector averaging is faster. By default, `fplot` will make a separate subplot for each station (thus $N_y \cdot N_x$ should equal $N_{\rm sta}$ to get single-page output), with the different subbands separated by dotted vertical lines and all using the same scale. The units on the y-axis are "unscaled" correlation coefficient (perfect correlation = 1). Color coding runs over the polarizations in each SB. If one SB has considerably more RFI than others, it can ruin the y-axis resolution for the others. In this case, you can use the additional parameter `multisub=F` to force separate plots by SB (thus, the product $N_y \cdot N_x$ should now be $N_{\rm sta} \cdot N_{\rm sb}$ to keep single-page output). Now each SB gets its own self-scaled y-axis. A similar `multipol=F`

parameter exists if you also want to separate polarizations. Aesthetically, guidelines for `yplot` & `xplot` include:

- for multi-SB $N_{\rm sta} \leq 12$ experiments: `yplot=`$N_{\rm sta}$`, xplot=1` if `multisub=T`
- for single-SB experiments, or globals with $N_{\rm sta} \geq 13$: `yplot=`$N_{\rm sta}/2$`, xplot=2` if `multisub=T` (*NB*: this `xplot` is the default, and could be omitted)
- for $N_{\rm sb} \leq 4$ experiments with uneven RFI across the SBs: `yplot=`$N_{\rm sta}$`, xplot=`$N_{\rm sb}$`, multisub=F`
- for $N_{\rm sb} = 8$ experiments with uneven RFI across the SBs: `yplot=`$N_{\rm sta}$`, xplot=`$N_{\rm sb}$`, multisub=F, landscape=T`

It's easy to play around with the various page-layout parameters by omitting the `outfile` parameter so that the plot goes to the screen directly — this is always more landscape-like than portrait-like.

iii — Baseline amp/phs(ν) plots

The disadvantages of the standard-plot baseline amp/phs(ν) plots include showing only baselines to a reference station and difficulty in controlling exactly what time-range to plot. One disadvantage of the "old-style" plots is that they don't combine amp & phase for a baseline into one `POSSM`-like compound plot, if you like that sort of thing. Here's how to generate "old-style" baseline amp/phs(ν) plots that address these points:

a `:= readms('`*MSname*`', {`*Ants*`}, [firsttime=`$Tstr_1$`, lasttime=`$Tstr_2$`), dur=`Δt`], [scan=`$N_{\rm scan}$`]`

`fplot(`a`,F,F,F,F,F,'amp', yplot=`N_y`,xplot=`N_x`, outfile='`*PLOTfilename*`')`

`fplot(`a`,F,F,F,F,F,'phas', yplot=`N_y`,xplot=`N_x`, plotline=F, globalscale=T, outfile='`*PLOTfilename*`')`

a `:= F` (once you've finished with the variable a)

The `firsttime`, `lasttime`, & `dur` parameters to `readms` provide direct control over what time-range to read in from the MS (use no more than two of the three; *cf.* ¶3.j.*i*). Use of the `scan` parameter can control which whole scans to read in ($N_{\rm scan}$ can be a vector of [non-contiguous] scans if desired). These provides the leverage to pick scans not accessible via the automatic standard-plot selection algorithm, and to adjust the interval within a scan as desired (shift in time, lengthen/shorten). If reading in just a few minutes of data, `readms` will be pretty quick for even the biggest MS. To check whether there are fringes in a specific time-range, you may first want to play around with phs(t) plots as described in the next sub-paragraph.

- If there are some stations you know you don't want to read in, you can use the

Ants entry: a vector of antenna IDs that limit `readms` to only those baselines (& autocorrelations) that can be formed from the set of stations in *Ants*. As this is the second parameter, you don't need to type the "`ants=`" when you use it in the order above. In this case, it's usually easiest to establish a vector of antenna-IDs that you do want to use prior to calling `readms` (for example, `gdsta := [0,2,5:8,11]`), and then use this variable in the `readms` call: `a := readms('MSname', gdsta, ...)`. This saves typing when you make subsequent `readms` calls for different time-ranges.

- For alternative ways to read in different subsets of the MS, you could use the parameters `source=`*Src_ID* or `scan=`*Scan_No*. The former will read in all data associated with the specified source(s), and the latter will read in all data associated with the specified scan(s). Both arguments take the form of a vector of integers.

As run above, `readms` loads all baselines into the variable `a`. For a 16-station global, this can be 120 baselines. `yplot=12` and `xplot=3` provides a reasonable compromise between minimizing the number of output pages and having legible plots. Getting much past 17×4 (good for 12 stations on 1 page) leaves the plots pretty small. If you want to plot a subset of baselines, you can use the first two parameters after the variable-to-plot: replace the first two `F`s by vectors of antenna-IDs to function in the manner of AIPS adverbs `ANTENNA` and `BASELINE`, respectively (see the discussion at the beginning of ¶3.j).

In `fplot`, the default is to vector average all the integrations in the time-range contained in the variable `a`. If you'd prefer a scalar average, then include `scalar=T`. The units for the amp(ν) plots are correlation-coefficient $\times 10^3$ (often termed "milli-amp" — but not to be confused with current....), and degrees for the phase(ν) plots. Note the two additional parameters for the phase(ν) plot:

- `plotline=F` plots points rather than connecting them with lines (the `fplot` default). This avoids clutter on the plot for SB/pols whose phase hugs $\pm 180°$.
- `globalscale=T` forces the y-axis range of all plots to be the same rather than scaling individually to fit each plot. This can be handy for phase plots, to allow a quick visual check on the "flatness" of the phase across the band, without having to worry about reading the y-axis ranges.

iv — Baseline amp/phs(t) plots

You can go beyond the standard plots by showing more SB/pols, a larger extent of the experiment, and/or more baselines. The trick is to read in more of the data without using up excessive memory, by doing the averaging across the band in the `readms` stage. Here's how to generate "old-style" baseline amp/phs(t) plots that address these points:

a `:= readms('`*MSname*`',` {`ants=`*Ants*}`,`{`antenna=`*Ant*}`,`{`baseline=`*Bsln*}`,`
{`source=`*Srcs*}`, [firsttime=`$Tstr_1$`, lasttime=`$Tstr_2$`, dur=`Δt`]`
`avfreq=T,` {*averaging-control params*}`)`

`tplot(`*a*`,F,F,F,F,F,'amp', yplot=`N_y`,xplot=`N_x`,`
`outfile='`*PLOTfilename*`')`

`tplot(`*a*`,F,F,F,F,F,'phas', yplot=`N_y`,xplot=`N_x`,`
`globalscale=T, outfile='`*PLOTfilename*`')`

a `:= F` (once you've finished with the variable *a*)

The differences between this and the previous calls of `readms` include:

- The explicit inclusion of all three parameters used for making subsets of baselines. By default, each is `F`, which means include everything (not excluded by one of the other parameters). The `ants` parameter is the same as seen in the previous sub-paragraph for amp/phs(ν) plots. The `antenna` & `baseline` parameters behave the same as the "first two `F`s" in `tplot/fplot` (*cf.* the initial example in ¶3.j). Each takes a vector of antenna-ID integers or station abbreviations. You can't use `ants=`*Ants* in combination with either `antenna=`*Ant* or `baseline=`*Bsln*. Taking account of the rules for parameter-passing in glish, examples of the ways you would likely want to run `readms` for amp/phs(t) plots include:
  - `a:=readms('`*MSname*`',gdsta,firsttime=.....)` — reads all baselines formed from the set of stations in the (pre-assigned) vector `gdsta` (if `gdsta` not pre-assigned, it's value will be `F`, so all stations will be read in).
  - `a:=readms('(`*MSname*`',F,[1,4],gdsta,firsttime=....)` — reads in all baselines to stations 1 & 4 formed by stations in `gdsta`. You don't have to worry about baseline order; if `gdsta` includes station 2, then baseline 2–4 will exist in `a`. Reading in all baselines to 2 stations is often a convenient way of avoiding the effects of an outage-period in the "real" reference station in a global, without going to the extreme of reading in all ∼16 stations.
  - `a:=readms('(`*MSname*`',F,gdsta,gdsta,firsttime=....)` — reads in exactly the same data as the first example, but the internal workings of the MS querying makes it much slower (the reason why the `ants` parameter was added in the first place).
- Use of the `source=` selection criteria allows you to limit the plot to appropriate sources, usually ones for which you have reasonable detections (to avoid plotting

unpedagogical near-zero amps and random phases). This can be a single source or a vector of sources, expressed either as source-ID integers (check these out via `mssum('MSname')` or as source-name strings.

- The inclusion of the `avfreq=T` and averaging-control parameters. `avfreq=T` averages the data across frequency for each baseline/SB/pol/integration as it is being read from the MS into the variable. The averaging process is controlled by three additional parameters:
  - `scalar=` : default is `F`, can be set to `T`
  - `x1=`$FP_{\rm low}$`, x2=`$FP_{\rm high}$: the lowest & highest frequency points to include in the average. If not are specified, then the middle $\simeq$80% of the passband is used (best to specify both or neither). The "middle" frequency point is defined as $N_\nu/2+1$ — thus 17 for a 32-point spectrum.
  - `width=` : specify the fraction ($0 \rightarrow 1$) of the band to average over, centered at the middle of the band, if `x1` & `x2` are not specified (default = 0.8).

For `tplot`, the default is to plot points rather than lines, so a separate `plotline=F` isn't needed for phases (of course, `plotline=T` could be used if desired for either plot). Again, there are 16 different colors in the color-scheme, enough for 8SB of 2pol or 4SB of 4pol. If you have 8SB of 4pol, then the last 4SB will all be black. You can avoid this by plotting a subset of SBs (the third "`F`", 0-based) or pols (the fifth "`F`", 1-based). You could also use `multisub=F` to separate the SBs per baseline, at the expense of $N_{\rm sb}$-times more sub-plots. I doubt that we could come up with 32 distinguishable colors (16 might even be considered a stretch...), but if demand is there, we could give it a go, or at least repeat the 16-colors twice rather than just default to all black after 16.

k. flagweight

This step establishes a low-weight floor. All visibilities with a weight lower than the specified weight cut-off will be flagged (actually, will have their weights w set to $-|w|$, and passed to the FITS file that way — the data will still exist in the FITS file). Weights >1 are also flagged, regardless of the weight cut-off used. I generally run `flagweight` last, so that the low-weight visibilities will show up on the standardplots, and thus provide a more honest view of the correlation. If `flagweight` were run before standard plots, then all the low-weight visibilities would have negative weights, and thus fall outside the plots, making the plots look too "clean" (of course, with disks this point is becoming moot). Here are the steps for doing the weight flagging:

i — {`whist`} prints out a tabular histogram of how many autocorrelations & baselines would remain unflagged for various weight cut-offs (0 to 0.95 in steps of 0.05). Due to previous operations (mostly flagging for specific conditions when making the MS & later `2bitVV`), some visibilities will already have $w < 0$ by this point. `whist`

also provides a count of visibilities with $w > 1$ (which usually signifies some sort of correlator bug), which will also be flagged by `flagweight`. The syntax is:

`whist('`*MSname*`', '`*OUTPUTfilename*`')`

The outputfile will be simple text. With the advent of universal disk recordings, `whist` has lost much of its interest, since weights are now much more likely to be nearly perfect or ≤ 0.

ii — `flagweight` does the actual flagging. In the standard usage, the syntax is simply:

`flagweight('`*MSname*`',` $W_{\rm cut}$`)`

where $W_{\rm cut}$ is the value of the weight cut-off, usually somewhere in the range 0.2–0.7. The `whist` output file plus the weight plots lets you judge the consequences of your choice on the amount of data that get through. Except for some bad-track problems that cause autocorrelations to drop to $\sim$0.25, $W_{\rm cut}$ as low as 0.2 will weed out the truly bad data. Higher weight cut-offs (> 0.5) could be used when there are no specific subband/polarization-channel problems. Nowadays the low-weight tail of the $N_{\rm vis}$–W diagram is so steep that changing $W_{\rm cut}$ by 0.2 usually results in incrementally rejecting/accepting only a few $\times 0.1\%$ of the data.

There are additional parameters in `flagweight` for flagging based on antenna(s), SB(s), source(s), scan(s), or time-range, and also an `undo=T` parameter for unflagging (setting the negative weights back to positive). Flagging based on these sorts of criteria would be done in a separate execution of `flagweight`, either before or after flagging by $W_{\rm cut}$ The syntax for time-range flagging uses the `firsttime` & `lasttime` parameters as used by `readms` above (¶3.j.*i*). As for any glish function, you can see all of the parameters (& their default values) by typing in `flagweight` without any following parentheses (or parameters that live inside them). If you include any of these extra-weight flagging parameters, `flagweight` will first form a subset of the MS according to them, and then apply the weight cut-off criterion to this subset. If you want to flag everything to a station or everything in a time-range, you'll need to set $W_{\rm cut}$ to 1.0.

4. Making the FITS Files.

a. tConvert

Now that you've finished all the reviewing/flagging operations on the MS(s) in glish, `tConvert` turns an MS(s) into FITS file(s). The standard usage produces a set of IDI FITS files, each no larger than 2 GB. These can be read directly into AIPS via `FITLD`, either altogether (with `NCOUNT` set to the number of files) or individually. The syntax for standard production of FITS files is simple:

`tConvert` *MSname* *FITSname*

There are some optional parameters (run `tConvert` without arguments for a summary), but you will ∼never need them. Upon starting, `tConvert` will figure out how many individual FITS files will result, and will provide a running update of its progress in each one at the bottom of the screen.

i — The only tricky bit is getting the output `FITSname` correct, so that it will work properly with the pipeline (¶6). The form of the file name is:

FITSname = *exp_C_P*.`IDI`

where:

C is an integer to denote correlation passes done with different correlation parameters that should pipelined independently. Examples include continuum/line experiments (different $N_{\rm lag}$ during correlation, resulting in continuum & line MSs), multiple-field experiments handled by changing source coordinates in different passes (*e.g.*, EZ013, ES051), and experiments where the PI prefers separate FITS files for each mode, corresponding to variously Doppler-shifted spectral line sources (*e.g.*, EL032).

P is an integer to denote correlation passes done for sub-sets of SBs that can be pipelined together (*i.e.*, combined straightforwardly with `VBGLU`). The order of *P* should increase with increasing frequency: *P*=1 for the lowest sub-set of SBs to $P{=}N_{\rm pass}$ for the highest sub-set. If the frequencies overlap, then the FITS files should be distinguished with the *C*-integer. Examples of experiments that would use the *P*-integer include wide-field mapping experiments correlated in multiple passes because only one SB would fit into the correlator at once (*e.g.*, GG053B). The new pipeline can handle `VBGLU`ing FITS files resulting from separate correlation passes for the RCP & LCP channels of a dual-pol experiment, so these also can distinguished via the *P*-integer.

One thing to keep in mind when planning to use `VBGLU` is that it has a "simultaneity criterion" of one-tenth of an integration period for combining SBs. Thus if the epochs of the integrations in your two FITS files differ by more than this, the data will appear to be lost in the combined data set. This condition can result in our data because we have no absolute control over the integral

second in which data starts flowing out of the correlator. Thus, for example, if $t_{\rm int} = 2$s, then the first pass may have odd integration epochs and the second pass even ones. Using $t_{\rm int} \leq 1$s avoids this complication, but this may require pre-correlation liaison with the PI to ensure that such short integrations (and the consequent data size) are okay.

`tConvert` will append an additional integer after the "`IDI`" extension, to denote individual FITS files belonging to the same "dataset" — files whose complete filenames differ only by the n should be identical in all respects, except for the time-range covered. If the total size of the output FITS data for a given run of `tConvert` is <2 GB, then there will be no integer appended. However, in this case you should rename the file extension from `IDI` to `IDI1` for proper operation of subsequent steps.

ii — If C or P exceed 1 for any FITS file(s) created, you should create a simple text file called *exp*`.README` to explain the significance of the naming convention. This `README` file will be bundled together with the FITS files on the EVN Data Archive (*cf.* ¶4.c). Examples shown in Listing 8.

```
GG053B was run in two passes, with each IF done separately.
Each of the two IFs has its own set of FITS files:

gg053b_1_1.IDI*   IF1  -  62 parts, 109.4 GB total
gg053b_1_2.IDI*   IF2  -  64 parts, 111.7 GB total



ez013_1_1.IDI<n>:  phase centers on W44C & W44F
                     n = 1,2,...8
                     t_int = 1s

ez013_2_1.IDI<n>:  phase centers on W44B & W44E
                     n = 1,2,...15
                     t_int = 0.5s
```

Listing 8: Examples of *exp*`.README` files (for GG053B, EZ013).

b. Cover Letter

You can find the cover letter template for your just-correlated experiment in ~`jops/piletters/`. The filename of the cover letter is *exp*`.piletter`, and of the associated experiment summary *exp*`.expsum`. The cover letter begins with a query to the PI about the preferred method for retrieval of the FITS files. If they want the FITS file sent to them on physical media (*e.g.*, DATs, DVDs), they should get back to you. There follows some boiler-plate (which may change as time progresses). The cover letter ends with the opportunity for you to fill in some information you have learned while reviewing the correlation up to this point: problems at stations, other points to bring to the PI's attention, *etc.* There are plenty of examples of cover letters on the EVN Data Archive (under the standard-plots menu item). I probably write excessive cover letters (leading the PI through each type of the plots, trying to find something to say about each). The principal goals are to announce the avail-

ability of the PI's long longed-for FITS files and to persuade them that we've done a conscientious job ensuring the quality of their correlation (*viz.* proper correlation parameters used; problems have already been found/investigated/understood, & determined not to be improvable by recorrelation). Also keep in mind that other people using the EVN Data Archive might rely on the cover letter to help them decide whether they want to mine data from this specific experiment (the descriptions in the pipeline would also help here).

```
project summary on 20/09/2005

                           EK020C

entry last modified on:    05/09/2005

Type and status of exper:  USER   DONE, evaluation not yet complete
             date release: --/--/----
        date distribution: --/--/----
          date completion: 05/09/2005
        archive is public: --/--/----

             observe date: 08/06/2005

Project information:       The Circumnuclear Region of OH Megamaser Galaxies
   Principal Investigator: Kloeckner  (hrk@astro.ox.ac.uk)
    observation resources: 11x10hr@5Disk   18cm
     scheduled telescopes: Ef Cm Nt Tr Jb Wb On Mc Ur Hh Ar
    sub-netting detection: Number networks: 9, max 10 min 8 telescopes


1 observe modes:

          recorded format: 4 bands with 8MHz,
                           using Usb, and DPOL
                           and 2 bit sampling
                  tapeform: 1:4@4MbpstoDisk
        UT range observed: 159d23h14m00s - 160d09h38m00s


2 correlator passes:

Correlator configuration:  11 Tels, 2 band X 2 pols,
                           each with 128 frequency points,
                           4 sec dump correlator time.
      status and capacity: DONE OK
           hours and date: 10hr, complete on 05/09/2005
                     note: Continuum pass

Correlator configuration:  11 Tels, 1 band X 2 pols,
                           each with 256 frequency points,
                           4 sec dump correlator time.
      status and capacity: DONE OK
           hours and date: 10hr, complete on 05/09/2005
                     note: Line pass

configuration in skd:      (11 Tels, 2 band X 2 pols,)
                           (each with 256 frequency points,)
                           (4 sec dump correlator time.)

Handling at JIVE:          Bignall   (bignall@jive.nl)
            tapes at JIVE:  13 disk
        release procedure: automatic by JIVE, after 2 weeks
        net network hours: 10hr

Source list:
  src =        3C286, type = fringefinder (confd), use = YES (confd)
  src =       I20550, type =       target (confd), use =  NO (confd)
  src =     2134+004, type =   calibrator (confd), use = YES (confd)
  src =   J2139+1423, type =    reference (confd), use = YES (confd)
  src =   J2052+1619, type =    reference (confd), use = YES (confd)


Status pipelining:         no information


Inquired publ. status:     not set
        status processing: no information
```

Listing 9: Example of an *exp*`.expsum` files (for EK020C).

The `expsum`-file (*e.g.*, Listing 9) is for your use, providing information about the correlation parameters, PI contact information (at least, from the time they ran sched for this experiment), and the public/private nature of the individual sources in the experiment (needed to know when pipelining). Sources having "`use = NO`" are private, and their pipeline plots & post-`SPLIT` FITS need to have the

same 1-year proprietary period as do the raw FITS files for the experiment as a whole (*cf.* ¶6).

The *exp*`.piletter` & *exp*`.expsum` files are not (yet) automatically generated and put into the proper location. So it's not out of the realm of possibility that you may not find them for your experiment when you look. If this should happen, it means I've forgot to put make them & put them there, and you should direct complaints towards me.

c. archive

The next step is to populate the EVN Data Archive, both with standard plots and FITS files. This paragraph describes these steps, and also some actions that may be required a little later pending liaison with the PI. The perl script `archive` takes care of transferring material to the Archive, with slightly different syntax depending on what is being transferred. The thing you need to know before you start archiving is the reference-day for the experiment. This is associated explicitly with the experiment name, and has the format *yymmdd*. You can confirm this date on `ccsops`:

`ls -d /ccsops/var/log2vex/logexp_date/`*EXP*`*`

There should be a single directory found; the appended year-month-day string will match exactly what you need to use in archiving, once you omit the first two digits of the year. This should also match the "observe date" in the *exp*`.expsum` file (but if it doesn't, the date associated with the experiment in the logbook databasing structure above is what the archive will expect. Let me know if you find discrepancies.

i — Standard plots.

First, `gzip` the standard plot output. If you have no other filenames of the form *exp*`*.ps`, then this is of course simply `gzip` *exp*`*.ps`. Then transfer the standard plots (and cover letter) via:

`archive -stnd -e` *EXP_yymmdd* *exp*`*.ps.gz` *exp*`.piletter`

The first parameter tells what kind of material follows (& where it should be placed in the archive). The `-e` parameter directs the material to the proper experiment. If you make a mistake (usually in the *yymmdd*, if you're like me), you won't necessarily get an error, but the data won't wind up in the Archive (writing to the archive is a multi-step process, of which running the `archive` script on E^3 initiates the first step). You can always confirm successful population by checking that the files you expect to be there actually reside on the archive. As you can see above, `archive` can handle normal wild-card use in filename specification. The above example, with both standard plots and the cover letter being passed to a single execution of `archive`, has equivalent action to running `archive` separately for the standard plots and then for the cover letter.

If for some reason you're archiving plots made outside `standardplots.g`, you'll also need to make a `plotdescription.txt` file to describe the plots you are archiving, and include that in the `archive` line. Every experiment's standard-plot page on the Archive has a `plotdescription.txt`; it uses the standard template unless you explicitly overwrite that with a "private" version for the experiment. You can check any of these on the Archive for an example of the contents. The only times I remember supplying such a "private" `plotdescription.txt` were for experiments such as GG060, GG053B, EB032* (extra-huge wide-field mapping experiments that didn't get along well with `standardplots.g`).

ii — FITS files.

To archive the FITS files & *exp*`.README`, just use a different first parameter.

`archive -fits -e` *EXP_yymmdd* *exp*`_*IDI*` *exp*`.README`

This will most likely take considerably longer, due to the `scp`'ing the FITS files to `jop15` (it should take about 5–6 minutes per individual $\lesssim$2 GB FITS file).

iii — Cover letter distribution. Mail out the cover letter to the PI (info `jops@jive.nl`). I've held off the actual mailing until now, since the cover letter says that the standard plots and FITS files are on the EVN Data Archive, and they're actually not until now.

iv — Setting downloading authorization

If the PI gets back in touch to say they want to download their FITS files from the Archive, you next have to arrange a user-name & password with the PI. There are no explicit guidelines for this process. The syntax for assigning the protection to an experiment's FITS files is:

`archive -auth -e` *EXP_yymmdd* `-n` *user-name* `-p` *password*

We haven't been keeping an internal register of user-names & passwords (arguably for extra security — now there's no records that could fall into enemy hands & there's no one person for them to try to turn). If a PI forgets the password, we can just re-run the above to overwrite the old one.

Before this authorization step, no one can download the FITS files from the Archive. After this step, the individual FITS filenames on the Archive are clickable links that bring up a dialogue box to enter user-name and password. The FITS files become public one-year after the distribution date (of the last part, if it is a multi-part experiment), unless the PI takes action through the PC Chairman to extend the proprietary period. Adjusting the public-release date in such cases is my responsibility, rather than the support scientist's.

d. {Transfer to Physical Media}

If the PI downloads the FITS files off the archive, this paragraph can be skipped. If they want us to send them their data on physical media, you first have to work out (with them) what they could use. The most convenient for us would probably by DDS-4 DATs; however, some PIs may not have access to a compatible reader. So far, we have distributed data on DATs of all kinds, on CDs (once), and on DVDs (once). The paragraphs below describe making distribution DATs and DVDs (the ∼0.7 GB capacity of CDs make them too small to discuss further).

i — DATs

There is a new perl script called `fits2dat` to write the FITS files onto DATs. I've never actually used it yet for a real experiment, so what follows is based on simple test runs, the documentation on the JIVE how-to wiki, and following through the code. It can be run either interactively or from a single command line (interactive seems easier, and is the only way described below). It can get the FITS files either from the local working directory or from the Archive (local access would seem faster, since the `scp` step is avoided). However, there is currently no DAT drive on PCInt. It may be quicker overall to `scp` the FITS files yourself from PCInt to somewhere on `juw27_[`*0*`|`*1*`]` and then treat that as your working directory. It can also plan when to prompt you to change DATs, since you can tell it what sort of DATs you will be using (a limitation seems to be that you need to use only one sort for the whole transfer). The interactive syntax is just to execute `fits2dat`, and then answer the following questions:

- `name of DAT device: /dev/rmt/[`*0*`|`*1*`]n:`
  - you need to supply the integer number of the drive you want to use — `juw27` has two DAT drives; enter `0` (upper one) or `1` (lower one). The default is `0`.
- `capacity of DAT [GB]:`
  - You should answer based on the type of DAT you'll be using: DDS-4 (150m) = `19`, DDS-3 (125m) = `11.4`, DDS-2 (120m) = `3.7`. These numbers would provide some space for block/file overhead. The DAT drives on `juw27` can't write to DDS-1 DATs, but some of the drives attached to computers upstairs can. The default corresponds to DDS-4 DATs.
- `Read FITS from archive (jop15)?`
  - answer `y` to get the FITS from the archive, or `n` to get the FITS from the local working directory.
- `name of experiment:`
  - if getting the FITS files from the local disk, just enter *EXP*; if getting them from the archive, enter *EXP_yymmdd* (*cf.* ¶4.c for details about the *yymmdd* experiment suffix).

- `list of filenames:`
  - the default is *exp*`_*_*.IDI*`, which should be sufficient anytime you want to move all FITS files to DAT. If you have multiple correlation passes (the *C* integer, *cf.* ¶4.a.*i*), and you want to make sure that each DAT has FITS files for only one or the other, you could replace the first `*` with a specific *C* integer, and run `fits2dat` multiple times.

`fits2dat` will then proceed to write the DATs, prompting you to insert new DATs as required. No FITS file should span more than one DAT. At the end, it will ask whether you want to make a printed label for the DAT cases.

In cases where you have more than one *C*- or *P*-number, the standard labeling for the DAT label doesn't fully reflect this. It's easy to take care of these cases manually via the command `datlabel`:

- `fits2dat` makes a file called *EXP*`.datlog`, which `datlabel` will use as input. This file contains a list of individual FITS-file names, file creation dates, sizes, and the DAT number on which each resides. If you make multiple runs of `fits2dat`, as described in the last bullet above, then this file will be overwritten in each subsequent run. There are two disadvantages to this:
  - You lose the "template" with which to remake the DAT label trivially.
  - The "name" of all DATs from the same experiment will be just *EXP*, with no further qualification (*i.e.*, cont, line, *etc.*) — although the *C*- and *P*-numbers will be shown in the individual FITS-file names.
- If you move the *EXP*`.datlog` file to a slightly different name, you can overcome the first of these two disadvantages. If you choose the new name to provide some information about the purpose of this pass (such as *EXP*`.line.datlog`) then you can overcome the second disadvantage as well, because `datlabel` will use the part of this file-name before the `.datlog` extension as the "name" of the DAT.
- Here's an example syntax for *EXP* = EM058A, for which the above file renaming has occurred. We also know from `fits2dat` that two DATs are needed.

  ```
  datlabel -e EM058A.line -t 1
  datlabel -e EM058A.line -t 2
  ```

  where the `-t` parameter specifies for which of the DATs to make a label. The result of each run is that the label appears on the screen, and you can print it to a selection of printers using the buttons. Figure 5 shows what you would see.

Should you ever need to transfer a FITS file manually to a DAT, the fundamental command is:

`dd  if=`*FITSname* `of=/dev/rmt/`*D*`n  bs=28800`

where *D* is the integer identifier for the drive to use. The block size of 28800 is consistent with what AIPS will expect.

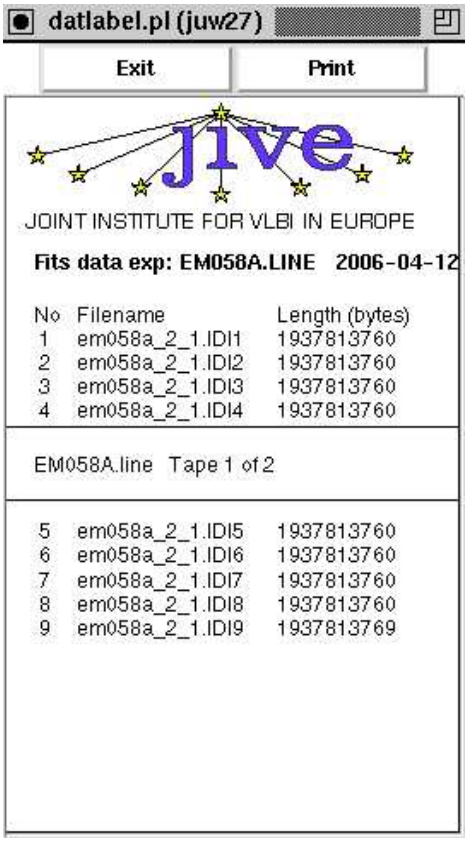

Figure 5: DAT label produced for first DAT for the line pass for EM058A, following the "manual" procedure above.

ii — DVDs

Making distribution DVDs obvsiously requires a machine with a DVD burner. The only time I've done this (for EZ013) has been on `jop25` (in the Visitors' Room), which I will use as an example here. One trick about these linux machines is that you can't log in as `jops` — something as yet unidentified in the login scripts seems to prevent KDE from starting up. I just logged in as myself.

- The first step if to ftp the FITS files from E^3 (or the Archive). I put them onto `/jop25_0/images`.
- Then start up the DVD burner (`k3b`). Via the KDE start menu, this was under multimedia, then CD/DVD burning.
- Once `k3b` starts up, pick "create DATA DVD". In the upper part of the screen, do what you need to do in order to get the listing of files in the directroy where you put the FITS files to appear in the upper-right window.
- Then drag-and-click files to the bottom window to assemble the contents to burn. A DVD holds ∼4.7 GB, so two 2 GB FITS files fit nicely. The last FITS file made by `tConvert` may be noticeably shorter than 2 GB; if you have an odd number of FITS this last FITS file may also be able to fit onto the last DVD.
- Once the bottom window has the files to include in the present DVD, click on

the Burn button at the lower right. You'll then see a set of configuration menu-tabs; most of these seem fine as they are. You'll want to check Settings (no multi-session) and Volume Desc (add a name?). In the Writing tab, make sure the device is right (`jop25` had only one burner, so this was trivial), and that `on-the-fly` is checked.

- Finally, click on the Burn button in top right of the window. For EZ013, it took ∼15 min per DVD (which works out considerably faster than writing DDS-2 DATs, which have about the same amount of space, ∼4 GB). We don't have any dedicated CD/DVD label writing software; I just added hand-written annotations (giving the data that personal touch...).

iii — Regardless of the specific media format used, give them to Nico for shipping out (along with an address for the PI).

5. Housekeeping.

a. D^3: After your experiment is distributed, let Nico know that its data (correlator jobs) on D^3 can be removed. By now, all of the jobs should have been archived to DAT anyway. You can check by going to `/jaw0_1/jops/cat`, and running `grep` *EXP* `*.cat` — make sure that the latest job from your experiment is listed; to be extra thorough, that all jobs in your `lis`-file(s) are listed. (`jaw0` used to be C^3.)

b. E^3: Back ups of PCInt are envisioned to be done in a communal fashion, but the explicit procedures have yet to solidfy. In general, the philosophy is that it would be useful to have a back-up of as much of the experiment's post-correlation review stages as reasonably practical, in case you (or someone else) would ever need to go back and re-do something. Typically, this would include at least the processed MSs and any other small stuff like plots, scripts, `lis-files` — the jobs are already saved as above, and the FITS files on the Archive are backed up separately by Bauke (a weekly incremental back-up, and a full back-up twice a year). But if it's not a terribly big experiment, backing up the jobs as well would give you a single place from which to recover them, rather than the official correlator-job archive, that for your specific experiment is likely spread over several DATs. For the moment, there's no real responsibility for the support scientist here, except maybe to delete clearly temporary things (test MSs, re-done MSs),,, and perhaps the final FITS files after enough time to be sure they've been backed up from the archive, to give a more honest view of what needs to be backed up from E^3.

c. C^3: If you've put any large files on `/ccs/expr/EXP` on `ccsops`, move or delete them. Get rid of any unneeded `core` file(s).

d. Paper Files: Put the experiment folder in the folder-archive (the middle black cabinet for EVN experiments, the one to the right for globals & NMEs/tests). Within the folder archive, experiments are alphabetized for easy retrieval; as time goes by, more and more drawers will be needed, so the specific contents of each drawer may change (hence the post-it labels in place of something more permanent). This step could wait until after the pipelining is done.

6. Pipelining.

The pipeline performs the initial calibration of the FITS data, making the first few versions of `CL` and other tables the PIs may use to save some time. It also fringe-fits the data, and makes preliminary images of all the sources (private sources get the same 1-year proprietary period protection enjoyed by the raw FITS files). These results can be retrieved via the `fitsfinder` utility on the EVN archive, or via the Bologna archive of EVN observations. More thorough information is available on the JIVE how-to wiki.

Appendix. Behind the Scenes

This appendix gathers the more lengthy of the "Behind-the-Scenes" vignettes that look into the workings of various steps in the post-correlation review process. The section numbering reflects from where in the main text the individual discussions have been drawn.

1.e: datasum.pl & daily review

i — `data_handler.log`

Every job has a `data_handler.log`, which forms the basis for much of the "statistical" summaries about the correlation (*e.g.*, the 5 numbers in the `Integrations` column of `showlog` [¶1.c], or at a lower level, `datasum.pl`). The `data_handler.log` lives in the job-directory (on one of the D^3 data disks); all subjobs in the job will be included within this single log. Listing 10 shows a very brief example of `data_handler.log` extracts (here, from EJ007A — 9 sta, 4 SB, 4 pol — job `507281739`).

Integration #301 illustrates a "normal" situation. For this integration, you can see:

- which sub-job you're in (#1).
- how many interferometers you got back from the correlator compared to the expectation (`1368 out of 1368`). Note that numbers in this line are actually $2 \times N_{\rm if}$, because both "stations" of the interferometer are counted separately. Thus, Table 1 (¶1.c) tells us that a 9-sta, 4-pol experiment will have 171 interferometers per SB; the 4 SBs make 684 interferometers, and the $\times 2$ finally results in the 1368 listed here in the `data_handler.log`. The fraction of got/expected is also expressed as a percentage.
- what correlator-frame numbers were found for the start & end of the integration (`210, 273`), and how many interferometers had those correlator-frame numbers (here, all of them). The number of correlator frames you'd expect in an integration depends on the correlator-frame rate (often referred to as *BOCF*-rate, given in Hz: *BOCF*=16 is the standard operational rate, tested limits = 4–64) and the integration time. Here we see 64 correlator frames (the end frame = the start frame + $t_{\rm int} \times$ *BOCF* − 1).
  - Changing the *BOCF* rate requires re-starting the real-time processes; there are scripts for the more common rates, but we're quickly getting beyond the scope of this document...
  - The *BOCF* count wraps back to 0 every 10 min, or at 9600 for the standard *BOCF*=16. The *BOCF* count is truly **the** fundamental clock for assigning time-stamps to the visibilities coming out of the correlator — a (spurious) jump in the *BOCF* count will impart a corresponding jump in the time stamps in the resulting correlator output data for the job, and hence in the

```
Read #301 from buffer #9, data address 0xfd7163e8

SubJobHandler( 1): Got correlator data
SubJobHandler( 1): Start processing new integration....

Statistics based on 1368 out of 1368 (100.00%)

Found start=210 (1368)
Found end=273 (1368)
SubJobHandler( 1): BOCF EXPECT/FOUND = 273/273
### TIME ###
Start: (Data Handler) Sysclks : 4.3620522000000000e+14 (158 18:30:13)
End:   (Data Handler) Sysclks : 4.3620534800000000e+14 (158 18:30:17)
BOCFs were start=210, end=273


Read #302 from buffer #11, data address 0xfd4163a8

SubJobHandler( 1): Got correlator data
SubJobHandler( 1): Start processing new integration....


*************** AAARGHHHHH ***************
BOCF framenrs differ across integration!!!
*****************************************

Statistics based on 1368 out of 1368 (100.00%)

Found start=274 (1368)
Found end=337 (1216)
Found end=0 (152)
PROBLEM found on X part of 0-7:0  3-3 ifnr=17
FR# 1= 274 DOFF=1734462219 DRAT=     -1708 POFF= 763442131 PRAT=   -350522 PACC=         0 FR# 2=   0 SUID= 7 CHAN= 1 KCFG= 1 OVSAMP= 1
PROBLEM found on X part of 0-4:3  3-4 ifnr=641
FR# 1= 274 DOFF=1734462219 DRAT=     -1708 POFF=1217334963 PRAT=   -355645 PACC=         0 FR# 2=   0 SUID= 7 CHAN= 7 KCFG= 1 OVSAMP= 1
SubJobHandler( 1): BOCF EXPECT/FOUND = 337/337
### TIME ###
Start: (Data Handler) Sysclks : 4.3620534800000000e+14 (158 18:30:17)
End:   (Data Handler) Sysclks : 4.3620547600000000e+14 (158 18:30:21)
BOCFs were start=274, end=337


Read #303 from buffer #4, data address 0xfdd16470

SubJobHandler( 1): Got correlator data
SubJobHandler( 1): Start processing new integration....

Statistics based on 1216 out of 1368 ( 88.89%)

Found start=338 (1216)
Found end=401 (1216)
SubJobHandler( 1): BOCF EXPECT/FOUND = 401/401
### TIME ###
Start: (Data Handler) Sysclks : 4.3620547600000000e+14 (158 18:30:21)
End:   (Data Handler) Sysclks : 4.3620560400000000e+14 (158 18:30:25)
BOCFs were start=338, end=401


Read #304 from buffer #6, data address 0xfda16430

SubJobHandler( 1): Got correlator data
SubJobHandler( 1): Start processing new integration....


*************** AAARGHHHHH ***************
BOCF framenrs differ across integration!!!
*****************************************

Statistics based on 1216 out of 1368 ( 88.89%)

Found start=402 (1216)
Found end=0 (304)
Found end=465 (912)
```

Listing 10: `data_handler.log` extracts (from EJ007A).

MS and FITS file arising out of that job.

- the start and stop times of the integration. Here, $t_{\rm int}$ is clearly 4s. The resolution of the quoted time is integral seconds, so for sub-second $t_{\rm int}$, you would often see identical start/end times. The start/end *BOCF* counts would correct reflect the sub-second $t_{\rm int}$, though.

Sometimes, however, the expectations for the *BOCF* start/end counts can get out of kilter with what is seen in some of the data coming back from the correlator. The technical term for this is an "AAAARGHHHH". Integration #302 illustrates one of these. You can see that all 1368 interferometers started on the same *BOCF* count=274, but that 152 of them ended at 0 instead of 337 (=274+64–1). If there

is an AAAAARGHHHH, then there will follow a "`PROBLEM found on...`" line for each "responsible" half of the "discrepant" interferometers. Thus there would be ∼152 entries for this integration, of which I kept two. Here, `X` refers to the first station and `Y` to the second. In the syntax *I*`-`*J*`:`*K* *M*`-`*N*,

- *I* & *J* refer to the two stations making up the interferometer (0-based, in the order of stations as listed in the `$STATION` section of the VEX file used to control this job, but omitting stations that didn't participate at the time of job-preparation from the ordinal count).
- *K* refers to the (0-based) subband.
- *M* & *N* refer to the polarizations, where `3` denotes LCP & `4` RCP (the other possibilities for these fields include `0` = unknown, `1` = X, and `2` = Y).

The `infr` can be used to track down problems in the hardware path taken by the data via `dzbcp` (*cf.* ¶App.1.e.*ii*). Note that for this specific AAAAAARGHHHH, the amount of interferometers lost corresponds to one station (9 stations started the job; 152/1368 = 1/9). You could also confirm this had I not deleted almost all of the "`PROBLEM found`" lines — all of them involved the station `0`. The other interesting clue is that the problem with the "discrepant" interferometers is that their end-*BOCF* was exactly 0. These features (a station's worth of interferometers that end at *BOCF*=0) are clear symptoms of a specific sort of problem, called a BSE (Biggs Syndrome Event, *cf.* ¶App.1.g.*i*.α). A more generic AAAAAARGHHHH could have problems with the start or end, for an arbitrary number of interferometers, and even more than one "discrepant" *BOCF* count in the start or end.

Continuing on in time, the 8 remaining stations' worth of data in integration #303 have no problem. In integration #304 (only the beginning part of it shown), there appears to be the loss of the 2$^{\rm nd}$ and 3$^{\rm rd}$ stations' data (304/1216 = 2/8) with end *BOCF*=0, continuing the typical BSE behavior.

Information in the "`out of`" line in the `data_handler.log` can also point out a different sort of problem, when the percentage of good data doesn't express a fraction $(N - L)/N$, where N is the total number of stations that started the job, and L is the number of stations lost. A loss of data that can't be attributed to an integral number of stations is difficult to attribute to the DPUs or SUs — most likely culprits could include loss of some serial links or a localized casualties in the correlator (input boards or correlator boards themselves).

The 3$^{\rm rd}$ and 4$^{\rm th}$ of the 5 numbers in the `Integrations` columns seen in `showlog` output (*cf.* ¶1.c) are computed from the first number (good interferometers) of the "`out of`" lines: `datasum.pl` builds an internal histogram of this number over all integrations (for each subjob). The 4$^{\rm th}$ number is simply the mode of this histogram (corrected for the extra factor of 2 in the `data_handler.log`), and the 3$^{\rm rd}$ number is just the number of integrations that don't have this number of good interferometers (either fewer or more).

The start/stop scan boundaries in the `showlog` output also derive from process-

ing the `data_handler.log`. Sometimes, `showlog` will report an end-scan that is one higher than what is actually in the data (seen, for example, from the `plotweight.pl` plots, *cf.* ¶1.d). This situation arises because it takes a short (but finite) amount of time to shut down the subjob handler. `datasum.pl` checks the times in the `data_handler.log` against times derived from the `$SCHED` section of the VEX file. Therefore times in the `data_handler.log` that come while the subjob handler is shutting down may trick `datasum.pl` into thinking that some data from the next scan is present (a few seconds, tops). A series of safeties in `j2ms2` and the output-VEX file (*cf.* ¶2.c.*ii*) prevent such integrations from getting into the output MS. Since `plotweight.pl` reads the data directly, it doesn't see these integrations, and its quoted scan ranges at the top of its plots is therefore more robust.

ii — Correlator Hardware Investigation

If you see problems with specific interferometer numbers in the `data_handler.log`, you can use the program `dzbcp` to trace down the hardware path associated with those interferometer numbers. Essentially, you need to start the program, and answer a series of questions. The key point is that each different correlator configuration (*i.e.*, set of $N_{\rm sta}, N_{\rm sb}, N_{\rm pol}, N_{\rm lag}$) will have a different map, so you need to run `dzbcp` right after a job that you're interested in (or at least before a job with a different correlator configuration runs) — another good reason for daily review of the previous day's jobs. Here's what you should input to get the hardware map.

- start the program (as `jops` on `ccsops`): `/users/evn/bin/dzbcp`
- Now you'll get the list of questions; enter (in order below):
  - `sl` (select list)
  - `6` (interconnect info)
  - `3` (interferometer hardware path)
  - `-2` (setup-id = global — in a different sense than global/local validity)
  - `0 1` $N_{\rm if}$ (first, increment, and number of interferometers — the default should be okay for this entry; $N_{\rm if}$ should be the same as the "`out of` N" in the `data_handler.log`, divided by 2)
  - `2` (auto & cross)
  - { *ml* | *mt* } (no prompt for this: `ml` = output to printer & file, `mt` = output to terminal)
  - `/l` (execute the program according to the inputs above)
  - `/e` (terminate & exit)

It's a feature that to get a copy of the output saved to file, you'll also get it printed out on `jet5mj`. There's also a possible complication about saving the output to file. The output file you want will be `/HP-RT/users/evnfra/txt_files/SLU.TXT`, It's not unprecedented, however, that Albert has run `dzbcp` previously, making an

SLU.TXT file for which jops doesn't have write permission (to overwrite it with the new one). When I have the occasion to run dzbcp, I usually move the output file to the ∼/expr/*EXP*/ directory for safe keeping.

```
******** DZBCP on jaw0 ********               Fri Aug 15 17:20:03 2003

  *** Data format descriptor print out ***

 Jobs= 1 Subarrays= 1 Interferometers= 156 Datablocks= 32 Total data= 393216

 Job format descr: Job nr= 0 Job id= 308151658 Nsubarray = 1 SAindex= 0
                   Nintfr= 156 IFindex= 0 Ncorfn = 156 CFindex= 0
       Data sizes: Job= 393216 Max Subarray= 393216
                   Max Intfr= 2048  Max Corfn= 2048

 Subarray format descr: Subarray nr= 0 SAindex= 0
                        Nintfr = 156 IFindex= 0 Ncorfn= 156 CFindex= 0
            Data sizes: Subarray= 393216 Max Intfr= 2048 Max Corfn= 2048

 IF data sizes:  NlagpIf= 2048 NlagpCf= 2048 Mon= 4 Sigsrc= 10
.
.
.
        Stn Un  SU suim    DDU presel  DDU posel  COR input     COR corr
 Intf   Un Sig  Tx Sig     Bd Rx Sig   Bd Tx Sig  Un Bd Rx Sig  Un Bd Chip

   20:X 15   0   0   0      2  7   0    1  0   0   0  0  1   0   0  2   16
     :Y  3   0   0   0      0  3   0    1  0   3   0  0  1   3
   21:X  6   0   0   0      0  6   0    1  0   1   0  0  1   1   0  2   24
     :Y  3   0   0   0      0  3   0    1  0   3   0  0  1   3
   22:X 15   0   0   0      2  7   0    1  0   0   0  0  1   0   0  2   24
     :Y  0   0   0   0      0  0   0    1  0   2   0  0  1   2
   23:X  7   0   0   0      0  7   0    1  0   5   0  0  1   5   0  3    0
   24:X  4   0   0   0      0  4   0    1  0   4   0  0  1   4   0  3    0
   25:X  4   0   0   0      0  4   0    1  0   4   0  0  1   4   0  3    0
     :Y  7   0   0   0      0  7   0    1  0   5   0  0  1   5
   26:X  6   2   0   2      0  6   2    0  1   1   0  1  0   1   0  4    0
     :Y  4   2   0   2      0  4   2    0  1   4   0  1  0   4
   27:X 15   2   0   2      2  7   2    0  1   0   0  1  0   0   0  4    0
     :Y  7   2   0   2      0  7   2    0  1   5   0  1  0   5
   28:X  6   2   0   2      0  6   2    0  1   1   0  1  0   1   0  4    8
     :Y  7   2   0   2      0  7   2    0  1   5   0  1  0   5
   29:X 15   2   0   2      2  7   2    0  1   0   0  1  0   0   0  4    8
     :Y  4   2   0   2      0  4   2    0  1   4   0  1  0   4

        Stn Un  SU suim    DDU presel  DDU posel  COR input     COR corr
 Intf   Un Sig  Tx Sig     Bd Rx Sig   Bd Tx Sig  Un Bd Rx Sig  Un Bd Chip

   30:X  3   2   0   2      0  3   2    0  1   3   0  1  0   3   0  4   16
     :Y  4   2   0   2      0  4   2    0  1   4   0  1  0   4
   31:X  0   2   0   2      0  0   2    0  1   2   0  1  0   2   0  4   16
     :Y  7   2   0   2      0  7   2    0  1   5   0  1  0   5
   32:X  3   2   0   2      0  3   2    0  1   3   0  1  0   3   0  4   24
     :Y  7   2   0   2      0  7   2    0  1   5   0  1  0   5
   33:X  0   2   0   2      0  0   2    0  1   2   0  1  0   2   0  4   24
     :Y  4   2   0   2      0  4   2    0  1   4   0  1  0   4
   34:X  6   2   0   2      0  6   2    0  1   1   0  1  0   1   0  6    0
     :Y 15   0   0   0      2  7   0    1  1   0   0  1  1   0
   35:X 15   2   0   2      2  7   2    0  1   0   0  1  0   0   0  6    0
     :Y  6   0   0   0      0  6   0    1  1   1   0  1  1   1
   36:X  6   2   0   2      0  6   2    0  1   1   0  1  0   1   0  6    8
     :Y  6   0   0   0      0  6   0    1  1   1   0  1  1   1
   37:X 15   2   0   2      2  7   2    0  1   0   0  1  0   0   0  6    8
     :Y 15   0   0   0      2  7   0    1  1   0   0  1  1   0
   38:X  3   2   0   2      0  3   2    0  1   3   0  1  0   3   0  6   16
     :Y 15   0   0   0      2  7   0    1  1   0   0  1  1   0
   39:X  0   2   0   2      0  0   2    0  1   2   0  1  0   2   0  6   16
     :Y  6   0   0   0      0  6   0    1  1   1   0  1  1   1
```

Listing 11: dzbcp output (from GP036B).

Listing 11 shows an extract from a dzbcp output file, from GP036B (2 SB, 4 pol, 512 lag). Note that this is a different experiment/configuration than shown in the data_handler.log example in Listing 10. After the header, the list of interferometers follows, in blocks of 10. The first column lists the interferometer number (here, we look at #20–39), which directly correspond to the listed "*ifnr*" in AAAARGHHHs in the data_handler.log. After the colon(s) come station(s) participating in the interferometer, expressed as an X & Y part (X = 1st station, in our parlance); either a single-station auto-correlation (*e.g.* Intf #24) or a two-station cross-correlation (*e.g.,* all others in Listing 11). The stations are referred to by their SU locations in the job. The UnSig column gives the 0-based channel number (in the

VEX-file sense) used for each station. You can see what this channel-ID means from the `$FREQ` section of the VEX file used to control this job. Typically, channel 0 corresponds to SB0/RCP; in a dual-pol observations, the even channels would be RCP and the odd channels would be LCP (in a single-pol observation, the channel-ID would directly give SB). There is one possible complication for interpretation of the `UnSig` column, which is illustrated in Listing 11: for fan-out=4 experiments (such as this one), the odd-channels are skipped in forming the data to go to the correlator, so here `UnSig=0` corresponds to SB0/RCP and `UnSig=2` corresponds to SB0/LCP. (those with long memories will recall that the LSB/USB mask's inconsistent treatment of this fact led to the "bad-LSB" problem, which has since been repaired, and hence not mentioned in ¶App.1.g.) You can see some Stokes=RR baselines (Intf #20–23, 25), some Stokes=LL baselines (Intf #26–33), and some cross-pol baselines (Intf #34–39). Note that Intf #36–37 are actually cross-pol "autocorrelations" — a station's RCP & LCP signals correlated with each other. The rest of the groups of columns trace the signal flow through the Data Processor: which parts of the DDU pre-selector board, the DDU post-selector board, the correlator-input boards, and the correlator itself were used to handle each interferometer's data (note that once inside the correlator and baselines have been formed, there's only one entry in the `COR corr` columns. For the correlator-related columns, `Un` refers to the crate (*0–3*) and `Bd` refers to the board within the crate (*0–7*). Individual interferometers having many lags may be spread out over multiple chips (the only constraint is that an interferometer must fit onto a single correlator board, which can handle 4096 lags in local validity — *cf.* ¶3.1 of the "*Field-of-View Calculations*" document on the JIVE how-to wiki, or on the EVN web site `www.evlbi.org/user_guide/fov/`).

If you have some known subset of interferometers that are having a problem, such as persistent AAAARGHHHs affecting only a couple `ifnr`'s, try to come up with the common factor among them — a correlator board, a portion of an correlator input board (the two most recurring suspects), *etc.* Inform Sjouke of specific hardware problems once identified.

1.g: A Correlator-Problem Bestiary

Here, we'll try to touch on each of the regularly-occurring problems you might encounter in reviewing the correlation of your experiment. Many of these are typically caught during correlation, and affected scans immediately recorrelated (see ¶2.a.*iii* for techniques of editing the `lis`-file to exclude specific scan-ranges from going to the Measurement Set, and thus ultimately the FITS file(s) the PI gets).

i — Problems that you can detect/diagnose from the `plotweight.pl` weight plots (on `/juw26_6/data/PltWgt/`*EXP*) by themselves:

α) Biggs Syndrome Event (BSE). In a BSE, individual stations drop out of the job one to a few at a time, often in rapid succession over 10's of seconds to a couple minutes. In the `plotweight.pl` weight plots, their lines just appear to stop in

the middle of the job. In the `data_handler.log` all the drop-outs occur when the end-*BOCF*-count is 0. The number of interferometers dropping out can always be attributed to an integral number of stations. See ¶App.1.e.*i* for an example from an affected `data_handler.log`. BSEs seem to occur rather more frequently for disk stations than for tape stations. Data from baselines that include a BSE'd station are rubbish, but they are now pre-flagged (to $w < -1$, so that any subsequent un-flagging of data will leave the weight >1, thus still eligible for flagging via `flagweight`, *cf.* ¶3.k). Recorrelation should begin from the scan in which the BSE first started to occur (or maybe the scan before that, to avoid the 30s correlator start-up-time in that first BSE-affected scan — in this case the first recorrelated scan would be omitted from the MS).

β) Sudden Onset AAAARGHHHHs (SOA). As distinct from BSEs, a Sudden Onset AAAARGHHHH is when a (sub)job that has been running perfectly well (all 100% data in the `data_handler.log`) suddenly starts to experience messy AAAARGHHHHs that persist through the rest of the (sub)job, which can't be attributed to a specific set of stations. It would be quite possible for there to be multiple different start and/or end *BOCF* counts, with varying numbers of interferometers associated with them. Sometimes it's possible to trace "rogue" `BOCF` counts moving in parallel with the correct count, and sometimes it's possible to see the same few `BOCF` counts repeating themselves in a cyclical pattern over just a few integrations (rather than the 10 min `BOCF`-wrap period). Sometimes, there's no readily discernible pattern. When you encounter an SOA, find out the time of the last "uninfected" integration (go to the first occurrence of an AAAARGHH in the `data_handler.log`, and scroll up to the first preceding good integration), and check to which scan it belongs (*i.e.*, from the `vexsum`-file or equivalent). Re-correlate that and following scans in the (sub)job, and use `j2ms2`'s `lis`-file scan-range checking feature to exclude the infected scans from the original correlation.

γ) Diagonal Weight Incident (DWI). This occurs almost exclusively when the `ENABLE_CRM_SERVO` is on — which is required for experiments that have 16 MHz subband bandwidths (*cf.* the "16 MHz jump" problem below in ¶App.1.g.*iii.*ν). The characteristic weight-plot signature is a station suddenly dropping from $w \simeq 1.0$ to ~ 0.5, and then decreasing in a straight diagonal line towards 0. If there is a gap in the schedule long enough for the SU to be reconfigured, the DWI will be over at the start of the scan after the gap. There are two forms of diagonal weights:

- ∘) Slow: can take up to ∼20 min to reach 0. This is by far the most common DWI (∼10 times more common than the "fast" DWIs). Fringes on baselines to the station undergoing a DWI disappear before the manifest beginning of the DWI on the weight plots (*i.e.*, the drop from unity to half weight). This time range, for which there is no apparent problem in the weight plot but no fringes to the station, depends on the per-track recording rate: $\lesssim 5^{\rm m}34^{\rm s}$

for 4 Mb/s/tr, $\lesssim 3^{m}01^{s}$ for 8 Mb/s/tr, and $\sim 1^{m}$ for 16 Mb/s/tr (these are the longest recorded periods of premature fringe-loss prior to a slow DWI that we've seen empirically). Any re-do should start with the scan that was running at the onset of the DWI, less the appropriate time above. Because of this lag between fringe-loss & characteristic signature, it's quite possible that you won't be able to notice a period of fringe-loss should it fall too close to the end of a subjob or the beginning of a gap (in which the SUs get reconfigured). Unless you get "lucky" and see something in standard plots (¶3.h) or other plots (¶3.j), you'll probably never find this sort of premature fringe-loss period. There are two incidents I know of where this has likely happened — one was recoverable because it happened in the scan that (*i*) was used as the `standardplots.g` scan, and (*ii*) was also the scan used for clock-searching (so there was an "earlier correlation" for comparison).

- Fast: takes ∼4–5 min to reach 0. Fringes disappear on baselines to the affected station exactly a the onset of the DWI as seen from the weight plots.

In experiments with $t_{\rm int} < 1$ s, the diagonal decrease on the weight plot can become a rhombus: at low resolution (as on the weight plots), it appears there are two traces forming a rhombus after the jump down from unity to half weight. One trace starts off horizontally at the reduced weight, while the other decreases diagonally. When the decreasing trace hits zero, it turns horizontal and the hitherto horizontal trace starts to decrease at the same rate as the first one did at the beginning of the DWI. The bisector of the initial vertex therefore remains diagonal itself, and decreases at about the same rate as a DWI for $t_{\rm int} \geq 1$ s. However, at higher resolution, it is clear that the two legs of the rhombus that appear exist are really formed by the weights of alternating integrations jumping between the two traces. Occasionally, even weirder patterns in sub-second $t_{\rm int}$ jobs can be seen (an "×" in the middle of the rhombus). Exceptionally unusual traces include very slow diagonally increasing weight after an unusually large initial jump down (seen twice) and a diagonal weight that appeared to reflect off $w = 0$, and started increasing at the same rate it was decreasing (seen once).

DWIs should be recorrelated from the start of the scan that contains the time ∼3–5 min prior to the manifest start of the DWI (for a slow DWI) or from the scan that contains the DWI (for a fast DWI). Edit the start/end-scan columns in the `lis`-file to excise the scans affected by DWIs (¶1.f.*i*).

δ) Ghost Data. This occurs when a station's trace remains in the weight plot after it has actually stopped recording or otherwise participating in the subjob. The last integration continues to be output from the correlator over and over (seems to be able to last ∼7–12 min). `j2ms2` on PCInt handles ghost data at the MS-creation stage (it recognizes that the station really has left the schedule, on a scan-by-scan basis).

ϵ) Ghost Data Termination. However, one negative consequence of the Ghost Data is that when it stops in one station (say, $\mathcal{G}$), it can (but doesn't have to) effectively kill the weight of another station (say, $\mathcal{V}$) at the same time. For station $\mathcal{G}$, it will look like its weight-plot trace disappears (actually the weight goes to $-\pi$, but the weight plots have a fixed y-axis range). The ghost data termination (GDT) occurs only if $\mathcal{G}$ leaves the array and doesn't return in the subjob (otherwise the ghost data will remain constant weight until $\mathcal{G}$ does return). The weight on $\mathcal{V}$ will drop to 0 simultaneously to the GDT on $\mathcal{G}$. Characteristics of the data during the GDT event include:

- baselines $\mathcal{G}$–* are of course bad, since $\mathcal{G}$ has stopped observing for this time range.
- the autocorrelation $\mathcal{V}$–$\mathcal{V}$ is indeed bad — it's not just the weight spuriously dropping to 0.
- other baselines $\mathcal{V}$–* are okay. This clearly points to a correlator problem, since the autocorrlations and cross-correlations are done entirely independently.
- in spite of this, the baselines $\mathcal{V}$–* will not be useful if `2bitVV` (¶3.d) is run. `2bitVV` needs the autocorrelations to compute the 2-bit van Vleck corrections to apply to the baselines; without autocorrelations (*i.e.,* the weight less than the weight cut-off), `2bitVV` will flag all baselines $\mathcal{V}$-*.

A GDT event can also occur if $\mathcal{G}$ in the subjob doesn't begin observing from the start of the subjob. In this case, some other station $\mathcal{V}$ may have 0 weight until $\mathcal{G}$ comes in. Here, the "Termination" aspect of the name is something of a misnomer, but the two aspects clearly stem from the same cause.

As mentioned, the GDT events are clearly a correlator bug. When the correlator computes autocorrelations (one station, one SB, one parallel-hand pol), it uses half the amount of correlator hardware as it does for a baseline (one baseline, one SB, one pol). This is because the autocorrelation will be symmetric about the peak lag (0 delay); the correlator computes only the top (or bottom?) half and subsequent processing uses the symmetry to reconstruct the full autocorrelation spectrum. We have determined empirically that the realtion between the autocorrelations for $\mathcal{G}$ & $\mathcal{V}$ is that they share the same correlator "footprint" that a baseline would have used (as traceable via `dzbcp`, *cf.* ¶App.1.e.*ii*). Sometimes, ghost data terminating on a station $\mathcal{S}$ doesn't knock any other station $\mathcal{G}$ out — in this case there wasn't another autocorrelation "sharing" the same baseline "footprint" with $\mathcal{S}$'s autocorrelation.

The recorrelation tactic when a GDT event is noticed is to limit the scan-range selection in the `runjob` gui to avoid the array missing a station at the beginning or end of the subjob(s). Note that some events that automatically trigger a subjob change (*e.g.,* a disk-pack change) could complicate the scan-range selection.

ζ) Premature Stop. It's quite possible for various things to die in the middle of a subjob (software: the CDI process, the shared-memory allocator; hardware: serial links, *etc.*). There may be obvious signs of this in the weight plot (only some stations/subbands/pols stop early) or not. Even if there are no such obvious signs, it should be clear from the end-time of the plots themselves — this would precede the last time in the last scan of the job. A more rigorous test would be to check the last reported time of a good integration in the `data_handler.log`. Such events are almost always caught during production, but if there's a final partial scan in the subjob, you'll need to edit the end-scan column of the `lis`-file to make sure it doesn't go into the MS (¶2.a.*iii*).

η) Servo'ing / Synching. Especially for tapes, it's not uncommon for synch to be lost and regained, often several times in a subjob. Sometimes recorrelation on the same unit helps, sometimes not. The operators usually notice such events, and try changing station location or checking various SU boards. With disks, servo'ing is rarely a problem, except maybe for some stations at 1 Gb/s.

θ) *BOCF* jump. This almost never occurs anymore. Because of the inherent 10-min cycle for the *BOCF* count, a "rogue" *BOCF* count returned from the correlator, coherent over enough interferometers that it is not interpreted as the count causing an "AAAAARGHHH" (*cf.* ¶App.1.e.*i*), could case a corresponding jump in the (UTC) time stamps associated with the data. Typically, this could be seen in the weight plots as a "blank" time-range, where no station has any weights. Also, the duration of the job would be up to 10 min longer than it should have been. Since there's no explicit scan information in the correlator output data, `plotweight.pl` figures out the scan boundaries in the data by referring back to the VEX file. Thus if $\tau_{\rm jump} \lesssim 10$ min are skipped in the data, the weight plot would "assign" data after the *BOCF* jump to whatever scan falls $\tau_{\rm jump}$ after the scan they really belong to. The scan-range of the plot as seen in the upper-right corner would extend to scans that weren't done in this (sub)job. Of course, if the job were stopped prematurely because the *BOCF* jump was noticed, this end-scan-is-too-big test may not be applicable, but the $\tau_{\rm jump}$ missing-weight interval will still be a diagnostic.

ii — Problems that you can detect from the `plotweight.pl` autocorrelation-amplitude plots:

κ) Sampler Stats. The amplitude of the autocorrelation peak for a given channel (subband/parallel-hand polarization) in the correlator output is now proportional to the fraction of high-bits (f_h) as recorded by the station in that channel. Thus you can get a quick check of the the stations' "sampler statistics" directly from the autocorrelation-amplitude plots made by `plotweight.pl`. The program `2bitVV` (*cf.* ¶3.d) will take care of computing and applying the 2-bit van Vleck correction, so there's no action to take at this point with respect to the

data in hand. But if the stations' f_h are too far off optimal (36.4% for us), then some of the sensitivity gains of 2-bit sampling will be irretrievably lost: $f_h = 1.0$ would just be 1-bit sampling, and if the f_h get too close to 0, then noise will play a larger role in the computed `2bitVV` corrections. The `plotweight.pl` autocorrelation plots provide a means to get feedback to the stations about their sampler stats, without having to wait for the analysis of the MS. (Even better, this sort of thing should have been looked at during clock-searching, which would have provided even faster feedback. If $f_h \lesssim 0.15$, Sergei suggests that we should force treat the station's recording as 1-bit, by making a separate `$TRACKS` section for it in the VEX file, with all the `MAG` bits commented out. This is another reason to check in clock-searching, prior to the production correlation. Of course, to shift a station from 2-bit to 1-bit, the low f_h would really have to be true for all SB/pol, since the number of bits in the recording is a station-based characteristic in our system.)

iii — Problems that you can't detect from any `plotweight.pl` plots, and would need further investigation of the data itself to discover:

λ) Byte Slips. There are two types of these, one arising in the DMMs (Delay Memory Modules) and another in the TRMs (Track Recovery Modules). Of the two, the TRM Byte Slips have largely be excised, although as we do more and more 64-track experiments, we may re-encounter some boards that were more prone to these events (we had moved them to the upper half of the SUs so that they wouldn't be used as much). The DMM Byte Slips seem more prevalent in (but not exclusive to) experiments with only 2 channels. There's really no smoking gun for these without an `olayspec` of the lag spectrum for baselines to a suspected station. There is a separate guide to Byte Slips available on the JIVE how-to wiki; it contains a series of illustrative figures for both kinds showing the characteristic behaviors in various sorts of plots. DMMs that appear to be vulnerable to DMM byte slips have been marked, and moved to the upper position in the upper SUs — the easiest approach is avoidance.

μ) *A priori* correlator model effects. Specific conditions in the *a priori* correlator model for a baseline may trigger problematic correlator output for a specific time range. There is a separate guide to these events available on the JIVE how-to wiki.

- When the *a priori* delay-rate on a baseline is close enough to 0 that the phase-cal tones in both stations can correlate with each other (a function of the subband frequency and the integration time). This causes a period of excess correlation amplitude in frequency points corresponding to the phase-cal locations, which bleeds through into averages across the band. In the past, we have also seen instances (twice that I know of) where Jb_1 has also had similar effects at times when harmonics of 50 Hz about the phase-cal tone also correlate against another station's phase-cal tones. If

one or more stations in the baseline don't have phase-cal on, then there is no problem. The glish program `plyflg.g` takes care of these events by flagging the appropriate time range (*cf.* ¶3.g).

- If using "global" validity, there is also a problem when the Mk4 tape-frame headers correlate with each other — in other words when the *a priori* delay on a baseline is near an integral multiple of the time represented by a tape frame (a function of the fan-out ratio, the subband bandwidth, and the $N_{\rm lag}$ used for correlation). Baselines including a VLBA-format station will not be affected (VLBA format doesn't use data-replacement headers). The only operational reason to use global rather than local validity is to squeeze an extra factor of 2 in $N_{\rm lag}$ out of the correlator (all other correlation parameters being equal). However, increasing $N_{\rm lag}$ just increases the time-range that would be affected by this problem, so it is essentially never worthwhile to use global validity (with Mk4-format recordings). (Also note that the standard correlator-load formulas assume the reduced correlator-output load appropriate to local validity, so while we can read-out the whole correlator to E^3 reliably in 0.25s in local validity, that's not necessarily true for global validity.) In any case, `plyflg.g` (*cf.* ¶3.g) can also flag time-ranges affected by this *a priori* delay condition.

ν) $BW_{\rm sb}$=16 MHz jumps. There's a circular memory buffer in the CRMs (Channel Recovery Modules) whose read & write pointers can overlap for data recorded with subband bandwidths of 16 MHz (or 32 Msamples/s). At this data rate, this buffer holds $\simeq$ 141 ms of data. When one station's pointers "overlap", fringes are lost to that station (it's data are offset by 141 ms with the others); when a second station's pointers "overlap", then fringes return on the baseline, but there's a 141 ms shift of the data with respect to the model. In other words, (once all stations' pointers have "overlapped") plots of delay-jump *vs. a priori* rate yield a straight line, as do plots of rate-jump *vs. a priori* acceleration, and so on for higher derivatives of delay, out to the level of noise in the data. The solution to this problem was to turn on the CRM-SERVO mechanism, which works to ensure the read/write pointers stay as far away from each other as possible. We've never seen a 16 MHz jump when `ENABLE_CRM_SERVO` was on. However, this feature seems to be involved in causing/allowing diagonal weights, so now it should be used only for experiments that have $BW_{\rm sb}$=16 MHz. Unfortunately, there is nothing in the output data or ancillary files which can tell you whether it was on or not for a specific job.

π) Any other loss-of-fringe problem that doesn't have a corresponding low-weight signature (*e.g.*, the fringe-loss in a slow DWI prior to the manifest diagonal weight signature). Another recent discovery (that appears to have been resolved) was the fact that SU5 prevented fringes in baselines to its mounted station for oversampled data. The hardest problems to find are of course the ones we don't know about yet.....

2.c — `j2ms2`

The correlator output data lives in job-directories (*YMoDyHrMi/*). Each job directory has one or more subjob subdirectories (*N/*), in which live the subjob's raw correlator output (`CDF`) and ancillary files (`HDRS, MAP`). The CDF is in lag-space, and can get pretty big — currently ∼14 GB/hr for a full-correlator, $t_{\rm int}$ (0.25 s) mode (more precisely, 1 MB per integration for the full correlator in local validity). These files contain all the correlation functions for all the interferometers/integrations, but have no intrinsic information to tie these to physical stations/baselines/SBs/pols. The correlator continues to pump out data as long as the job is running, regardless of gaps in the schedule (in which case of course, the data would be meaningless — you can often see this in the `plotweight.pl` weight plots, where strange weight patterns can be seen in periods of gaps in which the stations have stopped recording). A few other files are used to provide the ties back to the physical observing setup.

The SB/pol set up, including frequency assignments, is taken care of by the top-level VEX file (*cf.* ¶2.a.*ii*, ¶2.b.*ii*). This is why it's important to use the correct VEX-file in the first line of the `lis`-file, especially in experiments that have multiple correlator passes, each using a different subset of SBs/pols. The top-level VEX file also provides antenna information, so it's important not to edit the `$STATIONS` section (*e.g.*, remove antennas not used, change ordering) in the course of production correlation. Every run of `j2ms2` that contributes to the same MS should use the same top-level VEX file.

Each job directory also holds an "output VEX file", named *EXP_jobID*`.vex`. This is used to take care of the assignment of scan information, as well as to control what data actually makes it into the MS. Data at times that do not belong to a scan according to the output VEX file are not written to the MS. To be considered "in a scan", there is an additional test: the system knows which scans you are doing in each job (since you have to select them in the `runjob` window); the `source` for all other scans is set to "`unknown`" in the output VEX file. Data with times belonging to such scans are also not written to the MS. This is the safety referred to in ¶App.1.e.*i* — the fact that the output VEX file is a job-based rather than a subjob-based entity allows the possibility of "orphan data" sneaking through at the beginning of second (& subsequent) subjobs in a multi-subjob job. However, the scan-range checking capability of `j2ms2 -v` *exp*`.lis` (*cf.* ¶2.a.*iii*) provides a much more interactive way to constrain further what scans from individual subjobs contribute to the MS. The output VEX file also has a "header" comprising commented-out lines that provide you an oversight over the correlation parameters, and tell `j2ms2` what the versions of the on-line correlator software were for this job (`ABMajor, ABMinor`), so that it can interpret the correlator output data properly. The "`OLD_prep_job`" line shows the syntax for the older command-line interface for starting a job (*i.e.*, prior to the GUI-based `runjob.pl`). This line can be handy for testing/debugging (add a "`-l 3`" to the line, and only the CJD will be made without actually starting a job).

3.b: {badcorr.g}

`badcorr.g` flags visibilities that have good weights (above a weight cut-off, default $W_{\rm min} = 0.2$), but also have amplitude in their DC frequency point (first one for upper sideband channels; last one for lower sideband channels) greater than some unreasonable value (default = $\mathcal{A}_{\rm max} = 10.0$). The syntax is:

```
include 'badcorr.g'
badcorr('MSname', {Amax}, {Wmin}, live={F|T}, outfile='OUTFILEname',
   verbose={T|F})
```

where $\mathcal{A}_{\rm max}$ is the maximum allowable DC amplitude (visibilities above this will be flagged), $W_{\rm min}$ is the weight cut-off (visibilities below this won't be considered — we just want to treat data that otherwise might seem okay; low-weight points will be flagged later, *cf.* ¶3.k). The `live` parameter controls whether to actually apply the flagging or not (default = `F` — just report what it thinks should be flagged). The `verbose` parameter controls how much output you get:

- `T` (default) provides a time-range summary for all the intervals to be flagged, broken up by baseline/pol (*i.e.*, an isolated integration would have a time-range equal start/stop times; several consecutive integrations would be listed together in one line, with appropriate start/stop times. The day of the time range is given as day-of-year, for direct cut-n-pasting into the `firsttime` or `lasttime` parameters in several other glish subroutines.
- `F` provides only the total number of visibilities to flag per baseline/pol.

There is also an `undo=T` parameter to "unflag" visibilities meeting the selection criteria. In this context, flagging means setting the weights of the selected visibilities to $-|w|$ (*cf.* ¶3.k); unflagging to revert the weights to $|w|$. Listing 12 shows extracts from a `badcorr.g` run with `verbose=T`.

This program stems from trying to excise the effects of a correlator problem that led to ludicrously high power in the DC frequency points. At the time, there was a clear bifurcation in the DC amplitudes between "normal" visibilities and obviously discrepant ones; $\mathcal{A}_{\rm max} = 10.0$ was sufficient to separate these. Later, in GG053B we noticed that some autocorrelation-real values for a channel with weight=0.25 could also have a clearly wrong shape (high DC point, highly negative point ∼halfway through the band). Bad autocorrelations can now be notably troublesome, because the 2-bit van Vleck correction (¶3.d) computes baseline (amplitude) correction factors from parameters derived from the "participating" autocorrelations. `2bitVV` has some sanity checks built in, but it's probably better to flag obviously bad data first. The thing that makes me somewhat uncomfortable is that I don't have a good explanation for how cases like this come about, and certainly can't give any guarantees that there aren't other sorts of problems lurking that aren't addressed by what `badcorr.g` checks. I'm not even sure what to recommend for $\mathcal{A}_{\rm max}$ in a specific cases.

```
**** badcorr output for MS = gg053b_SB0.ms1 ****
Reporting data with DC amp. > 10, weight cutoff= 0.2
USB, DC channel 1

**** Summary for polarization 1 ****

 Bsln Hn-Hn:  053/00:46:08.250 - 053/00:46:08.250
1 bad times (out of 35958) for baseline Hn-Hn (1-1 in MS)

 Bsln La-La:  052/23:20:53.500 - 052/23:20:53.500
 Bsln La-La:  052/23:27:29.750 - 052/23:27:29.750
 Bsln La-La:  053/00:09:34.500 - 053/00:09:34.500
 Bsln La-La:  053/01:07:19.750 - 053/01:07:19.750
 Bsln La-La:  053/01:16:05.000 - 053/01:16:05.000
 Bsln La-La:  053/01:34:28.000 - 053/01:34:28.000
6 bad times (out of 35958) for baseline La-La (4-4 in MS)

Will flag 7 out of 4340184 times for polarization 1

**** Summary for polarization 2 ****

 Bsln Pt-Pt:  053/01:15:26.500 - 053/01:15:26.500
1 bad times (out of 35958) for baseline Pt-Pt (5-5 in MS)

 Bsln Kp-Kp:  052/23:23:48.500 - 052/23:23:48.500
 Bsln Kp-Kp:  052/23:28:28.750 - 052/23:28:29.250
 Bsln Kp-Kp:  052/23:28:29.750 - 052/23:28:30.000
 Bsln Kp-Kp:  052/23:28:36.250 - 052/23:28:36.250
 Bsln Kp-Kp:  052/23:28:40.500 - 052/23:28:40.750
 Bsln Kp-Kp:  052/23:28:45.500 - 052/23:28:45.750
 .
 .
36 bad times (out of 35958) for baseline Kp-Kp (6-6 in MS)
.
.

Will flag 51 out of 4340184 times for polarization 2
```

Listing 12: Extracts from `badcorr.g` output, with `verbose=T` (from GG053C).

3.d — `2bitVV`

All actions listed below are performed in a loop over each separate integrations in the MS. Thus the 2-bit van Vleck corrections for each integration are independent of those for other integrations, with no averaging period as in `ACCOR` in AIPS.

- read the `DATA, WEIGHT, DATA_DESC_ID, ANTENNA1`, and `ANTENNA2` columns from the MS. The `DATA_DESC_ID` columns provides the means to trace the SB/pol information from the `SPECTRAL_WINDOW` and `POLARIZATION` subtables.
- Loop over autocorrelations (`ANTENNA1 == ANTENNA2`, only parallel-hand pols):
  - Recreate the lag spectrum: keeping track of the upper/lower sidebandedness, "fold-over" the (real-only) N_ν-long frequency spectrum to $2N_\nu - 1$; compute the value for "freq #0" (such that lag #0 will be $0 + 0i$ after the FFT); do the FFT to lag-space.
  - Read off the value of the peak lag; scale by 0.364 to f_h; store in a table (dimensions $i_{\rm sta}, j_{\rm sb}, k_{\rm pol}$).
  - Compute the corrected lag-spectrum for this Sta/SB/pol using the table of coefficients A_{ij} determined by Sergei in his simulation:

$$\sum_{i=1}^{N_a=9} T_{2i-1}(\,\overline{\text{orig. lag spectrum}}\,) \cdot \sum_{j=1}^{N_c=8} A_{ji} T_{j-1}(2f_h - 1)$$

Here, $T_\ell(x)$ denote the Chebyshev polynomial of order ℓ (range $-1 \leq x \leq 1$). Note that each point in the autocorrelation lag spectra gets corrected separately (the overbar intends to signify treatment as a vector).

- FFT the corrected lag-space autocorrelation back to frequency-space; truncate from $N_{\rm lag} = 2N_\nu$ to N_ν, keeping track of lower/upper sidebandedness.

- Loop over baselines (cross-hand pols for `ANTENNA1 == ANTENNA2` are considered baselines, along with `ANTENNA1 != ANTENNA2`):
  - extract the appropriate f_{h_1}&f_{h_2} for the two participating Sta/SB/pol.
  - compute the single baseline-scaling factor ($\mathcal{B}$), using the table of coefficients X_{ij} determined by Sergei in his simulation:

$$\mathcal{B} = \sum_{i=1}^{N_p=8} T_{i-1}(2f_{h_1} - 1) \cdot \sum_{j=1}^{N_p=8} X_{ij} T_{j-1}(2f_{h_2} - 1)$$

  - Multiply the (complex) baseline frequency-space correlation function by $\mathcal{B}$.

Figure 6 shows the $\mathcal{B}$ "field". The green thick line is the locus of $\mathcal{B} = 1$. A box bounding $0.3 \leq f_h \leq 0.4$ for both stations is over-plotted (this forms something of an unofficial target for the stations to achieve). Note the numerical limitations of the correction: two stations with "optimal" f_h will have $\mathcal{B}$ of $\simeq 0.98$.

3.e — `fixfbs`

`fixfbs` computes its corrections to the phase slope across the band to compensate for the residual fractional bit shift (FBS). The basis for the correction is the relation that 1 lag of delay induces a 180° slope across the (upper or lower sideband) frequency bandpass, pivoting around the central frequency point ($N_\nu/2+1$). The goal will be to compute the net residual fractional bit shift for an integration, convert that to a phase per frequency point gradient, and apply it to the actual data in the MS.

The first step, just as in `plyflg.g` or `plyrecon.g`, is a loop over all jobs in the MS, loading the polynomial representations of the *a priori* models passed to the SUs for the participating stations (*i.e.*, the start & stop times and the 6 coefficients applicable over those time ranges). The model polynomials are evaluated via $\tau(t) = \sum_{i=0}^{5} C_i \cdot (t - t_0)^i$, where t_0 is the start time associated with the polynomial. Each polynomial can be valid no longer than two minutes ($t - t_0 \leq 120\,$s), but the explicit rules for how the start & stop times are assigned for the time range covered by a specific scan in the VEXfile are somewhat complicated, and may well change from job to job (depending on what range of scans the job covers). Of course, if the start & stop times differ between two jobs, the polynomial coefficients themselves

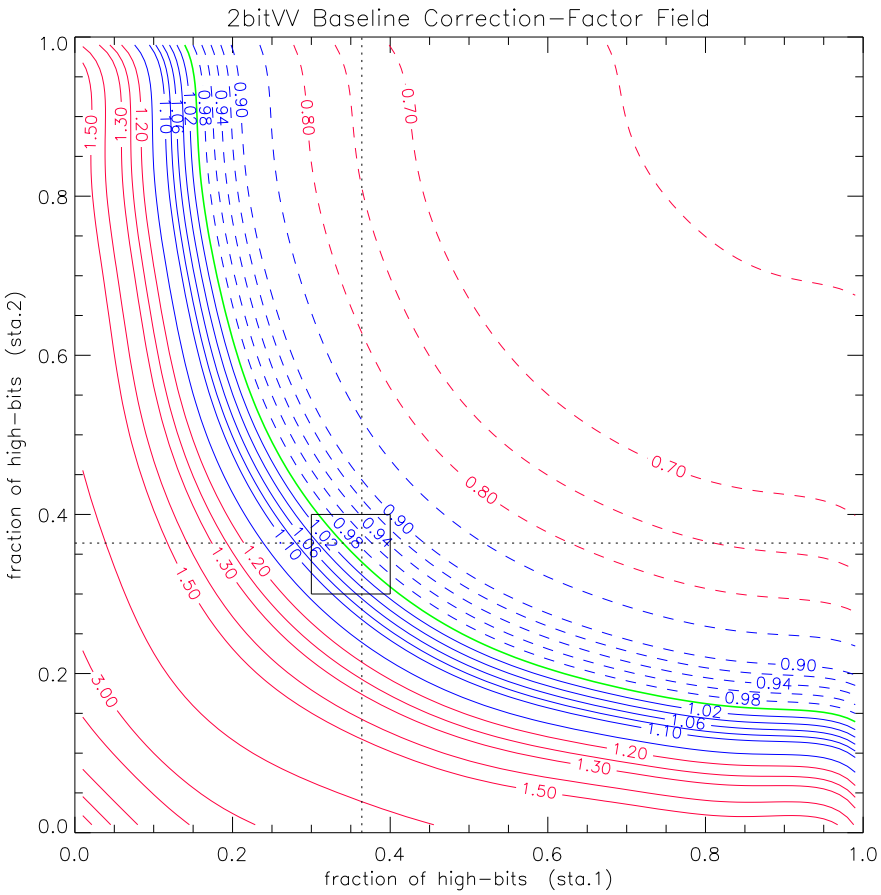

Figure 6: `2bitVV` baseline-scaling factor as a function of f_h for both stations.

will also be different, such that the evaluation of the total delay model remains the same.

Once the polynomials are loaded for all stations & jobs, there's a separate set of loops that go through all the baselines & integrations in the MS. For each bsln/int, the net residual FBS can be computed from the area in the (delay, time) plane above/below the $\tau(t)$ curve with respect to the closest (horizontal) integral lag "lobe" that are spaced $1/(2BW)$ apart in delay (and on which the FBS is instantaneously 0). The net area normalized by $t_{\rm int}$ provides a net FBS delay. Per integration, we compute a best-fit linear model to $5^{\rm th}$-order polynomial model. This linearization greatly assists the computation in two ways: determination of the crossing times of $\tau(t)$ with the equally spaced integral lag "lobes", and cancellation of above/below areas when $\tau(t)$ crosses >1 lag "lobe". With this linearization, a few geometrically-envisionable rules suffice to handle all possibilities of the relative configuration among the start/end fractional lag "lobes", the initial direction of the $\tau(t)$ curve, and the integral lag "lobes". The resulting net FBS is converted into a phase slope, `slp` := $d\varphi/\,d\nu_i$ [rad/frq.pt], and a N_ν-long vector of phase-offsets is computed: $\overrightarrow{\Delta\varphi}$:= $[1 : N_\nu]$ * `slp`. Loops over subbands and polarizations within the bsln/int are then done to read in, correct, and write back the actual data. The correction steps are:

$\vec{\varphi}_0$:= `arg(` *data* `)`

$\vec{\mathcal{A}}$:= | *data* |

$\vec{\varphi}_1 := \vec{\varphi}_0 - \overrightarrow{\Delta\varphi}$

$data_1$:= $\vec{\mathcal{A}} \exp($ `complex`$(0, \vec{\varphi}_1))$

6 — Setting the per-source protection for Pipeline output

Note that this requires that that `archive -auth` has already been run for the experiment (*cf.* ¶4.c.*iv*).

- go to the web site `www.jive.nl/archive/scripts/pipe/admin.php`. This will not work well under Netscape 4.76; does work well under Netscape 7 (= mozilla 1.4). Log-in as `jops` via the dialogue box.
- Select your experiment from the pull-down menu; click on the Submit Query button.
- Click on the experiment name (in column 0). A new window will appear. This has two columns of pipeline-generated plots: on the left, a list of experiment-wide plots, and on the right, a list of source-specific plots.
- In the upper right is a pull-down menu of sources. For each of the "private" sources (*cf.* the `expsum`-file), click on the source in this pull-down menu, click on the Select all source field button at the bottom, and then click on the Enter into database button. Then go on to the next "private" source.
- When you check the pipeline part of the Archive, you should be able to view any plot for non-"private" sources, but should get a dialogue box asking for a user-name/password for any plot for a "private" source (once you log-in once, no subsequent dialogue box will appear in the same Netscape session).