# GOES I-M XRS Processing Code

**P.L. Bornmann**

**Drafted April 12, 2000**

**Extended May 9 & 17 and August 8 &15, 2000**

## Introduction

The XRS data for GOES I-M is stored on CDs in binary format, so code to read XRS data from previous GOES could not be used. The code for reading the GOES I-M data is based on code written by Lorne Matheson (and Dave Lewis for the sellib files) to perform real-time XRS data processing. The code was extended by Pat Bornmann to extract information not provided by the real-time code and to provide analysis of the XRS data specific to questions regarding fluxes during offpointings. Pat also reviewed Lorne's code and added comments to the code where the operation was unclear.

The original real-time code was written in C for a UNIX machine and was successfully run on a Windows machine using Microsoft's Visual Studio to compile the code. The Visual Studio settings are available in files with the filename goesi_xrs_list and extensions dsp, ncb, opt, and plb. The library of support files known as sellib must be included.

The original c-code originally extracted some of the needed x-ray telemetry values and did not convert most of these to engineering units. In particular, the x-ray fluxes were converted in separate code that was not designed to run in the needed manner (either as a stand-alone program or as a module that could be called from the CD-reading program).

IDL was chosen as the language for analyzing the data, because of its facility with data arrays, analysis, and plotting. While it should be possible to call c code from IDL, it was easiest to use the c code to write data to a file and read this new file from IDL. The processing code includes finding times when XRS was offpointed and producing various plots.

The IDL code produces plots of all variables for the entire data file and plots when the SAS (Sun Angle Sensor) indicates the instrument was not pointed directly at the sun. This is provided to give an overview of system status so that a quick glance will show if a value changed significantly.

The c code is run first to extract the raw binary data from CD's holding the archive data. The IDL code is run next to produce plots of the data. The plots show all of the XRS-relevant data as a function of time for the entire 24-hour day and for just the times when the instrument was offpointed, in calibration mode, or in slew mode. Plots are also produced of the XRS flux signal as a function of SAS pointing direction. The code is automatic, so any date can be chosen for analysis. This code was run on data file 35718.5 (and the following day) per recommendation from Lorne Matheson. Indeed this date shows times of XRP off pointing whereas the following day did not. Additional files were not processed due to lack of time for this project.

All of the code was documented as it was written and tested. Each file starts with a header section containing information about the input and output variables, purpose and history of the code, and other pertinent information. (The information is formatted so that IDL's doc_library can be used to retrieve this information.) The IDL code also makes use of several routines in Pat's "library" of standard routines that she has written.

## Modifications and Additions to CD-reading code

- The code was corrected to start processing at the start of the data files
- The code was corrected for the location of the XRS preamp data in the subcomm telemetry stream (goesi_subcom.c)
- Preprocessor-type option (EXACT_SUBCOM) added code to provide the either exact subcommed values (with non values when not subcommed) or to provide the most recent value provided (goesi_subcom.pro).
- Added capabilities to return actual telemetered data values as well as values converted to engineering units (goesi_subcom.pro)
- Other minor changes so simplify the code, such as extracting repeated code to separate routines.
- Added capability to print telemetered values to a specified file (print_xrs.c) and set that file to be based on the name of the original data file (xxx.c).
- Added additional information to output, including the status bits as binary representations in ASCII (e.g. 1101 for four bits) (print_xrs.c)
- Values sent to file can be either telemetered values or engineering values, depending on values of preprocessor-type option (PR_TELEM) (print_xrs.c)
- Added solar array current data and solar array status (stepping, tracking) to the output file (print_xrs.c)
- Added code to send information sent to file to also be displayed on screen (could later remove or put it in a preprocessor-type conditional) (print_xrs.c)

- Changed file output to be completely fixed format, so it can be read simply by reading values in different columns (print_xrs.c)
- Reviewed code and added comments to document what code is doing and why. Based on conversations with Lorne Matheson.

## New IDL code

- Wrote code to do all the processing and plotting with a single command (doit4goesi.pro)
- Wrote code to read the ASCII files of telemetry data (rdgoesi.pro).
- Ported code to IDL to convert data to engineering units (parse_goesi.pro, calc_xrs_coef.pro, telem2eng.pro).
- Simplified the code to convert to engineering units. Save conversions coefficients in arrays and loop through for different conversion codes rather than repeat nearly identical code for each conversion. This, I believe, makes it easy to modify the values to for other calibrations, and certainly assures that all the functional code is identical. (I've verified calibrations for GOES-I but not the other spacecraft in the series I-M). (goesi_vcurve.pro)
- Wrote code to plot time sequences of all data that could be important for XRS analysis. (plot_goesi.pro)
- Wrote code to process times when the XRS was in calibration mode or when the XRP (X-Ray Positioner) was in slew mode and plot the results (do_xrscal.pro, get_xrpslew.pro, get_xrscal.pro).
- Wrote code to find times when the SAS indicated the XRS was not properly pointed plot the results (do_xrscal.pro, get_xrpslew.pro, get_xrscal.pro, plot_cal.pro).
- Converted code to IDL .(xrsi_conv.pro) to convert XRS fluxes to engineering units. (This function was not included in the original C code received from Lorne, which only read the telemetry from the archive data files.) Changed this code to process fluxes at all times, rather than return a nonvalue during offpointings, since data is needed during offpointings.
- Wrote code to display time series data during each slew, offpoint, or calibration interval. Plots include time-series information and plots of values as a function of pointing angle., (plot_cal.pro, plot_xrsoff.pro)

## C Code Structure:

| | |
|---|---|
| goesi_xrs_list.c | the main code |
| files_raw | opens file containing raw telemetry |
| goesi_unpk | converts byte frame to telemetry frames |
| goesi_wild_times | checks record for evidence of tar- and pax-induced errors |
| goesi_28s | finds DMS-induced 28-sec errors |
| goesi_frame_chk | checks that non-varying telemetered values did not change |
| print_xrs | does processing specific for XRS |
| eclipse | identifies eclipse times (see goesi_veclipse for comments) |
| goesi_subcom | extracts subcommed data |
| goesi_vcurve | converts telemetered to engineering values |
| goesi_sattde | gets Solar Array telemetry info |
| goesi_hrminsec | time conversion used by several modules |
| goesi_read.h | header file of structure definitions and prototypes |

goesi_veclipse not used, looks like eclipse.c which is used. Documentary comments were added to this one; these should merged into eclipse.c

## IDL Code

doit4goesi.pro

rdgoesi.pro

- get_filename.pro from PLB's library of routines, interface to get a legitimate file to open
- interp_path.pro from PLB's library of routines, used to extract parts of file path (for creation of output filename)
- parse_goesi.pro parses raw-data array to structure variables
- telem2eng.pro converts telemetry values to engineering values
  - goesi_vcurve.pro provides the conversion factors for the calibration curves
- plot_struct.pro plots then telemetered and engineering values side-by-side
- plot_goesi.pro plots all the xrs-related variables
  - plot_fwin2.pro from PLB's library of routines. Opens a plot window that fits the portable PC.
  - wait_user from PLB's library of routines. Requires user response if plots displayed on screen.
- do_xrscal.pro
  - get_xrpslew.pro gets the indices of data points where XRP was is slew mode
  - plot_goesi.pro used to plot values at times XRP was in slew mode
    - plotg.pro creates a plot with the data range included in the plot title
  - plot_xrscal.pro used to plot additional values during XRP slew times
- do_xrpoff.pro
  - get_xrpoff.pro locates all times when XRP/XRS are in the slew mode or in calibration mode and plots the results
    - get_off.pro gets the indices of data points where pointing exceeds specified limits
    - no_strays.pro used to eliminate isolated intervals that are too short
      - seqinfo.pro extracts temporal properties of intervals
    - gap_extend.pro used to extend intervals of offpoints
  - get_xrscal.pro used to get the times when the XRS is in calibration mode
  - plot_xrsoff.pro plots all values during time of calibration or slew
    - lim_range general routine that provides range values whether input is a single +/- value or a 2-element range

wait_user is used by several routines to give user a chance to view plots displayed on the screen. Has no affect when plot device is a postscript file.

## Data-Interpreting code

gi_xrs_coef.c

- xrsi_conv.c
- calc_xrs_coef.c

XRSFluxCode

- xrsi_conv.c
- calc_xrs_coef.c

## Additional Code

| | |
|---|---|
| dopath4rdgoesi | sets the current working directory and IDL search path. Is run prior to doit4goesi. (Setting path through the IDL preferences did not seem to work well for multiple projects. |
| patharr4lib | is used by dopath4goesi. It provides the paths to PLB's library of general IDL routines. |
| ps_start, /land | when run before doit4goesi, will capture all of the plots to a single postscript file. Should close the postscript device with output_plot) |
| output_plot | is required to finish the postscript file started by ps_start and save the file to disk. On UNIX machines it will also try to sent the plot to the printer. |
| xrsi_conv.pro | being developed to convert flux telemetry values to x-ray fluxes |
| plot_xrscal.pro | not currently used. Developed to plot values during XRS calibration-offpointings |

## ISSUES:

The telemetry-to-flux conversions were checked for GOES-I and found match those provided by Panametrics. The calibrations for the other spacecraft were not checked, but presumably, the components (thermistors, voltage monitors, etc.) are of the same batch and are subject to the same conversions. The XRS fluxes are known to differ and the values included are those according to Panametric's revised calibrations.

Eric Chipman produced a report dated 10/5/95 that examined the XRS angular response during PLT testing. He found the "locations and responsivity of the secondary peaks are different by approximately 10%. Also, the flatness of the predicted central responsivity maximum seems flatter than the data, which appears to show a subpeak near the north part of this peak." He also says "the E/W width which is within 2.5% of the peak is actually closer to 0.6° than to the design width of 1°." In the conclusions, he says "the West edge of the longwave FOV is different in detail from the predictions."

On-orbit verification of the XRS angular response will be complicated by the fact that the sun is not a point source. Bornmann and Matheson 1990, A&A, 231,525, used a lunar eclipse to extract the XRS contribution from four emitting regions that were significant contributors to the XRS flux at that time. Because there are no imaging solar x-ray monitors that cover exactly the same flux response as XRS, full analysis will require assumptions about the distribution of flux on the solar disk.

The thermal corrections are included in the code for calculating actual solar fluxes, but look like they will be a minor effect because the temperature does not change much during the short time when the instrument is offpointed. Analysis so far has focused on the variations of the telemetered (raw) data rather than solar fluxes in engineering units. The temperature of the XRS preamp, in telemetry units, varied from 181 to 188, over the course of a day (file 35718.5).

## Comments on Angular Response Analysis

1. The angular resolution of the Sun Angle Sensor (SAS) requires interpolation between SAS points to determine the pointing as a function of time. SAS resolution is 0.016 degrees. Solar Diameter is 0.5 degrees. X-ray emitting regions are around 100,000 km or about 0.03 degrees near disk center. XRS angular response is expected to vary as follows, according to Panametrics plots: Flat at 100% response from -0.5 to 0.75 degrees for 1-8A on the E-W axis, and linear variation from 95% at +/- 0.5 degrees to 100% on axis for the other cases. For both channels and directions, the response falls significantly beyond these limits.
2. XRS pointing varies during the day: Drift analysis done by Pat Bornmann for the SXI project showed E-W daily drifts up to 10 acmin = 0.166 degrees in both E-W. and N-S directions (GOES 8 October 19, 1995). The XRP, which controls the XRS pointing is recentered once per day, without consideration of where in the drift pattern the XRP is pointing at that time.
3. Solar X-ray Sources are distinct regions on the solar disk. Not a point source. Not a uniformly emitting disk. Proportion of flux from each source depends on the solar active region and the wavelength, because the flux ratio depends on the temperature of the regions and their emission measure. (See Bornmann and Matheson, 1990).

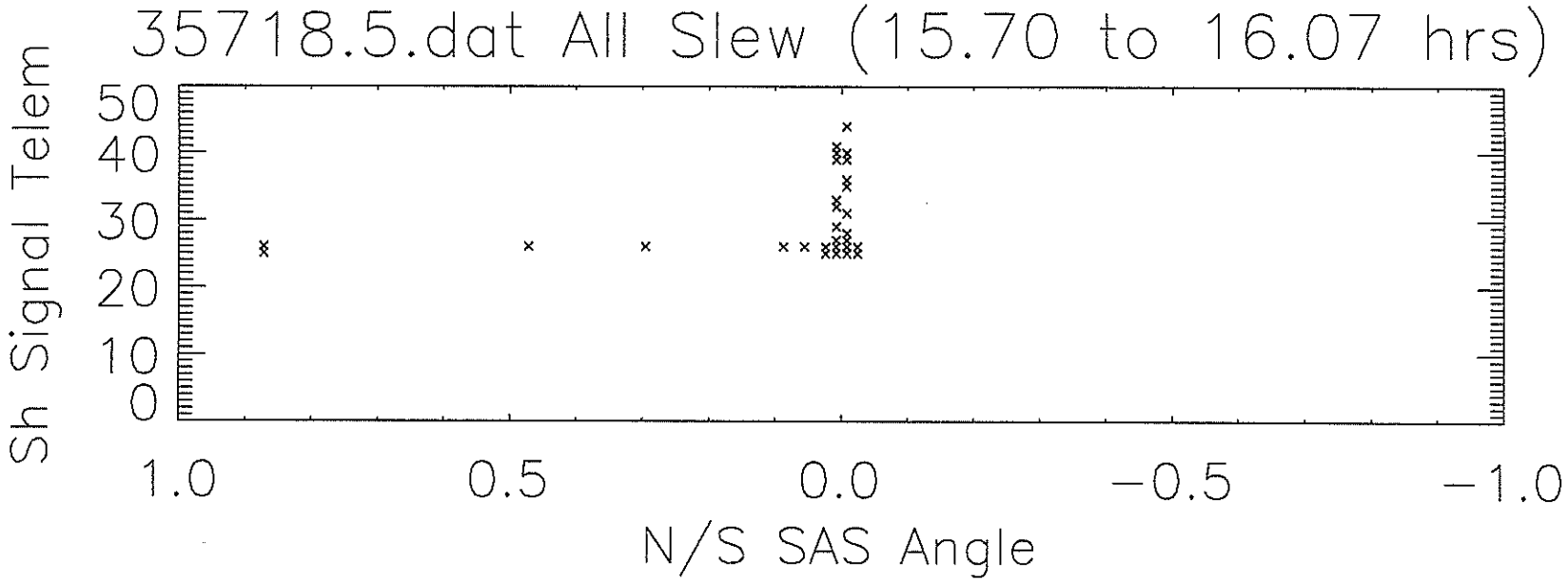
## Other Observations:

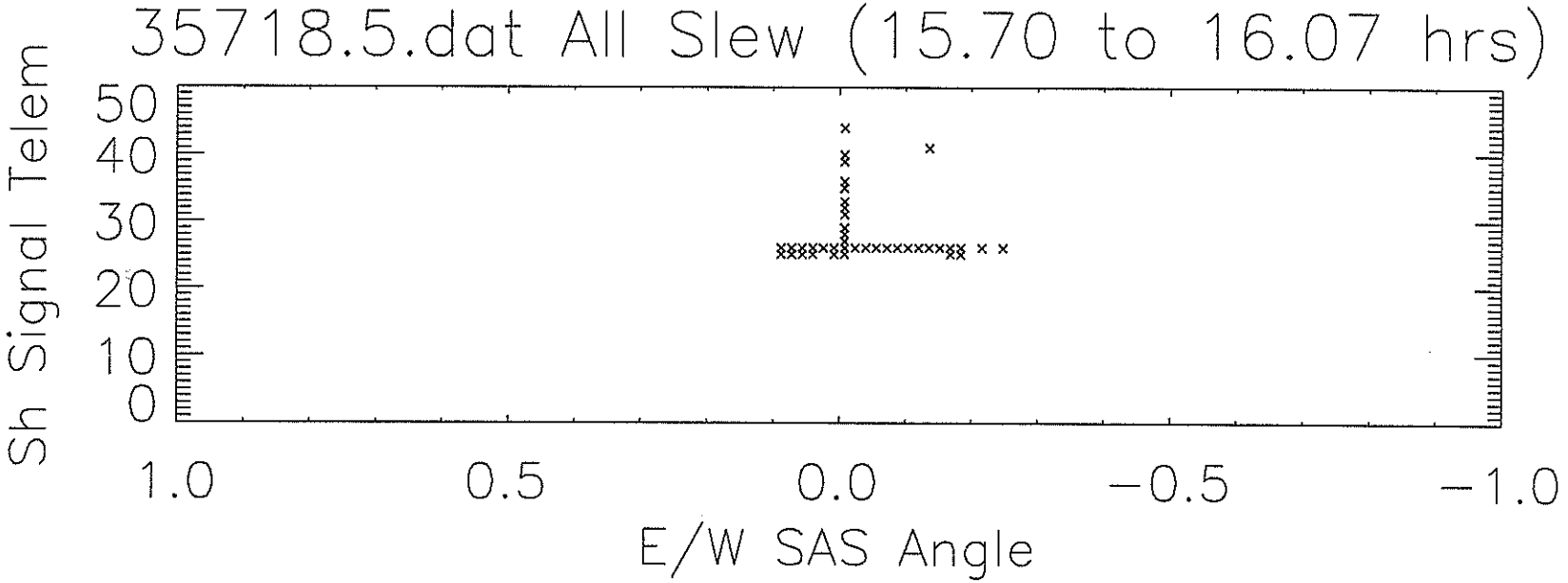
1. Temperature of the XRS preamp and XRP remained fairly constant over a 24 hour period. Plots of the XRP bearing temperature, XRP electronics temperature, and SAS temperature were pegged at zero, which could mean they were not properly extracted from the raw data or were not present in the raw data. This is not critical for initial analysis of the angular response and has not been pursued.
2. Voltages also did not vary much for the XRS calibration, reference and chamber.
3. The solar array, primary buss current, and telemetry control current show a periodicity in the "noise." This may be related to activities with the GOES Imager and Sounder. These affects are not critical for the initial analysis of the XRS's angular response, and do not vary noticeable during slew times.
4. The amplitude of the telemetered Coarse E/W readings are larger than those of the XRP/SAS E/W readings. This seems to be counter-intuitive. The N/S coarse position quantizations are 0.48 degrees, and the SAS N/S and E/W quantization is 0.016 degrees.

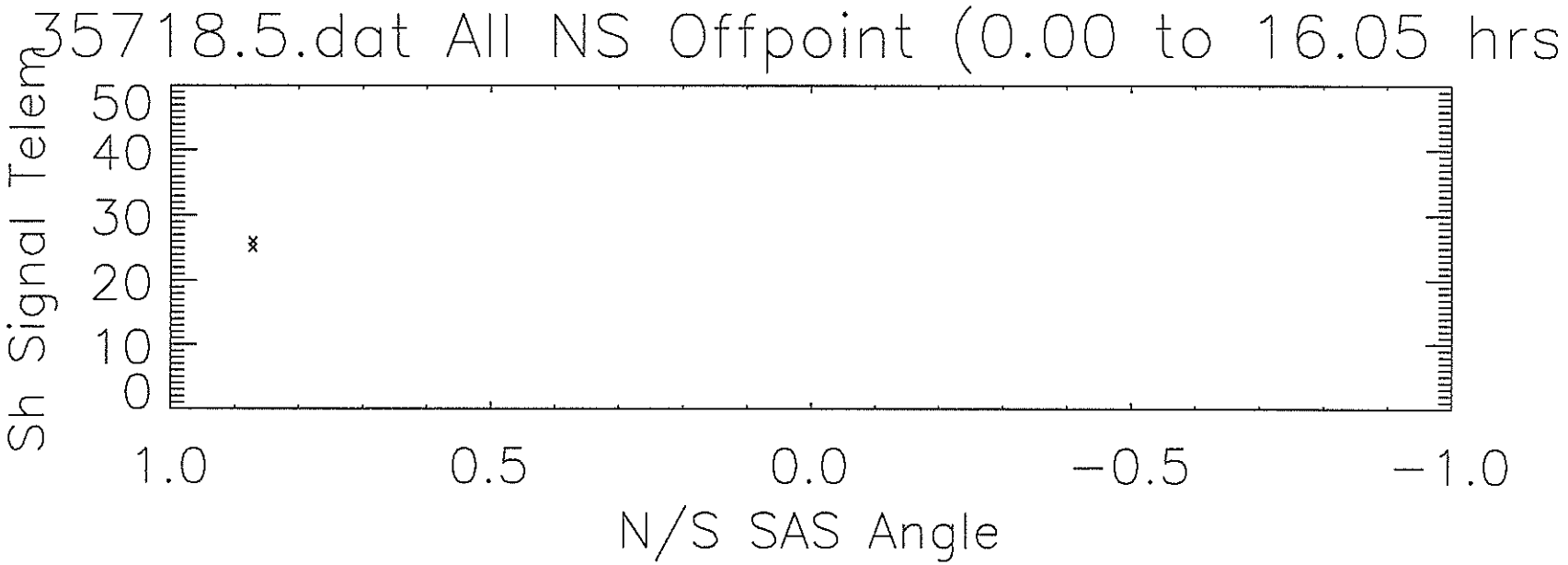
## References

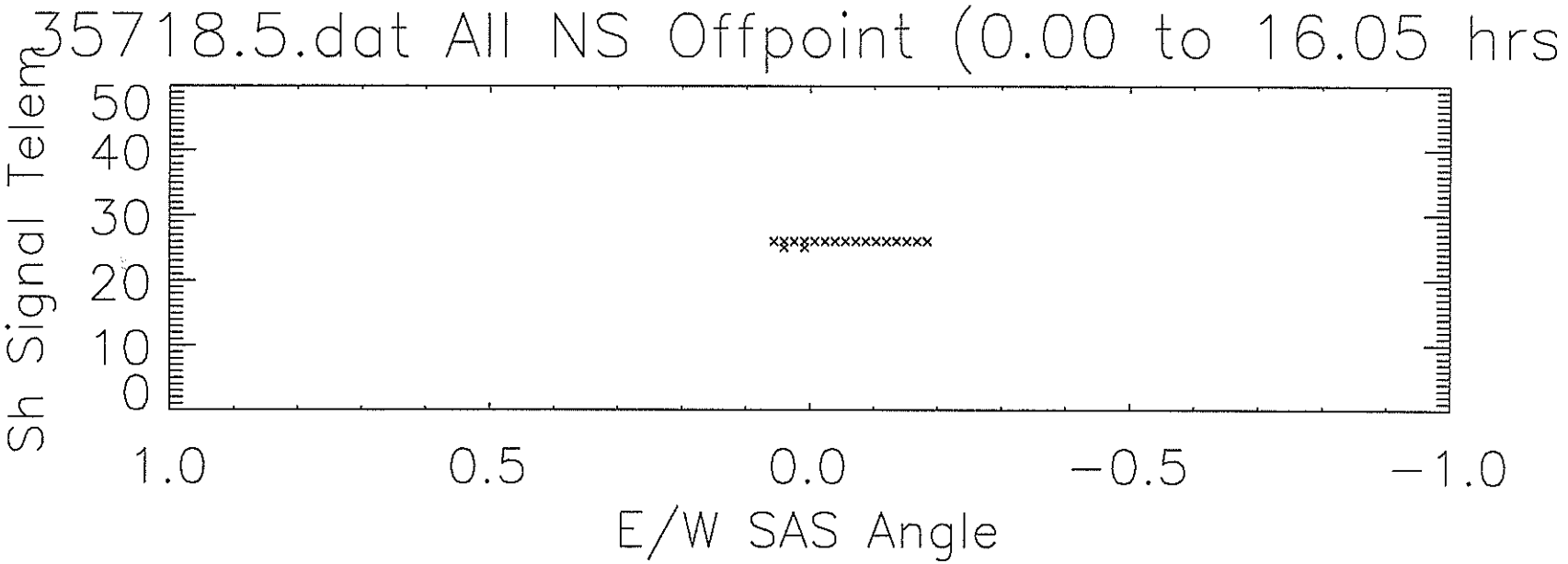
Bornmann, P. L. and Matheson, L. 1990, *Astronomy and Astrophysics*, 231, 525-535, "Solar Flare Plasma Properties Derived from the Disk-Integrating GOES X-Ray Sensors during an Eclipse."

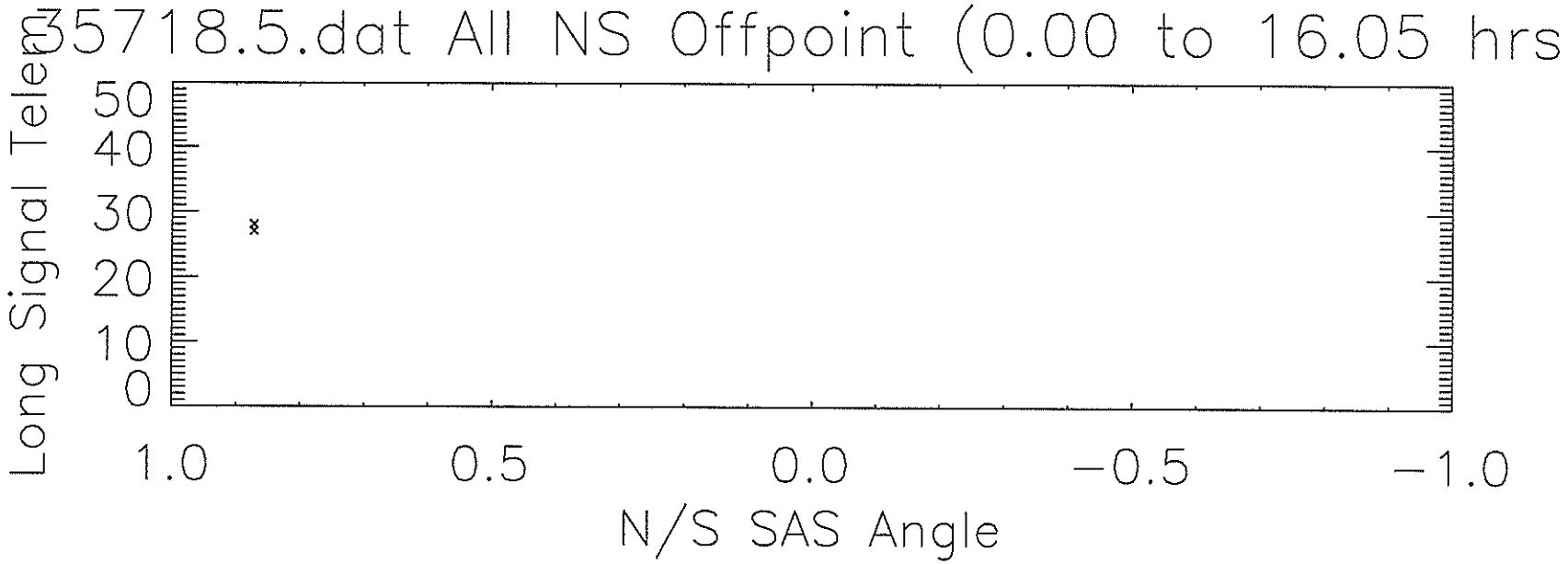
Chipman, Eric (not noted on report), October 10, 1995, Report on test EF0408, "X-Ray Positioner (XRPE) Operation XRS Field-of View Mapping"

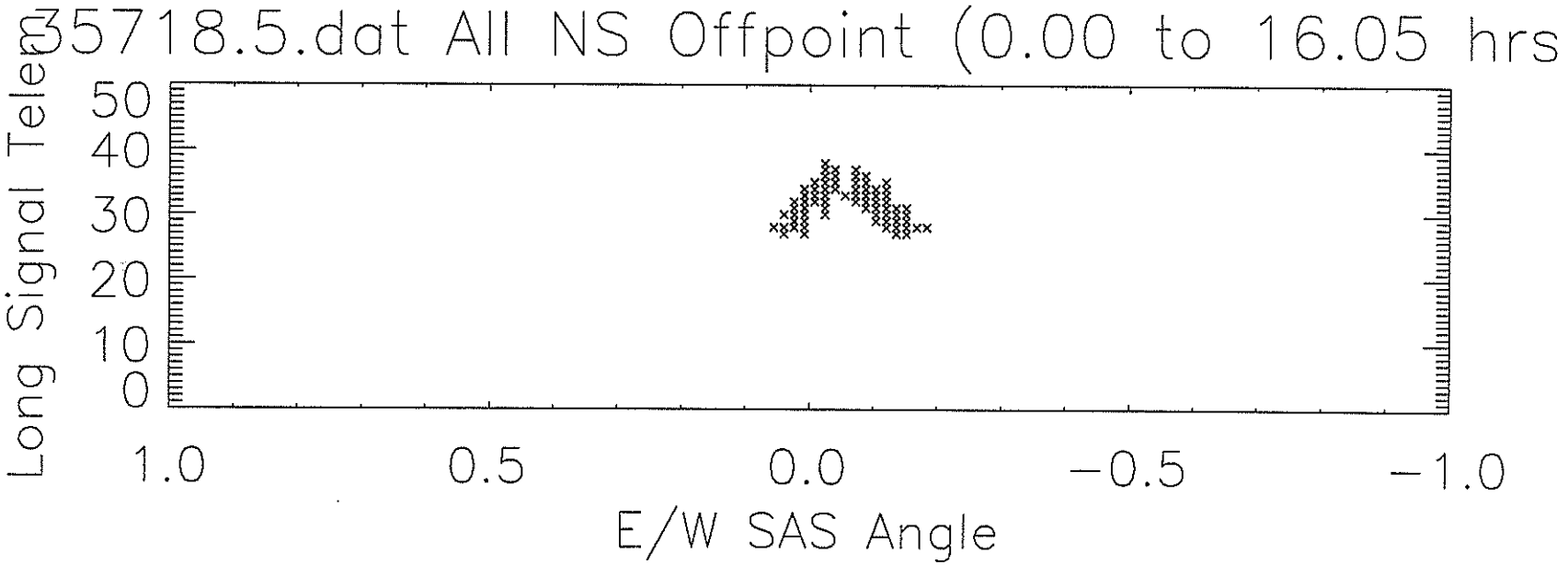

35718.5.dat All Slew (15.70 to 16.07 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

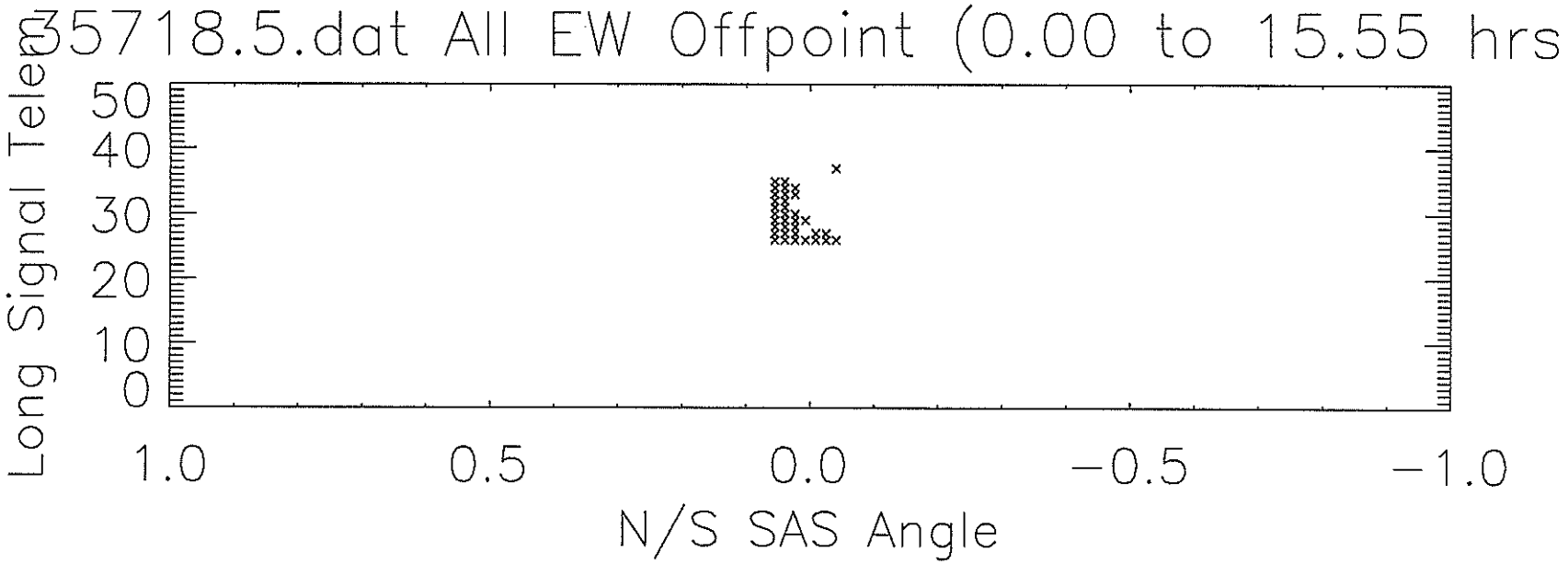

35718.5.dat All Slew (15.70 to 16.07 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

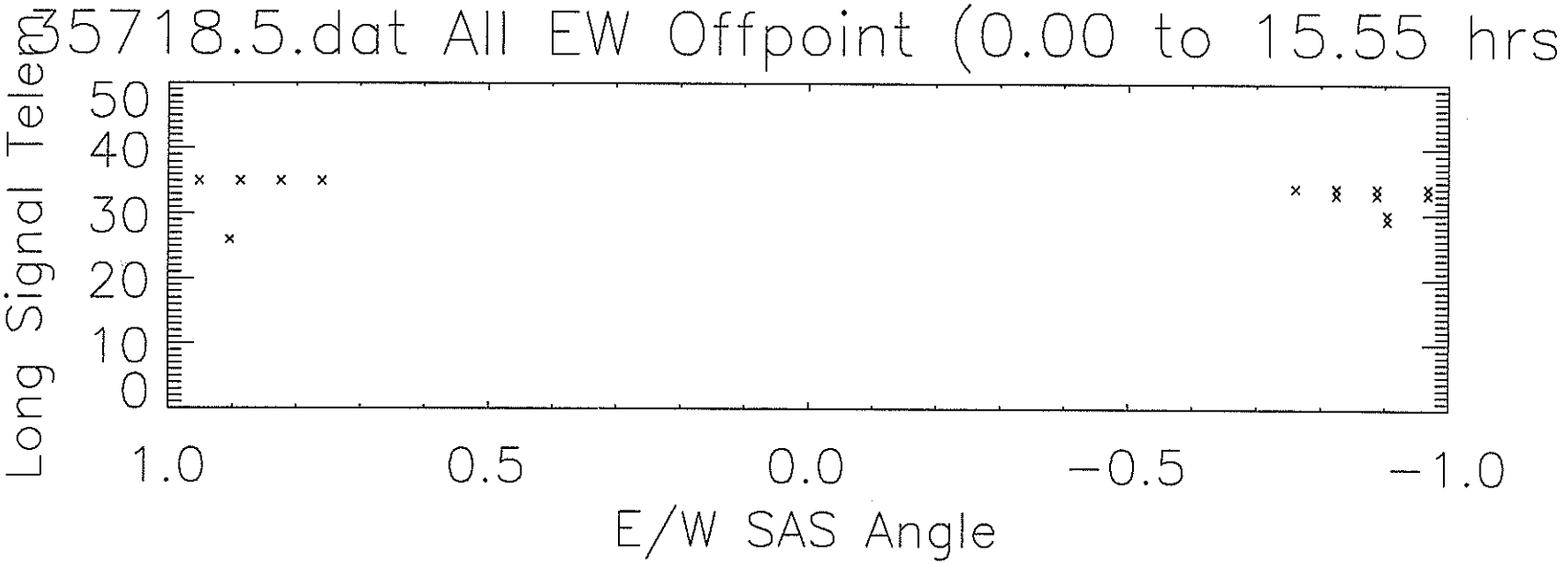

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

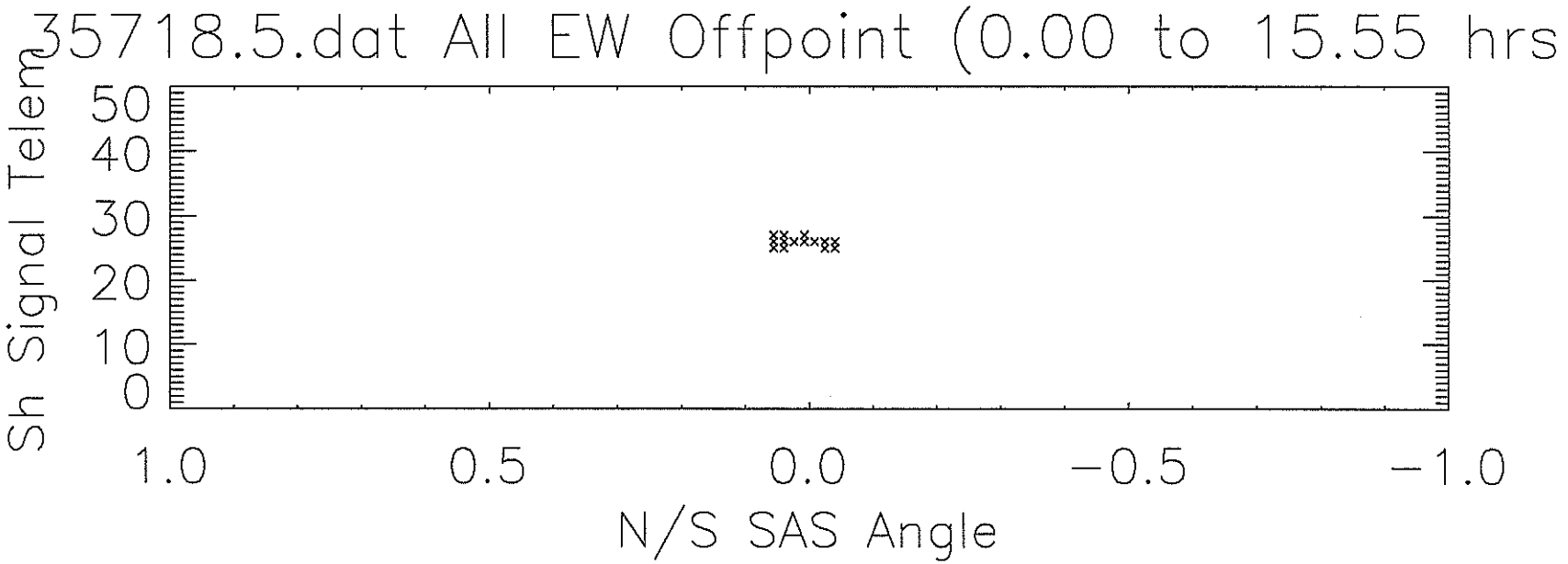

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

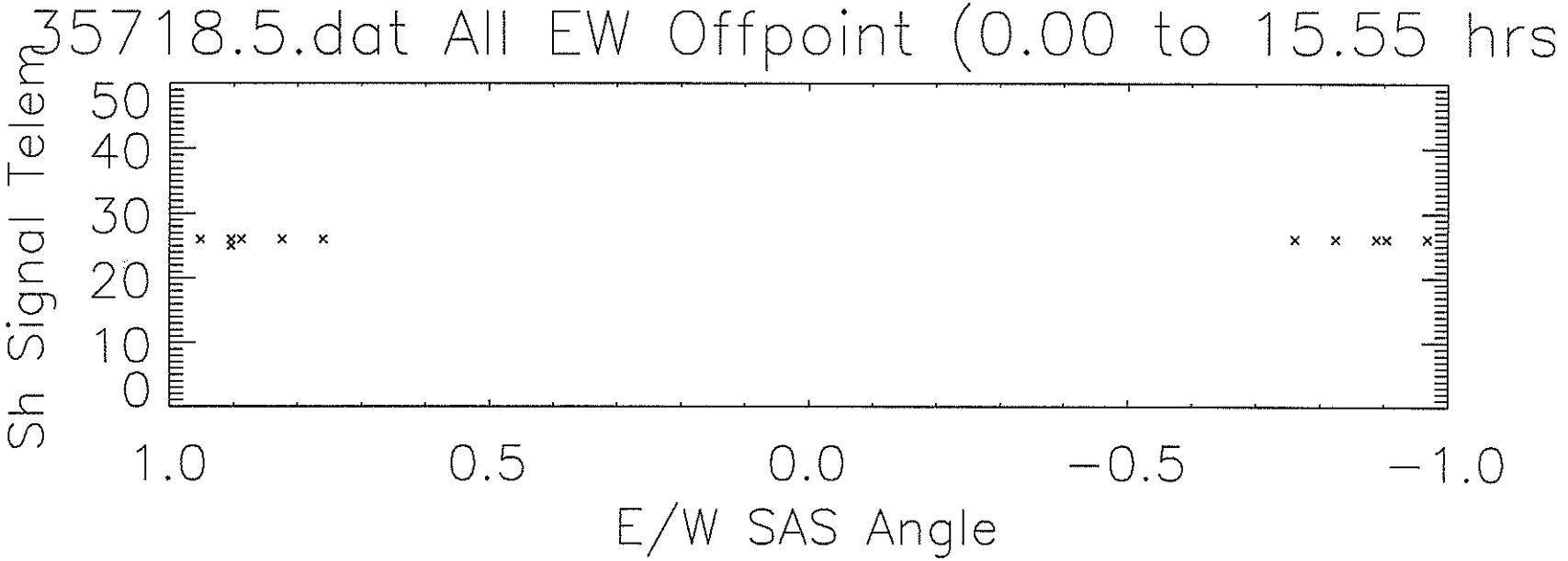

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

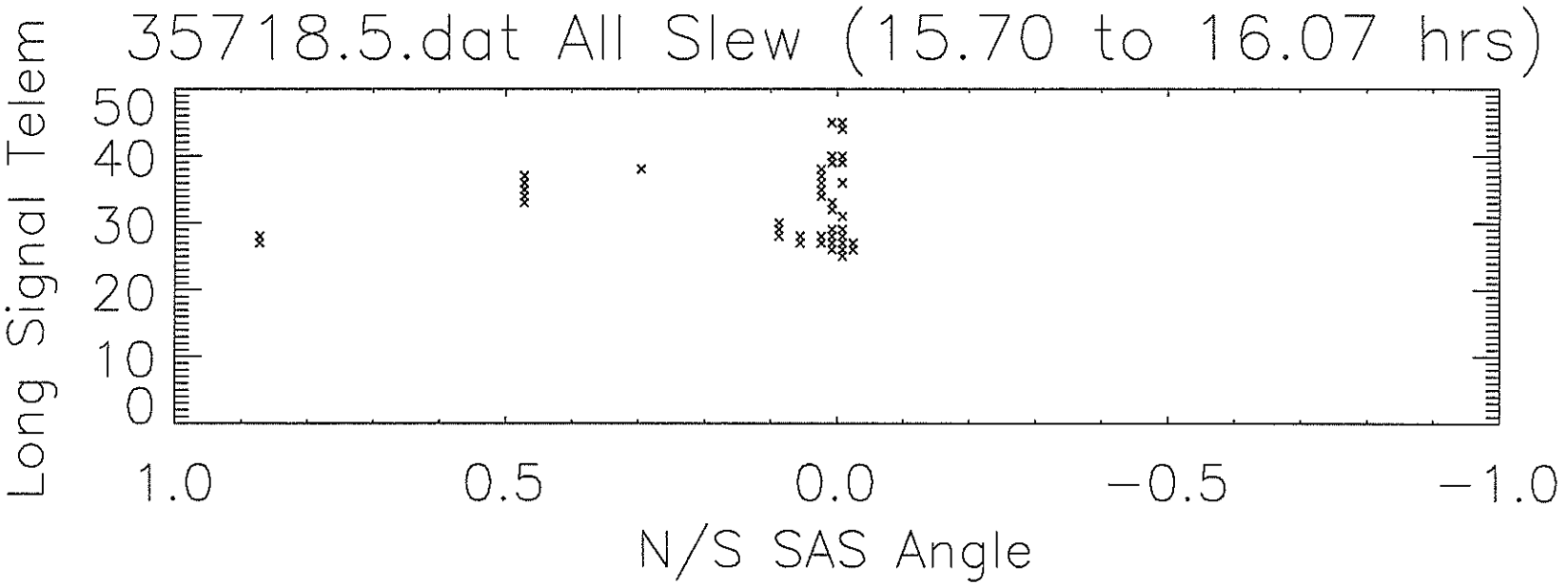

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

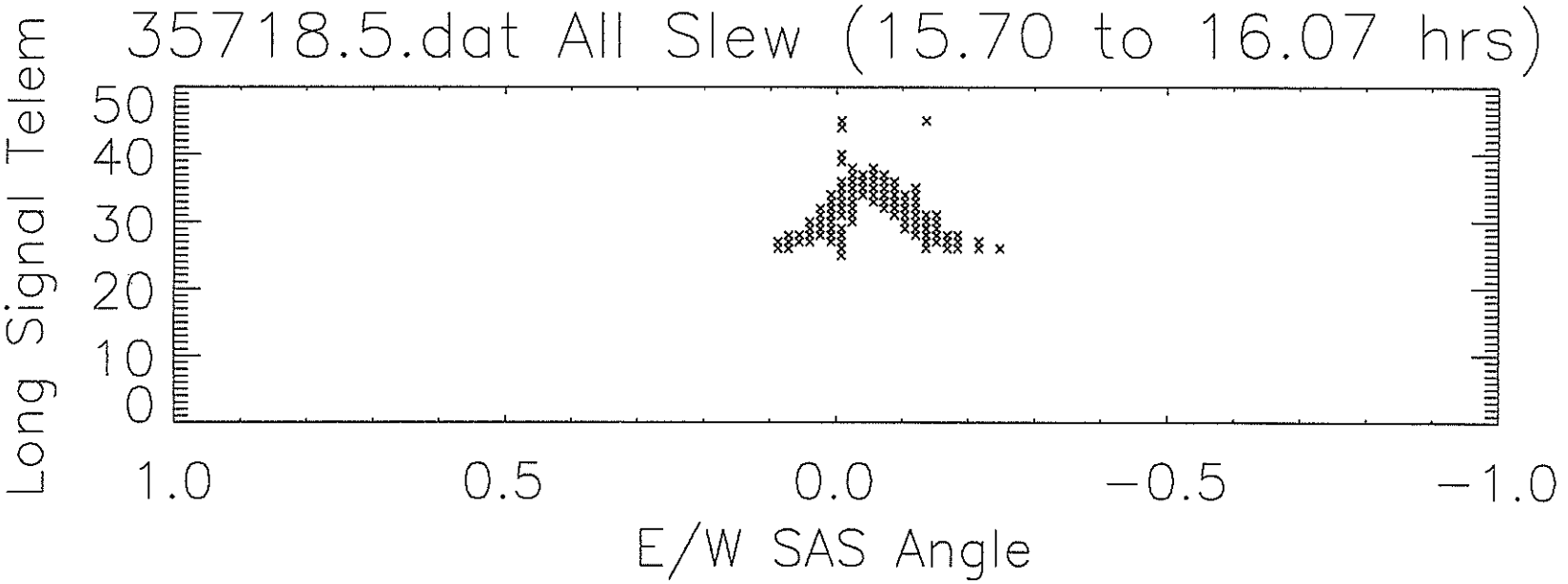

35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle


35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle


35718.5.dat All Slew (15.70 to 16.07 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat All Slew (15.70 to 16.07 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

~5/15/96

which S/C?

| | | |
|---|---|---|
| **REPORT DATE** | : | **October 10, 1995** |
| **TEST NUMBER** | : | **EF0408** |
| **TEST NAME** | : | **X-Ray Positioner (XRPE) Operation**<br>**XRS Field-of-View Mapping** |
| **TEST PHASE** | : | **ACT** |
| **TEST OBJECTIVE** | : | **Operate the XRP through its range by ground command, and verify each step in the telemetry. The XRS Channel A and Channel B responses to solar X-ray flux are verified. The Sun tracking mode is initialized and tested. The near and extended fields-of-view of both X-Ray channels are mapped, and the optimal SADA SAS and XRS SAS readings for X-Ray sensitivity determined. The magnetometer output is monitored for evidence of magnetic interference due to the XRS sweeper and bucking magnets and stepper motor magnet and/or solar array current. The effects of XRP motion in the SLEW and TRACK modes on spacecraft stability are determined.** |
| **DATE TEST PERFORMED** | : | **Day 169, Day 237, Day 285** |
| **SPEC COMPLIANCE** | : | **3.5.2.8** |

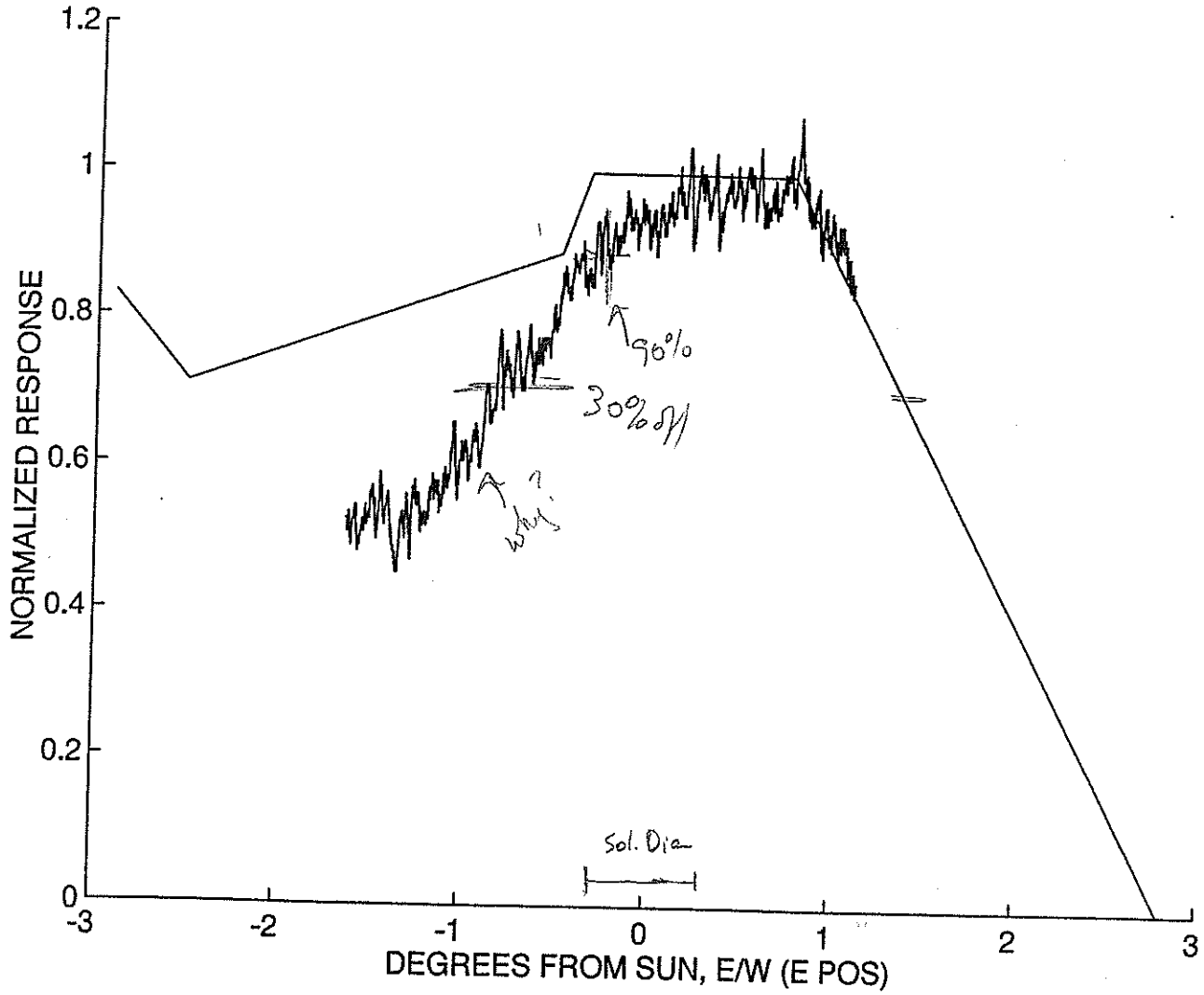
**TEST DESCRIPTION:**

The XRPE is enabled and verified in the telemetry. With the SADA coarse angle potentiometer reading at the calculated position of the sun in spacecraft azimuth, the XRP is slewed from stow through the sun to 23° north. It is then returned to the apparent position of the sun, and commanded to TRACK mode. The SADA is then slewed over a total of 90° centered on the sun to determine the near optimum E/W pointing angle as monitored by the SADA SAS, as well as mapping the extended E/W field of view. This optimum angle is verified by stepping the solar array through small angles about the optimum. Then the XRP is stepped through its N-S range to map the N/S field of view at that E/W position. This mapping is repeated for SADA E/W SAS angles of ±0.25° and ±0.5°. For XRS N/S SAS angles of 0 deg, ±0.25° and ±0.5°, the SADA is slewed 20 deg centered on the sun. The XRS field of view for both channels is thereby mapped in two dimensions.

Simultaneously, the magnetometer is active and the magnetometer data is searched for evidence of magnetometer susceptibility to the XRS sweeper and bucking magnets and XRP stepper magnet, as changes in dipole orientation occur. In addition, magnetic signatures are acquired in the slew mode to minimize the effect of ambient field drift and at four local times in order to characterize the magnetic signature over 360° of solar array rotation without rotating the solar array into eclipse. This phase should be coordinated with corresponding steps in PLT EF0404.

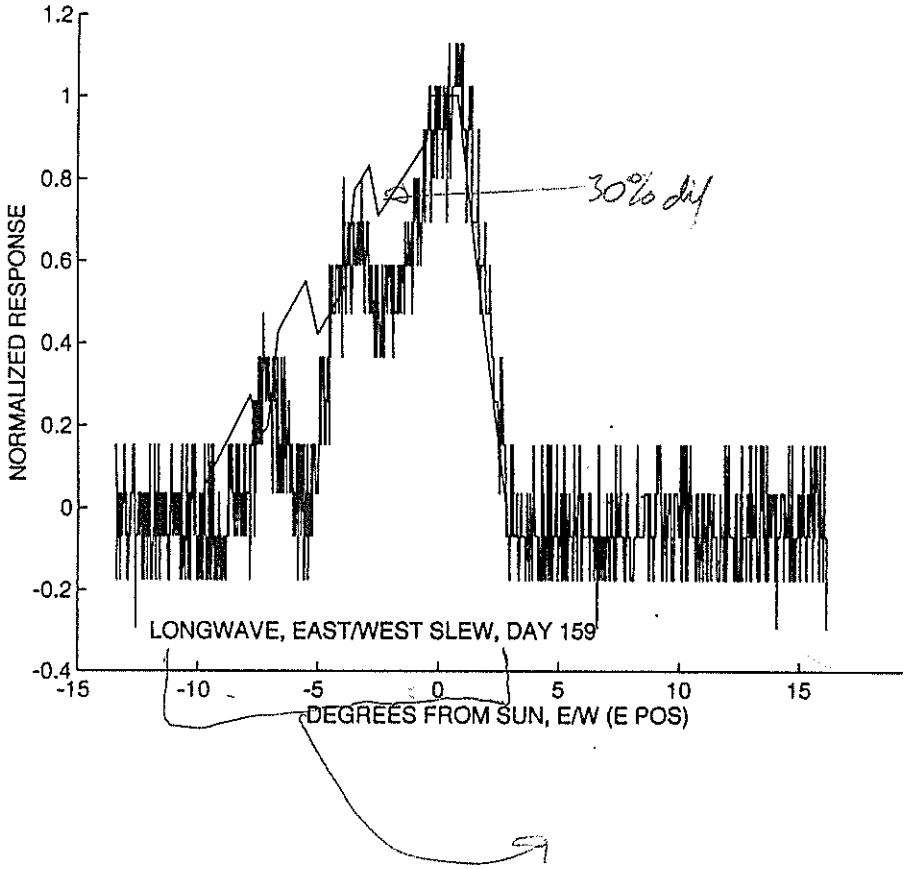
Since the X-Ray flux during active periods is quite variable, X-Ray data from another GOES spacecraft is desirable to normalize angular response of the XRS to fixed X-Ray flux levels in order to

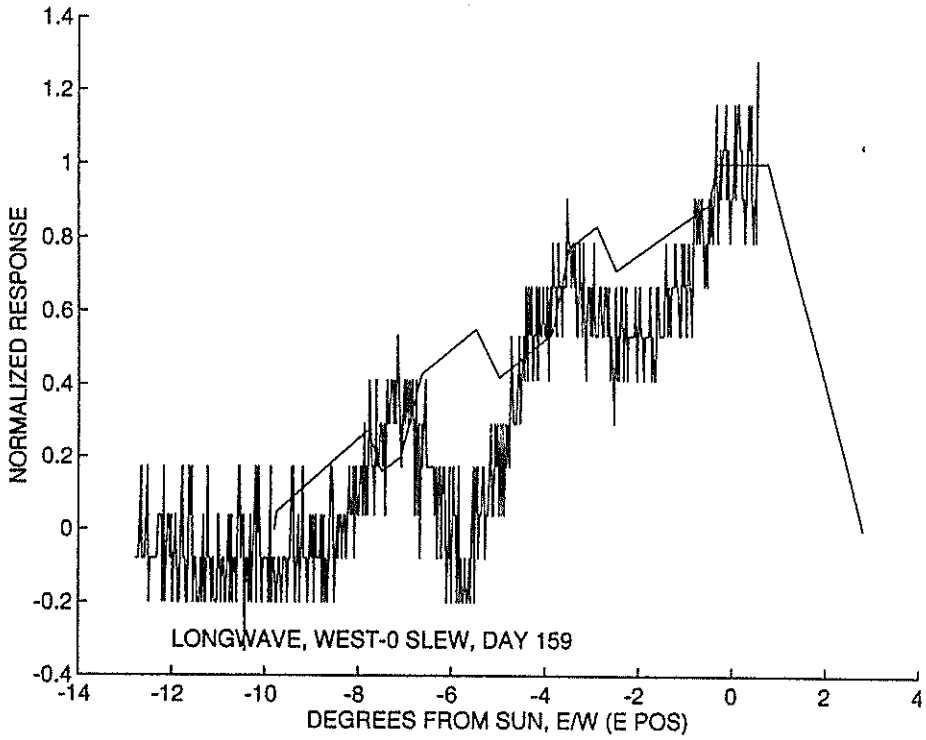
ITEM 10




LONGWAVE, FINE DRIFT, DAY 159

Why dif?

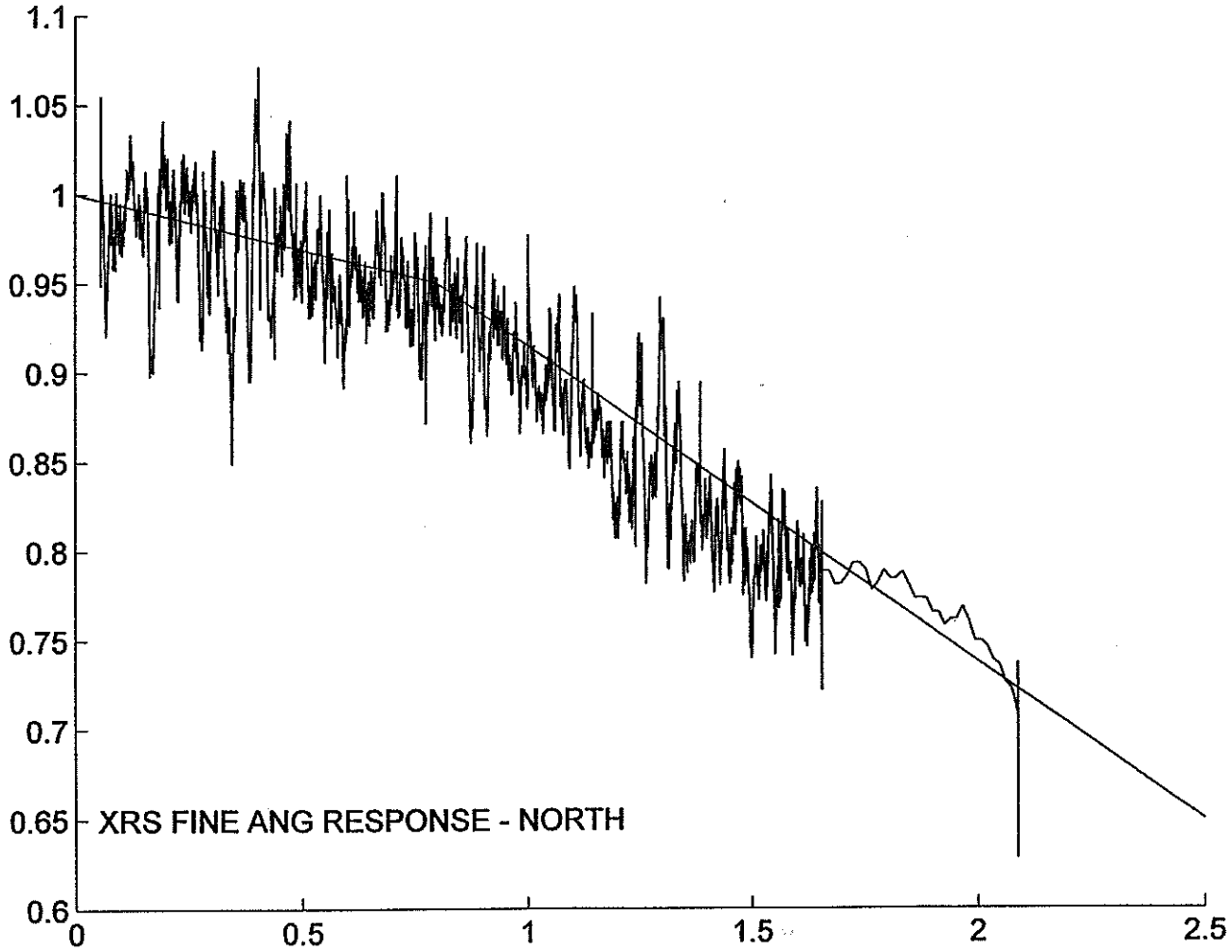


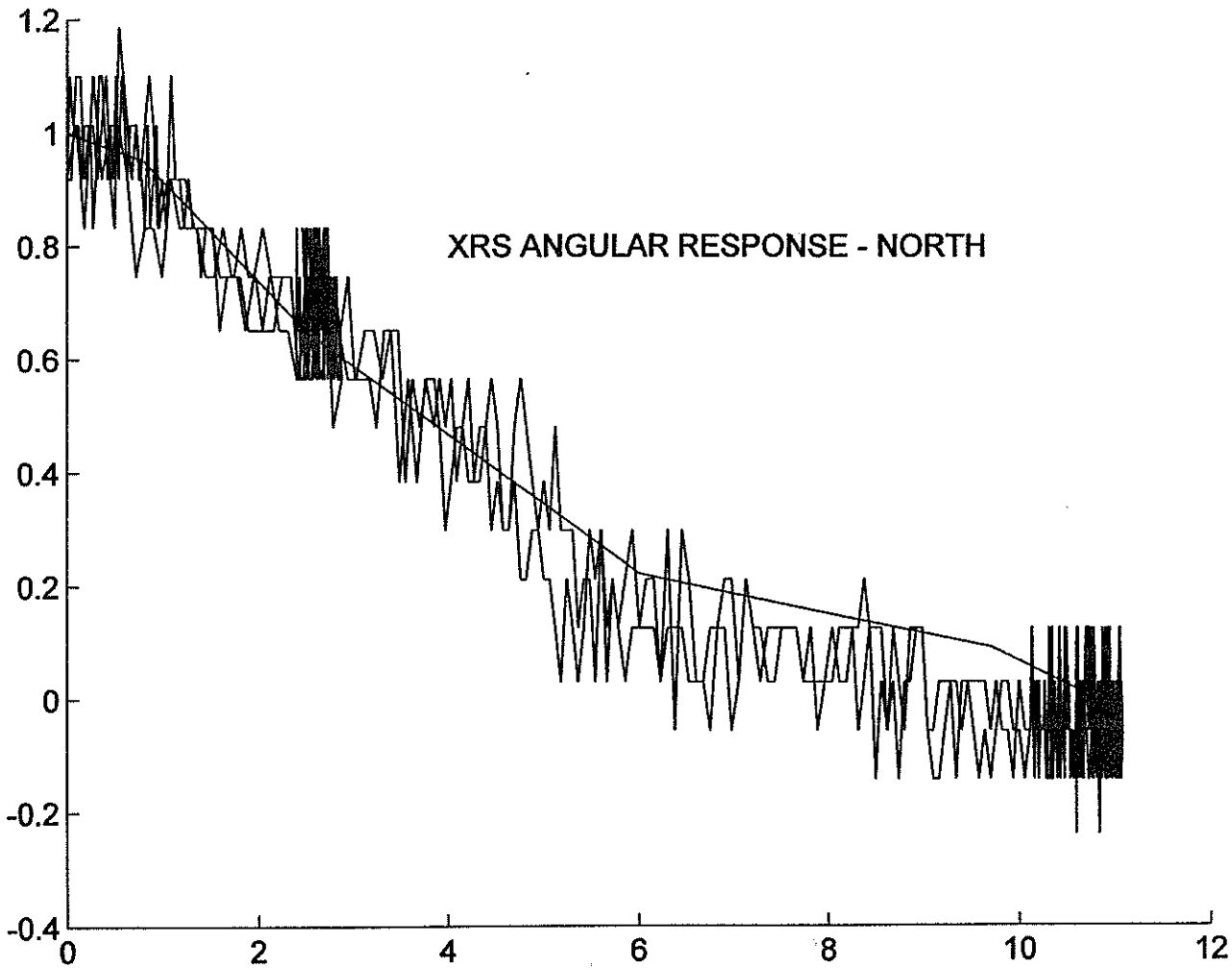

LA- 68 36±5 noise

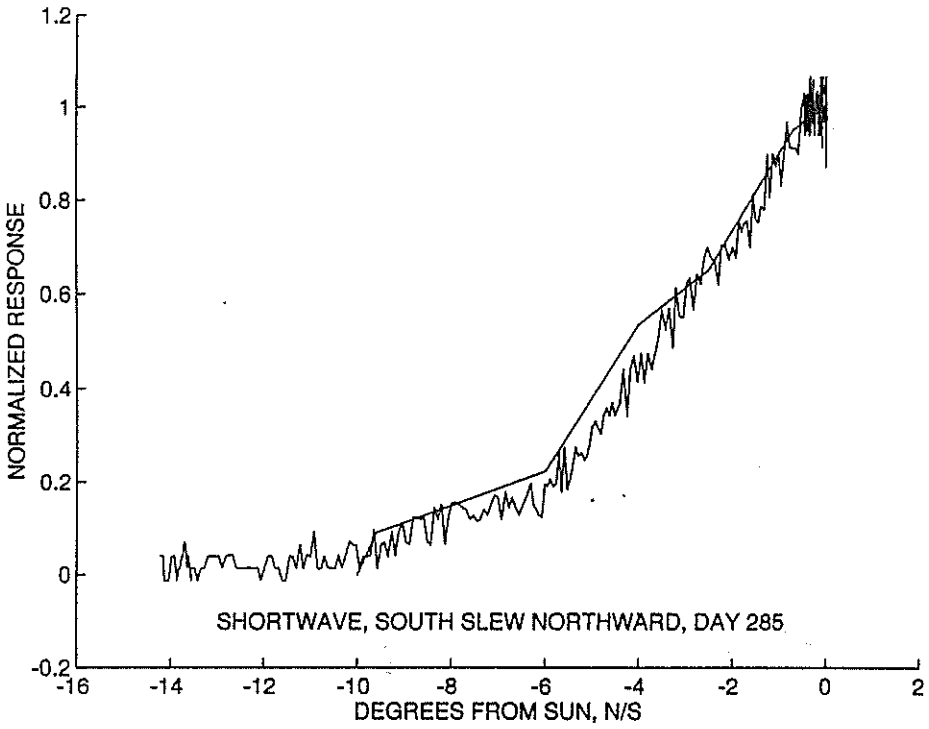
ITEM 9

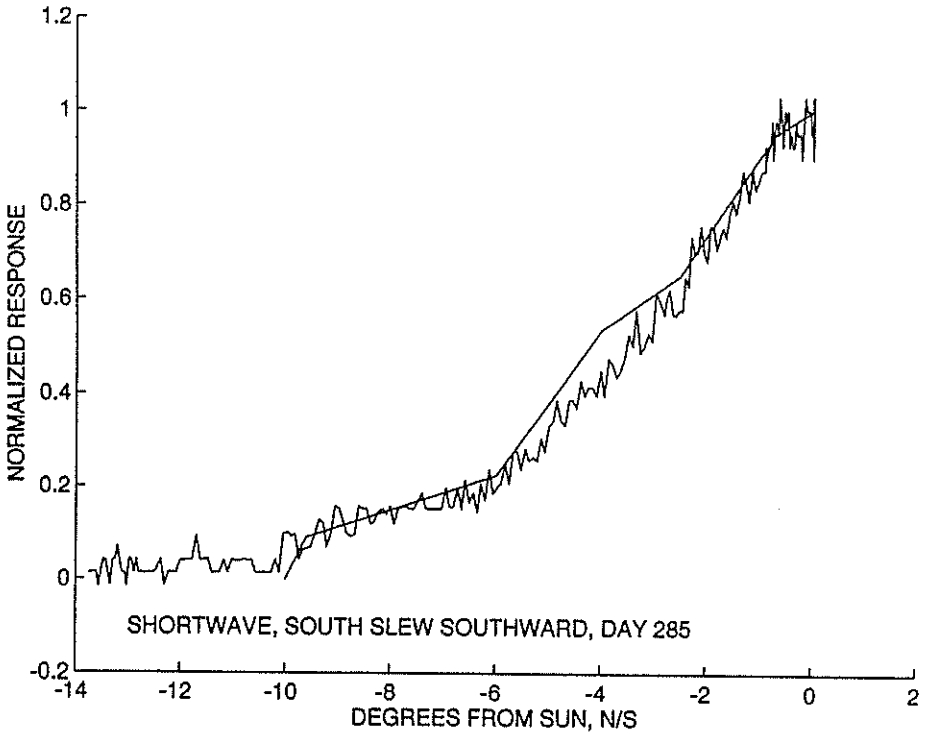
ITEM 8


1.1
1.05
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
0.5
1
1.5
2
2.5
XRS FINE ANG RESPONSE - NORTH

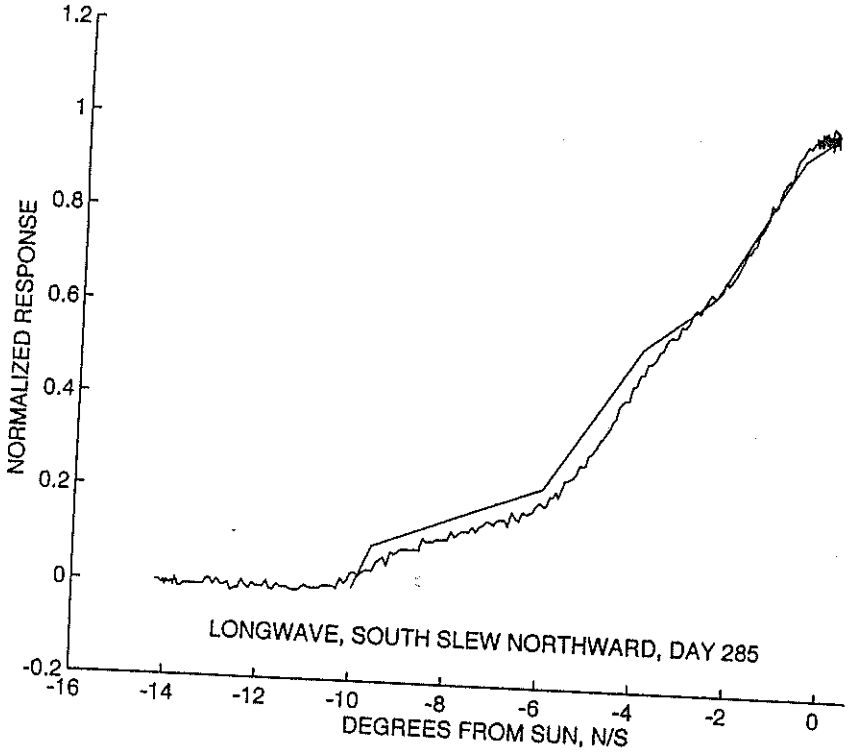
ITEM 7

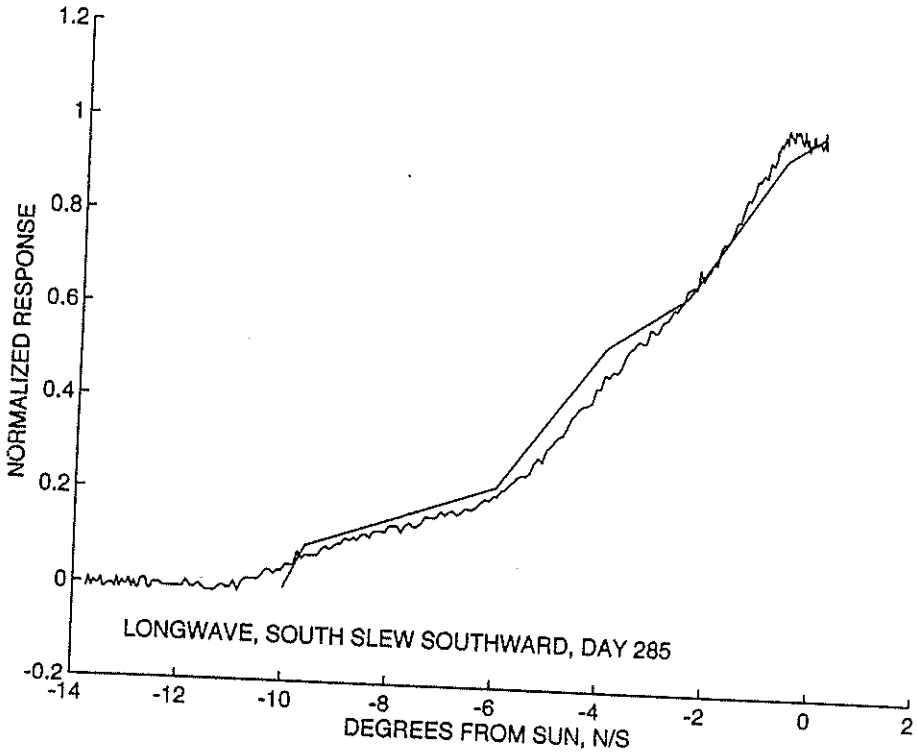



NORMALIZED RESPONSE
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-16
-14
-12
-10
-8
-6
-4
-2
0
2
SHORTWAVE, SOUTH SLEW NORTHWARD, DAY 285
DEGREES FROM SUN, N/S


NORMALIZED RESPONSE
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-14
-12
-10
-8
-6
-4
-2
0
2
SHORTWAVE, SOUTH SLEW SOUTHWARD, DAY 285
DEGREES FROM SUN, N/S

ITEM 6


NORMALIZED RESPONSE
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-16
-14
-12
-10
-8
-6
-4
-2
0
DEGREES FROM SUN, N/S
LONGWAVE, SOUTH SLEW NORTHWARD, DAY 285


NORMALIZED RESPONSE
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-14
-12
-10
-8
-6
-4
-2
0
2
DEGREES FROM SUN, N/S
LONGWAVE, SOUTH SLEW SOUTHWARD, DAY 285

ITEMS

```
/*                      goesi_proto.h
 *                                          started 1994 2 Aug 94  ldm
 *                                          2 Sep 1994 - ldl
 *                                          24 Feb 95 - ldm
 *                                          init_subcom, subcom_conv
 *                    part of goesiave preprocessor
 */

/*   process aocs part of telemtry for signs   */
int  aocsi_conv (void);

/*   process word 35 digital aocs telemetry, called by aocsi_conv  */
void aocs_wd35(int word);

/*   process eps/hepad telemetry   */
int  epsi_conv (void);

/*   convert eps/hepad counters to counts per second   */
float get_i_cps (int compres, int count);

/*   interface with the Goesi_ave methods */
int Goesi_init(void);
void gi_ave_time(long dmsday, long dmssec, int millisec);
void gi_ave_datum(int d_ch, float d);
void gi_ave_status(int d_ch, long status);

/*   initialize some g_stat variables   */
void init_gstat(void);

/*   initialize the subcom coefficients   */
int  init_subcom(void);

/*   keep status words for displays   */
void keep_gstat(void);

/*   process magnetometer telemetry   */
int  magi_conv (void);

/*   decode main frame words   */
int  maini_conv (void);

/*   decode subcom words   */
int  subcom_conv (void);

/*   write to log file if activated   */
void writei_log (const char *);

/*   decode xrs words   */
int  xrsi_conv (void);

/* interface from main() to conversion and averaging functions */
int  goesi_init(void);   /* initialize the conversion and averaging functions */
int  get_i_raw (void);   /* get next raw record */
int  ave_i_raw(void);    /* process a raw record */


/*============================================================================*/
```

```
/*   *   *   *   *
 *                      goesI_read.h
 *
 * PURPOSE:
 *    constants and structures for standardized reading of goes I
 *    daily raw data files.  There is a record of 288 bytes each second.
 *    Each record consists of two identidal 144 byte portions that are
 *    a telemetry frame from the satellite with time, etc.  The second
 *    frame in the set may be all zeroes.  There will be endian
 *    problems in the data to be sorted out.
 *
 * QUESTION:
 *   what is in the various elements of the data array in the raw data structure?
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *
 *   *   *   */

#ifndef GOESI_READ_H
#define GOESI_READ_H

/* The data is stored on the CDROMs as DMS records */
#define CHAR_PER_FRAME   128
#define CHAR_PER_REC     288
#define FRAMES_PER_REC     2

#define REC_PRT            1


/*  *    *    *
 *   used to pass counters for passing to subroutines
 */
struct counters
{
    long day_large;      /*   number of records with very large dms dates   */
    long day_neg;      /* will contain count of number of negative dates encountered */
    long sec_large;
    long sec_neg;       /* will contain count of number of negative times encountered */
};

/*   used for computations after breaking raw record apart into
 *       two frames.  This structure assumes longs are at least
 *       32 bits and ints are at least 16 bits.  Numbering of
 *       words in data matches Loral documentation, except that
 *       Loral word 128 is in data[0].
 */
struct    goesi_raw
{
    long   date;         /*   dms date   */
    long   time;         /*   dms seconds of day   */
    int milli;           /*   milliseconds, 0 to 999   */
    int agc;             /*   currently not used   */
    int port;            /*   input port identifier   */
    int node;            /*   QNX node where generated  */
    int sat;             /*   satellite identifier   */
    int chek;            /*   frame quality or validity
                          * 0x00  for normal, logical or'ed
                          * 0x01  wrong id
                          * 0x02  another satellite id
                          * 0x04  next synch words bad
                          * 0x08  not used, was text message
                          * 0x10  check sum, single bit error, fixed
                          * 0x20  check sum, single bit error, not fixed
                          * 0x30  check sum, multiple errors, not fixed
                          * 0x40  dwell mode is in effect
                          * 0x80  time quality suspect or unknown */
    /* Chek sets bits in most cases, except the check sum cases where
       it uses incremental values 0x10, 0x20, 0x30.  Thus all bits can
       be or'ed to find problems, except for these two bits.  */

    /* Dwell occurs when the spacecraft is commanded to fill all the
       subcom telemetry with a single variable.  When dwell occurs, the
       telemetry does not include the usual values.  Dwell is not
       available for SEM outputs. */
    int data[CHAR_PER_FRAME];  /* the 128 words of telem data stored in
                                  the s/c telemetered order  (see Spacecraft
                                  Operations Handbook), but note that c arrays
                                  start at zero and the handbook starts at one. */

/* Format of data array
   0  year
   1  day of year
   2  spacecraft ID
      time hh:mm:ss.msc
      frame counter (used to identify contents of subcomed frames)
      short wavelength range
      short wavelength value
      long wavelength range
      long wavelength flux
      subcommed data set one:
      fr = 13  XRS on/off, 0 is on
           18  second half ??  xrs preamp temperature
           20  xrpe slew/track, 1 is track
           21  slew north/south, 0 is north (note GOES 10 is flying
                   upside down, so directions are reversed from this)
           22  xr_s sun present, 1 is sun in fov
           23  xrs cal/data, 1 is data mode, 0 is calibration mode
           29  same as fr 13 (i.e. every 16 frames)
           04  same as fr 20 (i.e. every 16 frames)
           05  same as fr 21 (i.e. every 16 frames)
           06  same as fr 22 (i.e. every 16 frames)
           07  same as fr 23 (i.e. every 16 frames)
      subcommed data set two:
      fr = 18   XRS preamp temperature
           28   XRS reference voltage
           29   XRS bias voltage

  */


};  /* end of goesi_raw struct */

/*   function prototypes for necessary routines   */

int   goesi_frame_chk (struct goesi_raw , long, int);
int   goesi_frame_pr  (struct goesi_raw , long, long, int);
int   goesi_unpk (unsigned char *, struct goesi_raw *);
int   goesi_wild_times (long, struct goesi_raw *, struct counters *, int);
int   goesi_28s  (struct goesi_raw *, int, int);

/*   function prototypes for GoesI_sem_list programs   */

int   print_xrs  (FILE **, struct goesi_raw, int, int, long, int, int);
int   eclipse    (struct goesi_raw, float *, int, int, int);
int   files_raw  (char *, FILE **, FILE **, char *, char *);
int   goesi_subcom (struct goesi_raw fr, char *str_vals, char *str_hvals,
                    char *str_dvals, char *str_hdvals);
```

```
float goesi_vcurve (int telem, int curnum);
int   goesi_veclipse(struct goesi_raw fr, float *cur_cell);

int   goesi_hrminsec(struct goesi_raw fr, int *, int *, int *, char *);

int   goesi_sattde ( struct goesi_raw fr,
                  char str_engsattde1[16], char str_engsattde2[16],
                  char strh_engsattde1[16], char strh_engsattde2[16],
                  char str_tlmsattde1[16], char str_tlmsattde2[16],
                  char strh_tlmsattde1[16], char strh_tlmsattde2[16] );

/* end of #ifndef GOESI_READ_H */
#endif
```

```
/* * * *              goesi_subcom.h
 *                                                    Lorne Matheson
 *                                                  started 1 Sep 94
 *    Contains arrays of subcom data.  Each array consists of an array
 *    of subcom length and type.  In most analog subcoms, the data is
 *    converted into volts, etc.  Digital arrays are prefilled with -1,
 *    while the missing data are prefilled with BAD_DATA
 *
 *    The documentation perpetuates the old start with zero, start
 *    with one confusion.  The subcom typically numbered 1 to 32,
 *    with the corresponding frame counters being 0 to 31.  The
 *    numbers below are of the 1 to 32 variety, but the indexes in
 *    the arrays match the frame counters, or 0 to 31.
 *
 * * * *                                                          */

#define LEN_DSUB_36 16
#define FRAME36 frame16
/* * * *            configuration status digital submux
 *
 *   1  sounder
 *   2  sounder
 *   3  imager, sounder
 *   4  magnetometer, imager
 *   5  magnetometer, comm
 *   6  comm
 *   7  comm
 *   8  safe hold, sar, dcpr, cda
 *   9  dsn, imager, sounder
 *  10  dsn, mdl, ad, safe hold, spare
 *  11  autoload, safe hold, battery
 *  12  battery, sounder, imager
 *  13  imager, solar sail, solar array
 *  14  solar array, solar boom, spare, latch valves
 *  15  imager, sounder, earth sensor, latch valve
 *  16  sattde, imager, auto eclipse, safe hold
 *
 * * * *                                                          */
extern int gi_dsub36    [LEN_DSUB_36];


#define LEN_ASUB_63 32
#define FRAME63  frame
/* * * *           temperature data analog submux
 *
 *   1  eps dome temperature
 *   2  eps telescope temperature
 *   3  hepad temperature
 *   4  hepad electronics temperature
 *   5  eps/hepad digital processing unit temp
 *   6  magnetometer one sensor temperature
 *   7  magnetometer electronics temperature
 *   8  magnetometer two sensor temperature
 *   9  ad multiplexer temp
 *  10  ads sensor assy temp
 *  11  telemetry unit one temperature
 *  12  telemetry unit two temperature
 *  13  command unit one temperature
 *  14  command unit two temperature
 *  15  battery one temperature
 *  16  battery two temperature
 *  17  power control unit temperature
 *  18  spare type 'a'
 *  19  aoc thruster 6A fuel valve temperature
 *  20  aoc thruster 6A oxidizer valve temperature
 *  21  aoc thruster 6B oxidizer valve temperature
 *  22  aoc thruster 6B fuel valve temperature
 *  23  aoc thruster 7A oxidizer
 *  24  aoc thruster 7A fuel
 *  25  aoc thruster 7B oxidizer
 *  26  aoc thruster 7B fuel
 *  27  aoc thruster 2A oxidizer
 *  28  west panel temperature
 *  29  east panel temperature one
 *  30  magnetometer boom base temperature
 *  31  east panel temperature two
 *  32  apogee thruster flange temperature one
 *
 * * * *                                                          */
extern float gi_asub63  [LEN_ASUB_63];


#define LEN_ASUB_72  32
#define FRAME72 frame
/* * * *
 *   1  sounder patch control voltage
 *   2  spare
 *   3  sounder tlm +11v
 *   4  sounder tlm -8v
 *   5  sounder elec +17v
 *   6  sounder elec +8v
 *   7  sounder +10 reference voltage
 *   8  sounder elec -8v
 *   9  sounder elec -17v
 *  10  sounder servo +25v
 *  11  sounder servo -25v
 *  27  magnetometer one calibration reference voltage
 *  28  magnetometer two calibration reference voltage
 *  29  xrs reference voltage
 *  30  xrs -75 volt ion chamber bias voltage
 *  31  spare
 *  32  telemetry calibration voltage
 *
 * * * *                                                          */
extern float gi_asub72  [LEN_ASUB_72];

#define LEN_ASUB_88  32
#define FRAME88 frame
/* * * *
 *
 *   1  eps solid state bias 1 voltage
 *   2  eps solid state bias 2 voltage
 *   3  eps calibration reference voltage
 *   4  eps instant calibration voltage
 *   5  eps telescope electronics 8 volt reference voltage
 *   6  eps dome electronics 8 volt reference voltage
 *   7  hepad ssd bias voltage
 *   8  hepad electronics 8 volt reference voltage
 *   9  pressurant tank pressure
 *  10  fuel tank pressure
 *  11  oxidizer tank pressure
 *  12  cal word for extended range measurements
 *  13  fuel line pressure (extended range)
 *  14  oxidizer line pressure (extended range)
 *  15  fuel line pressure (extended range)
 *  16  oxidizer tank pressure (extended range)
 *  17  trim tab course position potentiometer 1
 *  18  trim tab fine position potentiometer 1a
 *  19  trim tab fine position potentiometer 1b
```

```
 *  20  sounder filter wheel motor temperature
 *  21  hepad pmt high voltage
 *  22  spare
 *  23  spare
 *  24  spare
 *  25  spare
 *  26  spare
 *  27  spare
 *  28  spare
 *  29  spare
 *  30  spare
 *  31  spare
 *  32  spare
 *
 * * * *                                                 */
extern float gi_asub88  [LEN_ASUB_88];


#define LEN_ASUB_100 32
#define FRAME100 frame
/* * * *
 *
 *   1  control bus current monitor (reserved)
 *   2  trim tab coarse positioner potentiometer 2
 *   3  trim tab fine positioner potentiometer 2a
 *   4  trim tab fine positioner potentiometer 2b
 *   5  xrs positioner coarse
 *   6  sun analog sensor (e/w)
 *   7  sun analog sensor (n/s)
 *   8  xri analog tlm (reserved)
 *   9  xri analog tlm (reserved)
 *  10  xri analog tlm (reserved)
 *
 * * * *                                                 */
extern float gi_asub100 [LEN_ASUB_100];


#define LEN_ASUB_101 16
#define FRAME101 frame16
/* * * *        Yoke Analog Temperature submux
 *
 *   1  sun analog sensor temperature
 *   2  xrs positioner bearing temperature
 *   3  xrs preamp temperature
 *   4  xrs positioner temperature
 *   5  xrs positioner drive electronics temperature
 *   6  spare type 'a'
 *   7  spare type 'a'
 *   8  spare type 'a' ESD
 *   9  spare typw 'a' ESD
 *  10  yoke hinge 1 temp (earth side)
 *  11  yoke hinge 2 temp (anti earth side)
 *  12  yoke hinge 3 temp
 *  13  spare type 'd'
 *  14  spare type 'd'
 *  15  trim tab motor temperature
 *  16  trim tab potentiometer temperature
 *
 * * * *                                                 */
extern float gi_asub101  [LEN_ASUB_101];

#ifndef NO_DATA
   #define NO_DATA -99999.
#endif
```

```
/*   *   *   *   *
/*                      goesiraw.h
 *      raw data record status bits for goes-8 to -12
 *                      status and time structure
 *                      expanded from char to int
 *      added suspect time documentation in raw_frame and g_status
 *                                                  1997 Apr 16  ldm
 *   *   */

struct raw_frame
{
    long dmsday;          /*  days since Jan 0, 1900   */
    long dmssec;          /*  seconds into the day */
    int  milli;           /*  milliseconds after the second */
    int  agc;             /*  from receiver, not yet available */
    int  port;            /*  1F (-6), 2F (-7) or 3F (other) */
    int  node;            /*  node number, 1 though 255 */
    int  sat;             /* */

    int  chek;          /* 0 is normal, otherwise its bits are:
                         *   0x01 wrong id
                         *   0x02 other satellite id
                         *   0x04 next synch words bad
                         *   0x08 unused, was text
                         *   0x10 check sum, single bit error, fixed
                         *   0x20 check sum, single bit error, not fixed
                         *   0x30 check sum, multiple errors, not fixed
                         *   0x40 dwell mode is in effect
                         *   0x80 timing suspect
                         */

    int raw_w[128];     /* the telemetry frame */
};
extern struct raw_frame g_raw;

struct g_status
{
    long stat1;
    long stat2;

/*              word1  0x01        1   No torquer current sign (AOCS)
 *                     0x02        2   Torquer current sign change
                       0x04        4   0.256 sec Torquer I Update
                       0x08       10   20 minute Torquer I change
                      0x010       20   Aocs anomaly
                      0x020       40   aocs, torquer, patch status change
                                           aocs on/off, cahge
                                           torquer relay changes
                                           dira on/off
                                           dira/dss select
                                           patch on/off
                      0x040      100   Single Bit error (corrected)
                      0x080      200   Telemetry Noise (more than 1 bit)
                     0x0100      400   Bad or Wrong Satellite Ident
                     0x0200     1000   goesiave restart
                     0x0400     2000   (unused)
                     0x0800     4000   Time Jump
                    0x01000    10000   Time Reverse
                    0x02000    20000   Frame Count Jump
                    0x04000    40000   (unused)
                    0x08000   100000   EPS/HEPAD Off
                   0x010000   200000   HEPAD Off
                   0x020000   400000   EPS/HEPAD Calibration On
                   0x040000  1000000   EPS Transient
                   0x080000  2000000   S4, S3 Ratio Bad
                 word2 0x01        1   X-ray Off
                       0x02        2   X-Ray Calibration
                       0x04        4   X-Ray Transient
                       0x08       10   X-Ray Long saturation
                      0x010       20   X-Ray Short saturation
                      0x020       40   Long Channel Range Change
                      0x040      100   Short Channel Range Change
                      0x080      200   XRS Elevation Angle Change
                     0x0100      400   XRS Pointing Error
                                           (SADA, CASS, yoke)
                     0x0200     1000   Eclipse
                     0x0400     2000   (unused)
                     0x0800     4000   Magnetometer 2 on
                    0x01000    10000   Magnetometers Off
                    0x02000    20000   Magnetometer Calibration
                    0x04000    40000   Magnetometer transient
                    0x08000   100000   Minor Mag Disturbance
                   0x010000   200000   (unused)
                   0x020000   400000   timing suspect
*                  0x040000  1000000   Dwell
*                  0x080000  2000000   (unused)
*   *   */

                /*   time of record status   */
    long  ioffset;              /*   dms day number last Dec 31   */
          /*   torquer correction to be subtracted from x, y, z   */
    float x_corr;
    float y_corr;
    float z_corr;
    int   year;                 /*   1994, etc   */
    int   rec_hour;
    int   rec_min;
    int   rec_sec;
    int   milli;                /*   milliseconds past minute   */
                /*   spacecraft, telemetry status   */
    int   satid;                /*   8, 9, 10, etc for printouts   */
    int   frame;                /*   main frme word, 0 to 31   */
    int   dwell;                        /*   >0 if dwell, else 0   */
    int   sub_inh;
    int   noise;            /*   1 if noise to be rejected, else 0   */
    int   fram_corr;        /*   1 if single bit corrected, else 0   */
    int   del_frame;        /*   number of missing frames, usually 0   */

                /*   aocs status   */
    int   aocs_rate;        /*   0 if 20 minutes, 1 if .256 seconds   */
    int   aocs_on;          /*   -1 unknown, 0 aocs1, 1 aocs2 on      */
                /*   -1 unknown, 0 no dira on, 1 dira 1 or 2 on   */
    int   dira_on;
    int   dss_dira;         /*  -1 unknown, 0 dss selected, 1 dira    */
    int   no_sign;          /*   1 if no torquer current signs avai   */
    int   up_date;          /*   > 0 if 20 min and current change   */
    int   tor_sign;         /*   > 0 if torquer current sign change   */
    int   tor_sign1;        /*   1 if positive, 0 if don't know
                             *   or off, -1 if negative, -2 don't know*/
    int   tor_sign2;        /*   1 if positive, 0 or -1   */

    /*   eps/hepad status   */
    int   eps_off;
    int   hepad_off;
    int   eps_cal;
    int   eps_sat;                  /*   eps counter saturation   */

    /*   mag status   */
    int   mag_off;
```

```
    int   mag_2_sel;
                        /*   mag on,  -1 don't know or off,     */
    int   mag_inst;     /*   0 mag 1 on, 1 mag 2 on             */
    int   mag_cal;      /*   > 0 treat as cal, 0 normal   */

    /*   xrs status   */
    int   xrs_off;              /*   > 0 if instrument off, else 0   */
    int   xrs_cal;       /*   > 0 if instrument calibrating, else 0   */
    int   xrs_sgain;    /*  > 0,  gain change in short channel   */
    int   xrs_lgain;    /*  > 0,  gain change in long channel   */
    int   xrs_point;    /*  > 0,  suspicious pointing   */
    int   x_sh_ra;      /*  current short range, usually 0 to 3   */
    int   x_lo_ra;      /*  current long range state              */

};

extern struct g_status g_stat;

#ifndef NO_DATA
    #define NO_DATA -99999.
#endif
```

```
/* ***** eclipse.c
 *
 * RETURN VALUE:
 *       function return is -1 for termination
 *                           0 for for normal
 *                           1 for eclipse
 *
 *       bad records passed in for timing, etc
 *       do not need end-of-run call
 *
 * ALGORITHM:
 *      This function only works for Goes-8 through Goes-M.  It looks at
 *       changes in the current generated by the solar cells to determine
 *       if an eclipse occurrs.  A simple algorithm is used that identifies
 *       the start of the eclipse when the solar cell current drops by 5
 *       per cent.  (3 per cent would probably also work)  This occurrs
 *       when 5 percent of the visible sun energy is obscured by the earth
 *       or the moon.  (Does not consider solar limb darkening.)
 *       The algorithm examines the difference between a short term
 *       smoothing (seconds) of the current with a long term smoothing
 *       (hours) of the current.  The per cent of total current defining
 *       eclipse is EL_FAC (set to 5 percent).
 *
 * ASSUMPTIONS:
 *       The current generated by the solar cells is divided between the
 *       solar panel current (sent over the yoke) and the control buss
 *       current (disapated by resistors on the solar array (to radiate
 *       away excess available powere.)  Other small
 *       (leakage, instrument) currents are ignored, but the results are
 *       probably accurate to a per cent or two.
 *
 *   Does not identify the times of first and last contact of the eclipse.
 *
 * INPUT:
 *    year
 *    doy    day of the year (1 indicates Jan 1).  Used for printouts.
 *
 * OUTPUT:
 *
 * RETURN VALUE:
 *    -1  if fewer than num_rec_(default is 60) records are printed as eclipse points
 *    otherwise, returns the the status of whether eclipse was found
 *
 * PROTOTYPE:
 *    int  eclipse(struct goesi_raw fr, float *cur_cell,
 *                 int year, int doy, int num_rec_prt)
 *
 * INFORMATION:
 *       If the XRS unit is automatically turned off during eclipse, by
 *       the detection of less than 5 amperes of solar array current, it
 *       is turned on 140 seconds after the solar array current reaches
 *       7 amperes (autoload group 6).
 *
 *   The XRS is typically turned on about 120 s (2 minutes) after the
 *
 * CALLS:
 *
 * CALLED BY:
 *    print_xrs.c
 *
 *  REFERENCE:
 *   For information about eclipses with the previous generation of GOES,
 *   see Bornmann, P. L. and Matheson, L. 1990, Astronomy and Astrophysics,
 *   231, 525-535, "Solar Flare Plasma Properties Derived from the
 *   Disk-Integrating GOES X-Ray Sensors during an Eclipse."
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *
 * FUTURE MODIFICATION:
 *   Make the call to goes_veclipse to get the eclipse voltages etc.
 *   Change min to minit, to avoid any confusion with a minimum function
 *
 * QUESTIONS:
 *   why/how are xrs long and short range bits (w8) shuffled? *       *         *
*/

#include <stdio.h>
#include "goesi_read.h"

/*   time interval (in seconds) of data printed  */
#define INT_TIME  150L

/*   ECL_FAC is the deviation factor from the smoothed current that defines
 *   an eclipse   */
#define ECL_FAC   .95
/* ECL_MAX is the number of points required to identify changes as an eclipse */
#define ECL_MAX   15

/* FAST_FAC and SLOW_FAC are arbitrary values used as timescales
 * to estimate whether an eclipse is occurring.  The fast factor
 * identifies events that change by 25% between the 0.5 sec data,
 * which is roughly a 2 minute time constant.  The fast factor is
 * used to filter out noise in the solar panel current.
 * The slow factor has a timescale of hours and is used to discard
 * current variations due to the diurnal and seasonal changes of the
 * solar angle (solar declination) and radiative heating from
 * sunlight reflected off the Earth.
 * FAST_FAC of 0.25 gives 1/4 weighting to the new point.
 * SLOW_FAC of 0.00015 gives 1/6666 weighting to the new point
 */
#define FAST_FAC  .25
#define SLOW_FAC  .00015

/* PRT_LEV: 0 is least print, 2 is most   */
#define PRT_LEV   0
int  eclipse(struct goesi_raw fr, float *cur_cell,
             int year, int doy, int num_rec_prt)
{
    static long actual_prt = 0L;
    static int  l_ecl    =  0;   /* Last (previous) eclipse state */
    static int  num_ecl  =  0;
    static int  sat_id   = -1;
    static int  sat_pid  = -1;
    static int  sat_idx  = -1;   /*  if >= 0, index into *_co arrays   */
    static int  start    = -1;
    static int  teclipse = 0;
    static int  Prt_Lev = PRT_LEV;

    int mf_ct   = -1;        /*   main frame counter, 0 to 31   */
    int fr_chek = -1;
    static long int_end = 0L;

    static long int_start = -8000L;
    /* Last (previous) date and time values */
    static long l_date = -1L;
    static long l_time = -1L;
    int hr = -1, min = -1, sec = -1;
```

```
/*   number of solar cell current measurements processed   */
static long n_cur = -1;

/*  number of current measurements for the fast and slow averaging
 *  n_$_start defines how many points are simply averaged instead
 *   of exponential decay to make startup more stable  */
static long n_f_start = -1;
static long n_s_start = -1;

static long nrec_tot = 0L;
static long nrec_int = 0L;
static long statwd1  = -1;
static long statwd2  = -1;


/*   Coefficients for spacecraft main currents for five GOES satellites,
 *   Goes-8 through Goes-N.  GOES-N filled in with GOES-8 values
 *   until available */
/*   coefficients for curve 131  primary bus current   word 74   */
static const float pbc_co[5][2] = {{2.754870e-01, 1.020744e-01},
    {1.76586e-01,  1.02990e-01},    {-7.15659e-02, 1.02369e-01},
    {-2.74036e-01, 1.02801e-01},    {2.754870e-01, 1.020744e-01}};
/*   coefficients for curve 133  solar arrray current  word 75   */
static const float sac_co[5][2] = {{2.448892e-01, 1.003481e-01},
    {-2.42757e-01, 1.01458e-01},    {9.831199e-02, 1.00072e-01},
    {-2.01376e-01, 1.00436e-01},  {2.448892e-01, 1.003481e-01}};
/*   coefficients for curve 132  control bus current   word 76   */
static const float cbc_co[5][2] = {{3.907000e-01, 7.926784e-02},
    {3.19468e-01,  7.91152e-02},    {1.94570e-01,  8.07924e-02},
    {1.94570e-01,  8.07924e-02},    {3.907000e-01, 7.926784e-02}};

/* currents for the solar array, control bus, and primary bus */
float arr_cu = 0.0,
      con_bu = 0.0,
      pri_bu= 0.0;

static float cur_slow     = -10.,
             cur_fast     = -10.,
             cur_sol_cell = -0.;
float cur_slow_t= 0.0;

float fra_ecl= 0.0;    /* NOT USED */
int ecl = -1;
int i = -1;
int fr_ch = -1;        /*   main frame counter, 0 to 31   */

int thr = -1, tmin = -1, tsec = -1;
int t_satid = -1;
int ret = 0;      /* The value to be returned by this routine.  */

/* This is an array that matches the unique bit patters that
 * identify the spacecrafts.  The unusual sequence is due to
 * the Hamming distance (or equivalent) used to avoid single-bit
 * errors causing misidentifications */
/* You had a question in here on why were the xrs ranges put through a table.
   Some programmers start at 0, some start at 1.  Some engineers shift right
   and some shift left.   The range bits in the telemetry have the least
   significant bit in the most significant bit.  The table is my way of
   changing the bit order of a two bit range word. */

char sat_str[16][5] = {{"UNK"}, {"UNK"}, {" GL"}, {"UNK"},
     {"UNK"}, {"UNK"}, {" G9"}, {"UNK"},          {"UNK"}, {" G8"},
     {"G10"}, {"UNK"},          {"UNK"}, {" GM"}, {"UNK"}, {"UNK"}};

int  sat_indx[16] = {-1, -1,  3, -1,   -1, -1,  1, -1,
                     -1,  0,  2, -1,   -1,  4, -1, -1};

    /*   *   *   *
     *         documentation numbers bits from 1 to 8, 1 being the most
     *             significant bit, 8 the least significant bit
     *   wd 2   subframe id 0-31
     *   wd 3   if bit 8 == 1, dwell, ignore
     *          bits 1-4 are satellite ident
     *   wd 5   xrs long, short range, bits shuffled
     *     6    eps (Energetic Particle Sensor) counter/cal
     *    36    sub 4(16), bit 8   magnetometer 1 = on
     *          sub 5,     bit 1   magnetometer 2 = on
     *    37    sub 2, bit 3,  xrs positioner bearing heaters status
     *          off = 0, enabled = 1 (thermostatically controlled )
     *          sub 2, bit 4,  xrs heaters status
     *          off = 0, enabled = 1 (thermostatically controlled )
     *    45    eps/hepad cal
     *    52    bit 8, sub  5 (16)  xrpe slew = 0, track = 1     *                    6
(16)  xrpe slew north = 0, south = 1
     *                      7 (16)  xrs sun not present = 0, present = 1
     *                      8 (16)  xrs cal = 0, data = 1
     *                 sub 14 (16)  xrs on = 0, off = 1
     *    53    2 mag cal bits
     *    71    battery 1 charge current curve 136
     *    73    primary s/c bus voltage  curve 130
     *    74    primary s/c bus current  curve 131
     *    75    solar array current      curve 133
     *    76    control bus current      curve 132
     *    79    battery 2 charge current curve 140
     *  Note: 75 conv + 76 conv is total solar cell current
     *        (16) indicates values are repeated twice per 32-value
     *        sequence (i.e. every 16 subframes
     *   *   *   */

    /*   initialize  */
    if (start != 0)
    { /* First call to routine, start = -1 */
       n_cur = 0L;  /* Initialize number of points processed so far */
       /* The weightings for how much the slowly and fast varying averages
          are adjusted for each new measurement */
       n_s_start = (long) 1.0/SLOW_FAC;  /* the number of points needed for a slow
change */
       n_f_start = (long) 1.0/FAST_FAC;  /* the number of points needed for a fast
change */
#if PRT_LEV > 0
          /*  ecl refers to the module eclipse.c   */
          (void) printf ("  ecl start, n_s_start %6ld, n_f_start %6ld\n",
             n_s_start, n_f_start);
#endif
       start = 0;
    } /* end of initializations for first call to this routine */

    ecl = l_ecl;  /* set current eclipse status to the previous (last) value */
    ret = -1;     /* Change from initialized value of 0 */

    /*   decode seconds into hrs, min, sec   */
    hr  = (int) fr.time/3600L;
    sec = (int) fr.time - 3600L * (long) hr;
    min = sec/60;
    sec = sec - 60*min;
    fr_chek = fr.chek & 0x6f;  /* checks for all unfixed errors (allows 0x10) */
    mf_ct = fr.data[2] >> 3;    /* Bit shift */

    /*  are we in new time interval  */
    if ((fr.date != l_date) || (fr.time >= int_end))
```

```
    {
        if (fr.date != l_date)
        {  /* New date */
            if (Prt_Lev > 0)
            {
                    (void) printf ("  ecl new date is %5ld, %d %3.3d\n",
                        fr.date, year, doy);
            }
            l_date = fr.date;
            int_start = fr.time/INT_TIME;
        }
        if (nrec_tot + nrec_int > 0)
        {
            thr = (int) int_start/3600L;
            tsec = (int) int_start - 3600L * (long) thr;
            tmin = tsec/60;
            tsec = tsec - 60 * tmin;
            (void) printf ("  old time interval    %2.2d:%2.2d:%2.2d  "
               " %7ld %7ld, %5ld %5ld\n", thr, tmin, tsec, statwd1,
               statwd2, int_end, nrec_int);

            /* Loop stops after one day or when the break occurs */
            for (i=0; i<3000;i++)  /* stops when int_end > fr.time */
            {
                int_start = int_end;
                int_end = int_start + INT_TIME;
/* This deemed illegal break */
                if (int_end > fr.time)
                    break;  /* go to increment of actual_prt. */
                tsec = (int) int_start % 3600L;
                tmin = tsec / 60;
                tsec = tsec - 60 * tmin;
#if PRT_LEV > 0
                    (void) printf ("  empty time interval  %2.2d:%2.2d\n",
                        tmin, tsec);
#endif
            } /* End of 3000 loop */
            actual_prt++;  /* this only increments if date changes or
                              fr.time exceeds int_time */
        } /* end of if ((fr.date != l_date) || (fr.time >= int_end)) */
        statwd1 = 0L;
        statwd2 = 0L;
        int_end  = int_start + INT_TIME;
        nrec_tot = nrec_tot + nrec_int;
        nrec_int = 0L;
    } /* end of if fr_date */

    /* Check the satelite ID */
    t_satid = fr.data[3] >> 4;  /* bit shift*/
    if ((sat_id != t_satid) && (fr_chek == 0))
    {
#if PRT_LEV > 0
        (void) printf ("   ecl  %2.2d:%2.2d:%2.2d.%3.3d    "
            "satellite id changes from %2d to %2d  %s, sat_indx is %ld\n",
            hr, min, sec, fr.milli, sat_id, t_satid, sat_str[t_satid],
            sat_indx[t_satid]);
#endif
        if (sat_id < 0)
        {  /* had a bad sat_id, but frame gives a new value */
            sat_id = t_satid;
            sat_pid = t_satid;
            sat_idx = sat_indx[sat_pid];
        }
        else
        {
            sat_id = t_satid;
            if (t_satid == sat_pid)
            {
                sat_idx = sat_indx[sat_pid];
            }
            else
            {  /* sat_id != t_satid */
                sat_idx = -1;
            }
        }
    }
}  /* end of if sat_id */

/*   calculate currents and test for eclipse. */
if ((sat_idx >= 0) && (fr_chek == 0))
{   /*   good data in this frame   */
    /*   use satellite dependant primary bus current coefficients   */
    pri_bu = pbc_co[sat_idx][0] + ((float) fr.data[74]) *
        pbc_co[sat_idx][1];
    /* x  use satellite dependant solar array current coefficients   */
    arr_cu = sac_co[sat_idx][0] + ((float) fr.data[75]) *
        sac_co[sat_idx][1];
    /*   use satellite dependant control buss current coefficients   */
    con_bu = cbc_co[sat_idx][0] + ((float) fr.data[76]) *
        cbc_co[sat_idx][1];   /* Uses Control Bus Currrent coefficients */
    cur_sol_cell = arr_cu + con_bu;  /* Solar Cell current is sum of
                                        * solar array current and s/c bus current */
    *cur_cell = cur_sol_cell;

    /* Calculate long and short duration averages of the solar panel current */
    /* Either update the time-scale weighted average if full number of
     * points available, or else update the average based on the number
     * of available points and their deviations from the value to be added */
    n_cur++;  /* increment count of the total number of current measurements
                 that are available for averaging */
    if (n_cur > n_f_start)  /* n_f_start is 1/fast_fac */
    {   /* This adjusts the current estimate by a factor of 1/n_fast_start
           times the change in measured current.  n_f_start = 4 for
           FAST_FAC of 0.25. */
        cur_fast = cur_fast + FAST_FAC * (cur_sol_cell - cur_fast);
    }
    else
    {  /* This adjusts the current estimate by a larger factor than if FAST_FAC
              were used, because n_cur > N */
        cur_fast = cur_fast +(cur_sol_cell - cur_fast)/((float) n_cur);
    }
    if (n_cur > n_s_start)  /* n_s_start is 1/slow_fac */
    {  /* This adjusts the current estimate by a factor of 1/n_slow_start
           times the change in measured current.  n_f_start = 6666 for
           FAST_FAC of 0.00015. */
        cur_slow_t = cur_slow + SLOW_FAC * (cur_sol_cell - cur_slow);
    }
    else
    { /* This adjusts the current estimate by a larger factor than if SLOW_FAC
              were used, because n_cur > N */

        cur_slow_t = cur_slow + (cur_sol_cell - cur_slow)/
            ((float) n_cur);
    }

   /*   Nominal average solar cell current is 29 or 30 amperes.
    *   If fast current (approx 2-min average) is < .95 (ECL_FAC)
    *   slow current (which averages over a time longer than the
    *   max 92 min eclipse) or if fast current
    *   is less than 5 amperes, the satellite is in eclipse. */
    if ((ECL_FAC * cur_slow_t > cur_fast) || (cur_fast < 5.0))
```

```
        {
            /*  in eclipse   */
            if (n_cur >= 20)
            {  /* because eclipse is in progress, add back 20% of the contribution
                  that dropped out */
               cur_slow_t = cur_slow + .2*SLOW_FAC*(cur_sol_cell - cur_slow);
            }
            num_ecl++;    /* increment static counter for total number of eclipse points
*/
            if (ECL_MAX > num_ecl)
                teclipse = num_ecl + 1;  /* count number of points in potential
                                            eclipse event.  Not yet exceed the limit
                                            required to call it an eclipse. */
            else
                teclipse = ECL_MAX;      /* Exceeded the number of points required for an
                                            eclipse.  So just set to preset maximum */
        }
        else
        {   /* not in eclipse */
            if (cur_fast > cur_slow_t)
            {
                /*  above average, so not an eclipse.  */
                num_ecl = 0;
            }
            else
            {
                /*  slightly below average, but above eclipse   */
                if (num_ecl > 0)
                    num_ecl--;  /* undo the increment to the eclipse counter */
            }
        }
        cur_slow = cur_slow_t;
        if (n_cur < 0L)
        {
            (void) printf ("  ecl cur %2.2d:%2.2d %2.2d "
                "%7.3f %7.3f %7.3f, %7.3f, fast, slow %7.3f %7.3f\n",
                min, sec, mf_ct, pri_bu, arr_cu, con_bu, cur_sol_cell,
                cur_fast, cur_slow);
        }
        fra_ecl = cur_fast/cur_slow;  /* NOT USED */
        if (teclipse > 0)
        {
            teclipse--;
            ecl = 1;
        }
        else
        { /* total time of event less than eclipse duration */
            ecl = 0;
        }
        /*   end of current, eclipse computations  */

        if (l_ecl != ecl)

        {   /* Change in the eclipse status */
            (void) printf ("              %2.2d:%2.2d:%2.2d.%3.3d %2d "
                "eclipse from %2d to %2d\n",
                hr, min, sec, fr.milli, mf_ct, l_ecl, ecl);
            l_ecl = ecl;  /* Save this eclipse status */
        }
    }  /* End of eclipse search, which started at the  if ((sat_idx >= 0) && (fr_chek ==
0)) */


    /*   are we through printing   */
    l_time = fr.time;
    ret = l_ecl;  /* Return the eclipse status */
    if (actual_prt > num_rec_prt)
    { /* Change return status if not encounter enough times when actual_prt (date
         changed or fr.time exceeded int_time) */
        ret = -1;
    }
#if PRT_LEV > 0
    if (n_cur < 5)
    {
        (void) printf("  eclipse return is %2d, %6.3f\n", ret, *cur_cell);

    }
#endif
    return ret;
}  /* end of eclipse */
```

```
/* ***** files_raw.c
 *
 * PURPOSE:
 *     Opens the files of raw telemetry.  Opens next file if
 *     user-supplied file contains a list of files to be processed.
 *
 * INPUT:
 *    file_name      character string specifying either the full path name
 *                   of a data file or a file containing a list of
 *                   data file names
 *    Note:  if file_name starts with a capital F, it is assumed
 *           to be a file contiaining a list of file names
 *
 * OUTPUT:
 *    f_descr        a file descripter pointing to the raw data file
 *    f_output       a file descripter pointing to an opened file
 *                   that will be used to store the processed results
 *    binput    input filename
 *    fname_output   output filename
 *
 * RETURN VALUE:
 *     return is 0 for data file available and opened
 *               1 for no more files available
 *              -1 for inability to open a file
 *
 *
 * PROTOTYPE:
 *   int files_raw (char *file_name, FILE **f_descr, FILE **f_output,
 *                  char *fname_input, char *fname_output)
 *
 * COMPILER OPTIONS:
 *   PRT_LEV adjusts amount of output to stdout
 *
 * CALLS:
 *
 * CALLED BY:
 *   main (goesI_xrs_list)
 *
 * INPUTS:
 *    filename  name of file to be processed, or name of file containing
 *              a list of file names to be processed.  Filename must start
 *              with F for a listing file. (see files_raw.c)
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modified 10/18/99 by P.L. Bornmann
 *     added output that can be used to as an output file
 *     that will be used by print_xrs.
 *   Modified 10/19/99 by P.L. Bornmann
 *     added preprocessor conditional for forward/reverse slashes.
 *     for MS windows, set preprocessor definition WIN32
 *   Modified 11/1/99 by P.L. Bornmann
 *     to return the names of the input and output file names
 *
 * FUTURE MODS:
 *    check whether capital F is valid test for file type under Windows
 *    *    */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "goesi_read.h"

int files_raw (char *file_name, FILE **f_descr, FILE **f_output,
               char *fname_input, char *fname_output)
  {

#define LEN_BUF 120
#define PRT_LEV   1       /*   0 least, 2 most printout   */

    static FILE *list_file;
    int len = -1;

    int ret  = -1;
    char *p = 0 ;              /* pointer to start character of filename in path*/
    char buff[LEN_BUF] = "";  /* the name of the next file to process */

    static int many_f = 0;    /*   many_f = 0 is only file name,
                               *          = 1 is data file list   */
    static int start  = 0;

    /*   first time in, check to see if name is data or file list   */
#if PRT_LEV >= 1
    (void) printf ("files_raw.c: start of file %d (many_files? %d)\n",
        start, many_f);
#endif

    if (start == 0)
      {

       /* ***************
        ** FIRST PASS **
        ****************/

        start = 1;
        /*   does file part of full path start with F?   */
        len = strlen (file_name);

       /* Find position of start of filename within full path name */
       /* Preprocessor determines if operating system uses forward or reverse slash */
#ifdef WIN32
        /* MS Windows uses backward slash for path delimiter */
        p = strrchr (file_name, '\\');
#else
        p = strrchr (file_name, '/');
#endif

        /* Examine first character of file name, if F the file is
         * a list of filenames */
        if (p != NULL)
          {  p++;
#if PRT_LEV >= 2
            (void) printf ("files_raw.c: after last occurance of /  %s\n", p);
#endif
          }
        else  /* p is null, no slashes in file name */
          { p = file_name;  /* pointer to start of file's name is the start of file_name
*/
          }
        if (p[0] != 'F')    /* will Windows handle this properly? */
          { /*   name is name of data file, not name list   */
            /* Open this data file */
            if ((*f_descr = fopen(file_name, "rb")) == NULL)  /* will read binary file */
              { (void) printf("files_raw.c: cannot open %s, QUIT\n", file_name);
                return -1;
              }
            /* Will pass back the filename for the opened input data file */
            (void) strcpy (fname_input, file_name);
```

```
#if PRT_LEV >= 1
            (void) printf("files_raw.c: opened %s \n"
                          "              equals %s \nf"
                          "              for raw data reads\n", file_name, fname_input);
#endif

            /* Open file for the output data */
            fname_output = strcat(file_name, ".dat");
            if ((*f_output = fopen(fname_output, "w")) == NULL)  /* will write ascii file
*/
              { (void) printf("files_raw.c: cannot open output file %s, QUIT\n",
file_name);
                return -1;
              }
#if PRT_LEV >= 1
            (void) printf("files_raw.c: opened %s for output data\n", fname_output);
#endif

            return 0;
          }
        else
          { /* p[0] == 'F' indicating file contains list of files to process */
            /* file is name of list of data files   */
            if((list_file = fopen(file_name, "r")) == NULL)
              { /*  cannot open list of data file names   */
                (void) printf ("files_raw.c: cannot open file list %s\n", file_name);
                return -1;
              }
            else
              {  /* Multiple files */
#if PRT_LEV >= 1
                (void) printf ("files_raw.c: file %s is list of data files\n",
file_name);
#endif
                many_f = 1;
              }
          }
      }  /* end of start == 0 */

    else  /* start != 0 */

        /* ******************
         ** REPEAT ENTRIES **
         ******************/
      { /* start is not equal 0, not first time through   */
        if (*f_descr == NULL)
          {  (void) printf ("files_raw.c: data file descripter is NULL, quit\n");
            return -1;
          }
        else
          { /* Close the currently open file, should have been read by now */
            ret = fclose(*f_descr);
            if (ret != 0)
              {  (void) printf ("files_raw.c: cannot close data file successfully\n");
                return -1;
              }
            /* Close the output file */
            ret = fclose(*f_output);
            if (ret != 0)
              { (void) printf ("files_raw.c: cannot close output file successfully\n");
                return -1;
              }
          }
      } /* end of f_descr if */

      /* ******************
       ** DETERMINE FILE **
       ******************/

      if (many_f == 0)
        {(void) printf ("files_raw.c: have called after single file open, QUIT\n");
          return 1;
        }
      else
        {
          /*   try to read another data file name   */
          p = fgets (buff, LEN_BUF, list_file);
          if (p == NULL)
            {  (void) printf ("files_raw.c: cannot read another file label, quit\n");
              return -1;
            }
          ret = strlen(buff);
#if PRT_LEV >= 2
          (void) printf ("files_raw.c: length of nth file name is %d\n", ret);
#endif
          if ((ret > 0) && (buff[ret-1] == '\n'))
            buff[ret-1] = '\0';
          /*   could this be an empty line, quit   */
          if (ret < 3)
            {  return -1;
            }

          /* Open the next data file */
          if ((*f_descr = fopen(buff, "rb")) == NULL)
            { (void) printf("files_raw.c: cannot open %s as raw data file, QUIT\n", buff);
              return -1;
            }
          else
            {
              (void) strcpy (fname_input, buff);

              /* Will pass back the filename for the opened input data file */
              (void) strcpy (fname_input, file_name);
              fname_output = strcat(file_name, ".dat");
              if ((*f_output = fopen(fname_output, "w")) == NULL)  /* will write ascii file
*/
                { (void) printf("files_raw.c: Cannot open output file %s, QUIT\n",
file_name);
                  return -1;
                }
              /* Summarize  */
              (void) printf("files_raw.c: opened %s for raw data reads\n", buff);
              (void) printf("files_raw.c: opened %s for input data\n", fname_input);
              (void) printf("files_raw.c: outfile %s\n", fname_output);
            }
        }  /* end of else if many_f == 0 */

      /* Summary */
      (void) strcpy (fname_input, file_name);
      (void) printf ("files_raw.c: input  file %s\n", fname_input);
      (void) printf ("files_raw.c: output file %s\n", fname_output);
      return 0;

    } /* end of files_raw */
```

```
/* ***** xrsi_conv.c
 *
 * PURPOSE:
 *       decode most of the xrs related spacecraft words and status
 *             push data on to averaging
 *
 *  REFERENCES:
 *       Telemetry Reference is Goes IJK/LM DRL 0-14 Telemetry and
 *       Command lists
 *       Instrument and spacecraft reference is Goes Program,
 *       Spacecraft Operations Handbook, Volume III, Spacecraft
 *       Description, DRL 503-02, Goes I-M
 *
 *       Telemetry conversion coefficients are in Space Operations
 *       Handbook, Volume VI, Spacecraft Data, DRL 503-02, Goes I and
 *       Goes-I Data and Calibration Report for Space Environment
 *       Monitor, DRL 311-01 as well as the Panametrics Calibration
 *       Report
 *
 *  MODIFICATION HISTORY
 *          started 5 Aug 94, ldm
 *          95 Mar 11  eclipse suppression, housekeeping  ldm
 *             May 16  added Goes-9 xrs coefficients
 *             Dec 20  corrected error in g9 using g8 kx,
 *                     updated kx to new coefficients
 *          96 Jan  3  sun lock log entry, new FAC entries
 *                  8  new FAC for long and short
 *          96 Jul 18  fixed logging error in range changes
 *          96 aug 20  $ $ PATCH to suppress xrs on Goes-9 during
 *             motor saving maneuvers, sas checking
 *          97 Feb 24  Goes-10 added, no xrs_pos
 *          98 Oct  6  redo startup of xrs, make longer
 *          1999-02-12 LDL: add Goes 11, cloned from Goes 8
 *          11/24/99 P.L. Bornmann adding explanatory comments
 *
 * QUESTIONS:
 *  Is g_stat a global variable?  Not defined prior to this.
 *  Are the temperature coefficients constant for all s/c, as
 *             assumed here.  Probably are if used same thermistors
 * USAGE:
 *   Must set PREPROCESSOR VALUE FOR GSAT during compile
 *
 * FUTURE MODIFCATIONS:
 *   PLB may want to make the conversion coefficients into
 *   arrays that include the spacecraft, rather than create different
 *   code for each s/c
 *   Case statment might be more efficient because it has breaks
 *   Looks like g_stat.x_sh_ra is previous short-wavelength range
 *              g_stat.xrs_sgain counter for RC transient
 * CALLS:
 *   calc_xrs_coef
 *   writei_log     This simply writes (%s) the string to a DMS log file
 *   gi_ave-dateum  This is C++ object code LDL wrote to calculate the
 *                  average
 *  gi_ave_datum is Dave Lewis's averaging and messageing to the dms machines,
 *  written in C++.  It calculates averages of various intervals and types.  It
 *  ignores bad data.  The one minute 'averages' are actually a linear fit to
 *  the data over the minute, with the linear fit evaluated at 30 seconds.  This
 *  way the long and short averages apply to the same time for ratios.
 *
 * ALGORITHM:
 * Looks like also set to treat change to calibration as RC transient
 *
 *  *    */

                                /*   include system .h files   */
#include <stdio.h>
                                /*   include goesiave .h files   */
#include "Goesi_channels.h"
#include "goesiraw.h"
#include "goesi_proto.h"
#include "goesi_subcom.h"
#ifdef LDM
   #include "string.h"
#endif

#define FAC_T     0.05
/* value used to remove RC transient after range change (same value for
 * short and long channels */
#define XRS_SW    8
#define XRS_SW_ON 13

/* Factors used to make fluxes agree with those from the spinning GOES */
#if   GSAT == 8
    #define FAC_LONG  0.70
    #define FAC_SHORT 0.85
#elif GSAT == 9
    #define FAC_LONG  0.70
    #define FAC_SHORT 0.85
#elif GSAT == 10
    #define FAC_LONG  0.70
    #define FAC_SHORT 0.85
#elif   GSAT == 11
    #define FAC_LONG  0.70
    #define FAC_SHORT 0.85
#else
    #error GSAT not defined properly in xrsi_conv
#endif

                    /*   word  56   short xrs flux
                     *   word  57   long  xrs flux
                     *              range bits are lsb first
                     *   word   5, bits 1-2   Short xrs range
                     *   word   5, bits 3-4   Long xrs range
                     *   word  36  11(16) bit 6, Group 6 autoload
                     *                    control status
                     *         36  16     bit 1 sattde 2 slew direction
                     *                    select status
                     *                      bit 2 sattde 2 run/slew
                     *                    select status
                     *                      bit 3 sattde 2 sada/trim tab
                     *                    select status
                     *   word  59   SADA output channel A
                     *   word  60   SADA output channel B
                     *   word  51, bits 1,2   SADA 1,2 on,off status
                     *   word  51, bit 7      SADA/trim tab select
                     *   word  52,  5(16) bit 8     xrpe track/slew
                     *                    0 is slew,  1 is track
                     *   word  52,  6(16) bit 8     xrpe slew dir
                     *                    0 is north, 1 is south
                     *   word  52,  7(16) bit 8     sun lock
                     *                    0 sun present  1 absent
                     *                    0 during eclipse and cal
                     *   word  52,  8(16) bit 8     xrs cal/data
                     *                    0 is cal, 1 is data
                     *   word  52, 14(16) bit 8     xrs on/off
                     *                    0 is on, 1 is off
                     *   word  72, 29(32)  xrs reference voltage
                     *         72, 30      xrs -75 volt ion chamber bias
```

```
                    *   word  75,  solar array current
                    *   word  76   control bus current
                    *        sum of word 75, 76 currents is total
                    *        current produced by solar cells
                    *   word 100,  5(32)  xrs posit coarse
                    *        100,  6      analog sun (n/s) was e/w
                    *        100,  7      analog sun (e/w)
                    *        101,  1(16)  sun analog sensor temp
                    *        101,  2      xrs posit bearing temp
                    *        101,  3      xrs preamp temp
                    *        101,  4      xrs posit temp
                    *        101,  5      xrs posit drive elec temp
                    *                                              */

/* prototype needed */
void calc_xrs_coef(float temp, int ran_s, int ran_l);

static char line[90];

static float xrs_coef[2][2];      /*   first index 0-short, 1-long
                                   *   second      0-volt offset
                                   *               1-sensitivity
                                   */

/* *****  xrsi_conv
```

```
************************************/
int xrsi_conv(void)
{
    static int la_xrs_cal = -1;   /*   holds the last cal/data bit
                                   *   0 is calibrate, 1 is data/off  */
    static int la_xrs_on =-1;     /*   holds the last on/off bit
                                   *   0 is on, 1 is off   */
    static int la_xrs_sun =-1;    /*   0 sun present, 1 sun absent
                                   *   ?  0 during eclipse and cal  */
    static int la_xrp_slew= -1;   /*   0 north, 1 south   */
    static int la_xrp_track =-1;  /*   0 is slew, 1 is track   */

    static int start = 1;         /*   first time flag   */
    static float min_short;
    static float min_long;
    static float sens_temp = 4.;
    static float temp_in;
    static float sas_ew = -5.;    /*   east-west sun analog sensor  */
    static float sas_ns = -5.;    /*   north-south sas              */

    /*   convert two-bit fields to xrs ranges   */
    const static int xrs_ga[4] = {0,2,1,3};
    const static float
                kss  = { 2.238e-9},     /* quantization level of lowest short range */
                ksl  = { 9.314e-9};     /* quantization level of lowest long  range */

    /*   temperature coefficients and conversion will become external
     *   to this function   */
    const static float temp_coef[6] ={-52.51458, 1.510513, -0.01771354,
             1.287143e-04, -4.557991e-07, 6.370513e-10};

/*  const float short_f[4] = {0.894, 0.898, 0.940, 0.940};
    const float long_f[4] = {0.766, 0.808, 0.821, 0.830};
*/
#ifdef LDM
    /*   five coded strings, xrs_pos, sas_ns, sas_ew,
        xrs_ref, xrs_ion   */
    static char xrp[5][10] = {"      ", "       ", "      ", "       ",
        "       "};
#endif
    float longx, shortx;
    float tdat;            /* variable to hold data temporarily */
    float xrs_ion;         /*   xrs ion chamber voltage   */
    float xrs_pos, xrs_ref;


    int bits;             /* variable to hold a bit values */
    int sh_ra;             /*   short channel range   */
    int lo_ra;             /*   long channel range    */
    int frame16;           /*   frame number mod 16   */

    int ra_ch = 0;         /*   set to 1 if a range change   */
    char range_ch[2][30] = {"","","};



    if (start)
     {  /*   minimum flux is .4 of the nominal quantization interval of
         *   the most sensitive range
         */
        min_short = .4 * kss;
        min_long  = .4 * ksl;
        start = 0;
     }

    if (g_stat.noise)
     {
        gi_ave_datum (CH_XRS_LONG, NO_DATA);
        gi_ave_datum (CH_XRS_SHORT, NO_DATA);
        return 0;
     }

    if (g_stat.dwell)
        return 0;

    /*   count down history counters   */
    if (g_stat.xrs_sgain > 0)
        g_stat.xrs_sgain--;
    if (g_stat.xrs_lgain > 0)
        g_stat.xrs_lgain--;
    if (g_stat.xrs_cal > 0)
        g_stat.xrs_cal--;
    if (g_stat.xrs_off > 0)
        g_stat.xrs_off--;

    /* Frame number modulo 16, means ignore bits above 16 */
    frame16 = g_stat.frame & 0x0f;
    if (frame16 == 2)
     {   /*   get sensor temperature and smooth   */
        tdat = g_raw.raw_w[101];
        temp_in =temp_coef[0]+tdat*(temp_coef[1]+tdat*(temp_coef[2]+tdat*
            (temp_coef[3]+tdat*(temp_coef[4]+tdat*temp_coef[5]))));
        gi_ave_datum (CH_XSENT, temp_in);
        sens_temp = sens_temp + FAC_T * (temp_in - sens_temp);
     }

    if (g_stat.frame == 4)
     {   /*   get xrs coarse position (n-s)   */
        tdat = g_raw.raw_w[100];

; NOTE: it is possible that the values for 9 and 10
;       should be the same as for 8 and 11; they may
;       have been truncated in the report


#if   GSAT == 8
        xrs_pos = 28.37270 - tdat*0.4816012;
#elif GSAT == 9
        xrs_pos = 28.00    - tdat*0.481601;
#elif GSAT == 10
        xrs_pos = 28.00    - tdat*0.481601;
#elif GSAT == 11
        xrs_pos = 28.37270 - tdat*0.4816012;
#endif
#ifdef LDM
        sprintf (xrp[0], "%7.2f", xrs_pos);
#endif
        gi_ave_datum (CH_XPELEV, xrs_pos);
     }

    if (g_stat.frame == 5)
     { /*   get xrs sun angle sensor (n-s)  */
        tdat = g_raw.raw_w[100];
        sas_ns = -1.99200 + tdat*0.0160000;
#ifdef LDM
        sprintf (xrp[1], "%7.3f", sas_ns);
#endif
        gi_ave_datum (CH_SASNS, sas_ns);
     }
```

```
    if (g_stat.frame == 6)
     {  /*   get xrs sun angle sensor (e-w)  */
        tdat = g_raw.raw_w[100];
        sas_ew = -1.99200 + tdat*0.0160000;
#ifdef LDM
        sprintf (xrp[2], "%7.3f", sas_ew);
#endif
        gi_ave_datum (CH_SASEW, sas_ew);
     }

    if (g_stat.frame == 28)
     {  /*   get xrs reference voltage  */
        tdat = g_raw.raw_w[72];
        xrs_ref = -0.02000 - tdat*0.04000;
#ifdef LDM
        sprintf (xrp[3], "%7.3f", xrs_ref);
#endif
        gi_ave_datum (CH_XREFV, xrs_ref);
     }

    if (g_stat.frame == 29)
     {  /*   get xrs ion chamber voltage  */
        tdat = g_raw.raw_w[72];
        xrs_ion = -0.2000 - tdat*0.4000;
#ifdef LDM
        sprintf (xrp[4], "%7.2f", xrs_ion);
        printf (" xrp  pos%s, ns%s, ew%s, xrs  ref%s, ion%s\n",
            xrp[0], xrp[1], xrp[2], xrp[3], xrp[4]);
        (void) strncpy ( xrp[0], "      " , 9);
        (void) strncpy ( xrp[1], "      " , 9);
        (void) strncpy ( xrp[2], "      " , 9);
#endif
/*        gi_ave_datum (CH_XREFV, xrs_ref);*/
     }


    /*   get short channel and see if it has changed   */
    bits = g_raw.raw_w[5] >> 6 ;
    sh_ra = xrs_ga[bits];
    if (sh_ra != g_stat.x_sh_ra)
     {  /* New short wavelength range encountered.  Is range valid? */
        if ((sh_ra != 0) || (g_stat.x_sh_ra != -1))
         {  /* Valid range change */
            sprintf (range_ch[0], ", short %d to %d",
                g_stat.x_sh_ra, sh_ra);
            ra_ch = 1;
         }  /* no else, ignore invalide ranges */
        /*   set history counter to ignore r c transient   */
        if (g_stat.xrs_sgain < XRS_SW)
            g_stat.xrs_sgain = XRS_SW;
     }

    /*  Determine whether long channel range changed   */
    bits = (g_raw.raw_w[5] >>4) & 3;
    lo_ra = xrs_ga[bits];
    if (lo_ra != g_stat.x_lo_ra)
     {  /* New long wavelength range encountered.  Is range valid? */
        if ((lo_ra != 0) || (g_stat.x_lo_ra != -1))
         {  /* Valid range change */
            sprintf (range_ch[1], ",  long %d to %d",
                g_stat.x_lo_ra, lo_ra);
            ra_ch = 1;
         }
        if (g_stat.xrs_lgain < XRS_SW)
            g_stat.xrs_lgain = XRS_SW;
     }
    /*   print range change(s), if present   */
    if (ra_ch == 1)
     {
        sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrs range change"
            "%s%s\n",
            g_stat.rec_hour, g_stat.rec_min, g_stat.rec_sec,
            g_stat.frame, range_ch[0], range_ch[1]);
#ifdef LDM
        printf ("%s", line);
#endif
        if ((la_xrs_cal == 1) && (la_xrs_on == 0))
         { writei_log (line);
         }
     }

    if ((lo_ra != g_stat.x_lo_ra) || (sh_ra != g_stat.x_sh_ra) ||
        (frame16 == 2))
     {  /* Get new flux conversion coefficients after range change */
        calc_xrs_coef(sens_temp, sh_ra, lo_ra);
     }

    /*   look at status bits for current xrpe track/slew status */
    bits = g_raw.raw_w[52] & 1;

    /*   examine xrpe slew/track, 0 is slew, 1 track   */
    if (frame16 == 4)
     {
        if (la_xrp_track != bits)
         { /* New track/slew status */
            if ((la_xrp_track != -2) || (bits == 0))
             {
                sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrp_track"
                    " changes to %d\n",
                    g_stat.rec_hour, g_stat.rec_min,
                    g_stat.rec_sec, g_stat.frame, bits);
                writei_log (line);
#ifdef LDM
                printf ("%s", line);
#endif
             }
            la_xrp_track = bits;
         }
     } /* end of slew/track analysis */

    /*   examine xrpe slew direction, 0 is north, 1 south   */
    if (frame16 == 5)
     {
        if (la_xrp_slew != bits)
         { /* new slew direction */
            if ((la_xrp_slew != -2) || (bits == 0))
             {
                sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrp_slew "
                    "changes to %d\n",
                    g_stat.rec_hour, g_stat.rec_min,
                    g_stat.rec_sec, g_stat.frame, bits);
                writei_log (line);
#ifdef LDM
                printf ("%s", line);
#endif
             }
            la_xrp_slew = bits;
         }
     }
```

```
    /*   examine sun lock bit
    *   bits = 0 is sun present
    *   ? ? 0 during eclipse, cal
    *   n - s sas only              */
    if (frame16 == 6)
     {
        if (la_xrs_sun != bits)
         {
            if ((la_xrs_sun != -1) || (bits == 0))
             {
                sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrs_sun "
                    "changes to %d\n", g_stat.rec_hour, g_stat.rec_min,
                    g_stat.rec_sec, g_stat.frame, bits);
                writei_log (line);
#ifdef LDM
                printf ("%s", line);
#endif
             }
            la_xrs_sun = bits;
         }
     }
    /*   examine xrs cal/data bit, zero is cal         */
    if (frame16 == 7)
     {
        if (la_xrs_cal != bits)
         {
            if ((la_xrs_cal != -1) || (bits == 0))
             {
                sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrs_cal"
                    " changes to %d\n", g_stat.rec_hour, g_stat.rec_min,
                    g_stat.rec_sec, g_stat.frame, bits);
                writei_log(line);
#ifdef LDM
                printf ("%s", line);
#endif
             }
            la_xrs_cal = bits;
         }
     }

    if ((la_xrs_cal == 0) && (g_stat.xrs_cal < XRS_SW))
        g_stat.xrs_cal = XRS_SW;

    /*   examine xrs on/off bit, 1 is off */
    if (frame16 == 13)
     {
        if (la_xrs_on != bits)
         {
            sprintf (line, " %2.2d:%2.2d:%2.2d %2.2d xrs on/off "
                "changes to %d\n", g_stat.rec_hour, g_stat.rec_min,
                g_stat.rec_sec, g_stat.frame, bits);
            if ((la_xrs_on != -1) || (bits == 1))
             {
                writei_log(line);
#ifdef LDM
                printf ("%s", line);
#endif
             }
            la_xrs_on = bits;
         }
     }
    if ((la_xrs_on == 1) && (g_stat.xrs_off < XRS_SW_ON))
        g_stat.xrs_off = XRS_SW_ON;

    g_stat.x_sh_ra = sh_ra;
    g_stat.x_lo_ra = lo_ra;
    if (g_stat.xrs_off > 0)
     {
        g_stat.stat2 = g_stat.stat2 | 1;
        return 0;
     }

    /* ****************************
     * Calculate the X-ray Fluxes *
     ****************************/
    /*   calculate short flux   */
    if ((g_raw.raw_w[56] == 0) || (g_raw.raw_w[56] == 255))
        g_stat.stat2 = g_stat.stat2 | 020;
    shortx = (float) g_raw.raw_w[56] * .02;
    shortx = (shortx - xrs_coef[0][0])* xrs_coef[0][1];
    shortx = FAC_SHORT * shortx;
    if (shortx < min_short)
        shortx = min_short;

    /*   calculate long flux   */
    if ((g_raw.raw_w[57] == 0) || (g_raw.raw_w[57] == 255))
        g_stat.stat2 = g_stat.stat2 | 010;
    longx = (float) g_raw.raw_w[57] * .02;
    longx = (longx - xrs_coef[1][0])* xrs_coef[1][1];
    longx = FAC_LONG * longx;
    if (longx < min_long)
        longx = min_long;

    if (g_stat.xrs_lgain > 0)
     {
        g_stat.stat2 = g_stat.stat2 | 040;
        longx = NO_DATA;
     }
    if (g_stat.xrs_sgain > 0)
     {
        g_stat.stat2 = g_stat.stat2 | 0100;
        shortx = NO_DATA;
     }

    /*   is xrs calibrating   */
    if (g_stat.xrs_cal > 0)
     {
        g_stat.stat2 = g_stat.stat2 | 2;
        longx = NO_DATA;
        shortx = NO_DATA;
     }

    /*   is n-s pointing locked   */
    if (la_xrs_sun == 0)
     {
        g_stat.stat2 = g_stat.stat2 | 0400;
        longx = NO_DATA;
        shortx = NO_DATA;
     }
                                        /*   are we in eclipse   */
    bits = (int) (g_stat.stat2 & 01000);
    if (bits != 0)
     {
        longx = NO_DATA;
        shortx = NO_DATA;
     }

    /*   check n-s, e-w sas   */
    if ((sas_ns > 1.2) || (sas_ns < -1.2))
```

```
    {
       longx = NO_DATA;
       shortx = NO_DATA;
       if (sas_ns > -4.)
        {
           g_stat.stat2 = g_stat.stat2 | 0400;
        }
    }
   if ((sas_ew > 1.2) || (sas_ew < -1.2))
    {
       longx = NO_DATA;
       shortx = NO_DATA;
       if (sas_ew > -4.)
        {
           g_stat.stat2 = g_stat.stat2 | 0400;
        }
    }


#ifdef LDM
   if (g_stat.frame == 16)
       printf(" %2.2d:%2.2d:%2.2d %2.2d  temp%5.1f, sh_ra %1d %2d"
           " %10.3e,  lo_ra %1d %2d %10.3e\n",
           g_stat.rec_hour, g_stat.rec_min, g_stat.rec_sec,
           g_stat.frame, temp_in, sh_ra, g_raw.raw_w[56],
           shortx, lo_ra, g_raw.raw_w[57], longx);
#endif
   gi_ave_datum (CH_XRS_LONG, longx);
   gi_ave_datum (CH_XRS_SHORT,shortx);
   return 0;
}
```

```
/* **** calc_xrs_coef */
void calc_xrs_coef(float temp, int ran_s, int ran_l)
{
/* PURPOSE
 *   calculate sensitivity and offset as a function of temperature
 *   for both channels.  The final form will be of flux = (v-offset)
 *   / sensitivity

 * INPUTS:
 *   ran_s and ran_l the short and long wavelength ranges
 * RETURN VALUE:
 *   xrs_coef[2][2];         first index 0-short, 1-long
 *                           second      0-volt offset
 *                                       1-sensitivity
 */

    /*   from Panametrics calibration book
     *   ssx, lsx are short and long channel sensitivities
     *   scx, lcx are short and long voltage offsets
     *   first set at 25 C, second set at -20 C
     *
     *   error had both s/n 1,2 with old s/n 2
     *       kx of 1.69e-5, 4.54e-6, corrected Dec 20, 95   */

   const float
//PLB will make this an array
#if GSAT == 8
            /*   s/n 002      F6751   */
            ssx[2][4]= {{5.29e11, 5.01e10, 5.14e9, 5.00e8},
                        {5.74e11, 5.34e10, 5.35e9, 5.18e8}},
            scx[2][4]= {{.504,    .501,    .503,   .501},
                        {.514,    .503,    .514,   .502}},
            lsx[2][4]= {{4.73e11, 4.52e10, 4.56e9, 4.45e8},
                        {5.01e11, 4.73e10, 4.75e9, 4.60e8}},
            lcx[2][4]= {{.500,    .501,    .501,   .501},
                        {.516,    .503,    .516,   .503}};
            const float kx[2] = {1.599e-5, 4.163e-6};

#elif GSAT == 9
            /*   s/n 001      F6747   */
            ssx[2][4]= {{5.37e11, 5.18e10, 5.23e9, 5.13e8},
                        {5.76e11, 5.45e10, 5.39e9, 5.28e8}},
            scx[2][4]= {{.519,    .503,    .518,   .503},
                        {.525,    .506,    .554,   .507}},
            lsx[2][4]= {{4.75e11, 4.56e10, 4.65e9, 4.57e8},
                        {5.07e11, 4.79e10, 4.81e9, 4.71e8}},
            lcx[2][4]= {{.507,    .501,    .507,   .501},
                        {.539,    .505,    .540,   .505}};
            const float kx[2] = {1.618e-5, 3.989e-6};
#elif GSAT == 10
            /*   s/n 003      F6749   */
            ssx[2][4]= {{5.37e11, 5.09e10, 5.11e9, 5.00e8},
                        {5.69e11, 5.32e10, 5.29e9, 5.16e8}},
            scx[2][4]= {{.499,    .501,    .500,   .501},
                        {.513,    .503,    .512,   .503}},
            lsx[2][4]= {{4.95e11, 4.75e10, 4.65e9, 4.53e8},
                        {5.37e11, 5.03e10, 4.79e9, 4.65e8}},
            lcx[2][4]= {{.508,    .502,    .507,   .501},
                        {.550,    .507,    .541,   .506}};
            const float kx[2] = {1.631e-5, 3.824e-6};
#elif GSAT == 11
            /*   s/n 002      F6751   */
            ssx[2][4]= {{5.29e11, 5.01e10, 5.14e9, 5.00e8},
                        {5.74e11, 5.34e10, 5.35e9, 5.18e8}},
            scx[2][4]= {{.504,    .501,    .503,   .501},
                        {.514,    .503,    .514,   .502}},
            lsx[2][4]= {{4.73e11, 4.52e10, 4.56e9, 4.45e8},
                        {5.01e11, 4.73e10, 4.75e9, 4.60e8}},
            lcx[2][4]= {{.500,    .501,    .501,   .501},
                        {.516,    .503,    .516,   .503}};
            const float kx[2] = {1.599e-5, 4.163e-6};

#endif

    float delta_temp;               /*   fraction for interpolation   */
    float sens_s, sens_l;

    delta_temp = (25.0 - temp)/45.;

    /*   calculate the short coefficients for the present short range   */
    /* coef is scaled by scx values at ???? */
    if (ran_s < 0)
        ran_s = 0;
    else if (ran_s > 3)
        ran_s = 3;
    xrs_coef[0][0] = scx[0][ran_s] + delta_temp * (scx[1][ran_s] -
        scx[0][ran_s]);
    sens_s = ssx[0][ran_s] + delta_temp * (ssx[1][ran_s] -
        ssx[0][ran_s]);
    xrs_coef[0][1] = 1./(sens_s * kx[0]);

    /*   calculate the long coefficients for the present long range   */
    if (ran_l < 0)
        ran_l = 0;
    else if (ran_l > 3)
        ran_l = 3;
    xrs_coef[1][0] = lcx[0][ran_l] + delta_temp * (lcx[1][ran_l] -
        lcx[0][ran_l]);
    sens_l = lsx[0][ran_l] + delta_temp * (lsx[1][ran_l] -
        lsx[0][ran_l]);
    xrs_coef[1][1] = 1./(sens_l * kx[1]);

#ifdef TEMP
    printf (" calc_xrs_coef, temp %4.1f, ranges %d %d, fract %f\n",
        temp, ran_s, ran_l, delta_temp);
    printf (" xrs_coef %f %e, %f %e\n",  xrs_coef[0][0],
        sens_s, xrs_coef[1][0], sens_l);
    printf (" sens %e %e\n", xrs_coef[0][1], xrs_coef[1][1]);
#endif

    return;
}
```

```
/* ****** goesi_28s.c
 *    part of generic read/unpack/check goesi raw records on multiple
 *    platforms
 *
 * RETURN VALUE:
 *   return of 0 means not 28sec record error,
 *                    check further and possibly use,
 *           >0 means bad and ignore
 *            1 if zero date in first record
 *            2 if zero sat id in first record
 *            3 if zero node in first record
 *            4 if something about the time....??
 *            5 if date and time do not change
 *            6 if zero date in second record
 *            9 if ....? (like return of 4)
 *           10 if zero sat id in second record
 *           11 if zero node in second record
 *
 * INPUTS:
 *   dar_i
 *   zero_2nd  value of 0 means no frame in second half, because all bytes tested by
 *             the unpack routine, goesi_unpk, were zero
 *             this happens regularly: the DMS can only handle times that differ by a
 *             second or more, so the times within the same second (they are separated
 *             by 0.512 (TBV) sec) are added to the second frame in the record.
 *   full_prt is 0 if minimize print, 1 if fuller print
 *
 * OUTPUT:
 *
 * RETURN VALUE:
 *
 * CALLED BY:
 *   goesi_xrs_list (the main program)
 *
 * CALLS:
 *    goesi_hrminsec
 *
 * PROTOTYPE
 *    int goesi_28s (struct goesi_raw *dar_i, int zero_2nd, int full_prt)
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modification 10/28/99 by P.L. Bornmann
 *     Changed time calculation to call to goesi_hrminsec
 *
 * FUTURE MODIFICATIONS:
 *   Could test on chek first, then the three values.
 *    *    */

#include <stdio.h>
#include "goesi_read.h"

#define PRT_28S  3
#define ZF_PRINT 0   /*   number of second half zero frame prints + 1   */

int goesi_28s (struct goesi_raw *dar_i, int zero_2nd, int full_prt)
   { static long num_rec         = 0L;  /* count of total number of records examined */
    static long num_28s         = 0L;  /* count of number of records having DMS-induced
28s problems */
    static long num_zero_frames = 0L;  /* number of frames with no values (all zeros) */

    int half_2nd= 0;

    long date0, date1;
    long time0, time1;
    int  chek0, chek1;
    int  ret = 0;

    /* for time string */
    char str_time[13];

    num_rec++;   /* increment count of total number of records examined */
    date0 = dar_i[0].date;
    date1 = dar_i[1].date;
    time0 = dar_i[0].time;
    time1 = dar_i[1].time;
    chek0 = dar_i[0].chek;
    chek1 = dar_i[1].chek;

    /* extract the check bits that specify check sum errors (0x60) and */
    /* bad data (0x0f), but not those that have been fixed (0x10) See goesI_read.h */
    chek0 = chek0 & 0x6f;
    chek1 = chek1 & 0x6f;

    /*   check first telemetry frame   */
    if (date0 == 0)
       {  /* year is zero*/
        ret = 1;
       }
    else if ((dar_i[0].data[125] == 0) && (chek0 == 0))
       { ret = 1;
       }
    else if ((dar_i[0].data[126] == 0) && (chek0 == 0))
       { ret = 1;
       }
    else if ((dar_i[0].data[127] == 0) && (chek0 == 0))
       { ret = 1;
       }
    else if (dar_i[0].sat == 0)
       { ret = 2;
       }
    else if (dar_i[0].node == 0)
       { ret = 3;
       }
    else if (zero_2nd == 0)  /* no frame in second half */
       {   /* no zero bytes were found during unpack */
        /* what are these array elements?  Some for of time??? */
        if ((dar_i[0].data[126] + dar_i[0].data[127]) == 0)
            ret = 4;
        if ((dar_i[0].data[125] + dar_i[0].data[126]) == 0)
            ret = 4;
       }
    /*   check for non-consistant times first   */
    else if ((date0 == date1) && (time0 != time1))
       {   /* adjacent records have identical time and date */
        ret = 5;
       }
    else if ((date0 != 0) && (date1 != 0) && (time0 != time1))
       {   /* non-zero dates, but identical times in adjacent records */
        ret = 5;
       }
    else if (chek1 != 0)
       { /*   keep ret == 0   */
       }
    else
       {  /* Passed tests for first frame */
        /*   we now know if second frame of record is all zeroes (determined during
unpack) */
        if (zero_2nd == 0)
```

```
          {
           num_zero_frames++;
          }
        else
          {  /* Second frame has data */
           if (date1 == 0)
              { ret = 6;
              }
           else if ((dar_i[1].data[127] == 0) && (chek1 == 0))
              { ret = 9;
              }
           else if ((dar_i[1].data[126] == 0) && (chek1 == 0))
              { ret = 9;
              }
           else if ((dar_i[1].data[125] == 0) && (chek1 == 0))
              { ret = 9;
              }
           else if (dar_i[1].sat == 0)
              { ret =10;
              }
           else if (dar_i[1].node == 0)
              { ret = 11;
              }
          } /* end of else from if zero_2nd */
       } /* end of else from start of frame tests */


    /* Diagnostic printouts if zero records are found */
    if (ret != 0)
       { num_28s++;
        if ((num_28s < PRT_28S) || (full_prt))
          { char str1[40];
           char str2[60];
           int ihr, min, sec;
           if ((date0 == date1) || (date1 == 0L))
              { (void) sprintf (str1, "numrec %5ld", num_rec);
              }
           else
              { (void) sprintf (str1, " %5ld %5ld %ld,",
                   num_rec, date0, date1);
              }
           if (time0 == time1)
              {
//               ihr = (int) (time0/3600L);
//               sec = (int) (time0 - (long) ihr *3600L);
//               min = sec/60;
//               sec = sec - 60*min;
                 (void) goesi_hrminsec (dar_i[0], &ihr, &min, &sec, str_time);
//               (void) sprintf (str2, "        %2.2d:%2.2d:%2.2d 28S IBAD "
                 (void) sprintf (str2, "        %s 28S IBAD %d, %s",
                              str_time, ret, str1);
              }
           else
              { (void) sprintf( str2, "        28S IBAD %d, %s %5ld %ld,",
                   ret, str1, time0, time1);
              }
           (void) printf ("%s - %2.2x %2.2x %2.2x     %2.2x %2.2x %2.2x\n",
               str2, dar_i[0].data[126], dar_i[0].data[127], dar_i[0].chek,
                    dar_i[1].data[126], dar_i[1].data[127], dar_i[1].chek);
          }
       } /* end of ret != 0 */

    return ret;

   } /* end of goesi_28s */
```

```
/* ***** *    *
 *                goesi_frame_chk.c
 *
 * PURPOSE:
 *   part of goesI_file_read, the quick check of a goesI raw data file
 *   checks values that should not change to be sure the did not change
 *
 * INTPUT:
 *
 * OUTPUT:
 *
 * RETURN VALUE:
 *   positive if detect a problem in telemetered value (via. file_ok)
 *   zero if test values are within expectations.
 *
 * CALLS:
 *
 * CALLED BY:
 *   goesI_xrs_list
 *
 * ALGORITHM:
 *   Check that S/C ID remains constant.
 *   Check that Energetic Particle Sensor (EPS) counter not exceed hardware limit.
 *   Check for zero voltage in s/c battery 1.
 *   Check for zero voltage in s/c battery 2.
 *   Check the cell voltage of s/c battery 1.
 *   Check the cell voltage of s/c battery 2.
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *
 *  *    */

#include <stdio.h>
#include "goesi_read.h"

int goesi_frame_chk (struct goesi_raw fr, long rec_inp,
    int fr_no)     /*   0 or 1 for first or second frame in record   */
{
    int eps_count;
    int frame;
    int file_ok;
    /*  frame check less timing error and single corrected bit   */
    int fr_chek;
    static int lsat_id = -1;
    int sat_id;
    int hour;
    int min;
    int sec;

    static long num_bat1_volt = 0;
    static long num_bat2_volt = 0;
    static long num_bat1_cell = 0;
    static long num_bat2_cell = 0;
    static long num_sat_id = 0;     /* count of the number of changes in the (constant)
satellite ID */
    static long num_eps_ctr = 0;

    const char sat_nam [16][11] =    {{"UNKNOWN),"},{"UNKnown),"},
          {"GOES-L), "},{"UNKnown),"},{"UNKnown),"},{"UNKnown),"},
          {"goes-9). "},{"UNKnown),"},{"UNKnown),"},{"goes-8), "},
          {"goes-10),"},{"UNKnown),"},{"UNKnown),"},{"GOES-M), "},
          {"UNKnown),"},{"UNKnown),"}};

    frame = fr.data[2] >> 3;  /* bit shift */
    fr_chek = fr.chek & 0x6f;  /* Check for unfixed checksum errors */
    file_ok = 0;

    /*   check for satellite id changes   */
    sat_id = ((int) fr.data[3]) >> 4; /* bit shift */    if (sat_id != lsat_id)
    {  /* Satellite ID has change. */
        if (fr_chek == 0)
        {  /* fr_chek does not indicate a known error */
            num_sat_id++;   /* Increment count of changes of the Satellite's ID */
            if (num_sat_id < 20)
            {   /* Print out message for first 20 errors */
                hour = (int) (fr.time /3600L);
                min  = (int) ((fr.time -3600L * (long) hour)/ 60L);
                sec  = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);

                (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1d sat "
                    "ident is %x (%s chek is %2x %2x %2x\n",
                    hour, min, sec, rec_inp, fr_no, sat_id, sat_nam[sat_id],
                    fr.chek, fr.data[6], fr.data[127]);
            }
            if ((sat_nam[sat_id][0] =='U') || (num_sat_id > 2))
            {  /* Satellite's ID is an unknown satellite.  What if num_sat_id <0 or > 16?
*/
                if (file_ok == 0)
                    file_ok = 1;
            }
            lsat_id = sat_id;
        }
    } /* end of check on satellite ID number */

    /*   check for bad eps counters > 79.  Spacecraft harware will not
         permit these values, so they indicate an error in the frame.
         This could arise when the s/c antenna's fov is blocked by the
         s/c body, resulting in a signal dropout.  */
    eps_count = fr.data[6] >> 1;  /* bit shift */
    if ((eps_count > 79) && (fr_chek == 0))
    {
        num_eps_ctr++;
        if (num_eps_ctr < 20)
        {
            hour = (int) (fr.time /3600L);
            min = (int) ((fr.time -3600L * (long) hour)/ 60L);
            sec = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);
            (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1i "
                "EPS COUNTER is %d, chek is %2x\n",
                hour, min, sec, rec_inp, fr_no, eps_count, fr.chek);
        }
        if (file_ok == 0)
            file_ok = 1;
    }

    /*   check for zero battery 1 voltage (word 69)   */
    if ((fr.data[69] == 0) && (fr_chek == 0))
    {
        num_bat1_volt++;
        if (num_bat1_volt < 20)
        {
            hour = (int) (fr.time /3600L);
            min = (int) ((fr.time -3600L * (long) hour)/ 60L);
            sec = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);
            (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1d battery "
                "ONE VOLTAGE is %d, chek is %2x\n",
                hour, min, sec, rec_inp, fr_no, fr.data[69] , fr.chek);
            if (file_ok == 0)
```

```
                file_ok = 1;
            }
        }

        /*   check for zero battery 2 voltage (word 77)   */
        if ((fr.data[77] == 0) && (fr_chek == 0))
        {
            num_bat2_volt++;
            if (num_bat2_volt < 28)
            {
                hour = (int) (fr.time /3600L);
                min = (int) ((fr.time -3600L * (long) hour)/ 60L);
                sec = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);
                (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1d "
                    "battery TWO VOLTAGE is %d, chek is %2x\n",
                    hour, min, sec, rec_inp, fr_no, fr.data[77] , fr.chek);
                if (file_ok == 0)
                    file_ok++;
            }
        }

        /*   check battery one cell voltages (word 110)   */
        if (frame < 28)
        {
            if ((fr.data[110] == 0) && (fr_chek == 0))
            {
                hour = (int) (fr.time /3600L);
                min = (int) ((fr.time -3600L * (long) hour)/ 60L);
                sec = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);
                num_bat1_cell++;
                if (num_bat1_cell < 20)
                {
                    (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1d "
                        "battery ONE, CELL %2d voltage is %2d, chek is %2x\n",
                        hour, min, sec, rec_inp, fr_no, frame+1, fr.data[110],
                        fr.chek);
                if (file_ok == 0)
                    file_ok++;
                }
            }
        }

        /*   check battery two cell voltages   */
        if (frame < 28)
        {
            if ((fr.data[111] == 0) && (fr_chek == 0))
            {
                hour = (int) (fr.time /3600L);
                min = (int) ((fr.time -3600L * (long) hour)/ 60L);
                sec = (int) (fr.time - 3600L * (long) hour - 60L * (long) min);
                num_bat2_cell++;
                if (num_bat2_cell < 20)
                {
                    if (fr.data[111] == 0)
                        (void) printf ("        %2.2d:%2.2d:%2.2d, rec %5ld.%1d "
                            "battery TWO, CELL %2d voltage is %2d, chek is %2x\n",
                            hour, min, sec, rec_inp, fr_no, frame+1, fr.data[111],
fr.chek);
                    if (file_ok == 0)
                        file_ok++;
                }
            }
        }

        return file_ok;
    } /* end of goesi_frame_check */
```

```
/* ***** goesi_hrminsec.c
 *
 * PURPOSE:
 *       passes back hour, minute, and second from goes frame time
 *
 * INPUT:
 *       fr    goes frame, which contains time
 *       string  must be a character array with at least 12 elements
 *
 * OUTPUT:
 *       hr, min, sec   integer values for the time units
 *       string representing the time informat hh:mm:ss
 *
 * RETURN VALUE:
 *       -1 if times are out of bounds
 *        0 if success
 *
 * ALGORITHM:
 *
 * ASSUMPTIONS:
 *
 *
 * PROTOTYPE:
 *
 * INFORMATION:
 *
 * CALLS:
 *
 * CALLED BY:
 *    print_xrs.c
 *
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modification 10/26/99 by P.L. Bornmann
 *     extracted the current conversions from eclipse.c
 *   Modification 10/28/99 by P.L. Bornmann
 *      added validity time
 *      changed return value to string
 *
 * QUESTIONS:
 *   why/how are xrs long and
short range bits (w8) shuffled? *       *         *         */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "goesi_read.h"


/* PRT_LEV: 0 is least print, 2 is most   */
#define PRT_LEV   0
int goesi_hrminsec (struct goesi_raw fr, int *hr, int *min, int *sec, char *str_time)
{
    int ret;
    int vhr, vmin, vsec;
    int i;

#if PRT_LEV > 0
    (void) printf ("goesi_hrminsec.c: started\n");
#endif

    /*   decode seconds into hrs, min, sec   */
    vhr  = (int) fr.time/3600L;
    vsec = (int) fr.time - 3600L * (long) (vhr);
    vmin = vsec/60;
    vsec = vsec - 60 * (vmin);

#if PRT_LEV > 0
    (void) printf ("goesi_hrminsec.c: vhr vmin vsec is %d:%d:%d\n", vhr, vmin, vsec);
#endif

    /* Zero out entire string */
    for (i = 0; i < 13; i++) str_time[i] = 0;

    if ((vhr > 24 || vmin > 60) || (vsec > 60) ||
        (vhr <  0 || vmin <  0) || (vsec <  0) )
      {  /* invalid time */
#if PRT_LEV > 0
    (void) printf ("goesi_hrminsec.c: invalid time %d:%d:%d\n", vhr, vmin, vsec);
#endif
         /* Zero the time, string is aleady zero */
         *hr  = 0;
         *min = 0;
         *sec = 0;
         ret = -1;
      }
    else
      { /* valid time */
        *hr  = vhr;
        *min = vmin;
        *sec = vsec;

        /* time as character string */
        (void) sprintf (str_time, "%2.2d:%2.2d:%2.2d",
                        vhr, vmin, vsec);
#if PRT_LEV > 0
        (void) printf ("goesi_hrminsec: str_time = %s\n", str_time);
#endif
        ret = 0;
      }

#if PRT_LEV > 0
    (void) printf("goesi_hrminsec: returns str_time: %s\n", str_time);
    (void) printf("goesi_hrminsec: %2.2d:%2.2d:%2.2d\n",
                  *hr, *min, *sec);
#endif
//    (void) exit(EXIT_SUCCESS);
    return ret;
}  /* end of goesi_hrminsec */
```

```
/* ***** eng_sattde.c
 *
 * PURPOSE:
 *   Get the Solar Array and Trim Tab Drive Electonics status from
 *   the telemetry data
 *
 * INPUT:
 *    fr  the goesi_raw structure containing all the telemetry data
 *
 * OUTPUT:
 *
 * PROTOTYPE:
 *
 * ALGORITHM:
 *    Reports changes in some subcommed data.
 *    Extracts data from telemetry.
 *    Calls goesi_subcom to extracted subcommed data.
 *
 * CALLS:
 *    (nothing)
 *
 * CALLED BY:
 *    print_xrs
 *
 * MODIFICATION HISTORY:
 *  Extracted from print_xrs 2/7/00 by P.L. Bornmann
 *
 * TO DO:
 *
 * FUTURE MODS:
 *
 * QUESTIONS:
 *
 * ALGORITHM DETAILS:
 *       *         *         */

#include <stdio.h>
#include <stdlib.h>

#include "goesi_read.h"

/* Temporary for testing */
#include <string.h>

/* Will print telemetered values to file if PR_TELEM == 1 */
#define PRT_LEV 0

int  goesi_sattde ( struct goesi_raw fr,
                    char str_engsattde1[16], char str_engsattde2[16],
                    char strh_engsattde1[16], char strh_engsattde2[16],
                    char str_tlmsattde1[16], char str_tlmsattde2[16],
                    char strh_tlmsattde1[16], char strh_tlmsattde2[16] )
 {

    int pr_level;         /* whether to print, holds PRT_LEV */
    int  eng_sattde1[6] = {0,0,0,0,0,0};
    int  eng_sattde2[6] = {0,0,0,0,0,0};
    int  tlm_sattde1[6] = {0,0,0,0,0,0};
    int  tlm_sattde2[6] = {0,0,0,0,0,0};

    pr_level = PRT_LEV;
    /* ****************************/
    /* THE SADA ELECTRONICS STATUS */
    /* ****************************/
    /* eng_sattde1[0] : eng_sattde1 electronics on 1 = on, 2 = no */
    /* eng_sattde1[1] : eng_sattde 2 used, 1 = no (eng_sattde1 used), 2 = yes */
    /* eng_sattde1[2] : eng_sattde1 direction, 1 = forward, 2 = backward */
    /* eng_sattde1[3] : sada slew enabled, 1 = will slew, 2 = will not */
    /* eng_sattde1[4] : single/double step 1 = single, 2 = double */
    /* eng_sattde1[5] : double step delay time, setting number 1 or 2 */
    /* eng_sattde1[6] : solar array of trim tab driven by eng_sattde , 1 = SA, 2 = TT */

    /* Capture the telemetry values for SATTDE */
    tlm_sattde1[0] = ((fr.data[51] & 0x001) >> 0);
    tlm_sattde1[1] = ((fr.data[51] & 0x002) >> 1);
    tlm_sattde1[2] = ((fr.data[51] & 0x008) >> 2);
    tlm_sattde1[3] = ((fr.data[51] & 0x010) >> 3);
    tlm_sattde1[4] = ((fr.data[51] & 0x020) >> 4);
    tlm_sattde1[5] = ((fr.data[51] & 0x080) >> 5);
    tlm_sattde1[6] = ((fr.data[51] & 0x100) >> 6);

    /* SATTDE values changed so 0 is nominal operations */
    eng_sattde1[0] = 2 - ((fr.data[51] & 0x001) >> 0);   /* Change from 0,1 to 2,1 */
    eng_sattde1[1] = 1 + ((fr.data[51] & 0x002) >> 1);   /* Change from 0,1 to 1,2 */
    eng_sattde1[2] = 1 + ((fr.data[51] & 0x008) >> 2);   /* Change from 0,1 to 1,2 */
    eng_sattde1[3] = 1 + ((fr.data[51] & 0x010) >> 3);   /* Change from 0,1 to 1,2 */
    eng_sattde1[4] = 1 + ((fr.data[51] & 0x020) >> 4);   /* Change from 0,1 to 1,2 */
    eng_sattde1[5] = 1 + ((fr.data[51] & 0x080) >> 5);   /* Change from 0,1 to 1,2 */
    eng_sattde1[6] = 2 - ((fr.data[51] & 0x100) >> 6);   /* Change from 0,1 to 2,1 */

    /* Save the results in a string */
    (void) sprintf (strh_engsattde1, "%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s",
                                "PrOn", "SecOff", "Dir", "Enable", "Profile", "Offset",
"TrimOff");
    (void) sprintf (strh_tlmsattde1, "%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s",
                                "PrOn", "SecOff", "Dir", "Enable", "Profile", "Offset",
"TrimOff");
    (void) sprintf (str_engsattde1, "%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d",
                          eng_sattde1[0], eng_sattde1[1], eng_sattde1[2], eng_sattde1[3],
                          eng_sattde1[4], eng_sattde1[5], eng_sattde1[6]);
    (void) sprintf (str_tlmsattde1, "%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d%1.1d",
                          tlm_sattde1[0], tlm_sattde1[1], tlm_sattde1[2], tlm_sattde1[3],
                          tlm_sattde1[4], tlm_sattde1[5], tlm_sattde1[6]);
    if (pr_level > 2)
      { (void) printf ("goesi_sattde.c: strh_engsattde1 %s\n", strh_engsattde1);
        (void) printf ("goesi_sattde.c:  str_engsattde1 %s\n", str_engsattde1);
        (void) printf ("goesi_sattde.c: strh_tlmsattde1 %s\n", strh_tlmsattde1);
        (void) printf ("goesi_sattde.c:  str_tlmsattde1 %s\n", str_tlmsattde1);
      }

    return 1;

 } /* end of eng_sattde */
```

```
/* ***** goesi_subcom.c    *        *        *        *

 * PURPOSE:
 *   used to get and convert some subcommed temperatures, voltages,
 *   angles, etc for goesI_xrs_list
 *
 * INPUT:
 *   fr
 *   str_vals      pointer to character string of at least XXX characters
 *
 * OUTPUT:
 *   print statements
 *   str_vals      a character string containing all the subcommed information
 *                 should be declared as at least 85 characters  (check return
 *                 value to verify string returned fit in allocated space)
 *   strh_vals     a character string containing the value names (for column headings)
 *                 should be declared same size as str_vals
 *
 * RETURN VALUE:
 *   str_vals_len  the length of the final string containing all the subcom data
 *
 *    xrs_dstat  [0] XRS on(1) or off(2)
 *               [1] XRS sun(1) or nosun(2)
 *               [2] XRS calibration off(1) or on(2)
 *               [3] XRS slew off(1) or on(2)
 *               [4] XRS slew direction north(1) or south(2)
 *               [5] XRS heaters enabled(1) or disabled(2)
 *               [6] XRP bearing heaters enabled(1) or disabled(2)
 *    NOTE: Status output differs from the telemetered notation.
 *          They are changed to facilitate XRS data processing, so that 1's all
 *          indicate good data)
 *    status values in output are set to
 *           0 if no data in that subcom,
 *           1 if data should be routine solar obs (or slew is north)
 *           2 if data differs from routine obs (e.g. cal, slew) (or slew is south)
 *   xrs_pos     [0] is SAS N/S (-2 to 2 degrees)
 *               [1] is SAS E/W (-2 to 2 degrees)
 *               [2] is coarse N/S (30 to -100 degrees)
 *   xrs_volt    [0] is reference voltage (0 to -12 volts)
 *               [1] is the -70 volt ion chamber bias voltage (0 to -120 volts)
 *               [2] is the tlm calib voltage  (??? volts )
 *   xrs_temp    [0] is preamp temp (-40 to 70 degrees Celcius)
 *               [1] is SAS temp    (-40 to 70 degrees Celcius)
 *               [2] is XRP temp    (-40 to 70 degrees Celcius)p
 *               [3] is XRP bearing temperature (-40 to 70 degC)
 *               [4] is XRP drive electronics temperature (-40 to 70 deg C)
 * NOTE:  telemetry is provided as 8 bits, so accuracy is 1/256 = 0.4% of range
 *  str_vals     will be a string with all the formated values in engineering units.
 *               must be declared as a string with at least 85 characters.
 *
 * ACRONYMS:
 *   XRS  X-Ray Spectrometer
 *   XRP  X-Ray Positioner
 *   SAS  Sun Angle Sensor
 *
 * RETURN VALUE:
 *   (none)
 *
 * PROTOTYPE:
 *   int goesi_subcom (struct goesi_raw fr, char *str_vals, char *strh_vals,
 *                     char *str_dvals, char *strh_dvals)
 *
 * USAGE:
 *    char str_vals[85];
 *    char strh_vals[85];
 *    (void) goesi_subcom (fr, str_vals, strh_vals, str_dvals, strh_dvals)
 *
 * CALLS:
 *   goesi_vcurve  for conversions to engineering units
 *
 * CALLED BY:
 *   print_xrs
 *
 * ALGORITHM:
 *     the telemtry converted and set by this function come from words
 *     72, 100 and 101.  The conversions are to six character strings.
 *     Digital subcommed status comes from words 36 and 37
 *    Compiler option, EXACT_SUBCOM determines whether status bit
 *     contains most recently known value or the value when actually subcommed.
 *   Setting preprocessor #define EXACT_SUBCOM 0 gives status bits set to
 *     value of most recent subcom reporting that value.
 *     Note: actual values may have changed occured up to 16 telemetry points earlier.
 *     Note: the first 16 telemetry values processed will not necessarily be
 *         correct; must wait till that value has been subcommed.
 *   Setting preprocessor #define EXACT_SUBCOM to 1 will show exact telemetry
 *     values at time they are subcommed.
 *
 * NOTE: The SAS (sun angle sensor) may required thermal correction factor.  This
 *    has not been included in the telemetry-to-engineering-unit conversions.
 *
*N  word  36 sub 10,26 [10(16)] incl autoload             digital
*N              16,32 [16(16)] incl SATTDE 2             digital
*N  word  37 sub  2,10,18,26
*N                    [ 2(8)] XRS positioner bearing
*N                            heaters status
*N                            (thermostatically controlled )
*N                            off (0), enabled (1)       bit 3     xrs_dstat[6]
*N               2,10,18,26
*N                    [ 2(8)] XRS heaters status
*N                           (thermostatically controlled )
*N                            off (0), enabled (1)       bit 4      xrs_dstat[5]
*Y  word  52 sub  5     [ 5(16)] xrpe slew = 0, track = 1  bit 8      xrs_dstat[3]
*Y                6     [ 6(16)] xrpe slew north = 0,
*Y                            south = 1                  bit 8      xrs_dstat[4]
*Y                7     [ 7(16)] XRS sun not present = 0,
*Y                            is present = 1             bit 8      xrs_dstat[1]
*Y                8     [ 8(16)] XRS cal = 0, data = 1     bit 8      xrs_dstat[2]
*Y               14     [14(16)] XRS on = 0, off = 1       bit 8      xrs_dstat[0]
*Y  word  72 sub 29     [29(32)] XRS reference voltage     curve 378  xrs_volt[0]
*Y               30     [30(32)] XRS -70 volt bias         curve 382  xrs_volt[1]
*Y                            (the -75 volt chamber bias voltage)
*Y               32     [32(32)] tlm calibration voltage   curve 220  xrs_volt[2]
* Fixed voltage applied XRS A/D converter to convert the analog
* values to digital values (0-5 volts)
*Y  word 100 sub  5     [ 5(32)] XRS coarse position       curve 379  xrs_pos[2]
*Y                6     [ 6(32)] SAS analog sun (N/S)      curve 380  xrs_pos[0]
*Y                7     [ 7(32)] SAS analog sun (E/W)      curve 381  xrs_pos[1]
*Y  word 101 sub  1,17  [ 1(16)] sun analog sensor temp    curve 233  xrs_temp[1]
*Y                2,18  [ 2(16)] XRS position bearing t    curve 233  xrs_temp[3]
*Y                3,19  [ 3(16)] XRS preamp temperature    curve 233  xrs_temp[0]
*Y                4,20  [ 4(16)] XRS positioner motor temp curve 233  xrs_temp[2]
*Y                5,21  [ 5(16)] XRS positioner drive
*Y                            electronics temperature    curve 233  xrs_temp[4]
*N               8,24  [ 8(16)] sxi power electronics temp (Goes-M)
 *
 * MODIFICATION HISTORY:
 *   Started Sep 99, ldm (Lorne Matheson)
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modified 10/24/99 by P.L.Bornmann
 *     to extract additional subcom data
```

```
 *     moved conversions to goesi_vcurve
 *   Modified 10/26/99 by P.L. Bornmann
 *     Corrected apparent error in location of the XRS preamp data.
 *     LM code says twice it is subcom 3, but code had it as
 *     subcom 21(32).  (Subcom 5,21 [5(16)] is XRP drive electronics
 *     temperature.)
 *  Modified 11/19/99 by P.L. Bornmann
 *     Corrected preprocessor logic for exact subcom vs. most recent value
 *     Added the integer telemetered value as output, the dvals (data values)
 *  Modified 2/10/00 by P.L. Bornmann
 *     Removed c_word101, since its values are returned in the interpreted string
 *
 * FUTURE MODIFICATIONS:
 *   pass back the arrays rather than just strings?
 *   include temperature correction in SAS
 *   double check all subcoms are being saved properly
 *
 * QUESTIONS:
 *   Is XRS reference voltage -70 or -75?
 *   Double check frequency of word 37 info.
 *   Double check XRPE temp
 *   Range of TLM calib voltage, and its meaning.
 *   Get more info on word 36
 *
 *  *    *    *    */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "goesi_read.h"

#define PRT_LEV 0

#define EXACT_SUBCOM 0

int goesi_subcom (struct goesi_raw fr, char *strh_vals, char *str_vals,
                  char *strh_dvals, char *str_dvals)
 {
    /* Counters */
    static int start = 0;
    int mf_ct;
    int mf_ct8;
    int mf_ct16;

    /* Formats */
#define MCHAR 50
    const char fmt_dstat[MCHAR] = "%1d%1d%1d%1d%1d%1d%1d ";
    const char fmt_dpos [MCHAR] = "%3d %3d %3d ";
    const char fmt_dvolt[MCHAR] = "%3d %3d %3d ";
    const char fmt_dtemp[MCHAR] = "%3d %3d %3d %3d %3d ";

    const char fmt_stat [MCHAR] = "%1d%1d%1d%1d%1d%1d%1d ";
    const char fmt_pos  [MCHAR] = "%6.2f %6.2f %6.2f ";
    const char fmt_volt [MCHAR] = "%6.2f %6.2f %6.2f ";
    const char fmt_temp [MCHAR] = "%6.2f %6.2f %6.2f %6.2f %6.2f ";

    /* Header Formats */
    const char fmth_dstat [MCHAR] =  "%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s ";
    const char fmth_dpos  [MCHAR] =  "%3.3s %3.3s %3.3s "          ;
    const char fmth_dvolt [MCHAR] =  "%3.3s %3.3s %3.3s "          ;
    const char fmth_dtemp [MCHAR] =  "%3.3s %3.3s %3.3s %3.3s %3.3s ";

    const char fmth_stat  [MCHAR] =  "%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s%1.1s ";
    const char fmth_pos   [MCHAR] =  "%6.6s %6.6s %6.6s "          ;
    const char fmth_volt  [MCHAR] =  "%6.6s %6.6s %6.6s "          ;
    const char fmth_temp  [MCHAR] =  "%6.6s %6.6s %6.6s %6.6s %6.6s ";

    int w52bit8;

#define NXRS_STAT 7
#define NSAS_POS  3
#define NXRS_VOLT 3
#define NXRS_TEMP 5

#if EXACT_SUBCOM
    /* reset all status values to zero for each call when they are not subcommed */
    int   xrs_dstat[NXRS_STAT] = {0,0,0,0,0,0,0};
    int   xrs_dpos [NSAS_POS ] = {0,0,0};
    int   xrs_dvolt[NXRS_VOLT] = {0,0,0};
    int   xrs_dtemp[NXRS_TEMP] = {0,0,0,0,0};
    float xrs_pos  [NSAS_POS ] = {0,0,0};
    float xrs_volt [NXRS_VOLT] = {0,0,0};
    float xrs_temp [NXRS_TEMP] = {0,0,0,0,0};
#else
    /* save values so arrays contain most recent information */
    static int   xrs_dstat[NXRS_STAT] = {0,0,0,0,0,0,0};
    static int   xrs_dpos [NSAS_POS ] = {0,0,0};
    static int   xrs_dvolt[NXRS_VOLT] = {0,0,0};
    static int   xrs_dtemp[NXRS_TEMP] = {0,0,0,0,0};
    static float xrs_pos  [NSAS_POS ] = {0,0,0};
    static float xrs_volt [NXRS_VOLT] = {0,0,0};
    static float xrs_temp [NXRS_TEMP] = {0,0,0,0,0};
#endif


    /* Telemetered values */
    char str_dstat[50]  = "";
    char str_dpos [50]  = "";
    char str_dvolt[50]  = "";
    char str_dtemp[50]  = "";


    /* Converted to engineering values */
    char str_stat[50]  = "";
    char str_pos [50]  = "";
    char str_volt[50]  = "";
    char str_temp[50]  = "";

    /* Headers */
    char strh_stat[20] = "";
    char strh_pos [50] = "";
    char strh_volt[50] = "";
    char strh_temp[50] = "";
    char strh_dstat[20] = "";
    char strh_dpos [50] = "";
    char strh_dvolt[50] = "";
    char strh_dtemp[50] = "";

    long len_strvals   = -1;

    /* These are used to establish the frequency of progress-reporting printouts */
    static long prog_cnt;     /* number of records processed */
    long pr_prog;              /* whether to report progress */

    if (start == 0)
     {   start = 1;

#if PRT_LEV > 1
       (void) printf ("goesi_subcom: frame start date %ld, time %ld\n",
                      fr.date, fr.time);
```

```
#endif
     }
    /* number of records between reports */
#define PROG_FREQ 20

    /* ************************/
    /* Create the header info */
    /* ************************/
    (void) sprintf (strh_dstat, fmth_dstat, "On", "Sun", "Obs", "Trk", "Ht", "Bht");
    (void) sprintf (strh_dpos,  fmth_dpos , "NS_SAS" , "EW_SAS"  , "CRS"       );
    (void) sprintf (strh_dvolt, fmth_dvolt, "RfV", "BV", "TRfV"     );
    (void) sprintf (strh_dtemp, fmth_dtemp, "PrT" , "ArT"   , "XPT",  "BrT", "DrT");

    (void) sprintf (strh_stat , fmth_stat , "On", "Sun", "Obs", "Trk", "Ht", "Bht");
    (void) sprintf (strh_pos,   fmth_pos  , "NS_SAS" , "EW_SAS"  , "NS_CRS"       );
    (void) sprintf (strh_volt,  fmth_volt , "RefV", "BiasV", "TlmRfV"     );
    (void) sprintf (strh_temp,  fmth_dtemp, "PreAmT" , "SAS_T"   , "XRP_T",  "BerT",
"drEleT");

    /* Combine the header strings into single string */
    (void) strcpy (strh_vals, "");
    (void) strcat (strh_vals, strh_stat);
    (void) strcat (strh_vals, strh_pos );
    (void) strcat (strh_vals, strh_temp);
    (void) strcat (strh_vals, strh_volt);

    (void) strcpy (strh_dvals, "");
    (void) strcat (strh_dvals, strh_dstat);
    (void) strcat (strh_dvals, strh_dpos );
    (void) strcat (strh_dvals, strh_dtemp);
    (void) strcat (strh_dvals, strh_dvolt);

    /* Main frame counter */
    mf_ct   = fr.data[2]>>3;  /* bit shift to get frame counter */
    mf_ct16 = mf_ct % 16;     /* counter for the values repeated in 2x per cycle */
    mf_ct8 = mf_ct % 8 ;     /* counter for the values repeated in 4x per cycle */
    prog_cnt++;               /* number of records processed (number of calls to routine)
*/
#if PRT_LEV > 1
    (void) printf ("counters  %3d %3d, %ld\n",
                   mf_ct, mf_ct16, prog_cnt);
#endif

    if (prog_cnt < 5)
     {  pr_prog = 0;
     }
    else
     { pr_prog = prog_cnt % PROG_FREQ;
     }
    if (PRT_LEV == 0) pr_prog = -1;

    /* ************************************
     ** PROCESS QUADRUPLY SUBCOMMED DATA  **
     **   XRS and XRP bearing heater stat **
     ************************************/
    /* Bit 8 of word 52 contains XRS status info */
    w52bit8    = fr.data[52] & 0x1; /* only need first bit of this subcommed word */

#if PRT_LEV > 1
    if (pr_prog == 0) (void) printf ("counters  %3d %3d, bit8 %d\n",
                                     mf_ct, mf_ct16, w52bit8);
#endif

#if PRT_LEV > 1
    (void) printf ("goesi_subcom.c: frames to examine (36,37,52,72,100,101): %4d %4d %4d
%4d %4d %4d\n",
                   fr.data[36], fr.data[37], fr.data[52], fr.data[72], fr.data[100],
fr.data[101]);
#endif


    switch (mf_ct8+1)   /* bit 8 has the status of the observations */
     {  /* check dual cycle values */
        case 2:
          { /* digital status, including XRS heater status   */
            /* word  37 sub  2,10,18,26  [ 2(8)] XRS heaters status
             *                                   off (0), enabled (1)       bit 4
xrs_dstat[5]
            /* XRP (x-ray positioner) heater enabled status (thermostatically controlled)
*/
            xrs_dstat[5] = fr.data[37] & 0x08;

            /* word  37 sub  2,10,18,26  [ 2(8)] XRP bearing heater status
             *                                   off (0), enabled (1)  bit 3
xrs_dstat[6] */
            xrs_dstat[6] = fr.data[37] & 0x02;
#if PRT_LEV > 1
            (void) printf ("goesi_subcom.c: xrs_dstat[5 and 6] = %d %d\n", xrs_dstat[5],
xrs_dstat[6]);
#endif
          }
          break;
        default:
          break;
     } /* end of switch (mf_ct8+1) */

    /* ***********************************
     ** PROCESS DOUBLY SUBCOMMED DATA **
     **   XRS temperatures            **
     **   XRS status                  **
     ***********************************/

    switch (mf_ct16+1)   /* bit 8 has the status of the observations */
     {  /* check dual cycle values */
        case 1:
          { /* word 101 sub  1,17  [ 1(16)]  sun analog sensor temp    curve 233
xrs_temp[1]
            xrs_dtemp[1] = fr.data[101];
            xrs_temp[1] = goesi_vcurve (fr.data[101], 233);
          }
          break;
        case 2:
          { /* word 101 sub  2,18  [ 2(16)]  XRS position bearing t    curve 233
xrs_temp[3]
            xrs_dtemp[3] = fr.data[101];
            xrs_temp[3] = goesi_vcurve (fr.data[101], 233);

          }
          break;
        case 3:
          { /* word 101 sub  3,19  [ 3(16)]  XRS preamp temperature    curve 233
xrs_temp[0] */
            xrs_dtemp[0] = fr.data[101];
            xrs_temp[0] = goesi_vcurve (fr.data[101], 233);
          }
          break;
        case 4:
          {/* word 101 sub  4,20  [ 4(16)]  XRS positioner temp       curve 233
xrs_temp[2] */
```

```
            xrs_dtemp[2] = fr.data[101];
            xrs_temp[2] = goesi_vcurve (fr.data[101], 233);
          }
          break;
        case 5:
          { /* word 101 sub  5,21  [ 5(16)]  XRS positioner drive ele  curve 233
xrs_temp[4]
            xrs_dtemp[4] = fr.data[101];
            xrs_temp[4] = goesi_vcurve (fr.data[101], 233);
            /* XRS slew status */
            /* word 52 sub [ 5(16)] xrpe slew = 0,
                                    track = 1     bit 8   xrs_dstat[3]*/
            xrs_dstat[3] = 2 - w52bit8;  /* xrs_dstat 1 for solar track, 2 for slewing
*/
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_dstat[3] = %d\n", xrs_dstat[3]);
#endif
          }
          break;
        case 6:
          { /* XRS slew direction */
            /* word 52 sub [ 6(16)] xrpe slew north = 0,
             *                              slew south = 1    bit 8   xrs_dstat[4]*/
            xrs_dstat[4] = w52bit8 + 1;  /* xrs_dstat 1 for slew north, 2 for slew south
*/
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_dstat[4] = %d\n", xrs_dstat[4]);
#endif
          }
          break;
        case 7:
          { /* word 52 sub [ 7(16)] XRS sun not present = 0,
             *                          sun acquired = 1        bit 8  xrs_dstat[1]*/
            xrs_dstat[1] = 2 - w52bit8;  /* xrs_dstat 1 for solar obs, 2 for no sun */
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_dstat[1] = %d\n", xrs_dstat[1]);
#endif
          }

          break;
        case  8:
          { /* XRS calibration status */
            /*   word 52 bit 8 sub [ 8(16)] XRS cal = 0, data = 1    xrs_dstat[2]*/
            xrs_dstat[2] = 2 - w52bit8;  /* xrs_dstat 1 for data, 2 for cal */
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_dstat[1] = %d\n", xrs_dstat[1]);
#endif
          }
          break;
        case 14:
          { /* XRS operation status */
            /*   word 52 bit 8 sub [14(16)] XRS on = 0, off = 1      xrs_dstat[0]*/
            xrs_dstat[0] = w52bit8 + 1;  /* xrs_dstat 1 for XRS on, 2 for off   */
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_dstat[0] = %d\n", xrs_dstat[0]);
#endif
          }
          break;
        default:
          break;
    } /* end of switch (mf_ctl6+1) */

   /* *********************************
    ** PROCESS SINGLY SUBCOMMED DATA **
    **   XRS status bits             **
    **   sun position                **
    **   XRS voltages                **
    *********************************/
   switch (mf_ct+1)
    {  /* process data that only comes up once per 32-frame subcom cycle */
       case  5:
         { /* coarse N-S XRP position, -94 to 28 degrees   curve 379   */
           /*   word 100 sub  5  XRS coarse position       curve 379  xrs_pos[2] */
           xrs_dpos[2] = fr.data[100];
           xrs_pos[2] = goesi_vcurve(fr.data[100], 379);
#if PRT_LEV > 1
           (void) printf ("goesi_subcom.c: xrs_pos[2] = %d\n", xrs_pos[2]);
#endif
         }
         break;
       case  6:
         { /*   word 100 sub  6  SAS (N/S) curve 380  xrs_pos[0] */
           /*   may need temperature compensation   */
           xrs_dpos[0] = fr.data[100];
           xrs_pos[0] = goesi_vcurve (fr.data[100], 380);
#if PRT_LEV > 1
           (void) printf ("goesi_subcom.c: xrs_pos[0] = %d\n", xrs_pos[0]);
#endif
         }
         break;
       case  7:
         { /*   word 100 sub  7  SAS (E/W) curve 381  xrs_pos[1]
           /*   may need temperature compensation   */
           xrs_dpos[1] = fr.data[100];
           xrs_pos[1] = goesi_vcurve (fr.data[100], 381);
#if PRT_LEV > 1
          (void) printf ("goesi_subcom.c: xrs_pos[1] = %d\n", xrs_pos[1]);
#endif
         }
         break;
       case 16:
         { /* digital status, including SATTDE 2   */
          /*N  word  36 sub 10,26 [10(16)] incl autoload             digital */
          /*N               16,32 [16(16)] incl SATTDE 2             digital */
         }
         break;
       case 26:
         break;
       case 29:
         {/*   word  72 sub 29  XRS ref voltage curve 378  xrs_volt[0]*/
           xrs_dvolt[0] = fr.data[72];
           xrs_volt[0] = goesi_vcurve (fr.data[72], 378);
         }
         break;
       case 30:
         { /*   word  72 sub 30  XRS -70 volt bias  curve 382  xrs_volt[1]*/
           /*   (the 75 volt chamber bias voltage)*/
           /*  the ion chamber voltage according to xrsi_conv.c */
           xrs_dvolt[1] = fr.data[72];
           xrs_volt[1] = goesi_vcurve (fr.data[72], 382);
         }
         break;
       case 32:
         { /*  word  72 sub 32  tlm calib voltage curve 220  xrs_volt[2] */
           xrs_dvolt[2] = fr.data[72];
           xrs_volt[2] = goesi_vcurve (fr.data[72], 220);
         }
           break;
       default:
           break;
```

```
    }
#if PRT_LEV > 1
     if (pr_prog == 0)
        {
          for (i=0; i<NSAS_POS ; i++) (void) printf("xrs_pos [%1d] = %5.3f\n", i, xrs_pos
[i]);
          for (i=0; i<NXRS_VOLT; i++) (void) printf("xrs_volt[%1d] = %5.3f\n", i,
xrs_volt[i]);
          (void) printf ("goesi_subcom: finished single subcom, start double \n");
        }
#endif

#if PRT_LEV > 1
    for (i=0; i<NXRS_STAT; i++) (void) printf("goesi_subcom.c: xrs_dstat[%1d] = %d\n", i,
xrs_dstat[i]);
    for (i=0; i<NSAS_POS ; i++) (void) printf("goesi_subcom.c: xrs_pos [%1d] = %6.2f
%3d\n",
                                          i, xrs_pos [i], xrs_dpos[i]);
    for (i=0; i<NXRS_VOLT; i++) (void) printf("goesi_subcom.c: xrs_volt[%1d] = %6.2f
%3d\n",
                                          i, xrs_volt[i], xrs_dvolt[i]);
    for (i=0; i<NXRS_TEMP; i++) (void) printf("goesi_subcom.c: xrs_temp[%1d] = %6.2f
%3d\n",
                                          i, xrs_temp[i], xrs_dtemp[i]);
#endif

    /* Create fixed format strings for each type of value */
    /* string lengths: str_stat = 6x1 + 1 = 7; str_pos = 7x3 +1 22;
     *                 str_volt = 7x3 + 1 = 22; str_temp = 7x5+1 = 36; */
    (void) sprintf(str_dstat, fmt_dstat,
                   xrs_dstat[0], xrs_dstat[1], xrs_dstat[2], xrs_dstat[3], xrs_dstat[4],
xrs_dstat[5]);
    (void) sprintf(str_dpos , fmt_dpos,
                   xrs_dpos[0], xrs_dpos[1], xrs_dpos[2]);
    (void) sprintf(str_dvolt, fmt_dvolt,
                   xrs_dvolt[0], xrs_dvolt[1], xrs_dvolt[2]);
    (void) sprintf(str_dtemp, fmt_dtemp,
                   xrs_dtemp[0], xrs_dtemp[1], xrs_dtemp[2], xrs_dtemp[3], xrs_dtemp[4]);

    (void) sprintf(str_stat, fmt_stat,
                   xrs_dstat[0], xrs_dstat[1], xrs_dstat[2], xrs_dstat[3], xrs_dstat[4],
xrs_dstat[5]);
    (void) sprintf(str_pos , fmt_pos,
                   xrs_pos[0], xrs_pos[1], xrs_pos[2]);
    (void) sprintf(str_volt, fmt_volt,
                   xrs_volt[0], xrs_volt[1], xrs_volt[2]);
    (void) sprintf(str_temp, fmt_temp,
                   xrs_temp[0], xrs_temp[1], xrs_temp[2], xrs_temp[3], xrs_temp[4]);

#if PRT_LEV > 1
    (void) printf ("goesi_subcom.c: strh_stat %s\n", strh_stat);
    (void) printf ("goesi_subcom.c: strh_pos  %s\n", strh_pos );
    (void) printf ("goesi_subcom.c: strh_temp %s\n", strh_temp);
    (void) printf ("goesi_subcom.c: strh_volt %s\n", strh_volt);
    (void) printf ("goesi_subcom.c: str_stat %s\n", str_stat);
    (void) printf ("goesi_subcom.c: str_pos  %s\n", str_pos );
    (void) printf ("goesi_subcom.c: str_temp %s\n", str_temp);
    (void) printf ("goesi_subcom.c: str_volt %s\n", str_volt);
    (void) printf ("goesi_subcom.c: str_dpos  %s\n", str_dpos );
    (void) printf ("goesi_subcom.c: str_dtemp %s\n", str_dtemp);
    (void) printf ("goesi_subcom.c: str_dvolt %s\n", str_dvolt);
#endif

    /* Combine the formated strings into single string */
    /* Initialize the values to a character string */
```

```
#if PRT_LEV > 1
(void) printf ("goesi_subcom.c: will concatenate strvals (%s)\n     with str_stat
(%s)\n",
                str_vals, str_stat);
#endif
    (void) strcpy (str_vals, "");
    (void) strcat (str_vals, str_stat);
    (void) strcat (str_vals, str_pos );
    (void) strcat (str_vals, str_temp);
    (void) strcat (str_vals, str_volt);

    (void) strcpy (str_dvals, "");
    (void) strcat (str_dvals, str_stat);
    (void) strcat (str_dvals, str_dpos );
    (void) strcat (str_dvals, str_dtemp);
    (void) strcat (str_dvals, str_dvolt);

    /* number of characters in the final string */
    len_strvals = strlen(str_vals);

#if PRT_LEV > 0
    (void) printf ("goesi_subcom.c: returns %ld characters in string\n", len_strvals);
    (void) printf ("goesi_subcom.c: strh_vals, and str_vals follow\n%s\n%s\n",
                   strh_vals, str_vals);
    (void) printf ("goesi_subcom.c: strh_dvals, and str_dvals follow\n%s\n%s\n",
                   strh_dvals, str_dvals);
#endif

//(void) exit (EXIT_SUCCESS);

    return len_strvals+1;  /* Return actual string length, including the '\0' */

 }  /* end of goesi_subcom */
```

```
/* ***** goesi_unpk
 *
 * PURPOSE:
 *           break and unpack a 288 byte frame record into two
 *           expanded telemetry frames
 *
 * INPUT:
 *   ar_i   the raw data for record i
 *
 * OUTPUT:
 *   dat_i  data array for the two frames in record i
 *
 * RETURN VALUE:
 *    num_zerby   the number of non-zero bytes  (Note: non-zero, not zero bytes)
 *                0 if all tested bytes were zero
 * CALLED BY:
 *   goesi_xrs_list.c
 *
 * CALLS:
 *  (nothing)
 *
 * PROTOTYPE:
 *   int goesi_unpk (unsigned char *dat_i , struct goesi_raw *ar_i)
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *
 *   *   *   */

#include <stdio.h>
#include "goesi_read.h"
#define NUM_PRT  1
#define PRT_LEV 0

int goesi_unpk (unsigned char *dat_i , struct goesi_raw *ar_i)
{
    static long nrec = 1L;
    int i, j, k;
    long temp_lo;

    int num_zerby;       /* the number of zero bytes encountered */
    int log_w[4] = {0};

    /* Loop over the two frames in the record */
    for (i=0; i<2; i++)
    {
        num_zerby = 0;
        j = 144*i;   /* the equivalent position in the second frame of the DMS record */
        temp_lo = ((long) dat_i[j]) | ((long) dat_i[j+1] << 8) |
            ((long) dat_i[j+2] << 16) | ((long) dat_i[j+3] <<24);
        if (temp_lo == 0xffff)
            temp_lo = -1L;
        if (temp_lo != 0)
            num_zerby++;   /* increment if non-zero found */
        ar_i[i].date = temp_lo;

        /* Save and check for time values */
        temp_lo = ((long) dat_i[j+4]) | ((long) dat_i[j+5] << 8) |
            ((long) dat_i[j+6] << 16) | ((long) dat_i[j+7] << 24);
        if (temp_lo != 0)
            num_zerby++;
        ar_i[i].time = temp_lo;

        ar_i[i].milli = ((int) dat_i[j+ 8]) + ((int) dat_i[j+ 9] << 8);
        if (ar_i[i].milli != 0)
            num_zerby++;

        /* Currently this is not used, per .h file */
        ar_i[i].agc   = ((int) dat_i[j+10]) + ((int) dat_i[j+11] << 8);


#ifdef FULL_PRT
        if ((nrec<4) || (( nrec>253) && (nrec < 258)))
            (void) printf (" rec %5ld, %d, first four (dms date) "
                "%2x %2x %2x %2x %10ld,\n                "
                "next four (dms time) %2x %2x %2x %2x %10ld \n",
                nrec, i, dat_i[j], dat_i[j+1], dat_i[j+2], dat_i[j+3],
                ar_i[i].date, dat_i[j+4], dat_i[j+5], dat_i[j+6],
                dat_i[j+7], ar_i[i].time);
#endif
        if (nrec < NUM_PRT)
        {
            (void) printf ("            %d, milli agc  %2x %2x %2x %2x"
                " %5d %5x\n",
                i, dat_i[j+ 8], dat_i[j+ 9], dat_i[j+10], dat_i[j+11],
                ar_i[i].milli, ar_i[i].agc);
        }

        /* Check for DMS computer port value */
        ar_i[i].port = (int) dat_i[j+12];
        if (ar_i[i].port != 0)
            num_zerby++;
        ar_i[i].node = (int) dat_i[j+13];
        if (ar_i[i].node != 0)
            num_zerby++;

        /* Check for satellite value */
        ar_i[i].sat  = (int) dat_i[j+14];
        if (ar_i[i].sat != 0)
            num_zerby++;

        ar_i[i].chek = (int) dat_i[j+15];

        /* Examine all remaining bytes for zeros */
        for (k=0; k < CHAR_PER_FRAME; k++)
        {
            ar_i[i].data[k] = dat_i[j+k+16];
            if (ar_i[i].data[k] != 0)
                num_zerby++;
        }
    }
#if PRT_LEV > 0
    (void) printf ("goesi_unpk.c: frames to examine (36,37,52,72,100,101):     "
                   "%4d %4d %4d %4d %4d %4d\n"
                   "                                                          "
                   "%4d %4d %4d %4d %4d %4d\n",
                   ar_i[0].data[36], ar_i[0].data[37], ar_i[0].data[52],
ar_i[0].data[72], ar_i[0].data[100], ar_i[0].data[101],
                   ar_i[1].data[36], ar_i[1].data[37], ar_i[1].data[52],
ar_i[1].data[72], ar_i[1].data[100], ar_i[1].data[101]);
#endif

    nrec++;
    return num_zerby;

} /* end of goesi_unpk */
```

```
/* ***** goesi_vcurve.c    *         *         *         *

 * PURPOSE:
 *   used to get and convert some subcommed temperatures, voltages,
 *   angles, etc for goesI_xrs_list, based on the conversion curve
 *   to convert telemetry to real values.
 *
 * INPUT:
 *   telem         the telemetry value from fr.data
 *   curnum        the conversion curve number:
 *                     003 for SADA output position Channel A (word 59, 0 to 360 degrees)
 *                     004 for SADA output position Channel B (word 60, 0 to 360 degrees)
 *                     220 for tlm calib voltage
 *                     233 for temperatures  (-40 to 70 degrees C)
 *                     378 for xrs reference voltage (0 to -12 volts)
 *                     379 for xrs coarse position  (30 to -100 degrees)
 *                     380 for SAS (Sun Angle Sensor) N/S (-2 to 2 degrees)
 *                     381 for SAS (Sun Angle Sensor) E/W (-2 to 2 degrees)
 *                     382 for xrs -70 volt bias  (0 to -120 volts)
 *
 * OUTPUT:
 *   c_word101     a character string of information from telem word 101
 *
 * RETURN VALUE:
 *   the telemetry value converted to engineering (scientific) units
 *
 * PROTOTYPE:
 * CALLED BY:
 *   goesi_subcom
 *
 * CALLS:
 *
 * ALGORITHM:
 *   Assumes all spacecraft (GOESI-M = GOES8-10) have same conversion
 *   factors.  Uses supplied telemetry value with specified conversion
 *   curve number to return value in engineering units.
 *
 * MODIFICATION HISTORY:
 *       started Sep 99, ldm (Lorne Matheson)
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modified 10/24/99 by P.L.Bornmann
 *     to extract additional subcom data
 *   Extracted from goesi_vcurve 10/25/99 by P.L. Bornmann
 *   Modification 11/19/99 by P.L. Bornmann
 *      curnum (the curve number) is passed as the actual number, so switch on this value
 *
 * TODO:
 *   Solar Array current
 *   What is range of curve 220 voltage?
 *
 * QUESTIONS:
 *
 *  *    *    *    */

#include <stdio.h>
#include "goesi_read.h"

#define PRT_LEV 0

float goesi_vcurve (int telem, int curnum)
 {
    static int start = 0;
    float  ftelem = 0;     /* float version of telemetry */

    /*   temperatures conversion coefficients for thermisters
         on yoke, xrpe, xrs.  Per curve 233   */
    const static float val_coef[6] =
                {   -52.51458,      1.510513,    -0.01771354,
                  1.287143e-04, -4.557991e-07, 6.370513e-10 } ;
   float val;

#if PRT_LEV > 1
    printf ("goesi_vcurve: input %d, calib curve %3d\n",
            telem, curnum);
#endif
    if (start == 0)
     {  start = 1;
     }

    switch (curnum)
     {
        case    3:
          { /*   SADA Output Position Channel A  curve 003   */
            if (telem > 235)
             {  /* Data is not valid if telem 235 */
                val = -99.9;
#if PRT_LEV > 1
                printf ("goesi_vcurve.c: SADA A position is %f for %d\n",
                        val, telem);
#endif
             }
            else
              { val = 270.10 + 1.41421*telem;
                if (val > 360) val = val - 360;
              }
          }
          break;
        case    4:
          { /*   SADA Output Position Channel A  curve 003   */
            if (telem > 237)
             {  /* Data is not valid if telem 237 */
                val = -99.9;
             }
            else
              {  val = -0.90000 + 1.41421*telem;
#if PRT_LEV > 1
                printf ("goesi_vcurve.c: SADA B position is %f for %d\n",
                        val, telem);
#endif
              }
          }
          break;
        case  220:
          { /*   telemetry calibration voltage  curve 220   */
            /* voltage used to convert analog sensors to digitial telemetry */
            val = 0.01 + .02*telem;
          }
          break;
        case  233:
          { /*   xrs pre-amp val,  curve 233   */
            val = val_coef[0] + telem*(val_coef[1] + telem*(val_coef[2] + telem*
                  (val_coef[3] + telem*(val_coef[4] + telem*val_coef[5]))));
          }
          break;
        case  378:
          { /*   xrs reference voltage,  curve 378   */
            val = -0.02 - telem*0.04;
          }
          break;
        case  379:
```

```
        { /*   coarse N-S xrp position, -94 to 28 degrees   curve 379   */
          val = 28.00 - telem*0.481601;
        }
        break;
      case  380:
        { /*   xrs n/s analog sun sensor    curve 380
          *       may need temperature compensation   */
          val = -1.992 + telem*0.0160;
        }
        break;
      case  381:
        { /*   xrs e/w analog sun sensor    curve 381
           *       may need temperature compensation   */
          val = -1.992 + telem*0.0160;
        }
        break;
      case  382:
        { /*   xrs -70 volt bias volt,  curve 382   */
          val = -0.2 - telem*0.4;
        }
        break;
      default:
        { (void) printf ("goesi_vcurve: ERROR. invalid curve number %d", curnum);
        }
        break;
    }


#if PRT_LEV > 0
    printf ("goesi_vcurve: input %d, output %5.3f, calib curve %3d\n",
            telem, val, curnum);
#endif

    return val;

 }  /* end of goesi_vcurve */
```

```
/* ***** goesi_veclipse.c
 *
 * PURPOSE:
 *       determines values for eclipse monitoring based on solar panel current
 *
 * RETURN VALUE:
 *
 * ALGORITHM:
 *      This function only works for Goes-8 through Goes-M.
 *
 * ASSUMPTIONS:
 *       The current generated by the solar cells is divided between the
 *       solar panel current (sent over the yoke) and the control buss
 *       current (disapated by resistors on the solar array (to radiate
 *       away excess available powere.)  Other small (leakage, instrument)
 *       currents are ignored, but the results are probably accurate to
 *       a per cent or two.
 *
 * INPUT:
 *
 * OUTPUT:
 *
 * RETURN VALUE:
 *
 * PROTOTYPE:
 *    int  goesi_veclipse(struct goesi_raw fr, int sat_id, float *cur_cell)
 *
 * INFORMATION:
 *       Nominal average solar cell current is 29 or 30 amperes.
 *       If the XRS unit is automatically turned off during eclipse, by
 *       the detection of less than 5 amperes of solar array current, it
 *       is turned on 140 seconds after the solar array current reaches
 *       7 amperes (autoload group 6).
 *
 *   The XRS is typically turned on about 120 s (2 minutes) after the
 *
 * CALLS:
 *
 * CALLED BY:
 *    print_xrs.c
 *
 * INFORMATION:
 *  Spacecraft documentation numbers bits from 1 to 8, 1 being the most
 *  significant bit, 8 the least significant bit
 *   wd 2   subframe id 0-31
 *   wd 3   if bit 8 == 1, dwell, ignore
 *          bits 1-4 are satellite ident
 *     71   battery 1 charge current curve 136
 *     73   primary s/c bus voltage  curve 130
 *     74   primary s/c bus current  curve 131
 *     75   solar array current      curve 133
 *     76   control bus current      curve 132
 *     79   battery 2 charge current curve 140
 *  Note: 75 conv + 76 conv is total solar cell current
 *
 *  REFERENCE:
 *   For information about eclipses with the previous generation of GOES,
 *   see Bornmann, P. L. and Matheson, L. 1990, Astronomy and Astrophysics,
 *   231, 525-535, "Solar Flare Plasma Properties Derived from the
 *   Disk-Integrating GOES X-Ray Sensors during an Eclipse."
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modification 10/26/99 by P.L. Bornmann
 *     extracted the current conversions from eclipse.c
 *   Modification 11/2/99 by P.L. Bornmann
 *     mf_ct was calculated but not used or declared
 *   Modifications 11/18/99 by P.L. Bornmann
 *     continue to clean up.  must supply valid satid, etc.
 *
 *
 * FUTURE MODIFICATIONS:
 *   have eclipse.c use this function
 *
 * QUESTIONS:
 *
 */

#include <stdio.h>
#include "goesi_read.h"

/* PRT_LEV: 0 is least print, 2 is most   */
#define PRT_LEV   0

//int  goesi_veclipse(struct goesi_raw fr, float *cur_cell,
//                    int year, int doy, int num_rec_prt)
int  goesi_veclipse(struct goesi_raw fr, int sat_id, float *cur_cell)
  {
    static long actual_prt = 0L;
    static int  prv_sat_id = -1;
    static int  t_satid    = -1;
    static int  t_satidx   = -1;   /*  if >= 0, index into *_co arrays   */
    static int  start      = -1;
    static long n_cur      = -1;

    //int i;
    int fr_chek;
    int ret = 0;      /* The value to be returned by this routine.  */
    int sat_idx;      /* value from sat_indx for this satellite */
    int sat_pid;      /* used, but is it needed?  holds value of t_satid*/

    /* This is an array that matches the unique bit patters that
     * identify the spacecrafts.  The unusual sequence is due to
     * the Hamming distance (or equivalent) used to avoid single-bit
     * errors causing misidentifications */
    /* Lorne Matheson says:  Some programmers start at 0, some start at 1.
     * Some engineers shift right and some shift left.  The range bits in
     * the telemetry have the least significant bit in the most significant
     * bit.  The table is my way of changing the bit order of a two-bit range word. */

    char sat_str[16][5] =   {{"UNK"  }, {"UNK"}, {" GL"}, {"UNK"},
           {"UNK"  }, {"UNK"}, {" G9"}, {"UNK"},          {"UNK"}, {" G8"},
           {"G10"  }, {"UNK"},          {"UNK"}, {" GM"}, {"UNK"}, {"UNK"}};

    int  sat_indx[16] =   {-1, -1,  3, -1,   -1, -1,  1, -1,
                           -1,  0,  2, -1,   -1,  4, -1, -1  };
    /*   Coefficients for spacecraft main currents for five GOES satellites,
     *   Goes-8 through Goes-N.  GOES-N filled in with GOES-8 values
     *   until available */
    /*   coefficients for curve 131  primary bus current   word 74   */
    /* Curve 131 range 0 to 30 amps */
    static const float pbc_co[5][2] =   {{2.754870e-01, 1.020744e-01  },
          {1.76586e-01,  1.02990e-01  },   {-7.15659e-02, 1.02369e-01},
          {-2.74036e-01, 1.02801e-01  },   {2.754870e-01, 1.020744e-01}};

    /*   coefficients for curve 133  solar arrray current  word 75   */
    /* curve 133 range 0 to 30 amps */
    static const float sac_co[5][2] =   {{2.448892e-01, 1.003481e-01  },
          {-2.42757e-01, 1.01458e-01  },   {9.831199e-02, 1.00072e-01},
```

```
        {-2.01376e-01, 1.00436e-01  },  {2.448892e-01, 1.003481e-01}};

    /*   coefficients for curve 132  control bus current   word 76   */
    /* curve 132 range 0 to 25 amps */
    static const float cbc_co[5][2] =   {{3.907000e-01, 7.926784e-02  },
          {3.19468e-01,  7.91152e-02  },   {1.94570e-01,  8.07924e-02},
          {1.94570e-01,  8.07924e-02  },   {3.907000e-01, 7.926784e-02}};

    /* currents for the solar array, control bus, and primary bus */
    float arr_cu,
          con_bu,
          pri_bu;
    float cur_sol_cell;

    /*   initialize  */
    if (start != 0)
      { /* First call to routine, start = -1 */
        n_cur = 0L;  /* Initialize number of points processed so far */
        start = 0;
      } /* end of initializations for first call to this routine */
    /* increment the number processed */
    n_cur++;

    /* check whether the data was considered to be ok */
    fr_chek = fr.chek & 0x6f;  /* checks for all unfixed errors (allows 0x10) */

    /* ************************
     **  NEED SPACECRAFT ID **
     *************************/

    /* Get the telemetered spacecraft ID */
    t_satid = fr.data[3] >> 4;  /* bit shift*/

    /* Check the satelite ID */
    if ((sat_id != t_satid) && (fr_chek == 0))
    {  /* Got a change in the spacecrart ID */
#if PRT_LEV > 0
        (void) printf ("goesi_veclipse: satellite id "
            "changes from %2d to %2d  %s, sat_indx is %1d\n",
            sat_id, t_satid, sat_str[t_satid],
            sat_indx[t_satid]);
#endif
        if (sat_id < 0)
         { /* had a bad sat_id, but frame gave a new value */
            sat_pid = t_satid;
            sat_idx = sat_indx[sat_pid];
         }
        else
         {    sat_id = t_satid;
            if (t_satid == sat_pid)
            { /* Valid satelite ID.  Get spacecraft index.*/
                sat_idx = sat_indx[sat_pid];
            }
            else
            {  /* sat_id != t_satid */
                sat_idx = -1;
            }
         }
    }  /* end of sat_id test */

    /* ****************************
     **  CALCULATE THE CURRENTS  **
     ****************************/

    /*  Calculate the currents. */

    if ((t_satidx >= 0) && (fr_chek == 0))
      { /*   good data in this frame   */

        /*   use satellite dependant primary bus current coefficients   */
        pri_bu = pbc_co[t_satidx][0] + ((float) fr.data[74]) *
                                       pbc_co[t_satidx][1];

        /* x  use satellite dependant solar array current coefficients   */
        arr_cu = sac_co[t_satidx][0] + ((float) fr.data[75]) *
                                       sac_co[t_satidx][1];

        /*   use satellite dependant control buss current coefficients   */
        con_bu = cbc_co[t_satidx][0] + ((float) fr.data[76]) *
                                        cbc_co[t_satidx][1];

        /* Calculate the solar array current. */
        /* Solar Cell current is sum of solar array current and s/c bus current */
        cur_sol_cell = arr_cu + con_bu;

        /* Will pass back the current */
        *cur_cell = cur_sol_cell;

      }  /* End of current calculations, started at the if ((t_satidx >= 0) && (fr_chek
== 0)) */


#if PRT_LEV > 0
    if (n_cur < 5)
      {
          {
            (void) printf ("goesi_veclipse:  ecl cur"
                "%7.3f %7.3f %7.3f, %7.3f\n",
                pri_bu, arr_cu, con_bu, cur_sol_cell);
          }
        (void) printf("goesi_veclipse: return is %2d, cur_cell is %6.3f\n", ret,
*cur_cell);
        n_cur++;
      }
#endif

    return ret;
  }  /* end of goesi_veclipse */
/*===========================================================================*/
```

```
/* ***** *    *
 *  goesi_wild_times.c
 *  (part of goesI_file_read)
 *
 * PURPOSE:
 *  check a one-second record for negative or very large frames
 *      most, > 99 %, detected so far are the result of
 *          the failures of tar, pax programs
 *
 *      arry[i].date, .time are dms time in days where day 1 is
 *          1900 Jan 01 and seconds of the day, usually less than
 *          86400.  Leap seconds not handled in ingest correctly.
 *
 * INPUTS:
 *   rec_inp     only used for print statements
 *   dar
 *   arry
 *   full_prt
 *
 * OUTPUT:
 *
 * RETURN VALUE:
 *   ret       set to 1 if bad values found
 *
 * ALGORITHM:
 *   checks for seconds in excess of a year, negative times and dates,
 *   and day numbers in excess of year approx 2012 AD
 *
 * CALLS:
 *   (nothing)
 *
 * CALLED BY:
 *    main (goesI_xrs_list)
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *
 *  *    *    */

#include  <stdio.h>
#include "goesi_read.h"

#define LARGE_DAY  41000L      /*   change about 2012 AD   */
#define LARGE_TIME 87000L      /*   more than seconds in day   */


int   goesi_wild_times (
    long rec_inp,              /*   input record number   */
    struct goesi_raw *arry,    /*   one or two unpacked frames   */
    struct counters  *dar,     /*   structure of status counters  */
    int full_prt)              /*   full_print control, 0 no, 1 full print  */
{

    int ret = 0;


    /*   check for very large times   */
    if ((arry[0].time > LARGE_TIME) || (arry[1].time > LARGE_TIME))
    {
         dar->sec_large++;
         if ((dar->sec_large == 1L) || (full_prt))
         {
             (void) printf ("        record %5ld had LARGE TIME of "
                 "%6ld, %6ld ignore rec\n",
                 rec_inp, arry[0].time, arry[1].time);
         }
         ret = 1;
    }

    /*   check for dms times < 0 and ignore   */
    else if ((arry[0].time < 0L) || (arry[1].time < 0L))
    {
        dar->sec_neg++;
        if ((dar->sec_neg == 1) || (full_prt))
        {
            (void) printf ("        record %5ld had NEG TIME of %6ld, "
                " %6ld ignore\n", rec_inp, arry[0].time, arry[1].time);
        }
        ret = 1;
    }


    /*   check for dms day numbers < -1 and ignore   */
    else if ((arry[0].date < -1L) || (arry[1].date < -1L))
    {
        dar->day_neg++;
        if ((dar->day_neg == 1L) || (full_prt))
        {
            (void) printf ("        record %5ld had NEG DATE of %6ld, "
                " %6ld ignore\n", rec_inp, arry[0].date, arry[1].date);
        }
        ret = 1;
    }

    /*   check for very large day numbers
     *       keep this test last, since LARGE_DAY will need to be
     *       adjusted during 2012 anno Domini
     */

    else if ((arry[0].date > LARGE_DAY) || (arry[1].date > LARGE_DAY))
    {
        dar->day_large++;
        if ((dar->day_large == 1L) || (full_prt))
        {
            (void) printf ("        record %5ld had LARGE DATE of "
                "%6ld, %6ld ignore rec\n",
                rec_inp, arry[0].date, arry[1].date);
        }
        ret = 1;
    }


   return ret;
} /* end of goesi_file_read */
```

```
/* ***** goesI_xrs_list.c
 *                                                    Apr 1997, ldm
 *
 * PURPOSE:
 *   This is the MAIN program for reading GOES I-M data from
 *   SEC archive CDs.
 *   a template for reading daily goesI raw telemtry data files,
 *   unpacking and checking them.  The program does not assume
 *   the identity of the spacecraft and can be used with any
 *   of the Goes-8, -9. -10 though -M satellites.
 *
 *  USAGE:
 *   executable_name filename
 *   executable_name filename, starttime
 *   executable_name filename, starttime, npfprint
 *   where starttime is given as hhmmss
 *
 *
 *  PROTOTYPE:  (goesI_xrs_list)
 *     int main (int argc, char *argv[])
 *
 *  INPUTS:
 *    filename  name of file to be processed, or name of file containing
 *              a list of file names to be processed.  Filename must start
 *              with F for a listing file. (see files_raw.c)
 *    starttime is the start time in units of hhmmss
 *    npfprint  number of pf (??) frames to be printed (for diagnostics?)
 *              this is roughly the number of records that will be printed
 *              (but eclipse processing...)
 *
 *    Note:   at least one argument is required, to pass to file open
 *            if second argument present, program will list some of
 *            the record time gaps
 *
 * ALGORITHM:
 *       The term record refers to a dms raw data record of 288 bytes,
 *       the record usually contains two time tagged telemetry frames
 *       of 144 bytes each, 128 bytes from the spacecraft and 16 bytes of
 *       time and other information.  The second half of the record is
 *       sometimes all zeroes, in which case it should be ignored.
 *
 * DEVELOPMENT HISTORY
 *       97 Jul 30  add logic for large and negative dms day numbers  ldm
 *
 *       Sep 99  changed goesI_sem_list.c to goesI_xrs_list.c for Pat B
 *
 *       October 99 PLB adding comments, etc.
 *
 *  CALLS:
 *     files_raw        to open raw data file
 *     goesi_unpk       returns number of non-zero bytes in the record
 *                      and fills ar_i variable with goes information such as
 *                      date, time, sat, and QNX info  (see goesIread.h)
 *     goesi_wild_times tests for illegal dates and times
 *     goesi_28s        determines if 28s errors were introduced by DMS computer
 *     goesi_frame_chk
 *     print_xrs        does all the instrument-specific processing
 *     goesi_hrminsec
 *
 * OTHER CALLS (in sellib libary and c's time lib, not LM's code):
 *     clock
 *     gmtime
 *     dmsyrda
 *     cdtdmst
 *
 * QUESTIONS:
 *    handling of time jumps when times remain the same?  Are times in msec?
 *
 * MODIFICATION HISTORY:
 *   Developed by Lorne Matheson
 *   Comments and Documentation added by Pat Bornmann October 1999
 *   Modification 10/28/99 by P.L. Bornmann
 *     Replaced multiple calculations of hr, min, sec with goesi_hrminsec
 *   Modification 10/29/99 by P.L. Bornmann
 *     changed multiple calculations of time to call to goesi_hrminsec
 *     and changed print statements accordingly
 *     last_time_r was not initialized before it was used.
 *   Modification 10/30/99 by P.L. Bornmann
 *     changed default start time from 1200L (12 minutes) to 0L, per
 *     Lorne statement that this was arbitrary start time.
 *   Modification 02/07/00 by P.L. Bornmann
 *     More records to print, was predefined in declarations
 *
 *   *   *   */

#include  <stdio.h>
#include  <stdlib.h>       /*   atol   */
#include  <time.h>

#include  "goesi_read.h"
#include  "sellib.h"

#define PROG_FREQ 100
#define PRT_LEV  0

int main (int argc, char *argv[])  /* goesI_xrs_list */
 {
    int pr_level;
    FILE *input_p;                 /* pointer to input data file */
    FILE *output_p;                /* pointer to output file */

    struct goesi_raw arry[2];      /* will hold data for the two
                                    * frames in the data record */
    struct counters dar;

    struct tm *now;                /* computer's time at start of processing */
    double dif_tim;
    clock_t clk_start, clk_end;    /*   for processor time used   */
    time_t time_start, time_end;   /*   for clock time used   */

    int i;
    int hour, min, sec;   /* hour, min, and second read from record   */
    int dow;              /* day of week, 0 is Sunday   */
    int doy;              /* day of year, 1 to 366   */
    int year;
    char str_time[13] = "";

    int end_file = 0;     /* if set to 1, no more records   */
    int file_ok = 0;      /* set to 1 if serious problems in file   */
    int frame_nzb = 200;  /* minumum encountered so far   */

    int full_prt = 0;     /* do we do a full print if 1 or not if 0 */

    int ret      = -1;
    int ret_byte;
    int ret_prt = 0;

    /*   four bit patterns unique to each satellite   */
    unsigned char rec_i [ CHAR_PER_REC ] ;   /* contains the 288 binary
                                    * bytes of telemetry information */
```

```
    char file_stat[2][20] = {{" understand file" },{" LOOK AT FILE"}};
    char day_ch    [40]    = "";
    char back_time[40]     = "";

    long gap_sec = 0;
    /*   flag for zeroed seconds frame in record   */
    long dms_offset;              /*   dms date of last Dec 31   */
    long last_date     = -1L;     /*   dms date of last valid frame   */
    long last_date_r   = -1L;     /* DMS date from last record */
    long last_time     = -1L;     /*   DMS date from last record   */
    long last_time_r   =  0;      /*   DMS time from last record   */
    long max_time_jmp  =  0L;     /*   longest record time jump so far   */
    long min_time_jmp  =  1L;     /*   largest negative time jump so far */
    long num_back_ti   =  0L;     /*   total number of records with
                                   *   backward time jumps */
    long num_bad_fr    =  0L;
    long num_corr_fr   =  0L;
    long num_date_ch   =  0L;
    //long num_rec_prt  = 60L;     /*   Number of records to print.  Routine returns
    long num_rec_prt   ;     /*   Number of records to print.  Routine returns
                                   *   -1 if fewer records are printed */
    long num_sus_tim   =  0L;     /*   number of suspect timing frames   */
    long num_time_jmps =  0L;     /*   total number of records with forward time jumps */
    long num_zero_fr   =  0L;
    long num_28s       =  0;      /*   total number of records with DMS-induced
                                   *   28-second errors */
    long rec_inp       =  0L;     /* number of current record in current data file */
    long rec_inp_tot   =  0L;     /* total number of records processed in all files */
    long sec_run;
    long start_sec     =  0L;     /*  starting seconds for print, to skip data */
                                  /*  start_sec arbitrary, so PLB reset it to zero */
    long time_jmp_sum  =  0L;     /*  total time having positive time jumps */
    long time_jmp_neg  =  0L;     /*  total time having negative time jumps */

    long pr_prog = 0L;            /* whether system should print progress */

    /* names and pointers for the input and output data files */
#define MC_FILENAME 200
    char infilename [MC_FILENAME] = "";
    char outfilename[MC_FILENAME] = "";

    /* ************************/
    /**  Initialize Values  **/
    /* ************************/

    /* The level of output prints */
    pr_level = PRT_LEV;

    input_p  = NULL;                /* will be a file descriptor provided */
    output_p = NULL;                /* when open these files via files_raw.c */

    { int i;
      for (i=0; i< CHAR_PER_REC; i++) rec_i[i]=0 ;
    }

    clk_start = clock();
    time_start = time(NULL);
    now = gmtime(&time_start);

    /* Initialize the dar time structure to zero */
    dar.day_large = dar.day_neg = dar.sec_large = dar.sec_neg = 0L;

    /* ****************************************/
    /* Check for properly call to this routine */
    /* ****************************************/
    if (argc < 2)
      { (void) printf ("\nERROR (goesi_xrs_list): no raw data input file given as
agument, QUIT\n");
        (void) printf ("\n");
        (void) printf ("  Usage is one of the following\n");
        (void) printf ("    goesi_xrs_list filename start_sec\n");
        (void) printf ("    goesi_xrs_list filename start_sec "
                       "number_of_frames_to_print\n");
        (void) printf ("  NOTE: use name of executable for goesi_xrs_list: \n"
                       "  %s\n", argv[0]);
        exit (EXIT_SUCCESS);
      }
    if (pr_level > 1) (void) printf ("goesi_xrs_list: try to open %s\n", argv[1]);

    /* ******************
     ** Get the file  **
     ******************/
    ret = files_raw (argv[1], &input_p, &output_p, infilename, outfilename);
    { /* Test length of filename */
      if ( (strlen(infilename ) > MC_FILENAME) ||
           (strlen(outfilename) > MC_FILENAME) )
       { printf ("goesi_xrs_list.c: allocated space insufficient to report file names\n"
                 " allocated %ld but needed %ld for input file and %ld for output
file\n",
                 MC_FILENAME, strlen(infilename), strlen(outfilename) );
       }
     }

    (void) printf ("goes_xrs_list.c: Input data from file: %s\nOutput data saved to
file:%s\n",
                   infilename, outfilename);
    if (ret != 0)
     { exit (EXIT_SUCCESS);
     }

    /*   get starting seconds of day   */
    if (argc > 2)
      { start_sec = atol (argv[2]);
        start_sec = (start_sec/10000L) * 3600L +
            ((start_sec/100L) % 100) * 60L + start_sec % 100;
      }
    /*   optionally set number pf frames (records) to print   */
    if (argc > 3)
      { num_rec_prt = atol(argv[3]);
      }
    else
      { /* One day's worth of 0.5 sec would be 2*60*60*24 = 172,800 */
        num_rec_prt = 180000L;
      }
    (void) printf ("\nProgram %s\n"
                   "    ran %d/%2.2d/%2.2d at %2.2d:%2.2d:%2.2d UTC\n"
                   "    input file is %s\n"
                   "    start sec is %5ld\n"
                   "    will print %3ld records\n",
                   argv[0],
                   now->tm_year + 1900, now->tm_mon + 1, now->tm_mday,
                   now->tm_hour, now->tm_min, now->tm_sec, argv[1],
                   start_sec, num_rec_prt);
    printf ("goesi_xrs_list.c: Input data from file: %s\nOutput data saved to file:%s\n",
            infilename, outfilename);
    if (argc > 3)
      full_prt = 1;
    if (input_p == NULL)
      { (void) printf ("goesi_xrs_list.c: prog input_p is NULL, quit/n");
        exit (EXIT_SUCCESS);
```

```
    }
/*  (void) fclose(stdout); */

    /* ****************************
     ** START LOOP OVER ALL DATA **
     ****************************/
    if (pr_level > 1) (void) printf ("goesi_xrs_list.c: start loop over all data \n");
    /*    forever loop, no limits  */
    for (;;)
     {
        rec_inp++;
        rec_inp_tot++;
        if (pr_level > 2) (void) printf ("goesi_xrs_list.c: record %ld, total %ld "
                          "progress frequency %ld\n",
                          rec_inp, rec_inp_tot, PROG_FREQ);

        if (rec_inp_tot < 10)
          {  pr_prog = 0;
          }
        else
          { //pr_prog = rec_inp_tot % 100;
            pr_prog = rec_inp_tot % PROG_FREQ;
          }

        if ((pr_prog == 0) && (pr_level > 0)) (void) printf ("goesi_xrs_list.c: record
%ld, total %ld "
                          "pr_prog = %ld\n",
                          rec_inp, rec_inp_tot,  pr_prog);

        /* ********************
         ** READ SOME DATA **
         ********************/
        /*   do not use   CHAR_PER_REC, 1,
         *       see The Standard C Library, Plauger, p 271   */
        ret = fread (rec_i, 1, CHAR_PER_REC, input_p);  /* read the 288 bytes */
        if ((pr_level > 2) &&  (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: read the
record\n");

        /*   test for eof before checking length   */
        if (feof(input_p))
         { (void) printf ("goesi_xrs_list.c: attempt to read record %ld is e o f,"
                " quit\n", rec_inp);
           /* ********************
            ** Get next file  **
            *********************/
           ret = files_raw (argv[1], &input_p, &output_p, infilename, outfilename);
           (void) printf ("goesi_xrs_list.c: Input data from file: %s\nOutput data saved
to file:%s\n",
                          infilename, outfilename);
          if (ret == 0)
           { rec_inp = 0L;
             continue;
           }
          else
           { end_file = 1;
             break;
           }
         } /* End of file */

        /*  Verify number of bytes read  */
        if (ret != CHAR_PER_REC)
         { (void) printf ("goesi_xrs_list.c: return from fread IS NOT %d, was %d char,
QUIT\n",
                          CHAR_PER_REC, ret);
          if (file_ok == 0) file_ok =1;
           break;
         }

        /* ********************
         ** UNPACK THE DATA **
         *********************/
        /*   unpack the record in bytes into two frames with ints, etc
         *   return is number of non-zero bytes in second frame   */
        ret_byte = goesi_unpk (rec_i, arry);  /* ret_byte is number of
                                               * non-zero bytes found */

        if (pr_level > 0)
          (void) printf ("goesi_xrs_list.c: frames to examine (36,37,52,72,100,101): "
                   "%4d %4d %4d %4d %4d %4d\n"
                   "                                                                  "
                   "%4d %4d %4d %4d %4d %4d\n",
                  arry[0].data[36], arry[0].data[37], arry[0].data[52],
                  arry[0].data[72], arry[0].data[100], arry[0].data[101],
                  arry[1].data[36], arry[1].data[37], arry[1].data[52],
                  arry[1].data[72], arry[1].data[111], arry[1].data[111]);

        if ((pr_level > 2) &&  (pr_prog == 0)) printf ("goesi_xrs_list.c: unpacked the
data\n");

        if ((ret_byte != 0) && (ret < frame_nzb))
          { (void) printf ("        record %5ld.1 has %2d non-zero "
                 "bytes\n", rec_inp, ret);
            frame_nzb = ret_byte;
          }

        /* is the dms date of the first frame -1   */
        if (arry[0].date == -1L)
         { if (full_prt)
             { (void) printf ("        record %ld had dmsdate of %ld "
                    "%ld, 2nd frame %ld, %ld, quit\n",
                    rec_inp, arry[0].date, arry[0].time,
                    arry[1].date, arry[1].time);
             }
           else
             { (void) printf ("        record %ld had dmsdate of %ld, "
                    "quit\n", rec_inp, arry[0].date);
             }
           ret = files_raw (argv[1], &input_p, &output_p, infilename, outfilename);
           printf ("goesi_xrslist.c: Input data from file: %s\nOutput data saved to
file:%s\n",
                    infilename, outfilename);
           if (ret == 0)
            { rec_inp = 0L;
            }
           else
            { end_file = 1;
              break;
            }
         } /* if arry[0].date */

        /* *************************
         ** CHECK FOR WILD TIMES **
         *************************/
        /*   check for very large, negative  times, dates   */
        ret = goesi_wild_times (rec_inp, arry, &dar, full_prt);
        if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: checked
for wild times\n");
        if (ret != 0)  /* bad values found */
         { if (file_ok == 0)  /* Why test?  Are there other values
                              * besides 0,1 for file_ok?*/
```

```
            file_ok = 1;
                /* Note that this looks like might not be structured as desired */
          continue;
        }  /* no else statement */

       /* ************************
        ** CHECK FOR TIME JUMPS **
        ************************/
       /*   check first times of records for time jumps   */
       if (last_date_r >= 0)  /* set to -1L when declared, changes during
                               * forever loop */
        {  /* have valid date for second frame in record */
          if ((last_date_r == arry[0].date) ||
              (last_date_r +1 == arry[0].date))
           { gap_sec = (arry[0].date - last_date_r)*86400L +
                 arry[0].time - last_time_r;

             if (gap_sec != 1L) /* gap_sec 0 or 1 for valid data */
              { (void) goesi_hrminsec (arry[0], &hour, &min, &sec, str_time);
                if (pr_level > 2) (void) printf ("goesi_xrs_list.c: str_time = %s\n",
str_time);
                gap_sec = gap_sec -1L;  /* gap_set shifted for this case to give */
                                        /* -1 or 0 for valid data */

                if (gap_sec > 0L)  /* more than 1 sec time jump */
                 {  if (gap_sec > max_time_jmp)  /* set to 0L in initialization */
                     {  /* update statistic on the maximum jump encountered */
                        if (full_prt)
                         { (void) printf ("              %2.2d:%2.2d:"
                                "%2.2d.%3.3d   rec %5ld "
                                " gap_sec is %2ld \n", hour, min,
                                sec, arry[0].milli, rec_inp, gap_sec);
                         }
                        max_time_jmp = gap_sec;  /* max jump may now be larger */
                     } /* end of analysis of max jump size (gap_sec > max_time_jmp)
*/
                    num_time_jmps++;
                    time_jmp_sum = time_jmp_sum + gap_sec;  /* total amount of lost
time */
                 } /* end of positive time gap */

                else  /* gap_sec 0 or 1 for valid data */

              {  /* valid data or negative time gap */

                   if (gap_sec < min_time_jmp)  /* initialized to 1L */
                    { /* could have gap of 0 and still be valid data */
                      if ((num_back_ti == 0L) || (full_prt))
                       { (void) printf ("        %2.2d:%2.2d:%2.2d, "
                              "rec %5ld, GAP_SEC IS NEGATIVE %2ld\n",
                               hour, min, sec, rec_inp, gap_sec);
                       }
                      min_time_jmp = gap_sec;  /* will set min_time_jmp */
                                               /* to 0 at some time */
                    } /* End of update on if min_time_jmp */
                   num_back_ti++;  /* increment the number of times time step is */
                                   /* 0 sec or negative */
                   time_jmp_neg = time_jmp_neg + gap_sec -1L;
                   if (file_ok == 0) file_ok = 1;
                 } /* else */
              } /* end of gap_sec */
          }  /* end of if last_date_r change is 0 or 1 */
        }  /* end of time jump test (if pos last_date_r); */
       last_date_r = arry[0].date;
       last_time_r = arry[0].time;

       if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: checked
for time jumps\n");


       /* ************************
        ** CHECK FOR DMS ERRORS **
        ************************/
       /*   check for 28 second dms errors and zero second frames   */
       ret = goesi_28s (arry, ret_byte, full_prt);  /* ret_byte is number of non-zero
bytes found by goesi_unpk */
       if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: checked
for DMS errors\n");
       if (ret != 0)
        { /* A 28-sec error was found */
          num_28s++;
          continue;
        }

       /* ************************
        ** PROCESS BOTH FRAMES **
        ************************/

      /*  loop on two possible frames per record   */
       for (i=0; i < FRAMES_PER_REC; i++)
        {  /*   ret_byte is number of non-zero bytes in second frame,
            *   occasionally normal for good data (1 of 19 seconds)  */
           /*   these occur when the 0.512 sec XRS data aliases with the 1 sec record
interval */
           if ((ret_byte == 0) && ( i == 1))
            {  num_zero_fr++;
               continue;
            }
           if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c:
process DMS frame %d\n", i);

           /*   check for backward time jumps   */
           if (last_date >= 0)
            { if ((last_date > arry[i].date) || ((last_date == arry[i].date)
                  && (last_time > arry[i].time)))
                { num_back_ti++;    /* increment count of number of backward times
encountered */
                  if (file_ok == 0) file_ok =1;
                }
            }
           if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c:
checked for backward time jump\n");

           /*   check to see how many day changes   */
           if (arry[i].date != last_date)
            { const char mon_s[13][4] =    {{"xxx"},{"Jan"},{"Feb"},
                    {"Mar" },{"Apr"},{"May"},{"Jun"},{"Jul"},{"Aug"},
                    {"Sep" },{"Oct"},{"Nov"},{"Dec"}};
              const char str_day[7][5] =   {{"Sun"},{"Mon"},{"Tue"},
                    {"Wed" },{"Thu"},{"Fri"},{"Sat"}};
              int mon, dom, hour, min, sec;   /* time, not telem data time */

              dow = (int) (arry[i].date % 7L);
              if (pr_level > 1) (void) printf ("goesi_xrs_list.c: dow = %d\n", dow);

              (void) dmsyrda (arry[i].date, 1, &year, &dms_offset);
              (void) cdtdmst (&arry[i].date, &year, &mon, &dom,
                   &hour, &min, &sec);
              if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c:
got DMS date \n");
              doy = (int) (arry[i].date - dms_offset);
```

```
            if (last_date >= 0) /* 0L for first pass */
             { if (num_date_ch < 5L)
                 {  /* print the first 5 date changes */
                    if (pr_prog == 0) (void) printf ("goesi_xrs_list.c: date
change\n");
                    (void) printf ("rec %6ld, date %4d, %s %2d\n",
                                   rec_inp, year, mon_s[mon], dom);
                    (void) printf ("        %2.2d:%2.2d:%2.2d, rec %6ld,"
                        " date changes to %4d %s %2d (%3.3d, %s)\n",
                        hour, min, sec, rec_inp, year, mon_s[mon], dom,
                        doy, str_day[dow]);
                 }
                num_date_ch++;
                if (num_date_ch > 4)
                 { if (file_ok == 0) file_ok = 1;
                 }
              }
             else
              { /* an invalid value for last_date.  negative last_date  */
                (void) printf ("        start data at dms day %5ld, "
                     "time %2ld, %4d %s %2d (%3.3d, %2.2d:%2.2d:%2.2d,) \n",
                     arry[i].date, arry[i].time, year, mon_s[mon],
                     dom, doy, str_day[dow], hour, min, sec);
              }
              last_date = arry[i].date;  /* update the last_date for next frame */
            } /* end of day-change test */
           if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesii_xrs_list.c:
date-change test completed \n");

           last_time = arry[i].time;

           /*   check non-zero check bytes   */
           if (arry[i].chek != 0)
            { /*   accumulate count of suspect timing frames   */
              if ((arry[i].chek & 0x80) != 0)  /* see goesI_read.h for defn */
               {  /* time data is suspect or unknown */
                  num_sus_tim++;
               }

              if ((arry[i].chek & 0x7f) != 0)
               {  /* 0x7f mask catches all bits except the time quality suspect or
unknown*/
                  if ((arry[i].chek & 0x7f) == 0x10)
                   {  /* Case of fixed check sum error */
                      num_corr_fr++;
                   }
                  else
                   { num_bad_fr++;  /* will not include case of 0x30, unfixed multi-
error checksum */
                      continue;
                   }
              }
            } /* end of check using chek */
           if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesii_xrs_list.c: chek
test completed \n");

           /* ***************************
            ** CHECK FOR FRAME ERRORS **
            ***************************/
           /*   check additional details of each frame for additional
            *   checking   */
           if ((pr_level > 2) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: will
check for frame errors\n");

           ret = goesi_frame_chk (arry[i], rec_inp, i);

           if ((pr_level > 2) &&  (pr_prog == 0)) (void) printf ("goesi_xrs_list.c:
checked for frame errors\n");
           file_ok = file_ok | ret;
           if (ret != 0)
            { /*  temporary for GoesI_stat_list   */
              if (pr_level > 2) (void) printf(" return from goesi_frame_chk is %2d\n",
ret);
              continue;
            }

           /* ********************
            ** PRINT THE DATA **
            ********************/
           /* Now print the XRS data */
           if ((pr_level > 1) && (pr_prog == 0)) (void) printf ("goesi_xrs_list.c: will
print XRS data\n");
           ret_prt = print_xrs (&output_p, arry[i], year, doy, start_sec, num_rec_prt,
                                end_file);
           if ((pr_level > 1) &&  (pr_prog == 0)) (void) printf ("goesi_xrs_list.c:
printed XRS data\n");

           /* End of infinite loop if print_xrs so indicates */
           if (ret_prt != 0) printf ("goesi_xrs_list.c: Either data to process or
exceeded limit of %d records",
                             num_rec_prt);
           if (ret_prt != 0)
              break;
         }  /* end of loop over the two frames per DMS data record */
       /* ********************************
        ** END OF LOOP OVER BOTH FRAMES **
        ********************************/

       if (ret_prt != 0)
           break;
       if (rec_inp < REC_PRT || rec_inp < 10)
        {  (void) printf ("goesi_xrs_list.c: record %ld\n", rec_inp);
           (void) printf ("goesi_xrs_list.c: synch words frame 0,1 are "
               "%2x %2x %2x and %2x %2x %2x\n",
               arry[0].data[125], arry[0].data[126], arry[0].data[127],
               arry[1].data[125], arry[1].data[126], arry[1].data[127]);
        }

    }   /* end of infinite for loop */
   /* ****************************************
    ** END OF LOOP READING ALL DATA RECORDS  **
    ****************************************/
   /* ***************************
    ** SUMMARIZE TEST RESULTS **
    ***************************/

//    hour = (int) (last_time / 3600L);
//    min  = (int) ((last_time - 3600L *
//           (long) hour) / 60L);
//    sec  = (int) (last_time - 3600L *
//                                 (long) hour - 60L * (long) min);
   (void) goesi_hrminsec (arry[0], &hour, &min, &sec, str_time);
   (void) printf ("        last time is dms day %ld %s, "
      "last time %ld\n", last_date, str_time, last_time);
   (void) printf ("    Frames: %2ld zero; %2ld bad, dwell or text;"
      " %2ld corrected, %2ld time suspect\n",
      num_zero_fr, num_bad_fr, num_corr_fr, num_sus_tim);

   if (num_date_ch == 0)
      (void) sprintf (day_ch, "no date changes");
   else if (num_date_ch == 1)
```

```
        (void) sprintf (day_ch, "one date change");
    else
     { (void) sprintf (day_ch, "%ld DATE CHANGES", num_date_ch);
     }
    if (num_back_ti == 0L)
         (void) sprintf (back_time, "time is forwards");
    else
     { (void) sprintf (back_time, "TIME JUMPS BACK %ld", num_back_ti);
     }
    (void) printf ("    there are %2ld 28_sec(dms) bad records, %s, %s"
        "\n", num_28s, day_ch, back_time);
    (void) printf ("    there are %2ld record gaps, total %ld seconds, "
        "max gap of %ld seconds\n",
         num_time_jmps, time_jmp_sum, max_time_jmp);

    /*   optional print line for negative, large day, time numbers   */
    if ((dar.day_neg > 0L) || (dar.day_large > 0L) || (dar.sec_neg > 0L)
        || (dar.sec_large > 0L))
     { (void) printf ("    records:  %2ld large TIME, %ld NEGative TIME,"
            " %ld NEG DAY, %ld LARge DAY\n",
            dar.sec_large, dar.sec_neg, dar.day_neg, dar.day_large);
     }

    /*   optional print line in negative time jumps   */
    if (num_back_ti > 0)
     { (void) printf("    there are %2ld NEGATIVE TIME GAPS, total %2ld"
                    " seconds, max neg gap of %2ld seconds\n",
                    num_back_ti, time_jmp_neg, min_time_jmp);
     }
    if (file_ok != 0)
     { file_ok = 1;
     }
    clk_end  = clock();
    time_end = time(NULL);
    sec_run  = (long) ((clk_end - clk_start)/ CLOCKS_PER_SEC);
    dif_tim  = difftime(time_end, time_start);
    (void) printf ("    processor time %2ld seconds, clock %3.0lf,"
        "%s\n",
        sec_run, dif_tim, file_stat[file_ok]);

    (void) printf ("goesi_xrs_list.c: end of main\n");
    return 0;

}  /* end of main */
```

```
/* ***** print_xrs.c
 *
 * PURPOSE:
 *   This function, and the functions it calls,
 *   does all of the spacecraft and instrument specific tasks.
 *
 * INPUT:
 *    fr  the goesi_raw structure containing all the telemetry data
 *    year (passed to eclipse and goesi_subcom routines)
 *    doy  (passed to eclipse and goesi_subcom routines)
 *    start_sec  the start time for printing data, in seconds since 0 UT.
 *
 * OUTPUT:
 *    values printed to stdio (to screen) and to file
 *    status values in output will be set to
 *        0 if no data in that subcom,
 *        1 if data should be routine solar obs (or slew is north)
 *        2 if data differs from routine obs (e.g. cal, sle/w) (or slew is south)
 *    Note that these differ from the telemetry.
 *    status values will be provided as single digits without spaces
 *    order will be [0] xrs on/off, [1] xrs sun/nosun
 *      [2] xrs calibration off/on, [3] xrs slew off/on,
 *      [4] xrs slew direction north/south
 *   NOTE:  Subcommed data may have changed during one of the previous 32 data frames.
 *   The following items are subcommed: xrs_stat, sas_pos, preamp_temp, xrs_volt.
 *
 * PROTOTYPE:
 *   int  print_xrs (struct goesi_raw fr, int year, int doy, long start_sec,
 *                   int num_rec_prt, int end_file, char *fname)
 *
 * ALGORITHM:
 *    Reports changes in some subcommed data.
 *    Extracts data from telemetry.
 *    Calls goesi_subcom to extracted subcommed data.
 *
 * CALLS:
 *   eclipse
 *   goesi_subcom  to print information about subcommed data
 *                 (temperatures, voltages, status, etc.)
 *   goesi_sattde
 *   goesi_hrminsec
 *   goesi_vcurve  to convert to engineering units
 *
 * CALLED BY:
 *   main (goesI_xrs_list)
 *
 * MODIFICATION HISTORY:
 *   Developed June 1999 by Lorne Matheson
 *      Originally part of the template (goesI_file_read) program and
 *      its derivatives.
 *  Modifications Oct 99 by P.L. Bornmann
 *      Added comments and documentation
 *  Modifications 10/18/99 by P.L. Bornmann
 *      send standard results to a file
 *  Modifications 10/22/99 by P.L. Bornmann
 *      changed construction of output.  Send to string first.
 *      only convert times once.
 *      added xrs status as output, instead of the changing w52.
 *  Modifications 11/19/99 P.L. Bornmann
 *    Will write to file the actual telemetry values, not the engineering values
 *    Added preprocessor code to make output either telemetered data or
 *    engineering values
 *  Modification 11/22/99 by P.L. Bornmann
 *    Added the currents used to calculate the solar array current
 *    Added code to extract the solar array status (stepping, tracking)
 *    Added code for engineering-unit conversion of solar array position
 *  Modification 1/10/00 by P.L. Bornmann
 *    Added print to stdio of what will be sent to the outputfile
 *  Modifications 1/11/00 by P.L. Bornmann
 *    Cleaned up formats for printing telemetry, engineering values not fixedx
 *  Modified 2/7/00 by P.L. Bornmann
 *    Replaced sprintf with strcat's to terminate the output string
 *    Removed solar cell from tlm output since it is derived quantity
 *  Modified 2/8/00 by P.L. Bornmann
 *    Moved code for SATTDE to separate file
 *
 * TO DO:
 *    PLB will want to trap word 52 bit 8 for XRP changes and for sun not present.
 *    PLB will want to print to a file (change printf's to fprintf's)
 *    Check word 51 bit encodings for Goes 10.  They changed from earlier s/c.
 *    which telem data to ignore during dwell? (word 3)
 *
 * FUTURE MODS:
 *    Extra SADA step for GOES-M (word 45)
 *    only use strlength needed for str_engvals
 *
 * QUESTIONS:
 *    Are these subcommed? word 36, 45
 *    Calibration curve for XRS flux (words 56,57)
 *
 * ALGORITHM DETAILS:
 *   wd 2   subframe counter (values 0 to 31)
 *   wd 3   if bit 8 == 1 telem is in dwell mode and should be ignored
 *          gives info on satid (see code)  No data is valid if dwel is on.
 *   wd 5   xrs long and short range, bits shuffled (see code)
 *     36   is subcommed and not implemented, it contains sattde 2 information
 *          for case when sattde1 is not operational(see word 51, sattde 1 is
 *          baseline, not 2) TT forward  = 0 slew sa opp normal or TT reverse = 1
 *     45   bit 1  (GOES M only) 1 = added SADA (solar array drive assembly) step
 *              2  (GOES-M only) 1 = subtracted SADA step
 *   The following values are repeated twice in each subcom cycle:
 *  Digital status from word 51 is not subcommed, but changes
 *  infrequently.  The longest gap is 8 frames or 4 seconds.  If
 *  we get to using this data, we might need to add it to the
 *  words put out every frame
 *     51  bit  1  sattde 1 (solar array trim tab electronics) on = 1, off = 0
 *              2  sattde 2 on = 1, off = 0
 *              3  sattde 1 slew sa in normal tracking direction or
 *                 TT in forward direction = 0, slew sa opp normal or TT reverse = 1
 *              4  sada will not process slew pulses = 1, will slew = 0
 *              5  solar-array stepping, single step = 0, double step = 1
 *              6  double-step profile (time delays before second step),
 *                 profile 1 = 1, profile 2 = 0
 *              7  what electronics by sattde 1 (solar array trim tab electronics)
 *                 is driven, 1 is solar array, 0 is trim tab motor
 *     52  is subcommed, contains, xrpe slew/track, slew direction north/south,
 *                       xrs sun not present/present,
 *                       xrs cal on/off, xrs on/off
 *    56   short wavelength solar flux
 *    57   long  wavelength solar flux
 *    59   Solar Array Position  (curve 3)
 *    60   Solar Array Position from other potentiometer.  (Use at times when
 *         first one passes its dead spots.)  (curve 4)
 *    72   is subcommed, contains xrs reference voltage
 *                       xrs -75 volt ion chamber bias
 *  For solar array current (cur_sol_cell = arr_cu + con_bu)
 *     74   primary buss current
 *     75   solar array current reading
 *     76   control buss current
 *    100  is subcommed, contains xrs coarse position (N-S),
```

```
 *                      xrs analog sun sense (N-S), and
 *                      xrs analog sun sense (E-W)
 *    101   is subommed, contains xrs preamp temperature
 *
 * NOTE: the values in parenthesis after the subframe number (16 or 32)
 *       indicate the frequency of the value in subcommed words.  Values
 *       of 16 are repeated twice in a 32-element cycle.
 *       *         *         */

#include <stdio.h>
#include <stdlib.h>

#include "goesi_read.h"

/* Temporary for testing */
#include <string.h>

/* Will print telemetered values to file if PR_TELEM == 1 */
#define PR_TELEM 1
#define PRT_LEV 0

/* Frequency for output of progress reports
 * number of records between reports */
#define PROG_FREQ  10
```

```
/* ***** print_xrs    */
int  print_xrs (FILE **fp_output, struct goesi_raw fr, int year, int doy, long start_sec,
    int num_rec_prt, int end_file)
 {
    static long actual_prt = 0L;
    static long prog_cnt = 0L;  /* Count of records processed */
    long pr_prog  = -1;         /* Flag of whether print this line as progress */
    int  pr_level = -1;         /* whether to print, holds PRT_LEV */
    int ret       =  0;         /* the return value */

    static long st_date;   /* the start date when routine first called */
    int hr = -1, min = -1, sec = -1;

    int mf_ct   = -1;      /*  main frame counter, 0 to 31   */
    int mf_ct16 = -1;      /*  main frame counter, mod 16, 0 to 15   */

    /* Variables to determine if telemetered values change */
    static int sat_id   = -1;
    static int xrs_on   = -1;
    static int xrs_cal  = -1;
    static int xrp_slew = -1;

    static int rng_chng[4] =  {0,2,1,3};

    int bit8     = -1;
    int ecl_flag = -1;
    int ra_short = -1;     /*  bit that identifies a short-range change */
    int ra_long  = -1;     /*  bit that identifies a long-range change */
    int t_sat_id = -1;     /*  the s/c ID read from the current telemetry */

    const int sat_ind[16] =  {0,0,3,0, 0,0,1,0, 0,0,2,0, 0,4,0,0 };
    float sol_cell_cur = -1.0;

    const char  cal_str[2][8]  =  { {"(cal) "}, {"(data)" }  };
    const char   on_str[2][7]  =  { {"(on) " }, {"(off)"  }  };
    const char slew_str[2][9]  =  { {"(slew)"}, {"(track)"} };

    /* sat_str used to bit match the actual bitstring used to represent the
     * various spacecraft.  The "wierd" order is the uniqueness pattern of
     * the bits (to maximize the Hamming distance so single bit flip is not
     * interpreted as a different spacecraft */
    const int sat_num[16]     =  { -1, -1, 11, -1,
                                   -1, -1,  9, -1,
                                   -1,  8, 10, -1,
                                   -1, 12, -1, -1};
    const char sat_str[16][5] =  { {"UNK"}, {"UNK"}, {" GL"}, {"UNK"},
                                   {"UNK"}, {"UNK"}, {" G9"}, {"UNK"},
                                   {"UNK"}, {" G8"}, {"G10"}, {"UNK"},
                                   {"UNK"}, {" GM"}, {"UNK"}, {"UNK"}};


    /* These strings are used for output.  The length is determined by goesi_subcom and
     * checked against allocated length when goesi_subcom returns. */
#define MCHARSUBCOM  300
    long subcom_strlen = -1;                /* Length of string returned by
goesi_subcom.c */
    static char strh_engvals [MCHARSUBCOM] = ""; /* string for column headings */
    static char strh_tlmvals [MCHARSUBCOM] = ""  /* string for column headings of string
of
                                            telemetered values */;
    char str_engvals    [MCHARSUBCOM] = ""; /* string of values in engineering units */
    char str_tlmvals    [MCHARSUBCOM] = ""; /* string of values in telemetry (data) units
*/

    /* Format codes */
    static char fmt_tlmdat  [MCHARSUBCOM] = "";
    static char fmt_engdat  [MCHARSUBCOM] = "";

    const char fmt_tlmtime  [MCHARSUBCOM] = "%2.2d %4.4d %3.3d %12.12s ";  /* sat_id,
year, month, day, str_time */
    const char fmt_engtime  [MCHARSUBCOM] = "%2.2d %4.4d %3.3d %12.12s ";  /* sat_id,
year, month, day, str_time */
    const char fmt_tlmsubcom[MCHARSUBCOM] = "%2.2d %1.1d ";                /* mf_ct,
tlmdwell */
    const char fmt_engsubcom[MCHARSUBCOM] = "%2.2d %1.1d ";                /* mf_ct,
tlmdwell */
    const char fmt_tlmflux  [MCHARSUBCOM] = "%3.3d %3.3d %3.3d %3.3d ";    /* range, flux
*/
    const char fmt_engflux  [MCHARSUBCOM] = "%1.1d %3.3d %1.1d %3.3d ";    /* range, flux
*/
    const char fmt_tlmpos   [MCHARSUBCOM] = "%3.3d %3.3d %8s ";            /* sada_posa,
sada_posb, str_engsattdel */
    const char fmt_engpos   [MCHARSUBCOM] = "%5.2f %5.2f %8s ";            /* sada_posa,
sada_posb, str_engsattdel */
    const char fmt_tlmecl   [MCHARSUBCOM] = "%1.1d %3.3d %3.3d %3.3d ";         /*
ecl_flag, ary_cur, primbuss, contrlbuss */
    const char fmt_engecl   [MCHARSUBCOM] = "%1.1d %5.2f %5.2f %5.2f %5.2f ";   /*
ecl_flag, sol_cell_cur, ary_cur, primbuss, contrlbuss*/
    const char fmt_tlmother [MCHARSUBCOM] = "%3.3d %s";         /* fr.data[51],
str_tlmvals*/
    const char fmt_engother [MCHARSUBCOM] = "%3.3d %s";         /* fr.data[51],
str_tlmvals/

    /* Formats for Header */
    /* These are time, subcom; flux, range; others; */
    const char fmth_tlmdat [MCHARSUBCOM]  = "%2.2s %4.4s %3.3s %12.12s "
                                            "%2.2s %1.1s "
                                            "%3.3s %3.3s %3.3s %3.3s "
                                            "%3.3s %3.3s %8.8s "
                                            "%1.1s %3.3s %3.3s %3.3s "
                                            "%3.3s %s\0";
    const char fmth_engdat [MCHARSUBCOM]  = "%2.2s %4.4s %3.3s %12.12s "
                                            "%2.2s %1.1s "
                                            "%1.1s %3.3s %1.1s %3.3s "
                                            "%5.5s %5.5s %8.8s "
                                            "%1.1s %5.5s %5.5s %5.5s %5.5s "
                                            "%3.3s %s\0";

    /* variables introduced by PLB */
#define MCHARVALS  300
    static char strh_engdat[MCHARVALS]  = "";  /* string of column headings for output
data */
    static char strh_tlmdat[MCHARVALS]  = "";  /* string of column headings for output
data */
    char str_engdat [MCHARVALS]  = "";  /* string containing the XRS output data */
    char str_tlmdat [MCHARVALS]  = "";  /* string containing the XRS output data */
    char str_time       [13]     = "";  /* 12 char time (hh:mm:ss.msc) plus end of str \0
*/
    char str_tlmtime    [13]     = "";  /* 12 char time (hh mm ss msc) plus end of str \0
*/
    char str_engtime    [13]     = "";  /* 12 char time (hh:mm:ss:msc) plus end of str \0
*/
    static char strh_tlmtime   [13]     = "";  /* time notation, used for column headings
*/
    static char strh_engtime   [13]     = "";  /* time notation, used for column headings
*/

    /* Poisition information */
    int   sada_dposa = -1, sada_dposb = -1;
    float sada_posa  = -1, sada_posb  = -1;
```

```
    /* Solar array currents */
    int   ary_dcur = -1, primbuss_dcur = -1, cntrlbuss_dcur = -1;
    float ary_cur  = -1, primbuss_cur  = -1, cntrlbuss_cur  = -1;

    /* For the Solar Array and Trim Tab Drive Electronics */
    char str_engsattde1[16]; char str_engsattde2[16];
    static char strh_engsattde1[16]; static char strh_engsattde2[16];
    char str_tlmsattde1[16]; char str_tlmsattde2[16];
    static char strh_tlmsattde1[16]; static char strh_tlmsattde2[16];

    int  tlmdwell  = -1;

    /* *************************/
    /**  End of Declarations  **/
    /* *************************/


    pr_level = PRT_LEV;

    /*   initialize  */
    if (sat_id < 0)
     { /* occurs first time routine is called */
        sat_id = fr.data[3] >> 4;  /* bit shift */
        st_date = fr.date;

       /* Time lables */
      (void) sprintf (strh_tlmtime, "%2s %2s %2s %3s",
                      "hr", "mi", "s", "msc");
      (void) sprintf (strh_engtime, "%2s:%2s:%2s.%3s",
                      "hr", "mi", "s", "msc");

//(void) printf ("print_xrs.c: strh_tlmtime = %s\n", strh_tlmtime);
//(void) printf ("print_xrs.c: strh_engtime = %s\n", strh_engtime);

        /* Call once to start, to get the header lables */
        subcom_strlen = goesi_subcom (fr, strh_engvals, str_engvals, strh_tlmvals,
str_tlmvals);
//(void) printf ("print_xrs.c: strh_tlmvals = %s\n", strh_tlmvals);
//(void) printf ("print_xrs.c:  str_tlmvals = %s\n",  str_tlmvals);
//(void) printf ("print_xrs.c: strh_engvals = %s\n", strh_engvals);
//(void) printf ("print_xrs.c:  str_engvals = %s\n",  str_engvals);
        goesi_sattde (fr, str_engsattde1, str_engsattde2, strh_engsattde1,
strh_engsattde2,
                      str_tlmsattde1, str_tlmsattde2, strh_tlmsattde1,
strh_tlmsattde2 );
//(void) printf ("print_xrs.c: strh_tlmsattde1 = %s\n", strh_tlmsattde1);
//(void) printf ("print_xrs.c:  str_tlmsattde1 = %s\n", str_tlmsattde1);

        /* Format for data output */
        strcpy(fmt_engdat, "");
        strcat(fmt_engdat, fmt_engtime);
        strcat(fmt_engdat, fmt_tlmsubcom);
        strcat(fmt_engdat, fmt_engflux);
        strcat(fmt_engdat, fmt_engpos);
        strcat(fmt_engdat, fmt_engecl);
        strcat(fmt_engdat, fmt_engother);

        strcpy(fmt_tlmdat, "");
        strcat(fmt_tlmdat, fmt_tlmtime);
        strcat(fmt_tlmdat, fmt_tlmsubcom);
        strcat(fmt_tlmdat, fmt_tlmflux);
        strcat(fmt_tlmdat, fmt_tlmpos);
        strcat(fmt_tlmdat, fmt_tlmecl);
        strcat(fmt_tlmdat, fmt_tlmother);

        (void) sprintf (strh_tlmdat, fmth_tlmdat,
                        "G", "year", "doy", strh_tlmtime, "mf", "dw", "rng", "F_s",
"rng", "F_l",
                        "p1a", "p2a", strh_tlmsattde1,
                        "ec", "arC", "prC", "ctlC",
                        "d51", strh_tlmvals);
         (void) sprintf (strh_engdat, fmth_engdat,
                         "G", "year", "doy", strh_engtime, "mf", "dw",
                         "r", "F_s", "r", "F_l",
                         "p1_a", "p2_a", strh_engsattde1,
                         "ec", "CELcu", "arC", "prC", "ctlC",
                         "d51", strh_engvals);
        if (pr_level > 1)
          { (void) printf ("print_xrs.c:  fmt_tlmdat = %s\n", fmt_tlmdat);
            (void) printf ("print_xrs.c: strh_tlmdat = %s\n", strh_tlmdat);
            (void) printf ("print_xrs.c:  fmt_engdat = %s\n", fmt_engdat);
            (void) printf ("print_xrs.c: strh_engdat = %s\n", strh_engdat);
          }
//exit(EXIT_FAILURE);
     }

    /* Counter of number of calls, used to control progress reports */
    prog_cnt++;
    if (PRT_LEV == 0)
     { pr_prog = 0;
     }
    else
     {  if (prog_cnt < 10)
          { pr_prog = 0;
          }
        else
          { pr_prog = prog_cnt % PROG_FREQ;
          }
     }

    /* Create time ID string */
    (void) goesi_hrminsec(fr, &hr, &min, &sec, str_time);
    (void) sprintf (str_time,  "%s %3.3d", str_time, fr.milli);
    (void) sprintf (str_tlmtime,  "%2.2d %2.2d %2.2d %3.3d", hr, min, sec, fr.milli);
    (void) sprintf (str_engtime,  "%2.2d:%2.2d:%2.2d:%3.3d", hr, min, sec, fr.milli);

    if ((pr_level > 1) && (pr_prog == 0))
       (void) printf ("print_xrs: time is %2.2d:%2.2d:%2.2d.%3.3d \n",
                      hr, min, sec, fr.milli);

    /*   decode some subcoms, etc   */
    t_sat_id = fr.data[3] >> 4;  /* bit shift */
    if (t_sat_id != sat_id)
     { /* Satellite ID changed value when should not */
       (void) printf (" sat_id changes from %2d to %2d, quit/n",
                      sat_id, t_sat_id);
       sat_id = t_sat_id;
     }

    /* *************************
     ** REPORT SUBCOM CHANGES **
     *************************/
    mf_ct   = fr.data[2] >> 3;    /* bit shift of word 2, for subframe counter value */
    mf_ct16 = mf_ct % 16;         /* counter for the values repeated twice per subframe
cycle */
    bit8    = fr.data[52] & 0x1;  /* only need first bit of this subcommed word */

    if ((pr_level > 1) && (pr_prog == 0)) (void) printf ("print_xrs.c: counters %d %d "
                    "now check for subcom changes\n", mf_ct, mf_ct16);
```

```
    switch (mf_ct16+1)
     {  /* check dual cycle values giving XRS status */
        case  8:
          /*   0 is cal, 1 is data   */
          if (bit8 != xrs_cal)
           { /* XRS calibration status changed */
             (void) printf ("                %12s"
                            " %2d  xrs cal/data changes from %1d to %1d %s\n",
                            str_time, mf_ct, xrs_cal, bit8, cal_str[bit8]);
             xrs_cal = bit8;
           }
          break;
        case 14:
          /*   0 is xrs on, 1 is off   */
          if (bit8 != xrs_on)
           { /* XRS on/off status changed */
             (void) printf ("              %12s"
                  " %2d  xrs  on/off  changes from %1d to %1d %s\n",
                  str_time, mf_ct, xrs_on, bit8, on_str[bit8]);
             xrs_on = bit8;
           }
          break;
        case 5:
          /*   0 is xrp slew, 1 is (sun) track */
          if (bit8 != xrp_slew)
           { /* XRP slew status changed */
             (void) printf ("               %12s"
                            " %2d  slew status  changes from %1d to %1d %s\n",
                            str_time, mf_ct, xrp_slew, bit8, slew_str[bit8]);
             xrp_slew = bit8;
           }
          break;
/* PLB: add other tests?? */
        default:
          break;
     }

    /* *********************
     ** TEST FOR ECLIPSE **
     *********************/
    if ((pr_level > 1) && (pr_prog == 0))
      (void) printf ("print_xrs.c: Check for eclipse\n");

    ecl_flag = eclipse (fr, &sol_cell_cur, year, doy, num_rec_prt);

    if ((pr_level > 1) && (pr_prog == 0))
       (void) printf ("print_xrs.c: Will print if time %ld >= %ld or date %ld > %ld\n",
                      fr.time, start_sec, fr.date, st_date);

    /*    do we start printing?   */
    if ((fr.time >= start_sec) || (fr.date > st_date))
     {  /* ************************
           ** PROCESS SUBCOM DATA **
         ************************/
        if ((pr_level > 1) && (pr_prog == 0))
            (void) printf ("print_xrs.c: about to call goesi_subcom\n");

        /* xrs preamp temperature */
        subcom_strlen = goesi_subcom (fr, strh_engvals, str_engvals, strh_tlmvals,
str_tlmvals);
         if (pr_level > 1)
         { (void) printf ("print_xrs.c: strh_tlmvals = %s\n", strh_tlmvals);
           (void) printf ("print_xrs.c: str_tlmvals  = %s\n",  str_tlmvals);
           (void) printf ("print_xrs.c: strh_engvals = %s\n", strh_engvals);
           (void) printf ("print_xrs.c: str_engvals  = %s\n",  str_engvals);
         }
        if (subcom_strlen > MCHARSUBCOM)
          { (void) printf ("print_xrs.c: ERROR.  Insufficient string length declared "
                           " for value returned by goes_i_subcom\n Declared %ld, returned
%ld\n",
                           MCHARSUBCOM, subcom_strlen);
            exit(EXIT_FAILURE);
          }
        if ((pr_level > 1) && (pr_prog == 0))
          { (void) printf ("print_xrs.c: returned from goesi_subcom\n");
          }


        /* Get the range change information */
        ra_long  = (fr.data[5] >> 4) & 0x3;   /* bit shift with mask */
        ra_short = (fr.data[5] & 0xc0) >> 6;  /* bit shift with mask */

        /* Get the solar array position from Channels A and B */
        sada_dposa = fr.data[59];
        sada_dposb = fr.data[60];
        sada_posa = goesi_vcurve(fr.data[59], 3);
        sada_posb = goesi_vcurve(fr.data[60], 4);

        /* telemetry dwell.  (Some values invalid if dwell on) */
        tlmdwell = (fr.data[3] & 0x080) >> 8;

        /* The currents used to calculate the solar array (eclipse) current */
        primbuss_dcur  = fr.data[74];
        ary_dcur       = fr.data[75];
        cntrlbuss_dcur = fr.data[76];
        primbuss_cur   = goesi_vcurve(fr.data[74], 3);
        ary_cur        = goesi_vcurve(fr.data[75], 4);
        cntrlbuss_cur  = goesi_vcurve(fr.data[76], 3);
        if (pr_level > 0) printf ("print_xrs.c: sada_pos %5.2f %5.2f, tlmdwell %1d,
currents %5.2f %5.2f %5.2f\n",
                  sada_posa, sada_posb, tlmdwell, ary_cur, primbuss_cur, cntrlbuss_cur);

        /* ******************************/
        /* THE SADA ELECTRONICS STATUS */
        /* ******************************/
        goesi_sattde (fr,  str_engsattde1,  str_engsattde2,
                          strh_engsattde1, strh_engsattde2,
                           str_tlmsattde1,  str_tlmsattde2,
                          strh_tlmsattde1, strh_tlmsattde2 );

      /* ***********************
       ** Print the XRS data **
       ***********************/

       /* ******************************/
       /* The data header and format  */
       /* ******************************/
       /* Print the column headings */
       if (actual_prt == 0)
        {

          if ((strlen(strh_tlmdat) > MCHARVALS) || (strlen(strh_engdat) > MCHARVALS))
            { printf ("\nprint_xrs.c: ERROR.  String length longer than allocated\n"
                      "                  Allocated %d, but needed %d and %d for strh_engdat
and strh_tlmdat\n",
                      MCHARVALS, strlen(strh_engdat), strlen(strh_tlmdat));
              printf ("print_xrs.c: strh_engdat is \n%s\n", strh_engdat);
              printf ("print_xrs.c: strh_tlmdat is \n%s\n", strh_tlmdat);
              exit(EXIT_FAILURE);
            }
```

```
         /* ************************************************/
         /* Print the header (variable names) to the file */
         /* ************************************************/
         if (PR_TELEM == 1)
           { (void) fprintf (*fp_output, "%s\n", strh_tlmdat);
             (void)  printf ("\nprint_xrs.c:To file %s\n", strh_tlmdat);
           }
         else
           { (void) fprintf (*fp_output, "%s\n", strh_engdat);
             (void)  printf ("\nprint_xrs.c: fprintf x strh_engdat:  %s\n",
strh_engdat);
           }

       }  /* actual print, Strings of header informat ion completed */

       /* Format for data output */
       /* TO DO: ONLY NEED TO CALL THIS ONCE IF MAKE STATIC */
       // (void) sprintf (fmt_engdat,  "%s%s%s%s%s\n",
       //                 fmt_engtime, fmt_tlmsubcom, fmt_engflux, fmt_engpos,
fmt_engother);


        /* Prepare data string for output */
       (void) sprintf (str_tlmdat, fmt_tlmdat,
                       sat_num[sat_id], year, doy, str_tlmtime,
                       mf_ct, tlmdwell,
                       rng_chng[ra_short], fr.data[56], rng_chng[ra_long], fr.data[57],
                       sada_dposa, sada_dposb, str_tlmsattde1,
                       ecl_flag, ary_dcur, primbuss_dcur, cntrlbuss_dcur,
                       fr.data[51], str_tlmvals
                       );

        /* Prepare data string for output */
       (void) sprintf (str_engdat, fmt_engdat,
                       sat_num[sat_id], year, doy, str_time,
                       mf_ct, tlmdwell,
                       rng_chng[ra_short], fr.data[56], rng_chng[ra_long], fr.data[57],
                       sada_posa, sada_posb,  str_engsattde1,
                       ecl_flag, sol_cell_cur, ary_cur, primbuss_cur, cntrlbuss_cur,
                       fr.data[51], str_engvals
                      );
       if (pr_level > 0)
        {  (void) printf ("\n");
           (void) printf ("print_xrs.c: fmt_engdat %s\n", fmt_engdat);
           (void) printf ("print_xrs.c: fprintf    %s\n", str_engdat);
           (void) printf ("print_xrs.c: fmt_tlmdat %s\n", fmt_tlmdat);
           (void) printf ("print_xrs.c: fprintf    %s\n", str_tlmdat);
           (void) printf ("\n");
         }

       /* Check the string lengths for overflow */
       if (strlen(str_engdat) > MCHARVALS || strlen(str_tlmdat) > MCHARVALS)
        { printf ("print_xrs.c: ERROR.  String length longer than allocated\n"
                  "             Allocated %d, but needed %d and %d for str_engdat and
str_tlmdat\n",
                  strlen(str_engdat), strlen(str_tlmdat));
          exit(EXIT_FAILURE);
        }

       if (pr_level > 0)
        { int hlen;
          int dat_len;
          hlen = strlen(str_engdat);
          dat_len = strlen(str_engdat);
         }

       if (pr_level > 0)
          { (void) printf ("\n");
            (void) printf ("print_xrs.c: str_engdat and str_engdat follow \n");
            (void) printf ("%s\n", strh_tlmdat);
            (void) printf ("%s\n", str_tlmdat);
            (void) printf ("%s\n", strh_engdat);
            (void) printf ("%s\n", str_engdat);
            (void) printf ("\n");
          }

        /* ************************************/
        /* Print the actual data to the file */
        /* ************************************/
        if (PR_TELEM == 1)
          { (void) fprintf(*fp_output, "%s\n", str_tlmdat);
            if (pr_prog == 0) (void)  printf("\nprint_xrs.c:To file: %s\n", str_tlmdat);
          }
        else
          { (void) fprintf(*fp_output, "%s\n", str_engdat);
             if (pr_prog == 0) (void)  (void)  printf("\nprint_xrs.c:To file: %s\n",
str_engdat);
           }

        actual_prt++;
        if (actual_prt >= num_rec_prt)
         { /* will return a one if data was actually printed */
           ret = 1;
         }
     }

    /*   are we through printing?   */
    if ((pr_level > 1) && (pr_prog == 0)) (void) printf ("print_xrs.c: return value
%d\n", ret);
//exit(EXIT_FAILURE);
    return ret;

 } /* end of print_xrs */
```

```
PRO calc_xrs_coef, temp, ran_s, ran_l, GSAT = GSAT

;+
; PURPOSE
;   calculate sensitivity and offset as a function of temperature
;   for both channels.  The final form will be of flux = (v-offset)
;   / sensitivity
;
; INPUTS:
;   temp      the xrs instrument/electronics temperature
;   ran_s     the short wavelength ranges
;   ran_l     the long  wavelength ranges
;
; OPTIONAL KEYWORD PARAMETER:
;   GSAT      the GOES satellite number (e.g 8, 9)
;
; RETURN VALUE:
;   xrs_coef[2, 2, ndata];  first index 0-short, 1-long
;                           second      0-volt offset
;                                       1-sensitivity
;   PLB thought should return offset constant and
;   sensitivity multiplier defined from
;     x-ray flux is VX * [1 / (SX * KX)] + [-CX/(SX * KX)]
;   or x-ray flux  is (VX - CX )/(SX * KX)]
;   This routine returns
;      xrs_coef constant = CX, and
;      xrs_coef multiplier = (1/(SX*KX))
;    which can be converted to x-ray flux with
;      x-ray flux = (voltage - xrs_coef constant)/ xrs_coef multiplier
;
; USAGE:
;    calc_xrs_coef, temp, ran_s, ran_l, GSAT = GSAT
;
; CALLED BY:
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;   Developed by L. Matheson
;   Correction Dec 20, 95 by LM
;     error had both s/n 1,2 with old s/n 2
;       kx of 1.69e-5, 4.54e-6 (per revised Panametrics report)
;   Future errors for unlaunched GOES-L noted by PLB 01/07/00.
;   Ported to IDL 3/6/00 by P.L. Bornmann
;     added the values for GOESI XRS to SAS alignment
;   PLB adding comments 5/10/00 trying to understand
;     what is returned
;
; TO DO:
;   not pass back the coefs for each temperature point but
;   pass back the necessary coefs for temp corrections
;
; QUESTIONS:
;
; ALGORITHM
;   from Panametrics calibration book
;     ssx, lsx are short and long channel sensitivities
;     scx, lcx are short and long voltage offsets
;     first set at 25 C, second set at -20 C
;  The x-ray flux is given by
;     Jx = (Vx - Cx)/(Sx - Kx) in W/m2
;     where Jx is the Xray flux, Vx is the chamber voltage,
;     Sx and Cx are constants, and Kx is a conversion factor in
;     Amps m2 / W
;  The chamber bias voltage is given by
;     VB = - Bias Monitor * 20
;  The IFC reference voltage is given by
;     Vifc = - Ref V. Monitr * 2
;
;
;   Includes the temperature correction, using delta_temp
;-

    if (not(keyword_set(GSAT))) then begin
      print, 'calc_xrs_coef.pro: must supply a satellite number as GSAT keyword'
      STOP
      RETURN
    endif

    ; Get the thermal factors SX and CX for the Short and Long
    ; channels (SSX, SCX) and (LSX, LCX).  Also get the
    ; calibration factor (KX) for conversion to x-ray fluxes
    ; The x-ray flux is given by (VX - CX) / (SX * KX), where
    ; VX is the XRS chamber output voltage
    ; Therefore, x-ray flux is VX * [1 / (SX * KX)] + [-CX/(SX * KX)]
    if (GSAT eq 8) then begin ;   s/n 002      F6751
       ; Note: The calibrations were changed in an 11/29/95 report
       ; The ssx, scx, lsx, and lcx agree with the GOES I calibr
       ; handbook -- P. Bornmann 4/7/00
       ; These arrays are of the form [, temperature] where
       ; first line applies at 25 degrees C and the second line is
       ; for -20 degrees C
       ssx =  [ [5.29e11, 5.01e10, 5.14e9, 5.00e8], $
                [5.74e11, 5.34e10, 5.35e9, 5.18e8]]
       scx =  [ [.504,    .501,    .503,   .501], $
                [.514,    .503,    .514,   .502]]
       lsx =  [ [4.73e11, 4.52e10, 4.56e9, 4.45e8], $
                [5.01e11, 4.73e10, 4.75e9, 4.60e8]]
       lcx =  [ [.500,    .501,    .501,   .501], $
                [.516,    .503,    .516,   .503]]
       kx  =  [1.599e-5, 4.163e-6]
       ; NOTE: the GOES I handbook lists 4.54 for the second coef
       ; but this was corrected in the revised Panametrics report

    endif else if (GSAT eq 9) then begin ;   s/n 001      F6747
       ssx =  [ [5.37e11, 5.18e10, 5.23e9, 5.13e8], $
                [5.76e11, 5.45e10, 5.39e9, 5.28e8]]
       scx =  [ [.519,    .503,    .518,   .503], $
                [.525,    .506,    .554,   .507]]
       lsx =  [ [4.75e11, 4.56e10, 4.65e9, 4.57e8], $
                [5.07e11, 4.79e10, 4.81e9, 4.71e8]]
       lcx =  [ [.507,    .501,    .507,   .501], $
                [.539,    .505,    .540,   .505]]
       kx =  [1.618e-5, 3.989e-6]

    endif else if (GSAT eq 10) then begin ;   s/n 003      F6749
       ssx =  [ [5.37e11, 5.09e10, 5.11e9, 5.00e8], $
                [5.69e11, 5.32e10, 5.29e9, 5.16e8]]
       scx =  [ [.499,    .501,    .500,   .501], $
                [.513,    .503,    .512,   .503]]
       lsx =  [ [4.95e11, 4.75e10, 4.65e9, 4.53e8], $
                [5.37e11, 5.03e10, 4.79e9, 4.65e8]]
       lcx =  [ [.508,    .502,    .507,   .501], $
                [.550,    .507,    .541,   .506]]
       kx  =  [1.631e-5, 3.824e-6]

    endif else if (GSAT eq 11) then begin ; These are defaults, not GOES-L/11 (PLB
1/7/00)
       ;   s/n 002      F6751
```

```
      ssx =  [ [5.29e11, 5.01e10, 5.14e9, 5.00e8], $
               [5.74e11, 5.34e10, 5.35e9, 5.18e8]]
      scx =  [ [.504,    .501,    .503,   .501], $
               [.514,    .503,    .514,   .502]]
      lsx =  [ [4.73e11, 4.52e10, 4.56e9, 4.45e8], $
               [5.01e11, 4.73e10, 4.75e9, 4.60e8]]
      lcx =  [ [.500,    .501,    .501,   .501], $
               [.516,    .503,    .516,   .503]]
      kx  =  [1.599e-5, 4.163e-6]
  endif

    ;float delta_temp;               ;   fraction for interpolation
    ;float sens_s, sens_l

    ndata = n_elements(ran_s)

    ; The temperature offset needed for thermal corrections
    delta_temp = (25.0 - temp)/45.

    ; calculate the short coefficients for the present short range
    ; coef is scaled by scx values at ????
    ran_s = ran_s > 0  ; value must be greater than zero
    ran_s = ran_s < 3  ; value must be less than three

    ; x-ray flux is [-CX/(SX * KX)] + VX * [1 / (SX * KX)]
    xrs_coef = fltarr(2,2,ndata) ; short/long, const/mult, temperatures

    ; Short-flux conversion constant  ??? SHOULD DIVIDE BY SX*KX??
    xrs_coef[0, 0, *] = scx[0, ran_s] + delta_temp * (scx[1, ran_s] - $
                        scx[0, ran_s])
    ; Short-flux conversion multiplier
    sens_s = ssx[0, ran_s] + delta_temp * (ssx[1, ran_s] - $
             ssx[0, ran_s])
    xrs_coef[0, 1, *] = 1./(sens_s * kx[0])

    ; calculate the long coefficients for the present long range
    ran_l = ran_l > 0
    ran_l = ran_l < 3

    ; Long-flux conversion constant  ??? SHOULD DIVIDE BY SX*KX??
    xrs_coef[1, 0, *] = lcx[0, ran_l] + delta_temp * (lcx[1, ran_l] - $
                        lcx[0, ran_l])

    ; Long-flux conversion multiplier
    sens_l = lsx[0, ran_l] + delta_temp * (lsx[1, ran_l] - $
             lsx[0, ran_l])
    xrs_coef[1, 1, *] = 1./(sens_l * kx[1])

    ; XRS to SAS alignment is required to be +/- 0.050 degrees
    ; Actual values for GOES I are (in degrees), where
    ; Yaw has a rotation axis about the XRP rotation axis (N/S)
    ; and Pitch is along the SADA rotation axis (E/W)
    ; +0.008 ; E/W Pitch
    ; +0.011 ; E/W Yaw
    ; -0.014 ; N/S Pitch
    ; +0.013 ; N/S Yaw

if (TEMP ne 0) then begin
    print, " calc_xrs_coef, temp %4.1f, ranges %d %d, fract %f\n", $
        temp, ran_s, ran_l, delta_temp
    print, " xrs_coef %f %e, %f %e\n",  xrs_coef[0, 0], $
        sens_s, xrs_coef[1, 0], sens_l
    print, " sens %e %e\n", xrs_coef[0, 1], xrs_coef[1, 1]
endif

RETURN
END
```

```
PRO doit4goesi, plot_all = plot_all

;+
; PURPOSE:
;   Process GOES I-M data that has been read
;   from CD and stored on an ASCII file
;   This is the main program that looks at the
;   GOES I-M data.
;
; INPUTS:
;   (none)
;
; OUTPUTS:
;   (none, just plots and processing)
;
; OPTIONAL KEYWORDS:
;   plot_all
;
; USAGE:
;   doit4goesi
;
; CALLS:
;   rdgoesi
;   parse_goesi  reads the data file and
;                creates structure variables
;   telem2eng
;   plot_struct
;   plot_goesi
;   do_xrscal
;   do_xrpoff
;
; MODIFICATION HISTORY:
;   Developed 2/10/00 by P.L. Bornmann
;   Modified 4/12/00 by P.L. Bornmann
;     to set limit parameters for offset plots
;-

  if (not(keyword_set(plot_all))) then plot_all = 1

  ; Read the data
  dir = 'd:\d_plb\plb_work\c_code\PLB_RdGoes\TestData'
  rdgoesi, data, varnames, filename

  ; these need to be non-zero for parsing to occur
  xrs_tlmstats    = 1
  xrs_tlmpos      = 1
  xrs_tlmsattde   = 1
  xrs_tlmvolts    = 1
  xrs_tlmtemps    = 1
  xrs_tlmeclipse  = 1

  parse_goesi, data, varnames, tlm_fluxes, hours, xrsstat, dwell, $
               xrsstats = xrs_tlmstats , pos     = xrs_tlmpos, $
               sattde   = xrs_tlmsattde, eclipse = xrs_tlmeclipse, $
               volts    = xrs_tlmvolts , temps   = xrs_tlmtemps

  ; Convert to engineering units
  telem2eng, xrs_tlmstats, xrs_tlmpos, xrs_tlmvolts, xrs_tlmtemps, $
             xrs_engstats, xrs_engpos, xrs_engvolts, xrs_engtemps

  xt = 'Time (hrs)'
  print, 'doit4goesi.pro: xt = ', xt
  ;wait_user
  interp_path, filename, dir, file, ext

  if (plot_all ge 1) then begin
    plot_struct, xvalues = hours[32:*], xrs_tlmpos  [32:*], xrs_engpos  [32:*], $
                 title = file, xtitle = xt

    plot_struct, xvalues = hours[32:*], xrs_tlmtemps[32:*], xrs_engtemps[32:*], $
                 title = file, xtitle = xt
    plot_struct, xvalues = hours[32:*], xrs_tlmvolts[32:*], xrs_engvolts[32:*], $
                 title = file, xtitle = xt
    plot_struct, xvalues = hours[32:*], xrs_tlmstats[32:*], xrs_engstats[32:*], $
                 title = file, xtitle = xt

    plot_goesi, tlm_fluxes, hours, xrsstats = xrs_tlmstats, dwell, $
                pos = xrs_tlmpos, sattde = sattde, eclipse =  xrs_tlmeclipse, $
                volts = xrs_tlmvolts, temps = xrs_tlmtemps, title = file + ' Telem',
go_slow = 0


  do_xrscal, tlm_fluxes, hours, xrs_tlmstats, dwell, $
             pos = xrs_tlmpos, sattde = sattde, eclipse =  xrs_tlmeclipse, $
             volts = xrs_tlmvolts, temps = xrs_tlmtemps, title = file, /slew
endif
  ; Make the first limit larger than the second to find "out-of-bounds" points
  oob = [1,-1]
  lim_ewxrp    = oob * 0.75
  lim_nsxrp    = oob * 0.75
  lim_ewcoarse = oob * 0.75
  lim_fxrp     = [0,50]
  do_xrpoff, tlm_fluxes, hours, xrs_engpos, $
             lim_fxrp = lim_fxrp, lim_ewxrp = lim_ewxrp, $
             lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse, $
             title = file

  print, "End of doit4goesi.pro for file ", file

STOP
END
```

```
PRO doit4goesi, plot_all = plot_all

;+
; PURPOSE:
;   Process GOES I-M data that has been read
;   from CD and stored on an ASCII file
;   This is the main program that looks at the
;   GOES I-M data.
;
; INPUTS;
;   (none)
;
; OUTPUTS:
;   (none, just plots and processing)
;
; OPTIONAL KEYWORDS:
;   plot_all
;
; USAGE:
;   doit4goesi
;
; CALLS:
;   rdgoesi
;   parse_goesi  reads the data file and
;                creates structure variables
;   telem2eng
;   plot_struct
;   plot_goesi
;   do_xrscal
;   do_xrpoff
;
; MODIFICATION HISTORY:
;   Developed 2/10/00 by P.L. Bornmann
;   Modified 4/12/00 by P.L. Bornmann
;     to set limit parameters for offset plots
;-

  if (not(keyword_set(plot_all))) then plot_all = 1

  ; Read the data
  dir = 'd:\d_plb\plb_work\c_code\PLB_RdGoes\TestData'
  rdgoesi, data, varnames, filename

  ; these need to be non-zero for parsing to occur
  xrs_tlmstats    = 1
  xrs_tlmpos      = 1
  xrs_tlmsattde   = 1
  xrs_tlmvolts    = 1
  xrs_tlmtemps    = 1
  xrs_tlmeclipse  = 1

  parse_goesi, data, varnames, tlm_fluxes, hours, xrsstat, dwell, $
               xrsstats = xrs_tlmstats , pos     = xrs_tlmpos, $
               sattde   = xrs_tlmsattde, eclipse = xrs_tlmeclipse, $
               volts    = xrs_tlmvolts , temps   = xrs_tlmtemps

  ; Convert to engineering units
  telem2eng, xrs_tlmstats, xrs_tlmpos, xrs_tlmvolts, xrs_tlmtemps, $
             xrs_engstats, xrs_engpos, xrs_engvolts, xrs_engtemps

  xt = 'Time (hrs)'
  print, 'doit4goesi.pro: xt = ', xt
  ;wait_user
  interp_path, filename, dir, file, ext

  if (plot_all ge 1) then begin
    plot_struct, xvalues = hours[32:*], xrs_tlmpos  [32:*], xrs_engpos  [32:*], $
                 title = file, xtitle = xt

    plot_struct, xvalues = hours[32:*], xrs_tlmtemps[32:*], xrs_engtemps[32:*], $
                 title = file, xtitle = xt
    plot_struct, xvalues = hours[32:*], xrs_tlmvolts[32:*], xrs_engvolts[32:*], $
                 title = file, xtitle = xt
    plot_struct, xvalues = hours[32:*], xrs_tlmstats[32:*], xrs_engstats[32:*], $
                 title = file, xtitle = xt

    plot_goesi, tlm_fluxes, hours, xrsstats = xrs_tlmstats, dwell, $
                pos = xrs_tlmpos, sattde = sattde, eclipse =  xrs_tlmeclipse, $
                volts = xrs_tlmvolts, temps = xrs_tlmtemps, title = file + ' Telem',
go_slow = 0


  do_xrscal, tlm_fluxes, hours, xrs_tlmstats, dwell, $
             pos = xrs_tlmpos, sattde = sattde, eclipse =  xrs_tlmeclipse, $
             volts = xrs_tlmvolts, temps = xrs_tlmtemps, title = file, /slew
endif
  ; Make the first limit larger than the second to find "out-of-bounds" points
  oob = [1,-1]
  lim_ewxrp    = oob * 0.75
  lim_nsxrp    = oob * 0.75
  lim_ewcoarse = oob * 0.75
  lim_fxrp     = [0,50]
  do_xrpoff, tlm_fluxes, hours, xrs_engpos, $
             lim_fxrp = lim_fxrp, lim_ewxrp = lim_ewxrp, $
             lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse, $
             title = file

  print, "End of doit4goesi.pro for file ", file

STOP
END
```

```
PRO dopath4rdgoesi, olddir = olddir, oldpath = oldpath, $
                  print_all = print_all

;+
; PURPOSE:
;   changes the working directory and IDL search
;   path to settings needed for doit4goesi.
;
; INPUTS:
;   (none)
;
; OUTPUTS:
;   changes system variable !path and !dir
;
; OPTIONAL OUTPUT PARAMETERS:
;   olddir   returns orgininal working directory
;   oldpath  returns original path
;   print_all prints listing of resulting path
;
; USAGE:
;   dopath4rdgoesi
;   dopath4rdgoesi, olddir = olddir, oldpath = oldpath
;   dopath4rdgoesi, /print_all
;   dopath4rdgoesi, olddir = olddir, oldpath = oldpath, /print_all
;
; CALLS:
;   patharr4lib
;   prpath
;
; CALLED BY:
;   (nothing)
;
; NOTE:
;   to verify the results, you can use function prpath to
;   print the resulting path
;
; MODIFICATION HISTORY:
;   developed 9/11/00 by P.L. Bornmann
;
    ; Desired working directory
    newdir = "d:\d_plb\plb_work\c_code\rdgoes8_10\plb_rdgoes\source\idlcode"

    ; Change the directory
    cd, newdir, current = olddir

    ; New search path.  Note that the path separators
    ; vary with operating system, so make this an array
    patharr4lib, patharr
    newpatharr = [newdir, patharr]
    ; Get the path deliminter
    opsys = strupcase(!version.os_family)
    case opsys of
      "WINDOWS":pathsep = ";"
      "UNIX":   pathsep = ":"
      "MACOS":  pathsep = ","
      "VMS":    pathsep = ","
      else: begin
              print, "dopath4rdgoesi.pro: unknown operating system ", opsys
              pathsep = ","
            end
    endcase

    ; Prepare the path string
    newpath = " "
    for ipath = 0, n_elements(newpatharr)-1 do begin
      newpath = newpath + pathsep + newpatharr[ipath]
    endfor

    ; Update the search path
    oldpath = !path
    !path = !path + newpath

    if (keyword_set(print_all)) then print, ' Working directory was ', oldir, ' now ',
new_dir
    if (keyword_set(print_all)) then prpath


RETURN
END
```

```
PRO do_xrpoff, fluxes, hours, pos, lim_fxrp = lim_fxrp, lim_ewxrp = lim_ewxrp, $
            lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse, title = title

;+
; PURPOSE:
;   Create plots showing the XRS signal as functions of
;   angles to the sun.
;
; INPUTS:
;   fluxes
;   hours
;   pos
;
; OUTPUTS:
;
; OPTIONAL KEYWORD PARAMETERS:
;   lim_fxrp, lim_ewxrp, lim_nsxrp, lim_ewcoarse   limits for the plots
;   title   text to add to plots
;
; USAGE:
;   do_xrpoff, fluxes, hours, pos, lim_ewxrp = lim_ewxrp, $
;             lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse
;
; CALLED BY:
;   doit4goesi.pro
;
; CALLS:
;  get_xrpoff
;  plot_xrsoff
;
; MODIFICATION HISTORY:
;   Developed 2/16/00 by P.L. Bornmann
;   Modified 4/12/00 by P.L. Bornmann
;     to pass limit parameters
;   Modified 4/17/00 by P.L. Bornmann
;      Added lim_fxrp to keyword parameters
;   Modified 9/11/00 by P.L. Bornmann
;      Added plot reporting case when no offpoints occurred.
;      Added keyword parameter title and included it on plots
;-

    if (not keyword_set(title)) then title = ''

   ; Get the offpoint times
   get_xrpoff, pos, flgoffpos, woffall, woffnsxrp, hours = hours, $
             woffewxrp, ewoffewcoarse, nwoffall, lim_ewxrp = lim_ewxrp, $
             lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse

   if (n_elements(woffall) le 0) then begin
     print, 'do_xrpoff.pro:  NO OFFSET POINTINGS FOUND'
     STOP
   endif

   ; Find the start and ends of offpoint times
   for i = 0, 1 do begin
     if (i eq 0) then begin
        woff = woffnsxrp
        offdir = "NS"
     endif else begin
        woff = woffewxrp
        offdir = "EW"
     endelse

     gaps = woff(1:*) - woff
     wwstarts = where(gaps gt 1, nwstarts)
     ntimes = n_elements(pos)
     if (nwstarts ge 1) then begin
       wstarts = woff(wwstarts+1)
       wends   = woff(wwstarts)
       nwoff = n_elements(woff)
       if (woff(0) eq 0 and wstarts(0) ne 0) then wstarts = [0, wstarts]
       if (wends(n_elements(wends)-1) ne woff(nwoff-1)) then $
           wends = [wends, woff(nwoff-1)]
       if (wstarts(0) eq wends(0)) then begin
         ; Eliminate start and end at same point
         STOP
       endif
     endif


   ; WANT TO MERGE LOTS OF CLOSE GAPS INTO A BIG UNIT
   ; WANT TO ELIMINATE TIMES WHEN POINTING JUST SLIGHTLY OFFSET FOR SHORT TIME

     if (i eq 0) then begin ; only do this once
       plot_xrsoff, fluxes, hours, pos, lim_fxrp = lim_fxrp, $
           lim_nsxrp = lim_nsxrp, lim_ewxrp = lim_ewxrp, title = title + "!cAll Times"
     endif

     ; Plot the data
     print, ' There are ', nwstarts, ' sets of ', offdir, ' offpoints'

     if (nwstarts eq 0) then begin
       xr = [min(hours), max(hours)]
       yr = [min(fluxes.shflux),max(fluxes.shflux)]
       plot,xr,yr, /nodata, title = title
       xyouts, 0.5*(xr[0]+xr[1]), 0.5*(yr[0]+yr[1]), "No offpoints occurred", align = 0.5
     endif else begin
       plot_xrsoff, fluxes(woff), hours(woff), pos(woff), $
              lim_fxrp = lim_fxrp, lim_nsxrp = lim_nsxrp, lim_ewxrp = lim_ewxrp, $
              title = title + " All Off-ptg."
       for iset = 0, nwstarts-1 do begin
          print, 'plot_xrpoff for offpointing ', iset, ' of ', nwstarts
          w1 = wstarts(iset)
          w2 = wends  (iset)
          tt = strcompress(title + " " + offdir + '!cOffpointing ' + $
                string(iset) + ' of ' + string(nwstarts))

          plot_xrsoff, fluxes(w1:w2), hours(w1:w2), pos(w1:w2), $
              lim_fxrp = lim_fxrp, lim_nsxrp = lim_nsxrp, lim_ewxrp = lim_ewxrp, $
              title = tt
       endfor
     endelse
   endfor

RETURN
END
```

```
PRO do_xrscal, fluxes, hours, xrsstats, dwell, $
             pos = pos, sattde = sattde, eclipse = eclipse, $
             volts = volts, temps = temps, title = title, $
             slew = slew, cal = cal

;+
; PURPOSE:
;   Extracts the times of calibrations and sends them to plots
;
; INPUTS:
;    fluxes
;    hours
;    xrsstats
;    dwell
;
;  OPTIONAL KEYWORD PARAMETERS:
;    if these are supplied with a non-zero value, then they
;    are extracted and the values are returned as an array
;       pos = pos, sattde = sattde, eclipse = eclipse,
;       volts = volts, temps = temps, title = title,
;
; OPTIONAL KEYWORD CONTROL PARAMETERS
;    slew   if set, process the times in slew mode
;    cal    if set, process the times in calibration mode
;  NOTE: slew and cal are treated independently.  If both are
;  selected, then will repeat the analysis, once for each mode.
;
; OUTPUTS:
;    keyword parameters
;
; USAGE:
;   do_xrscal, fluxes, hours, xrsstats, dwell, $
;              pos = pos, sattde = sattde, eclipse = eclipse, $
;              volts = volts, temps = temps, title = title, slew = slew
;
; CALLED BY:
;   doit4goesi
;
; CALLS:
;   get_xrpslew
;   get_xrscal
;   plot_goesi
;   plot_xrscal
;
; MODIFICATION HISTORY:
;   Developed 2/10/00 by P.L. Bornmann
;   Added explicit keyword for cal sequences
;-

  if (not(keyword_set(slew))) then slew = 0
  if (not(keyword_set(cal))) then cal = 0
  if (not(keyword_set(title))) then title    = ""

  for imode = 0, 1 do begin
    if (imode eq 0) then begin
      if (slew ge 1) then begin
        nwuse = get_xrpslew(xrsstats, wuse)
        title_add = " Slew "
      endif else begin
        ; type of analysis not selected
        goto, next_mode
      endelse
    endif else if (imode eq 1) then begin
      if (cal ge 1) then begin
        nwuse = get_xrscal(xrsstats, nwuse)
        title_add = " Cal "
      endif else begin
        ; type of analysis not selected
        goto, next_mode
      endelse
    endif
    if (nwuse le 0) then RETURN

    ; Get the calibration times

    ; Extract the values at calibration times
    wfluxes = fluxes(wuse)
    whours  = hours (wuse)

    ; Add the time range to the plot titles
    ttext = string(hours(wuse(0)), hours(wuse(nwuse-1)), $
            format = '(f6.2, " to ", f6.2, " hrs")')
    ttext = strcompress(ttext)
    title_add = title_add + ttext
    if (not(keyword_set(xrsstats))) then wxrsstats = 0 $
      else wxrsstats = xrsstats(wuse)
    if (not(keyword_set(dwell   ))) then wdwell    = 0 $
      else wdwell    = dwell   (wuse)
    if (not(keyword_set(pos     ))) then wpos      = 0 $
      else wpos      = pos     (wuse)
    if (not(keyword_set(eclipse ))) then weclipse  = 0 $
      else weclipse  = eclipse (wuse)
    if (not(keyword_set(sattde  ))) then wsattde   = 0 $
      else wsattde   = sattde  (wuse)
    if (not(keyword_set(volts   ))) then wvolts    = 0 $
      else wvolts    = volts   (wuse)
    if (not(keyword_set(temps   ))) then wtemps    = 0 $
      else wtemp     = temps   (wuse)

    ; Send raw data to plots
    plot_goesi,  wfluxes, whours, xrsstats = wxrsstats, dwell, $
                 pos = wpos, sattde = wsattde, eclipse = weclipse, $
                 volts = wvolts, temps = wtemps, $
                 title = title + title_add

    plot_xrscal, wfluxes, whours, wxrsstats, dwell, $
                 pos = wpos, sattde = wsattde, eclipse = weclipse, $
                 volts = wvolts, temps = wtemps, $
                 title = title + title_add

next_mode:
  endfor ; imode

RETURN
END
```

```
PRO gap_extend, flgoffpos, itype, ntimes, woff, nextend = nextend

;+
; PURPOSE:
;   This extends the offpointing flags by the specified amount
;   Used to include points that pass through "good" pointing range
;   during periods when offpointings were occuring
;
; INPUTS:
;   flgoffpos  array to hold the flags indicating times of offpointings
;   itype      first index in flgoffpos, used for multiple types of
;              offpoints (eg. N/S vs E/W) stored in the flgoffpos array
;   woff       indices where offpointings were found
;   nextend    number of indices to extend the offpointing flags
;
; OUTPUTS:
;   flgoffpos  is modified by this code
;
; CALLED BY:
;   get_xrpoff
;
; CALLS:
;   wait_user
;
; MODIFICATION HISTORY:
;   Extracted from get_xrpoff 2/15/00 by P.L. Bornmann
;-

    if (not(keyword_set(nextend))) then nextend = 2*60*60  ; Sixty minutes

    if (n_elements(woff) le 0) then begin
      print, 'gap_extend.pro: No offset indices were received'
      help, /tr
      wait_user , "gap_extend.pro: "
      return
    endif

    ; Set the flag for definite offpointings  - NEEDED HERE??
    flgoffpos(itype, woff   ) = 2

     ; Get the positions where gaps occur
     wgappos = where ((woff(1:*) - woff) gt 1, nwgappos)
     if (nwgappos le 0) then begin
       print, 'gap_extend.pro: no gaps found'
       RETURN
     endif

     ; First position might be part of a set of data,
     ; rather than part of a gap
     if (woff(1) eq woff(0)+1) then begin
       wgappos = [0, wgappos]
       nwgappos = nwgappos + 1
     endif
     ; Add last point as end "gap"
     wgappos = [wgappos, n_elements(woff)-1]
     nwgappos = nwgappos + 1

     ; All data same setting
     if (nwgappos le 0) then begin
       wgappos = [wgappos(0), wgappos(2)]
       nwgappos = nwgappos - 1
       STOP
     endif

     if (nwgappos ge 1) then begin

       for iwgap = 0, nwgappos-1 do begin

         ; Extend flags forward from end of offpointings
         w1 = (woff(wgappos(iwgap))) > 0
         w2 = (woff(wgappos(iwgap)) + nextend) < ntimes-1
         flgoffpos(itype,w1:w2) = 1
         print, ' offpoint flag extended from ', w1, ' to ', w2

          ; Extend flags backward from start of offpointings
         if (iwgap lt nwgappos-1) then begin
           w1 = (woff(wgappos(iwgap+1)) - nextend)>0
           w2 = woff(wgappos(iwgap+1)) < ntimes-1
           flgoffpos(1,w1:w2) = 1
           print, '               extended from ', w1, ' to ', w2
         endif

       endfor
       ; Reset the originals
       flgoffpos(itype, woff) = 2
     endif

return
end
```

```
FUNCTION get_off, pos_val, lim_val, woff

;+
; PURPOSE:
;   Gets the indices when the XRS position indicates
;   offpointings.  This is the generic part.  get_xrpoff handles
;   all three types of offpoint indicators
;
; INPUTS:
;   pos_val
;   lim_val  the limits for the values to be identified
;            if lim_val[0] < lim_val[1] the indices for
;            all points between these two limits are returned.
;            if lim_val[0] > lim_val[1] the indices for
;            "out-of-bounds" points are returned
;
; OUTPUTS:
;   woff     the indices to the offset points
;
; RETURNS:
;   the number of offset points
;
; USAGE:
;   noffsets = get_off (pos_val, lim_val, woff)
;
; CALLED BY:
;   do_xrpoff.pro
;
; CALLS:
;   (nothing)
;
; ALGORITHM:
;   if lim_val is a vector, then
;     woff = where(pos_val ge lim_val[0] and $
;                  pos_val le lim_val[1] , nwoff)
;   if lim_val is a scalar then
;      woff    = where(abs(pos_val) ge lim_val[0], nwoff)
;
; MODIFICATION HISTORY:
;  Extracted from get_xrpoff 4/12/00 by P.L. Bornmann
;    and modified to make this generic code
;  Modified 9/11/00 by P.L. Bornmann
;    Corrected where test and print statement for case when points not in range
;-

   medval = median(pos_val)

   ; Default threshold uses points that are beyond threshold of the median
   offthr = 10
   if (not(keyword_set(lim_val))) then lim_val = medval + [-1,1]*offthr

   ; Identify points beyond thresholds
   if (n_elements(lim_val) eq 2) then begin
     if (lim_val[0] le lim_val[1]) then begin
       woff    = where(pos_val ge lim_val[0] and $
                       pos_val le lim_val[1] , nwoff)
       print, 'get_off.pro: Found ', nwoff, ' points between ', lim_val
     endif else begin
       woff    = where(pos_val ge lim_val[0] or $
                       pos_val le lim_val[1], nwoff)
       print, 'get_off.pro: Found ', nwoff, ' points not in range of ', lim_val
     endelse
   endif else begin
     woff    = where(abs(pos_val) ge lim_val[0], nwoff)
     print, 'get_off.pro: Found ', nwoff, ' points within +/-', lim_val
     endelse

   ; End of get_off
   RETURN, nwoff
   END
```

```
PRO get_xrpoff, pos, flgoffpos, woffall, woffnsxrp, hours = hours, $
                woffewxrp, ewoffewcoarse, nwoffall, lim_ewxrp = lim_ewxrp, $
                lim_nsxrp = lim_nsxrp, lim_ewcoarse = lim_ewcoarse

;+
; PURPOSE:
;   Gets the indices when the XRS position indicates
;   offpointings
;
; INPUTS:
;   pos
;
; INPUT KEYWORD PARAMETERS:
;   hours
;
; OUTPUTS:
;   flgoffpos  an array indicating whether point is
;              offpoint (dir offpointed, ntimes)
;   woffall    indices where offpoint occured in any direction
;   woffnsxrp  indices where offpointed in N/S direction
;
; CALLED BY:
;   do_xrsoff.pro
;
; CALLS:
;   get_off
;   gap_extend
;   no_strays
;
; MODIFICATION HISTORY:
;  Developed 2/14/00 by P.L. Bornmann
;  Modified 4/12/00 by P.L. Bornmann
;     added limit parameters
;     changed to PRO to allow keyword parameters
;     extracted generic code to get_off
;     corrected call to no_strays
;  Modification 4/17/00 by P.L. Bornmann
;     added hours to inputs for plots of offpoint times
;-

  !p.multi = 0
  !p.multi(1) = 1
  !p.multi(2) = 4
  yt = ["All Pos ", "XRP N/S Pos", "XRP E/W Pos", "Coarse E/W Pos"]

   if (not(keyword_set(hours))) then begin
      hours = indgen(n_elements(pos))
      xthours = 'Data Number'
   endif else begin
      xthours = 'Time (hrs)'
   endelse

   ; Array of flags whether pointing is on or off target
   ntimes = n_elements(pos)
   flgoffpos = intarr(4, ntimes)

   ; get the offset indices
   print, 'get_xrpoff.pro: about to call get_off with ', $
          'lim_nsxrp = ', lim_nsxrp, 'lim_ewxrp = ', lim_ewxrp, $
          'lim_ewcoarse = ', lim_ewcoarse
   nwoffnsxrp    = get_off(pos.nsxrp   , lim_nsxrp   , woffnsxrp   )
   nwoffewxrp    = get_off(pos.ewxrp   , lim_ewxrp   , woffewxrp   )
   nwoffewcoarse = get_off(pos.ewcoarse, lim_ewcoarse, woffewcoarse)

   ; Remove isolated offpoints
   if (nwoffnsxrp    ge 1) then flgoffpos(1,woffnsxrp) = 2
   if (nwoffewxrp    ge 1) then flgoffpos(2,woffewxrp) = 2
   if (nwoffewcoarse ge 1) then flgoffpos(3,woffewcoarse) = 2
   flgoffpos(0,*) = flgoffpos(1,*) + flgoffpos(2,*) + flgoffpos(3,*)

   ; Remove isolated offpoints
   s = size(flgoffpos)
   for itype = 0, s(1)-1 do begin
     no_strays, flgoffpos, itype, ntimes, woff, keepthr = keepthr
   endfor

   ; Extend the offpoints
   if (nwoffnsxrp  ge 1) then begin
     itype = 1
     gap_extend, flgoffpos, itype, ntimes, woffnsxrp
     flgoffpos(itype, woffnsxrp   ) = 2
     plot, hours, flgoffpos(itype,*), yrange = [-1,3], $
           title = yt[itype], xtitle = xthours, ytitle = 'Flag'
   endif

   if (nwoffewxrp ge 1) then begin
     itype = 2
     flgoffpos(itype,woffewxrp) = 2
     gap_extend, flgoffpos, itype, ntimes, woffewxrp  ; this failed, woff was -1
     flgoffpos(woffnsxrp   ) = 2
     plot, hours, flgoffpos(itype,*), yrange = [-1,3], title = yt[itype], $
           xtitle = xthours, ytitle = 'Pos'
   endif

   if (nwoffewcoarse ge 1) then begin
     itype = 3
     flgoffpos(itype,woffewcoarse) = 2
     gap_extend, flgoffpos, itype, ntimes, woffewcoarse
     flgoffpos(itype,woffewcoarse) = 2
     plot, hours, flgoffpos(itype,*), yrange = [-1,3], title =  yt[itype], $
           xtitle = xthours, ytitle = 'Pos'

   endif

   ; Merge all offpoints
   flgoffpos(0,*) = flgoffpos(1,*) + flgoffpos(2,*) + flgoffpos(3,*)
   flgoffpos = flgoffpos < 1
   woffall = where(flgoffpos(0,*) ge 1, nwoffall)

   ; Plot the offpoints
   plot, hours, flgoffpos(0,*), yrange = [-1,3], title = 'All Offpoints', $
     xtitle = xthours, ytitle = 'Pos'

; End of get_xrpoff
RETURN
END
```

```
FUNCTION get_xrpslew, xrsstats, wslew

;+
; PURPOSE:
;   Gets the indices for when XRP was in slew mode
;
; INPUTS:
;    xrsstats
;
; OUTPUTS:
;   wslew   the indices of the slew points
;
; RETURN VALUE:
;   number of slew points
;
; USAGE:
;  nslewpts = get_xrpslew (xrsstats, wslew)
;
; CALLED BY:
;
; CALLS:
;   (nothing)
;
; RETURN VALUE
;   returns number of calibration points
;
; MODIFICATION HISTORY:
;  Developed 2/14/00 by P.L. Bornmann
;-

; Look for calibration times
wslew = where(xrsstats.slew eq 2, nwslew)
if (nwslew le 0) then begin
  print, 'get_xrpslew.pro:  NO XRS SLEW INTERVALS FOUND'
  plot, [0,n_elements(xrsstats)>1],[0,255], /nodata
  xyouts, 0.5*n_elements(xrsstats)>1, 128, "No Slew Times", align = 0.5
  RETURN, -1
endif

RETURN, nwslew
END
```

```
FUNCTION get_xrscal, xrsstats, wcal

;+
; PURPOSE:
;   Gets the indices for XRS calibration times
;
; INPUTS:
;   xrsstats
;
; OUTPUTS:
;   wcal   indices of points during XRS calibrations
;
; RETURN VALUE:
;   returns number of calibration points
;
; USAGE:
;  ncalpts = get_xrscal (xrsstats, wcal)
;
; CALLED BY:
;   do_xrscal.pro
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;  Developed 2/10/00 by P.L. Bornmann
;-

; Look for calibration times
wcal = where(xrsstats.cal eq 2, nwcal)
if (nwcal le 0) then begin
  print, 'plotxrscals.pro:  NO XRS CALIBRATION INTERVALS FOUND'
  plot, [0,n_elements(xrsstats)>1],[0,255], /nodata
  xyouts, 0.5*n_elements(xrsstats)>1, 128, "No Cal Data", align = 0.5
  STOP
  RETURN, -1
endif

RETURN, nwcal
END
```

```
FUNCTION goesi_vcurve, telem, curnum, plot_all = plot_all

;+
; PURPOSE:
;   used to get and convert some subcommed temperatures, voltages,
;   angles, etc for goesI_xrs_list, based on the conversion curve
;   to convert telemetry to real values.
;
; INPUT:
;   telem         the telemetry value from fr.data
;   curnum        the conversion curve number:
;                     003 for SADA output position Channel A
;                           (word 59, 0 to 360 degrees)
;                     004 for SADA output position Channel B
;                           (word 60, 0 to 360 degrees)
;                     220 for tlm calib voltage
;                     233 for temperatures  (-40 to 70 degrees C)
;                     378 for xrs reference voltage (0 to -12 volts)
;                     379 for xrs coarse position  (30 to -100 degrees)
;                     380 for SAS (Sun Angle Sensor) N/S (-2 to 2 degrees)
;                     381 for SAS (Sun Angle Sensor) E/W (-2 to 2 degrees)
;                     382 for xrs -70 volt bias  (0 to -120 volts)
;
; OPTIONAL KEYWORD CONTROL PARAMETER
;     plot_all   controls how many plots are produced
;
; OUTPUT:
;   c_word101     a character string of information from telem word 101
;
; USAGE:
;   values = goesi_vcurve(telem, curnum, plot_all = plot_all)
;
; RETURN VALUE:
;   the telemetry value converted to engineering (scientific) units
;
; PROTOTYPE:
;
; CALLED BY:
;   goesi_subcom
;
; CALLS:
;   (nothing)
;
; INCLUDE FILES:
;   "goesi_read.h"
;
; ALGORITHM:
;   Assumes all spacecraft (GOESI-M = GOES8-10) have same conversion
;   factors.  Uses supplied telemetry value with specified conversion
;   curve number to return value in engineering units.
;
; MODIFICATION HISTORY:
;       started Sep 99, ldm (Lorne Matheson)
;   Comments and Documentation added by Pat Bornmann October 1999
;   Modified 10/24/99 by P.L.Bornmann
;     to extract additional subcom data
;   Extracted from goesi_vcurve 10/25/99 by P.L. Bornmann
;   Modification 11/19/99 by P.L. Bornmann
;      curnum (the curve number) is passed as the actual
;      number, so switch on this value
;   Ported to IDL 3/5/00 by P.L. Bornmann
;     change case statements to array coefficients
;
; TODO:
;   Solar Array current
;   What is range of curve 220 voltage?
;   xrsi_conv had spacecraft-dependent values for the
;   coarse XRP positions
;
; QUESTIONS:
;
;
; ALGORITHM:
;    GOES I handbook says the N/S SunSensor's sun-present bit
;    indicates sun presence when the XRPE angle is less than +/- 0.4 degrees
;    it indicates no sun present for angles greater than +/-4 degrees
;    it also indiates no sunprent if the E/W angle exceeds 15 degrees
;-

    if (not(keyword_set(plot_all))) then plot_all = 0

    PRT_LEV = 0
;;    start = 0
;;    ftelem = 0.0;     ; float version of telemetry

    ;   temperatures conversion coefficients for thermisters
    ;    on yoke, xrpe, xrs.  Per curve 233
    val_coef = [   -52.51458,      1.510513,    -0.01771354, $
                  1.287143e-04, -4.557991e-07, 6.370513e-10 ]
    val = 0.0
       Textcode = [ $
         '220: ;   telemetry calibration voltage  curve 220', $
         '220:       voltage used to convert analog sensors to digitial telemetry', $
         '220:       val = 0.01 + .02*telem',            $
         '378:    xrs reference voltage,  curve 378',    $
         '378:       val = -0.02 - telem*0.04',          $
         '379:    coarse N-S xrp position, -94 to 28 degrees   curve 379', $
         '379:       val = 28.00 - telem*0.481601',      $
         '380:    xrs n/s SAS analog sun sensor    curve 380', $
         '380:       may need temperature compensation', $
         '380:       val = -1.992 + telem*0.0160',       $
         '381:    xrs e/w SAS analog sun sensor    curve 381', $
         '381:       may need temperature compensation', $
         '381:       val = -1.992 + telem*0.0160',       $
         '382:    xrs -70 volt chamber bias volt,  curve 382',   $
         '382:       val = -0.2 - telem*0.4']          ; NOTE: factor ten larger than ref
voltage

    ; NOTE: curve 233 uses multiple coefs, so tacked on the end and given unit coefs
    calcurve  = [      3,        4,  220,   378,       379,    380,    381,  382, 233]
    constcoef = [ 270.10, -0.90000, 0.01, -0.02,     28.00, -1.992, -1.992, -0.2,   0]
    multcoef  = [1.41421, 1.41421,  0.02,  0.04, -0.481601, 0.0160, 0.0160, -0.4,   1]
    ncalcurve = n_elements(calcurve)

    if (n_elements(curnum) gt 1) then begin
      print, "goesi_vcurve.pro: Not designed to treat multiple calibration curves."
      print, "                  supplied curve number must be a scalar."
      STOP
      RETURN, -1
    endif

    ; Get the case number for this curve
    icase = where(curnum eq calcurve, wicase)
    if (wicase le 0) then begin
      print, "goesi_vcurve.pro:  ERROR. invalid curve number %d", curnum
      print, '                   valid values are ', calcurve
      STOP
    endif

    ;*************************************************************
```

```
    ;**  The simple calibrated values (applies to most cases)  **
    ;*************************************************************
    calval = constcoef(icase(0)) + multcoef(icase(0)) * telem

    if (PRT_LEV gt  1)   then $
      print, "goesi_vcurve: begin input %d, calib curve %3d\n", $
             telem, curnum

;;    ; Keep track of whether this has been called already
;;    if (start eq 0) then start = start + 1

    ;*********************************************
    ;**  Corrections to the simple calibrations  **
    ;*********************************************

    ; SADA Output Position Channel A  curve 003
    if (curnum eq 3) then begin
      ; Data is not valid if telem 235
      wmax = where(telem gt 235, nwmax)
      if (nwmax ge 1) then calval(wmax) = -99.9
      wmaxcal = where(calval gt 360, nwmaxcal)
      if (nwmaxcal ge 1) then calval(wmaxcal) = calval(wmaxcal) - 360
      if (PRT_LEV gt 1) then print, $
          "goesi_vcurve.c:  SADA A position is %f for %d\n", $
                   val, telem
     endif

     ; SADA Output Position Channel A  curve 003
     if (curnum eq 4) then begin
       ; Data is not valid if telem 237
       wmax = where (telem gt 237, nwmax)
       if (nwmax ge 1) then calval(wmax) = -99.9
       if (PRT_LEV gt 1) then print, $
         "goesi_vcurve.c:  SADA B position is %f for %d\n", $
                   calval, telem
      endif

      if (curnum eq 233) then  begin ;   xrs pre-amp val,  curve 233
          calval =  val_coef[0] + telem*(val_coef[1] + telem*(val_coef[2] + telem* $
                (val_coef[3] + telem*(val_coef[4] + telem*val_coef[5]))))
      end

      if (PRT_LEV gt 0) then print, $
         "goesi_vcurve:  input %d, output %5.3f, calib curve %3d\n", $
              telem, calval, curnum

      if (plot_all) then begin
          plot, telem, title = 'telemetered ' + string(curnum, format = '(i4)')
          plot, calval, title = 'calibrated ' + string(curnum, format = '(i4)'), $
                yrange = [min(calval(32:*)), max(calval(32:*))]
      endif

  RETURN, calval
  END  ; end of goesi_vcurve
```

```
FUNCTION lim_range, lim = lim, default = default

;+
; PURPOSE:
;   Return a range array based on supplied limit keyword
;
; INPUTS:
;   lim     either a two-element array containing
;           the [lower, upper] limits for the plot range,
;           or a single value assumed to be the +/- limits
;
; OPTIONAL KEYWORD INPUTS:
;   default limit to be used if lim not supplied
;
; OUTPUTS:
;   -1 if no valid limit supplied
;
; USAGE:
;    xrange = lim_range (lim, default = default)
;
; CALLED BY:
;   plot_xrsoff.pro
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;   Developed 4/11/00 by P.L. Bornmann
;-

  if (keyword_set(lim)) then begin
    if (n_elements(lim) eq 1) then $
      lim_rng = [-1,1]*lim
    if (n_elements(lim) eq 2) then $
      lim_rng  = lim
  endif else if (keyword_set(default)) then begin
    if (n_elements(lim) eq 1) then $
      lim_rng = [-1,1]*lim
    if (n_elements(lim) eq 2) then $
      lim_rng  = lim
  endif else begin
     print, 'lim_range.pro: No valid limit supplied'
     lim_rng = -1
     STOP
  endelse

  ; Return this range

RETURN, lim_rng
END
```

```
PRO no_strays, flgoffpos, itype, ntimes, woff, keepthr = keepthr

;+
; PURPOSE:
;   This removes isolated offpoint flags that do not last longer than the
;   specified duration.
;
; INPUTS:
;   flgoffpos  array to hold the flags indicating times of offpointings
;   itype      first index in flgoffpos, used for multiple types of
;              offpoints (eg. N/S vs E/W) stored in the flgoffpos array
;
; OPTIONAL KEYWORD PARAMETERS:
;   keepthr    number of indices needed to keep a set of offpoints
;
; OUTPUT:
;   woff       indices where offpointings were found
;   flgoffpos  modified to eliminate isolated flags
;
; USAGE:
;   no_strays, flgoffpos, itype, ntimes, woff, keepthr = keepthr
;
; CALLED BY:
;   get_xrpoff.pro
;
; CALLS:
;   seqinfo
;
; MODIFICATION HISTORY:
;   Extracted from get_xrpoff 2/15/00 by P.L. Bornmann
;-

; Get the sequence info
 seqinfo, flgoffpos(0,*)<1, wevent, wstart, wend, durseq, durgap, flagval = 1
 ;if (not(keyword_set(keepthr))) then keepthr = 2*60*1  ; 1 minutes
 if (not(keyword_set(keepthr))) then keepthr = 5

    woff = where(flgoffpos(0,*) gt 0, nwoff)
    if (nwoff le 0) then begin
      print, 'gap_extend.pro: No flags greater than zero'
      RETURN
      STOP
      ;goto, end_delete
    endif

    ; Set the flag for definite offpointings  - NEEDED HERE??
    ;flgoffpos(itype, woff   ) = 2

     ; Get the positions where gaps occur, these are the
     ; end points to a sequence of offset pointing
     wwend = where ((woff(1:*) - woff) gt 1, nwgappos)

     ; Add last point as sequence end
     wwend = [wwend, n_elements(woff)-1]

     ; The end indices
     wend   = woff(wwend)

     ; The sequences start after another ends
     wstart = woff(wwend + 1)

     ; If first points started a sequence, include that
     if (woff(1) eq woff(0)+1) then wstart = [woff(0), wstart]

     ; Determine the duration of sequences and gaps
     durseq    = (wend - wstart) + 1
     durgap    = (wstart(1:*) - wend)

     ; Verify
     wbad = where(durseq le 0, nwbad)
     if (nwbad gt 1) then begin
       print, 'get_xrpoff.pro: This should not have happened.  Negative duration'
       STOP
     endif
;end_delete:

     ; Identify short events
     wshort = where(durseq le keepthr, nwshort)

     ; Remove short events
     if (nwshort ge 1) then begin
       print, ' Will remove ', nwshort, ' of ', n_elements(woff), ' points with durations
less than ', keepthr
       for ishort = 0, nwshort-1 do begin
         w1 = wstart(ishort)
         w2 = wend  (ishort)
         if (w2 gt w1) then begin
           flgoffpos(*,w1:w2) = -2
           print, '     Removed ', wstart(ishort) , ' to ', wend(ishort), ' (duration ',
durseq(ishort), ')'
         endif else begin
           flgoffpos(*,w1) = -2
           print, '     Removed ', wstart(ishort) ,  ' (duration ', durseq(ishort), ')'
         endelse

       endfor
     endif

RETURN

print, 'no_strays.pro: IF THIS HAPPENS THEN DO NOT DELETE THIS CODE'

     if (nwgappos le 0) then begin
       wgappos = [wgappos(0), wgappos(2)]
       nwgappos = nwgappos - 1
       STOP
     endif

     if (nwgappos ge 1) then begin
     ; nextend = 10

       for iwgap = 0, nwgappos-1 do begin

         ; Extend flags forward from end of offpointings
         w1 = (woff(wgappos(iwgap)) ) > 0
         ;w2 = (woff(wgappos(iwgap)) + nextend) < ntimes-1
         ;flgoffpos(itype,w1:w2) = 1
         ;print, ' offpoint flag ext ended from ', w1, ' to ', w2

          ; Extend flags backward from start of offpointings
         ;if (iwgap lt nwgappos-1) then begin
           ;w1 = woff((wgappos(iwgap+1) - nextend)>0 ) > 0
           w2 = woff(wgappos(iwgap+1)) < ntimes-1
           ;flgoffpos(1,w1:w2) = 1
           ;print, ' offpoint flag extended from ', w1, ' to ', w2
         ;endif

         print, ' Gap ', iwgap, ' from ', w1, ' to ', w2, ' duration ', w2-w1
         if (w2-w1 le keepthr) then begin
           ; Set these flags as removed
```

```
          flgoffpos(*,w1:w2) = -2

             print, ' Removed ', iwgap, ' from ', w1, ' to ', w2, ' duration ', w2-w1
          endif

        endfor
        ; Reset the originals
        ;flgoffpos(itype, woff) = 2
      endif
      ;plot, flgoffpos(itype,*), yrange = [-1,3], title = 'NS XRP offpoint flag'


  return
  end
;-------------------------------------------------------------------------------
```

```
PRO no_strays_v2, flgoffpos, itype, ntimes, woff, keepthr = keepthr

;+
; PURPOSE:
;   This removes isolated offpoint flags that do not last longer than the
;   specified duration.
;
; INPUTS:
;   flgoffpos  array to hold the flags indicating times of offpointings
;   itype      first index in flgoffpos, used for multiple types of
;              offpoints (eg. N/S vs E/W) stored in the flgoffpos array
;   woff       indices where offpointings were found
;   keepthr    number of indices needed to keep a set of offpoints
;
; CALLED BY:
;   get_xrpoff
;
; CALLS:
;   seginfo
;
; MODIFICATION HISTORY:
;   Extracted from get_xrpoff 2/15/00 by P.L. Bornmann
;   Modified 4/11/00 by P.L. Bornmann
;     to process based on itype, not fixed first argument of flgoffpos
;-

    ;if (not(keyword_set(nextend))) then nextend = 2*60*60  ; Sixty minutes
    ;if (not(keyword_set(keepthr))) then keepthr = 2*60*1  ; 1 minutes


; Get the sequence info
 seqinfo, flgoffpos(itype,*)<1, wevent, wstart, wend, durseq, durgap, flagval = 1

;goto, end_delete
    if (not(keyword_set(keepthr))) then keepthr = 5

    woff = where(flgoffpos(itype,*) gt 0, nwoff)
    if (nwoff le 0) then begin
      print, 'gap_extend.pro: No offset indices were received'
      STOP
    endif

    ; Set the flag for definite offpointings  - NEEDED HERE??
    ;flgoffpos(itype, woff   ) = 2

     ; Get the positions where gaps occur, these are the
     ; end points to a sequence of offset pointing
     wwend = where ((woff(1:*) - woff) gt 1, nwgappos)

     ; Add last point as sequence end
     wwend = [wwend, n_elements(woff)-1]

     ; The end indices
     wend   = woff(wwend)

     ; The sequences start after another ends
     wstart = woff(wwend + 1)

     ; If first points started a sequence, include that
     if (woff(1) eq woff(0)+1) then wstart = [woff(0), wstart]

     ; Determine the duration of sequences and gaps
     durseq    = (wend - wstart) + 1
     durgap    = (wstart(1:*) - wend)

     ; Verify
     wbad = where(durseq le 0, nwbad)
     if (nwbad gt 1) then begin
       print, 'get_xrpoff.pro: This should not have happened.  Negative duration'
       STOP
     endif
end_delete:

     ; Identify short events
     wshort = where(durseq le keepthr, nwshort)

     ; Remove short events
     if (nwshort ge 1) then begin
       print, ' Will remove ', nwshort, ' of ', n_elements(woff), $
              ' points with durations less than ', keepthr
       for ishort = 0, nwshort-1 do begin
         w1 = wstart(ishort)
         w2 = wend  (ishort)
         if (w2 gt w1) then begin
           flgoffpos(itype,w1:w2) = -2
           print, '     Removed ', wstart(ishort) , ' to ', $
                wend(ishort), ' (duration ', durseq(ishort), ')'
         endif else begin
           flgoffpos(itype,w1) = -2
           print, '     Removed ', wstart(ishort) ,  ' (duration ', durseq(ishort), ')'
         endelse

       endfor
     endif

RETURN

print, 'THE CODE AFTER THE RETURN GETS EXECUTED'
stop

     if (nwgappos le 0) then begin
       wgappos = [wgappos(0), wgappos(2)]
       nwgappos = nwgappos - 1
       STOP
     endif

     if (nwgappos ge 1) then begin
     ; nextend = 10

       for iwgap = 0, nwgappos-1 do begin

         ; Extend flags forward from end of offpointings
         w1 = (woff(wgappos(iwgap))) > 0
         ;w2 = (woff(wgappos(iwgap)) + nextend) < ntimes-1
         ;flgoffpos(itype,w1:w2) = 1
         ;print, ' offpoint flag ext ended from ', w1, ' to ', w2

          ; Extend flags backward from start of offpointings
         ;if (iwgap lt nwgappos-1) then begin
           ;w1 = woff((wgappos(iwgap+1) - nextend)>0 ) > 0
           w2 = woff(wgappos(iwgap+1)) < ntimes-1
           ;flgoffpos(1,w1:w2) = 1
           ;print, ' offpoint flag extended from ', w1, ' to ', w2
         ;endif

         print, ' Gap ', iwgap, ' from ', w1, ' to ', w2, ' duration ', w2-w1
         if (w2-w1 le keepthr) then begin
           ; Set these flags as removed
           flgoffpos(*,w1:w2) = -2
```

```
            print, ' Removed ', iwgap, ' from ', w1, ' to ', w2, ' duration ', w2-w1
          endif

        endfor
        ; Reset the originals
        ;flgoffpos(itype, woff) = 2
      endif
      ;plot, flgoffpos(itype,*), yrange = [-1,3], title = 'NS XRP offpoint flag'

return
end
```

```
PRO parse_goesi, data, varnames, fluxes, hours, xrsstat, dwell, $
                 pos = pos, sattde = sattde, eclipse = eclipse, $
                 xrsstats = xrsstats, volts = volts, temps = temps, $
                 print_all = print_all

;+
; PURPOSE:
;   Converts the block of XRS data into structure variables
;
; INPUTS:
;   data  The array of telemetry values
;
; OUTPUTS:
;   varnames, fluxes, hours, xrsstat, dwell,
;
; OPTIONAL KEYWORD OUTPUTS:
;    pos, sattde, eclipse, xrsstats, volts, temps
;
; OPTIONAL KEYWORD CONTROL:
;    print_all   controls amount of printed information
;
; USAGE:
;   parse_goesi, data, varnames, fluxes, hours, xrsstat, dwell, $
;                pos = pos, sattde = sattde, eclipse = eclipse, $
;                xrsstats = xrsstats, volts = volts, temps = temps, $
;                print_all = print_all
;
; CALLED BY:
;   doit4goesi
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;  Developed 2/9/00 by P.L. Bornmann
;
; TO DO:
;   There is a call to long, which IDL interprets as a user_function
;   Why use xrsstat, when xrsstats is the full array?
;   There is a call to long, which IDL interprets as a user function
;
; ALGORITHM:
;    These are the structure definitions used for the return values
;       fluxstr    = {fluxes, shflux: 0.0, shrng:-1, lflux: 0.0, lrng: -1}
;       posstr     = {pos, sada1:0, sada2:0, nsxrp:0, ewxrp: 0, ewcoarse:0}
;       sasttstr   = {sattstr, de1on: -1, de2on: -1, dir: -1, $
;                     slewon: -1, singlestep: -1, deltstep : -1, $
;                     sa_or_tt: -1}
;       eclstr     = {eclipse, eclflag: -1, arraycur: -1.0, $
;                     primcur: -1.0, ctrlcur: -1.0}
;       xrsstatstr = {xrsstatstr, on:-1, sun: -1, cal:-1, slew:-1, dir:-1}
;       tempstr    = {xrstempstr, tpreamp:-1, tsas:-1, txrp:-1, $
;                     txrpbearing:-1, txrpelectronics:-1}
;       voltstr    = {xrsvoltstr, vref:0, vchamber:0, vcal:0}
;-

if (not(keyword_set(print_all))) then print_all = 0

; Verify data
fmtrng = '("Variable ", i3, a10, " range is ", i6, " to ", i6, " (really ", i6," to ",
i6,")")'
for ivar = 0, n_elements(varnames)-1 do begin
  print, ivar, varnames(ivar), min(data(*,ivar)), max(data(*,ivar)), $
         min(data(32:*,ivar)), max(data(32:*,ivar)),format = fmtrng
endfor

; Convert time
msec = data(*,6)
hours = 1.0*data(*,3) + (data(*,4)/60.) + (data(*,5)/(60.*60.))
msec = data(*,6)

; Ignore data(*,7) for now, the main frame counter

; Dwell (invalid data)
dwell = data(*, 8)
wnz = where(dwell ne 0, nwz)
if (nwz ge 1) then begin
  print, 'parse_goesi.pro:  Found times when dwell was on'
  STOP
endif

ntimes = n_elements(dwell)

; Fluxes
fluxstr = {fluxes, shflux: 0.0, shrng:-1, lflux: 0.0, lrng: -1}
fluxes = replicate(fluxstr, ntimes)
fluxes.shrng  = data(*, 9)
fluxes.shflux = data(*,10)
fluxes.lrng   = data(*,11)
fluxes.lflux  = data(*,12)

; Pointing
if (keyword_set(pos)) then begin
  posstr = {pos, sada1:0, sada2:0, nsxrp:0, ewxrp: 0, ewcoarse:0}
  pos = replicate(posstr, ntimes)
  pos.sada1 = data(*,13)
  pos.sada2 = data(*,14)
  pos.nsxrp = data(*,22)
  pos.ewxrp = data(*,23)
  pos.ewcoarse = data(*,24)
endif

; Solar Array and Trim Tab Drive Electronics
if (keyword_set(sattde)) then begin
  sasttstr = {sattstr, de1on: -1, de2on: -1, dir: -1, $
              slewon: -1, singlestep: -1, deltstep : -1, $
              sa_or_tt: -1}
  sattde = data(*,15)
endif

; Eclipses
if (keyword_set(eclipse)) then begin
  eclstr = {eclipse, eclflag: -1, arraycur: -1.0, primcur: -1.0, ctrlcur: -1.0}
  eclipse = replicate(eclstr, ntimes)
  eclipse.eclflag  = data(*,16)
  eclipse.arraycur = data(*,17)
  eclipse.primcur  = data(*,18)
  eclipse.ctrlcur  = data(*,19)
endif

; Word 51
word51 = data(*,20)
; XRS status word
xrsstat = data(*,21)
xrsstatstr = {xrsstatstr, on:-1, sun: -1, cal:-1, slew:-1, dir:-1}
xrsstats = replicate(xrsstatstr, ntimes)
iarr = lonarr(ntimes, 6)
varr = lonarr(ntimes,6)
```

```
div = 10L^indgen(6)
cum = 0
fmt = "(a5, 11i8)"
for i = 0, 5 do begin
  iarr(*,i) = long(xrsstat/div(5-i))
  if (i eq 0) then $
    varr(*,i) = iarr(*,i)  $
  else $
    varr(*,i) = iarr(*,i) - cum
  if (print_all) then begin
    print, "stat", xrsstat(0:10), format = fmt
    print, "iarr", iarr(0:10,i), format = fmt
    if (i gt 0) then print, "cum ", cum (0:10)  , format = fmt
    print, "varr", varr(0:10,i), format = fmt
  endif
  cum = (cum + varr(*,i)) *10

endfor

  xrsstats.on   = varr(*,0)
  xrsstats.sun  = varr(*,1)
  xrsstats.cal  = varr(*,2)
  xrsstats.slew = varr(*,3)
  xrsstats.dir  = varr(*,4)

if (print_all) then begin
  print, xrsstats(0:10)
  print, ' Range of xrsstats.on   is ', min(xrsstats.on  ), max(xrsstats.on)
  print, ' Range of xrsstats.sun  is ', min(xrsstats.sun ), max(xrsstats.sun)
  print, ' Range of xrsstats.cal  is ', min(xrsstats.cal ), max(xrsstats.cal)
  print, ' Range of xrsstats.slew is ', min(xrsstats.slew), max(xrsstats.slew)
  print, ' Range of xrsstats.dir  is ', min(xrsstats.dir ), max(xrsstats.dir)
endif

; Temperatures
if (keyword_set(temps)) then begin
  tempstr = {xrstempstr, tpreamp:-1, tsas:-1, txrp:-1, $
             txrpbearing:-1, txrpelectronics:-1}
  temps = replicate(tempstr, ntimes)
  temps.tpreamp = data(*,25)
  temps.tsas    = data(*,26)
  temps.txrp    = data(*,27)
  temps.txrpbearing     = data(*,28)
  temps.txrpelectronics = data(*,29)
endif

; Voltages
if (keyword_set(volts)) then begin
  voltstr = {xrsvoltstr, vref:0, vchamber:0, vcal:0}
  volts = replicate(voltstr, ntimes)
  volts.vref     = data(*,30)
  volts.vchamber = data(*,31)
  volts.vcal     = data(*,32)
endif

RETURN
; End of parse_goesi.pro
END
```

```
PRO patharr4lib, patharr

;+
; PURPOSE:
;   returns string array of PLB's library directories
;   to use in the IDL search path
;
; INPUTS:
;   (none)
;
; OUTPUTS:
;   patharr
;
; USAGE:
;   patharr4lib, patharr
;
; CALLS:
;   (nothing)
;
; CALLED BY:
;   path4rdgoesi
;
; MODIFICATION HISTORY:
;   developed 9/11/00 by P.L. Bornmann
;-

  patharr = [ "D:\D_plb\PLB_work\IDL_dir\Lib_dir\DataProc", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\DataTests", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\DirFiles", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Display\IDLversions", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Display\Other", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Display", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\DisplayOutput", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\GOES", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\InputOutput", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\InputReads", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\RunControl", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\SearchFits\test_search_dir", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\SearchFits", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Statistics", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Strings", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Structures", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\SXI", $
              "D:\D_plb\PLB_work\IDL_dir\Lib_dir\Time" $
            ]

  RETURN
  END
```

```
PRO plotg, xvals, yvals, xt, yt, tt, yr

;+
; PURPOSE:
;   create plots for plot_goesi.pro
;   plot title will include the yvals range and the y axis title
;
; INPUTS:
;   yvals
;   xt      title for x-axis
;   yt      title for y-axis
;   xt      title for plot
;   yr      y axis range for plot
;
; OUTPUTS:
;    (a plot)
;
; USAGE:
;    plotg, yvals, xt, yt, tt, yr
;
; CALLS:
;  (nothing)
;
; CALLED BY:
;  plot_goesi.pro
;
; MODIFICATION HISTORY:
;   Developed 8/10/00 by P.L. Bornmann
;-

    ; Format for range uses either integer format for integers or general for others
    if (is_int(yvals(0))) then gformat = "('(',i10,'-', i10,')')" $
    else gformat = "('(',g10.3,'-', g10.3,')')"

    ; Get the yvals range and create a text string of this info
    rval  = [min(yvals), max(yvals)]
    trval = string (format = gformat, rval[0], rval[1])
    trvl  = strcompress(trval)

    ; Plot title includes data info provided in y axis title
    title = tt + " " + yt + " " + trval
    title = strcompress(title)

    ; Do the plot
    plot, xvals, yvals, yrange = yr, title = title, $
          xtitle = xt, ytitle = yt

RETURN
END
RO plot_cal, fluxes, xrsstat, xrsstats = xrsstats, dwell, $
             pos = pos, sattde = sattde, eclipse = eclipse, $
             volts = volts, temps = temps

;+
; PURPOSE:
;   Makes plots of specific interest for XRS calibrations
;
; INPUTS:
;   Assumes these are the calibration times.  Use get_xrscal to get the
;   indices for calibrations
;
; OUTPUTS:
;   (just plots)
;
; CALLED BY:
;
; CALLS:
;  (nothing)
;
; MODIFICATION HISTORY:
;  Developed 2/10/00 by P.L. Bornmann
;-

  ; Plot just what is most important for calibration
  !p.multi = 0
  !p.multi(1) = 2
  !p.multi(2) = 3

  !p.charsize = 2

  ; Convert time
  ;msec = data(*,6)
  ;time = 1.0*data(*,3) + (data(*,4)/60.) + (data(*,5)/(60.*60.))
  ;msec = data(*,6)
  xt = 'Time'

  ; Ignore data(*,7) for now, the main frame counter

  ntimes = n_elements(dwell)

  ; Plot the fluxes vs. time
  plot, time, fluxes.shflux, ytitle = "Short Fluxes", xtitle = xt
  plot, time, fluxes.lflux, ytitle = "Long Fluxes", xtitle = xt

  ; Pointing
  yt = 'XRS fluxes'
  yr = [0, 255]  ; for telemetry, not engineering units
  plot, pos.sadal, fluxes.shflux, yrange = yr, $
        xtitle = "SADA Pos", ytitle = yt
    oplot, pos.sadal, fluxes.lflux
  plot, pos.nsxrp, fluxes.shflux, yrange = yr,
        xtitle = xt, ytitle = "XRP N/S Pos"
    oplot, pos.nsxrp, fluxes.lflux
  plot, pos.ewxrp, fluxes.shflux, yrange = yr,
        xtitle = xt, ytitle = "XRP E/W Pos"
    oplot, pos.ewxrp, fluxes.lflux
  plot, pos.ewcoarse, fluxes.shflux, yrange = yr, $
        xtitle = xt, ytitle = "Coarse E/W Pos"
    oplot, pos.ewcoarse, fluxes.lflux

; Temperatures
  print, ' The Ranges for this period '
  print,  "temps.tpreamp        ",  min(temps.tpreamp ) ,  max(temps.tpreamp )
  print,  "temps.tsas           ",  min(temps.tsas    ) ,  max(temps.tsas    )
  print,  "temps.txrp           ",  min(temps.txrp    ) ,  min(temps.txrp    )
  print,  "temps.txrpbearing    ",  min(temps.txrpbearing ), max(temps.txrpbearing )
  print,  "temps.txrpelectronics",  min(temps.txrpelectronics ),
max(temps.txrpelectronics )

; Voltages
  print,  "volts.vref    ", min( volts.vref    ), max( volts.vref    )
  print,  "volts.vchamber", min( volts.vchamber), max( volts.vchamber)
  print,  "volts.vcal    ", min( volts.vcal    ), min( volts.vcal    )

RETURN
END
```

```
PRO plot_goesi, fluxes, hours, dwell, xrsstats = xrsstats, pos = pos, $
                sattde = sattde, eclipse = eclipse, volts = volts, temps = temps, $
                go_slow = go_slow, title = title

;+
; PURPOSE:
;   Plots xrs data
;
; INPUTS:
;   fluxes
;   hours
;   dwell
;
; OPTIONAL KEYWORD PARAMETERS:
;   if these are supplied, then they are plotted
;     xrsstats, dwell, pos, sattde, eclipse, volts, temps
;   title     title to be used for all plots
;
; OPTIONAL KEYWORD CONTROL PARAMETER
;   go_slow   if set, will pause after each screen of plots
;
; OUTPUTS:
;   (just plots)
;
; USAGE:
;   plot_goesi, fluxes, hours, dwell, xrsstats = xrsstats, pos = pos, $
;               sattde = sattde, eclipse = eclipse, volts = volts, temps = temps, $
;                go_slow = go_slow, title = title
;
; CALLED BY:
;    doit4goesi.pro
;
; CALLS:
;   port_fwin2
;   plotg
;   wait_user
;
; MODIFICATION HISTORY:
;  Developed 2/10/00 by P.L. Bornmann
;  Modified 4/15/00 by P.L. Bornmann
;    Only the first plot, of time, is ploted against data
;    point number.  All the rest have time as the x axis
;    Change !p.multi for first plots
;  Modification 8/10/00 by P.L. Bornmann
;    Corrected y-axis lable for the eclipse flag plot
;    Moved plot parameters to plotg subroutine
;
; TO DO:
;   Why temperatures remain at zero:  DrT, BrT, ArT
;-

if (not(keyword_set(go_slow))) then go_slow = 0

!p.charsize = 2
yrtlm = [-1,256]

!p.multi = 0
!p.multi(1) = 0
    !p.multi(2) = 4
    port_fwin2

    ; Plot time
    plot, hours, title = title, $
          xtitle = "Data Point number", ytitle = 'Time'

    ; Title for axis
    xt = 'Time (hrs)'
    print, 'plot_goesi.pro: xt = ', xt
    wait_user

    ; Dwell (invalid data)
    ;dwell = data(*, 8)
    ;plot, hours, dwell, xtitle = xt, ytitle = 'Dwell', yrange = [-1,3], $
    ;      title = title
    plotg, hours, dwell, xt, 'Dwell', title, [-1,3]

    ; For baselines
    ntimes   = n_elements(dwell)
    xdrange  = [min(hours), max(hours)]
    baseline = [0,0]

    ; Fluxes
;    gformat = "('(',g10.3,'-', g10.3,')')", reng[0], reng[1])
;    rval = [min(fluxes.shflux), max(fluxes.shflux)]
;    trval = string (format = gformat, rval[0], rval[1])
;    plot, hours, fluxes.shflux, yrange = [-5,255], title = title + yt + rval, $
;          xtitle = xt, ytitle = yt
    plotg, hours, fluxes.shflux, xt, "Short Fluxes", title, [-5,255]
      oplot, -fluxes.shrng
      oplot, xdrange, baseline, linestyle = 1

;    yt = "Long Fluxes"
;    rval = [min(fluxes.lflux), max(fluxes.lflux)]
;    trval = string (format = gformat, rval[0], rval[1])
;    plot, hours, fluxes.lflux, yrange = [-5,255],  title = title + yt + rval , $
;          xtitle = xt, ytitle = yt
    plotg, hours, fluxes.lflux, xt, "Long Fluxes", title, [-5,255]
      oplot, -fluxes.lrng
    if (go_slow) then wait_user

    ; Pointing
    if (keyword_set(pos)) then begin
      !p.multi = 0
      !p.multi(2) = 4

;    yt = "SADA Pos"
;    rval = [min(pos.shflux), max(pos.shflux)]
;    trval = string (format = gformat, rval[0], rval[1])
;    trval = strcompress(trval)
;    plot, hours, pos.shflux, yrange = [-5,255], title = title + yt + trval, $
;      plot, hours, pos.sada1, yrange = yrtlm, $
;            title = title + yt + trval, $
;            xtitle = xt, ytitle = yt
    plotg, hours, pos.sada1, xt, "SADA Pos", title, [-5,255]
        oplot, hours, pos.sada2

;    yt= "XRP N/S Pos"
;    rval = [min(pos.shflux), max(pos.shflux)]
;    trval = string (format = gformat, rval[0], rval[1])
;    trval = strcompress(trval)
;    plot, hours, pos.shflux, yrange = [-5,255], title = title + yt + trval, $
;      plot, hours, pos.nsxrp,    title = title + yt + trval,  $
;            xtitle = xt, ytitle = yt
    plotg, hours, pos.nsxrp, xt, "XRP N/S Pos", title, [-5,255]

;    yt = "XRP E/W Pos"
;    rval = [min(pos.shflux), max(pos.shflux)]
;    trval = string (format = gformat, rval[0], rval[1])
;    trval = strcompress(trval)
;    plot, hours, pos.shflux, yrange = [-5,255], title = title + yt + trval, $
```

```
;      plot, hours, pos.ewxrp,    title = title + yt + trval,  $
;            xtitle = xt, ytitle = yt
    plotg, hours, pos.ewxrp, xt, "XRP E/W Pos", title, [-5,255]

;    yt= "Coarse E/W Pos"
;    rval = [min(pos.ewcoarse), max(pos.ewcoarse)]
;    trval = string (format = gformat, rval[0], rval[1])
;    trval = strcompress(trval)
;      plot, hours, pos.ewcoarse, title = title + yt + trval, $
;            xtitle = xt, ytitle = yt
    plotg, hours, pos.ewcoarse, xt, "Coarse E/W Pos", title, [-5,255]

      if (go_slow) then wait_user
    endif

    ; Solar Array and Trim Tab Drive Electronics
    if (keyword_set(sattde)) then begin
      ;sasttstr = {sattstr, de1on: -1, de2on: -1, dir: -1, $
      ;            slewon: -1, singlestep: -1, deltstep : -1, $
      ;            sa_or_tt: -1}
      ;sattde = data(*,15)
    endif

    ; Eclipses
    if (keyword_set(eclipse)) then begin
      !p.multi = 0
      !p.multi(2) = 4
;     yt = 'Current'
;     plot, hours, eclipse.eclflag , yrange = [-1,2], $
;           title = title + " Eclipse Flag", xtitle = xt, ytitle = "Flag"
      plotg, hours, eclipse.eclflag, xt, "Ecl. Flag", title, [-1,2]
        oplot, xdrange, baseline, linestyle = 1

;     plot, hours, eclipse.arraycur, yr = yrtlm, $
;           title = title +  " Array Cur.", xtitle = xt, ytitle = yt
      plotg, hours, eclipse.arraycur, xt, " Array Cur.", title, yrtlm

;     plot, hours, eclipse.primcur , yr = yrtlm, $
;           title = title +  " Prim. Buss Cur.", xtitle = xt, ytitle = yt
      plotg, hours, eclipse.primcur , xt, " Prim. Buss Cur.", title, yrtlm

;     plot, hours, eclipse.ctrlcur , yr = yrtlm, $
;           title = title + "Control Cur.", xtitle = xt, ytitle = yt
      plotg, hours, eclipse.ctrlcur , xt, "Control Cur.", title, yrtlm
      if (go_slow) then wait_user
    endif

    ; Word 51
    ;word51 = data(*,20)

    ; XRS status word
    if (keyword_set(xrsstats)) then begin
      !p.multi = 0
      off = 3
      xdrh = [min(hours), max(hours)*1.2]
      plot, hours, xrsstats.on, /nodata, $
            xrange = xdrh, yrange = [0,5*off],  $
            title = title, xtitle = xt, ytitle = 'XRS Status'
        oplot, hours, xrsstats.on
        oplot, xdrange, baseline, linestyle = 1
        xyouts, xdrange[1], 1, 'XRS On'
        oplot, hours, xrsstats.sun +  off
        oplot, xdrh, baseline+off, linestyle = 1
        xyouts, xdrange[1], off+1,  'XRS has Sun '
        oplot, hours, xrsstats.cal +2*off
        oplot, xdrh, baseline+2*off, linestyle = 1
        xyouts, xdrange[1], 2*off+1, 'XRS Calib'
        oplot, hours, xrsstats.slew+3*off
        oplot, xdrh, baseline+3*off, linestyle = 1
        xyouts, xdrange[1], 3*off+1, 'XRS Slew'
        oplot, hours, xrsstats.dir +4*off
        oplot, xdrh, baseline+4*off, linestyle = 1
        xyouts, xdrange[1], 4*off+1,'XRS Dir'
      if (go_slow) then wait_user
    endif

    ; Temperatures
    if (keyword_set(temps)) then begin
      !p.multi = 0
      !p.multi(2) = 4
      yt = 'Temp'
;     plot, hours, temps.tsas    , yr = yrtlm,  $
;           title = title + " SAS Temp", $
;           xtitle = xt, ytitle = yt
      plotg, hours, temps.tsas    , xt, " SAS Temp", title, yrtlm

;     plot, hours, temps.txrp    , yr = yrtlm,  $
;           title = title + " XRP Temp", $
;           xtitle = xt, ytitle = yt
      plotg, hours, temps.txrp    , xt, " XRP Temp", title, yrtlm

;     plot, hours, temps.txrpbearing, yr = yrtlm,  $
;           title = title + " XRP Bear. Temp", $
;           xtitle = xt, ytitle = yt
      plotg, hours, temps.txrpbearing, xt, " XRP Bear. Temp", title, yrtlm

;     plot, hours, temps.txrpelectronics , yr = yrtlm,  $
;           title = title + " XRP Electr. Temp", $
;           xtitle = xt, ytitle = yt
      plotg, hours, temps.tsas    , xt, " XRP Electr. Temp", title, yrtlm

;     if (go_slow) then wait_user
    endif

    ; Voltages
    if (keyword_set(volts)) then begin
      !p.multi = 0
      !p.multi(2) = 3
      ytitle  = 'Volts'
;     plot, hours, volts.vref    , yr = yrtlm, title = title + " Ref Volt", $
;           xtitle = xt, ytitle = yt
      plotg, hours, volts.vref    , xt, " Ref Volt", title, yrtlm
          oplot, xdrange, baseline, linestyle = 1

;     plot, hours, volts.vchamber, yr = yrtlm, title = title + " Chamber Volt", $
;           xtitle = xt, ytitle = yt
      plotg, hours, volts.vchamber, xt, " Chamber Volt", title, yrtlm
          oplot, xdrange, baseline, linestyle = 1

;     plot, hours, volts.vcal    , yr = yrtlm, title = title + " TLM Cal Volt", $
;           xtitle = xt, ytitle = yt
      plotg, hours, volts.vcal    , xt, " TLM Cal Volt", title, yrtlm
          oplot, xdrange, baseline, linestyle = 1
    endif

; end of plot_goesi.pro
RETURN
END
```

```
PRO plot_struct, struct_1, struct_2, xvalues = xvals, $
                 title = title, xtitle = xt

;+
; PURPOSE:
;   plot all the values in a structure
;   if two stuctures are supplied, it plots them side-by-side
;   (ideal for comparing related stucture variables or intervals)
;
; INPUT:
;   struct_1   a structure containing all numerical values
;
; OPTIONAL INPUT:
;   struct_2   another structure containing all numerical values
;              assumed to have same number of tags as struct_1
;
; OPTIONAL KEYWORD INPUT:
;   xvalues  the x variable for the plots.  If not supplied,
;            x will indicate the point number
;   title    title to add to all plots
;   xtitle   title for x axis
;
; OUTPUT:
;   (plots)
;
; USAGE:
;   plot_struct, struct_1
;   plot_struct, struct_1, struct_2, xvalues = xvalues, $
;      title = title, xtitle = xtitle
;
; CALLED BY:
;   doit4goesi.pro
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;    Developed 4/11/00 by P.L. Bornmann
;    Modified 4/16/00 by P.L. Bornmann
;      added title and xtitle to inputs
;    Modified 5/17/00 by P.L. Bornmann
;      plots just one structure if second not supplied
;-

  ntags = n_tags(struct_1)
  if (not(keyword_set(x))) then $
    x = indgen(n_elements(struct_1))
  if (not(keyword_set(xtitle))) then xtitle = ''
  if (not(keyword_set( title))) then  title = ''
  if (n_params() lt 2) then dostruct2 = 0 else dostruct2 = 1

  print, 'plot_struct.pro:  title = ', title
  print, 'plot_struct.pro: xtitle = ', xtitle
  ;wait_user

  !p.multi = 0
  if (dostruct2 eq 1) then !p.multi(1) = 2
  !p.multi(2) = ntags
  tlmnames = tag_names(struct_1)
  if (dostruct2 eq 1) then engnames = tag_names(struct_2)


  gfmt = "(g10.3)"
  ifmt = "(i10)"
  for itag = 0,ntags-1 do begin
    if (is_int(struct_1.(itag))) then fmt = ifmt else fmt = gfmt
    rng1 = [min(struct_1.(itag)), max(struct_1.(itag))]
    srng1 = strcompress([string(rng1[0],format = fmt), string(rng1[1],format = fmt)])
    plot, xvals, struct_1.(itag), $
          title = title + ' Telem ' + tlmnames(itag) + ' ' + srng1[0] + " - " + srng1[1],
$
          xtitle = xt, ytitle = 'Telem Val'
    if (dostruct2 eq 2) then begin
        if (is_int(struct_2.(itag))) then fmt = ifmt else fmt = gfmt
        rng2 = [min(struct_2.(itag)), max(struct_2.(itag))]
        srng2 = strcompress([string(rng2[0],format = fmt), string(rng2[1],format = fmt)])

        plot, xvals, struct_2.(itag), $
          title = title + ' Eng ' + engnames(itag) + ' ' + srng2[0] + " - " + srng2[1], $
          xtitle = xt, ytitle = 'Eng Val'
    endif
  endfor

  print, 'plot_struct.pro: xtitle = ', xt
  wait_user, 'plot_struct completed'

RETURN
END
```

```
PRO plot_xrscal, fluxes, hours, xrsstats, dwell, $
             pos = pos, sattde = sattde, eclipse = eclipse, $
             volts = volts, temps = temps, title = title

;+
; PURPOSE:
;   Makes plots specifically designed for looking at
;   XRS calibration data
;
; INPUTS:
;   fluxes   Fluxes for the cal times
;   hours
;   xrsstats
;   dwell
;
; OPTIONAL KEYWORD PARAMETERS:
;   if these values are supplied then they are plotted
;      pos, sattde, eclipse, volts, temps,
;   title    title information for all plots
;
; OUTPUTS:
;   (plots)
;
; USAGE:
;   plot_xrscal, fluxes, hours, xrsstats, dwell, $
;             pos = pos, sattde = sattde, eclipse = eclipse, $
;             volts = volts, temps = temps, title = title
;
; CALLED BY:
;
; CALLS:
;   plot_goesi
;   wait_user
;
; MODIFICATION HISTORY:
;  Developed 2/10/00 by P.L. Bornmann
;-

; Send raw data to plots
plot_goesi,  fluxes, hours, xrsstats = xrsstats, dwell, $
             pos = pos, sattde = sattde, eclipse = eclipse, $
             volts = volts, temps = temps, title = title

; Plot controls
!p.multi = 0
!p.multi(1) = 2
!p.multi(2) = 3
!p.multi(2) = 4
!p.charsize = 2

xt = 'Time (hrs)'
print, 'plot_xrscal.pro: xt = ', xt
wait_user
ntimes = n_elements(dwell)

; Plot the fluxes vs. time
plot, hours, fluxes.shflux,  title = title, $
      xtitle = xt,  ytitle = "Short Fluxes"
;plot, hours, fluxes.shrng , ytitle = "Short Range", xtitle = xt
plot, hours, fluxes.lflux, title = title, $
      xtitle = xt, ytitle = "Long Fluxes"
;plot, hours, fluxes.lrng , ytitle = "Long Range", xtitle = xt

; Pointing
if (keyword_set(pos)) then begin
  yt = 'Pos'
  plot, hours, pos.sada1, yrange = [0,255], title = title + "SADA 1 Pos", $
        xtitle = xt, ytitle = yt
  plot, hours, pos.sada2, yrange = [0,255], title = title + "SADA 2 Pos", $
        xtitle = xt, ytitle = yt
  plot, hours, pos.nsxrp, title = title + "XRP N/S Pos", $
        xtitle = xt, ytitle = yt
  plot, hours, pos.ewxrp, title = title + "XRP E/W Pos", $
        xtitle = xt, ytitle = yt
  plot, hours, pos.ewcoarse, title = title + "Coarse E/W Pos", $
        xtitle = xt, ytitle = yt
endif

plot, pos.ewxrp, fluxes.shflux, title = title, xtitle = 'EW SAS', ytitle = 'XRS Signal'
  oplot, pos.ewxrp, fluxes.lflux
plot, pos.nsxrp, fluxes.shflux, title = title, xtitle = 'NS SAS', ytitle = 'XRS Signal'
  oplot, pos.nsxrp, fluxes.lflux

RETURN
; Eclipses
if (keyword_set(eclipse)) then begin
  !p.multi = 0
  !p.multi(2) = 4
  plot, hours, eclipse.eclflag  , title = title, $
        ytitle = "Eclipse Flag", xtitle = xt
  plot, hours, eclipse.arraycur , title = title, $
        ytitle = "Array Current", xtitle = xt
  plot, hours, eclipse.primcur  , title = title, $
        ytitle = "Primary Buss Currrent", xtitle = xt
  plot, hours, eclipse.ctrlcur  , title = title, $
        ytitle = "Control Current", xtitle = xt
  wait_user
endif

; XRS status word
if (keyword_set(xrsstats)) then begin
  !p.multi = 0
  !p.multi(2) = 5
  plot, hours, xrsstats.on  , title = title, ytitle = 'XRS On',       xtitle = xt
  plot, hours, xrsstats.sun , title = title, ytitle = 'XRS has Sun ', xtitle = xt
  plot, hours, xrsstats.cal , title = title, ytitle = 'XRS Calib',    xtitle = xt
  plot, hours, xrsstats.slew, title = title, ytitle = 'XRS Slew',     xtitle = xt
  plot, hours, xrsstats.dir , title = title, ytitle = 'XRS Dir',      xtitle = xt
  wait_user
endif

RETURN
END
```

```
PRO plot_xrsoff, fluxes, hours, pos, lim_fxrp = lim_fxrp, $
                 lim_nsxrp = lim_nsxrp, lim_ewxrp = lim_ewxrp, $
                 title = title

;+
; PURPOSE:
;   Create plots showing the XRS signal as functions of
;   angles to the sun.
;
; INPUTS:
;   fluxes  stucture array containing the (xrs) fluxes
;   hours   array containing the time  (not used)
;   pos     structure array containing the (xrs) position information
;
; OPTIONAL KEYWORD INPUTS:
;   lim_fxrp lim_nsxrp, lim_ewxrp
;           each these can be either a two-element array
;           containing the [lower, upper] limits for the plot range,
;           or a single value assumed to be the +/- limits
;   title   additional text for the plot titles
;
; OUTPUTS:
;   plots of fluxes as a function of position
;
; USAGE:
;   plot_xrsoff, fluxes, hours, pos, lim_fxrp  = lim_fxrp, $
;                lim_nsxrp = lim_nsxrp, lim_ewxrp = lim_ewxrp
;
; CALLED BY:
;  do_xrsoff
;
; CALLS:
;   lim_range
;   wait_user
;
; MODIFICATION HISTORY:
;   Developed 2/16/00 by P.L. Bornmann
;   Modified 4/11/00 by P.L. Bornman
;     Added limit keyword and code
;   Modified 4/17/00 by P.L. Bornmann
;     Added title as keyword parameter
;     Changed plot range because angles are eng,
;     rather than telemetered.
;   Modified 9/11/00 by P.L. Bornmann
;     change to four panel plots for offpointings
;   Modified 9/13/00 by P.L. Bornmann
;     xr change not used, so no need for izoom loop
;     return to two panel plots, change to plot both
;     directions together, fluxes on different plots
;
; TO DO:
;   plot as time sequences
;-

  if (not(keyword_set(title))) then title = ''

  xtns = "N/S SAS Angle"
  xtew = "E/W SAS Angle"
  ytsh = "Sh Signal"
  ytl  = "Long Signal"

  ; Model received telemetered fluxes
  ytsh = ytsh + ' Telem'
  ytl  = ytl  + ' Telem'

  ; Plot ranges
  tlm_ang = 0
  if (tlm_ang eq 1) then begin
    drflux  = [0,300]
    drnsxrp = [0,275]
    drewxrp = [0,275]

  endif else begin
    drflux  = [0,100]
    drnsxrp = [-0.75, 0.75]
    drewxrp = [-0.75, 0.75]
    print, 'plot_xrsoff.pro: angle range is ', drnsxrp, drewxrp
  endelse

  ; Plot properties
  ;!p.multi = 0
  ;!p.multi(2) = 2

  ; Four-panel plots for the two SAS direction and two wavelengths
  !p.multi = 0
  ;!p.multi(1) = 2
  !p.multi(2) = 2

  if (not(keyword_set(lim_fxrp ))) then lim_fxrp  = 0
  if (not(keyword_set(lim_nsxrp))) then lim_nsxrp = 0
  if (not(keyword_set(lim_ewxrp))) then lim_ewxpr = 0
  rflux  = lim_range (lim = lim_fxrp , default = drflux )
  rnsxrp = lim_range (lim = lim_nsxrp, default = drnsxrp)
  rewxrp = lim_range (lim = lim_ewxrp, default = drewxrp)

  tt = string(hours(0), format = '(f6.2)') + ' to ' + $
       string (hours(n_elements(hours)-1), $
       format = '(f6.2)') + ' hrs'
  tt = title + ' ' + tt
  tt = strcompress(tt)
  npos = n_elements(pos)
  pltmult = 10
  pltshft = 270
  pltxout = 150   ; NEED TO CHANGE THESE
  pltcout = 0.75
  ;for izoom = 0, 1 do begin   ; ARE TWO VALUES USE?
;    if (izoom eq 1) then begin
;      xr = 128 + [-1,1]*20
;    endif

      plot, pos.nsxrp, fluxes.shflux,  xrange = rnsxrp, yrange = rflux, $
            xtitle = xtns, ytitle = ytsh, title = tt, psym = 7,  symsize = 0.5
        ;oplot, pos.nsxrp, fluxes.shrng * pltmult + pltshft
        xyouts, pltxout, pltshft, 'Sh Rng', charsize = pltcout

      plot, pos.ewxrp, fluxes.shflux , xrange = rnsxrp, yrange = rflux, $
            xtitle = xtew, ytitle = ytsh, title = tt, psym = 7,  symsize = 0.5
        ;oplot, pos.ewxrp, fluxes.lrng*pltmult + pltshft
        xyouts, pltxout, pltshft, 'Sh Rng', charsize = pltcout

      plot, pos.nsxrp, fluxes.lflux  , xrange = rnsxrp, yrange = rflux, $
            xtitle = xtns, ytitle = ytl, title = tt, psym = 7,  symsize = 0.5
        ;oplot, pos.nsxrp, fluxes.lrng * pltmult + pltshft
        xyouts, pltxout, pltshft, 'Lng Rng', charsize = pltcout

      plot, pos.ewxrp, fluxes.lflux  , xrange = rnsxrp, yrange = rflux, $
            xtitle = xtew, ytitle = ytl, title = tt, psym = 7,  symsize = 0.5
        ;oplot, pos.ewxrp, fluxes.lrng*pltmult + pltshft
        xyouts, pltxout, pltshft, 'Lng Rng', charsize = pltcout
```

```
  ;endfor ; izoom

  if (!p.multi(0) eq 0) then wait_user

RETURN
END ; end of plot_xrsoff
```

```
PRO rdgoesi, data, varnames, filename, filedir, print_all = print_all

;+
; PURPOSE:
;   Reads the GOES I-M, (8- M) data that has been written
;   to an asicii file by the c program rdgicd.exe
;
; INPUTS:
;   (none, gets filename from user interaction via get_filename.pro)
;
; OUTPUTS:
;    data
;    varnames
;    filaname  the name of the selected file
;    filedir   the directory of the selected file
;
; USAGE:
;   rdgoesi, data, varnames, filename, filedir, print_all = print_all
;
; CALLS:
;   get_filename
;   interp_path
;
; ALGORITHM:
;   Uses get_filename to find file to process (starts with hardcoded
;   directory to look in).
;   Reads the data from the file the user selects.
;
; MODIFICATION HISTORY:
;  Developed 2/8/00 by P.L. Bornmann
;  Modification 4/11/00 by P.L. Bornmann
;    added missing commands to close and free the lun
;
;-

    if (not(keyword_set(print_all))) then print_all = 0

    ; Get the file
    file = get_filename(filedir = $
        'd:\d_plb\plb_work\c_code\rdgoes8_10\plb_rdgoes\testdata\', $
         wildfile = '*.dat')

    ; To return the filename and path
    interp_path, file, dirname, filename, ext

    ; Open the file
    openr, lun, file, /get_lun

        ; Read the header line
        colhead = ''
        readf, lun, colhead
        if (print_all) then begin
           print, ' Column headings are '
           print, colhead
        endif

        ; Parse column heading to get number of elements in file
        varnames = str_sep(strtrim(strcompress(colhead),2), " ", /trim)
        nvars = n_elements(varnames)
        if (print_all) then begin
          print, ' Found ', nvars, ' variable times'
          for i = 0, nvars-1 do print, i, ':  ', varnames(i)
        endif
        dataline = lonarr(nvars)
        nobspday = 24L*60*60*2
        data = lonarr(nobspday, nvars)

        fmt_start = "("
        fmt_end   = ")"
        fmt_date  = "i4, 1x, i3, 1x "
        fmt_time  = "i2, ':', i2, ':', i2, '.', i3"
        fmt_flux  = "i1, 1x, i3, 1x, i1, 1x, i3"
        fmt_sada  = "i3, 1x, i3"
        fmt_cont  = ", 1x, "
        fmt = fmt_start + "i7, 1x, i2, 1x, " + fmt_date + fmt_cont + fmt_time + fmt_cont
+ $
              "i3, 1x, i2 ,1x,  " + fmt_flux + fmt_cont + fmt_sada + fmt_cont + $
              "i8, 1x, i1, 1x, 4(i3,1x), i8, 1x,  11(i3,1x)" + fmt_end
        if (print_all) then print, fmt
        str_temp = str_sep (fmt, "i")
        hfmt = ""
        for isep = 0, n_elements(str_temp)-2 do begin
          hfmt = hfmt + str_temp(isep) + "a"
        endfor
        hfmt = hfmt + str_temp((n_elements(str_temp)-1))
        if (print_all) then begin
          print, str_temp
          print, hfmt
        endif
        iline = -1L
        print, "", varnames, format = hfmt

        ; Read all the data
        while (not(eof(lun))) do begin
          iline = iline + 1
          readf, lun, dataline
          if (iline mod 5000 eq 0) then print, iline, dataline, format = fmt
          data (iline,*) = dataline
        endwhile

    ; Close the file
    close, lun
    free_lun, lun

    ; Truncate the array to what was actually read
    print, 'rdgoesi.pro: Read a total of ', iline, ' lines, allocated ', nobspday
    data = data(0:iline,*)

    ; Print data
    print, "", varnames, format = hfmt
    fmtrng = '("Variable ", i4, a10, " range is ", i6, " to ", i6)'
    for ivar = 0, n_elements(varnames)-1 do begin
      print, ivar, varnames(ivar), min(data(*,ivar)), max(data(*,ivar)), $
             min(data(32:*,ivar)), max(data(32:*,ivar)), format = fmtrng
    endfor

; End of rdgoesi.pro
RETURN
END
```

```
PRO seqinfo, flags, wevent, wstart, wend, durseq, durgap, flagval = flagval

;+
; PURPOSE:
;   takes an array containing flag values and identifies the indices
;   indicating when "events" start and end, and provides the "event"
;   duration and gap durations
;
; INPUTS:
;   flags
;
; OPTIONAL KEYWORD INPUT:
;   flagval  the value in the flags array that denotes and "event"
;
; OUTPUT:
;   wevent
;   wstart
;   wend
;   durseq
;   durgap
;
; USAGE:
;   seqinfo, flags, wevent, wstart, wend, durseq, durgap, flagval = flagval
;
; CALLED BY:
;   no_strays.pro
;
; CALLS:
;  (nothing)
;
; MODIFICATION HISTORY:
;   Extracted from no_strays 2/16/00 by P.L. Bornmann
;-

    if (not(keyword_set(flagval))) then flagval = 1

    wevent = where(flags eq flagval, nwevent)
    if (nwevent le 0) then begin
      print, 'seqinfo.pro: No points flagged as ', flagval, $
             ' all flag settings between ', min(flags), 'and ', max(flags)
      RETURN
      STOP
    endif

     ; Get the positions where gaps occur, these are the
     ; end points to a sequence of offset pointing
     wwend = where ((wevent(1:*) - wevent) gt 1, nwend)
     if (nwend le 0) then begin
       print, 'seq_info: no "events" found.  Range of flags were ', $
              min(flags), ' to ', max(flags)
       RETURN
     endif

     ; Add last point as sequence end
     wwend = [wwend, n_elements(wevent)-1]

     ; The end indices
     wend   = wevent(wwend)

     ; The sequences start after another ends
     wstart = wevent(wwend + 1)

     ; If first points started a sequence, include that
     if (wevent(1) eq wevent(0)+1) then wstart = [wevent(0), wstart]

     ; Eliminate last point if necessary
     if (wstart(n_elements(wstart)-1) eq wend(n_elements(wend)-1)) then  begin
       wstart = wstart(0:(n_elements(wstart)-2))
     endif

     ; Determine the duration of sequences and gaps
     durseq    = (wend - wstart) + 1
     durgap    = (wstart(1:*) - wend)

     ; Verify
     wbad = where(durseq le 0, nwbad)
     if (nwbad gt 1) then begin
       print, 'seq_info: This should not have happened.  Negative duration'
       STOP
     endif

RETURN
END

;-------------------------------------------------------------------------
```

```
PRO seqinfo_v2, flags, wevent, wstart, wend, durseq, durgap, $
             flagval = flagval, print_all = print_all

;+
; PURPOSE:
;   takes an array containing flag values and identifies the indices
;   indicating when "events" start and end, and provides the "event"
;   duration and gap durations
;
; INPUTS:
;   flags
;
; OPTIONAL KEYWORD INPUT:
;   flagval  the value in the flags array that denotes and "event"
;
; OUTPUT:
;   wevent
;   wstart
;   wend
;   durseq
;   durgap
;
; CALLED BY:
;   no_stays
;
; CALLS:
;   (nothing)
;
; MODIFICATION HISTORY:
;   Extracted from no_strays 2/16/00 by P.L. Bornmann
;-
    if (not(keyword_set(flagval))) then flagval = 1
    if (not(keyword_set(print_all))) then print_all = 1

    wevent = where(flags eq flagval, nwevent)
    if (nwevent le 0) then begin
      print, 'seqinfo.pro: No offset indices were received'
      STOP
    endif

     ; Get the positions where gaps occur, these are the
     ; end points to a sequence of offset pointing
     wwend = where ((wevent(1:*) - wevent) gt 1, nwend)
     if (nwend le 0) then begin
       print, 'seq_info: no "events" found.  Flag range was ', $
              min(flags), ' to ', max(flags), ' and event indicated by ', $
              flagval
       RETURN
     endif

     ; Add last point as sequence end
     wwend = [wwend, n_elements(wevent)-1]

     ; The end indices
     wend   = wevent(wwend)

     ; The sequences start after another ends
     wstart = wevent(wwend + 1)

     ; If first points started a sequence, include that
     if (wevent(1) eq wevent(0)+1) then wstart = [wevent(0), wstart]

     ; Determine the duration of sequences and gaps
     durseq    = (wend - wstart) + 1
     durgap    = (wstart(1:*) - wend)

     ; Verify
     wbad = where(durseq le 0, nwbad)
     if (nwbad gt 1) then begin
       print, 'seq_info: This should not have happened.  Negative duration'
       STOP
     endif

     if (print_all) then begin
       print, ' wstart = ', wstart
       print, ' wend   = ', wend
       print, ' durseq = ', durseq
       print, ' durgap = ', durgap
     endif

RETURN
END
```

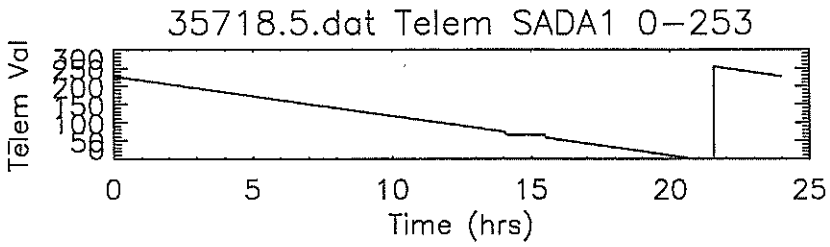
```
PRO telem2eng, xrs_tlmstats, xrs_tlmpos, xrs_tlmvolts, xrs_tlmtemps, $
               xrs_engstats, xrs_engpos, xrs_engvolts, xrs_engtemps, $
               plot_all = plot_all

;-
; FILE:
;   telem2eng.pro
;
; PURPOSE:
;   used to convert some temperatures, voltages, angles, etc from
;   telemetered values to engineering values
;
; INPUT:
;   xrs_tlmstats
;   xrs_tlmpos
;   xrs_tlmvolts
;   xrs_tlmtemps
;
; OUTPUT:
;   xrs_engtemps
;   xrs_engpos
;   xrs_engvolts
;   xrs_engstats
;
; OPTIONAL CONTROL KEYWORD PARAMETERS:
;   plot_all
;
; USAGE:
;   telem2eng, xrs_tlmstats, xrs_tlmpos, xrs_tlmvolts, xrs_tlmtemps,
;              xrs_engstats, xrs_engpos, xrs_engvolts, xrs_engtemps,
;              plot_all = plot_all
;
; CALLED BY:
;   doit4goesi.pro
;
; CALLS:
;   goesi_vcurve  for conversions to engineering units
;
; ALGORITHM:
;    applies the goesi_vcurve calibrations
;    NOTE: does not handle eclipse properties, since these are not
;    simple conversions
;
; NOTE: The SAS (sun angle sensor) may required thermal correction factor.  This
;    has not been included in the telemetry-to-engineering-unit conversions.
;
; MODIFICATION HISTORY:
;  Extracted from goesi_subcom.c 4/10/00 by P.L. Bornmann
;     to convert telemetry to engineering data within IDL
;  Modifications 4/17/00 by P.L. Bornmann
;     Activated plot_all for plots
;  Modification 8/10/00 by P.L. Bornmann
;     Added data range to title of plots
;
; FUTURE MODIFICATIONS:
;   sattde = sattde,
;   NEED TO DEAL WITH ARRAYS VS. STRUCTURES OF THE VARIABLES
;   need curve number for XRP electronics temperature
;
; QUESTIONS:
;   Is XRS reference voltage -70 or -75?
;   Double check XRPE temp
;   Range of TLM calib voltage, and its meaning.
;   Get more info on word 36
;
; ACRONYMS:
;   XRS  X-Ray Spectrometer
;   XRP  X-Ray Positioner
;   SAS  Sun Angle Sensor
;
; CALIBRATION DOCUMENTATION
; word 101 sub  3,19  [ 3(16)]  XRS preamp temperature    curve 233 xrs_engtemps[0]
; word 101 sub  1,17  [ 1(16)]  sun analog sensor temp    curve 233 xrs_engtemps[1]
; word 101 sub  4,20  [ 4(16)]  XRS positioner temp       curve 233 xrs_engtemps[2]
; word 101 sub  2,18  [ 2(16)]  XRS position bearing t    curve 233 xrs_engtemps[3]
;
; XRS operation status
;   word 52 bit 8 sub [14(16)] XRS on = 0, off = 1      xrs_engstats[0]
;   xrs_engstats 1 for XRS on, 2 for off
;
; word 52 sub [ 7(16)] XRS sun not present = 0,
;                      sun acquired = 1        bit 8  xrs_engstats[1]
;   xrs_engstats 1 for solar obs, 2 for no sun
;
; XRS calibration status
;   word 52 bit 8 sub [ 8(16)] XRS cal = 0, data = 1    xrs_engstats[2]
;   xrs_engstats 1 for data, 2 for cal
;
; word 101 sub  5,21  [ 5(16)]  XRS positioner drive ele  curve 233 xrs_engtemps[4]
;
; XRS slew status
;    word 52 sub [ 5(16)] xrpe slew = 0,
;                                 track = 1     bit 8   xrs_engstats[3]
;    xrs_engstats 1 for solar track, 2 for slewing
;
; XRS slew direction
;   word 52 sub [ 6(16)] xrpe slew north = 0,
;                                slew south = 1    bit 8   xrs_engstats[4]
;   xrs_engstats 1 for slew north, 2 for slew south
;
; word 100 sub  6  SAS (N/S) curve 380  xrs_engpos[0]
;   may need temperature compensation
;
; word 100 sub  7  SAS (E/W) curve 381  xrs_engpos[1]
;   may need temperature compensation
;
; coarse N-S XRP position, -94 to 28 degrees   curve 379
;   word 100 sub  5  XRS coarse position       curve 379  xrs_engpos[2]
;
;
; digital status, including SATTDE 2
;N  word  36 sub 10,26 [10(16)] incl autoload             digital
;N               16,32 [16(16)] incl SATTDE 2             digital
;
; word  72 sub 29  XRS ref voltage    curve 378  xrs_engvolts[0]
; word  72 sub 30  XRS -70 volt bias  curve 382  xrs_engvolts[1]
;   (the 75 volt chamber bias voltage)
; word  72 sub 32  tlm calib voltage  curve 220  xrs_engvolts[2]
;-

;#include "goesi_read.h"

    if (not(keyword_set(plot_all))) then plot_all = 0

    ; The curves to use for calibrations
    crvtag_temps = ['tpreamp', 'tsas', 'txrp', 'txrpbearing', 'txrpelectronics']
    crv_engtemps  = [233, 233, 233, 233, 233]

    crvtag_volts = ['vref', 'vchamber', 'vcal']
    crv_engvolts = [378, 382, 220]
```

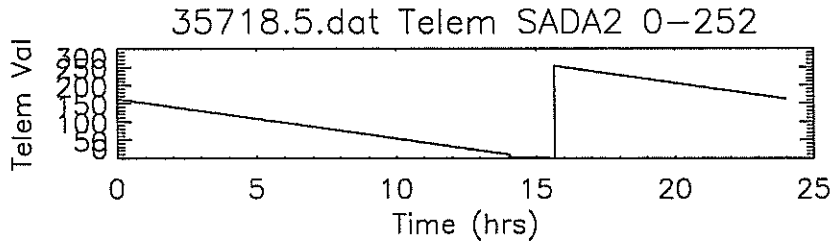
```
crvtag_pos   = ['sada1', 'sada2', 'nsxrp', 'ewxrp', 'ewcoarse']
crv_engpos   = [003, 004, 380, 381, 379]

; The conversion values for PLB conversions
crvtag_stats  = ['on', 'sun', 'cal', 'slew', 'dir']
xrs_engstats_const = [1,  2,  2,  2, 1]
xrs_engstats_mult  = [1, -1, -1, -1, 1]

; Structure definitions for results
xrsstatstr = {engstatstr, on:-1, sun: -1, cal:-1, slew:-1, dir:-1}
xrsposstr  = {engposstr , sada1      :-99.9, sada2      :-99.9, $
                          nsxrp      :-99.9, ewxrp      :-99.9, ewcoarse:-99.9}
xrsvoltstr = {engvoltstr, vref       :-99.9, vchamber   :-99.9, vcal     :-99.9}
xrstempstr = {engtempstr, tpreamp    :-99.9, tsas       :-99.9, txrp     :-99.9, $
                          txrpbearing:-99.9, txrpelectronics:-99.9}

; New structures for the results
xrs_engstats = replicate(xrsstatstr, n_elements(xrs_tlmstats))
xrs_engpos   = replicate(xrsposstr , n_elements(xrs_tlmpos  ))
xrs_engvolts = replicate(xrsvoltstr, n_elements(xrs_tlmvolts))
xrs_engtemps = replicate(xrstempstr, n_elements(xrs_tlmtemps))

; Apply the PLB conversion
ntags = n_tags(xrs_tlmstats)
tlmtagname = tag_names(xrs_tlmstats)
engtagname = tag_names(xrs_engstats)
for itlmtag = 0, ntags-1 do begin
  iengtag = where(tlmtagname(itlmtag) eq engtagname, nwengtag)
  icrvtag = where(tlmtagname(itlmtag) eq crvtag_stats, nwcrvtag)
  if (nwengtag ne 1 or nwcrvtag ne 1) then begin
    print, 'telem2eng.pro: structure tag name not supported ', $
              tlmtagname(itlmtag), 'valid eng tags are ', engtagname
  endif else begin
    xrs_engstats.(iengtag) = xrs_engstats_const[crvtag] + $
        xrs_engstats_mult[icrvtag] * xrs_tlmstats.(itlmtag)
  endelse
endfor

; Process structures that have calibration curves
for istruct = 0, 2 do begin
   case istruct of
      0:  vartxt = "temps"
      1:  vartxt = "pos"
      2:  vartxt = "volts"
   endcase
   cmd1 = 'eng_struct = xrs_eng' + vartxt
   cmd2 = 'tlm_struct = xrs_tlm' + vartxt
   cmd3 = 'crv = crv_eng' + vartxt
   cmd4 = 'crvtagname = crvtag_' + vartxt
   ok = execute(cmd1)
   ok = execute(cmd2)
   ok = execute(cmd3)
   ok = execute(cmd4)
   ntags = n_tags(tlm_struct)
   tlmtagname = tag_names(tlm_struct)
   engtagname = tag_names(eng_struct)
   crvtagname = strupcase(crvtagname)
   !p.multi = 0
   !p.multi(1) = 2
   !p.multi(2) = ntags
   for itlmtag = 0, ntags-1 do begin
     iengtag = where(tlmtagname(itlmtag) eq engtagname, nwengtag)
     icrvtag = where(tlmtagname(itlmtag) eq crvtagname, nwcrvtag)
     if (nwengtag ne 1 or nwcrvtag ne 1) then begin
       print, 'telem2eng.pro: structure tag name not supported ', $
              tlmtagname(itlmtag)
       help, eng_struct, /str
       help, tlm_struct, /str
       STOP
     endif else begin
       tlm    = tlm_struct.(itlmtag)
       tlmcrv = crv(icrvtag(0))
       eng    = goesi_vcurve (tlm, tlmcrv)
       print, 'telem2eng.pro:  Will plot ', vartxt + '.' + tlmtagname(itlmtag),$
              ' Curve ', tlmcrv
       if (plot_all eq 1) then begin
          rtlm = [min(tlm[32:*]), max(tlm[32:*])]
          reng = [min(eng[32:*]), max(eng[32:*])]
          rtlm = [min(tlm[32:*]), max(tlm[32:*])]
         trtlm = string (format = "('(',i5,'-', i5,')')", rtlm[0], rtlm[1])
         treng = string (format = "('(',g10.3,'-', g10.3,')')", reng[0], reng[1])
         trtlm = strcompress(trtlm)
         treng = strcompress(treng)
         plot, tlm[32:*], title = vartxt + '.' + tlmtagname(itlmtag) + ' Telem ' +
trtlm, $
               xtitle = 'Data Point Number', ytitle = 'Telem ' + tlmtagname(itlmtag)
         plot, eng[32:*], title = vartxt + '.' + tlmtagname(itlmtag) + ' Calib ' +
treng, $
               xtitle = 'Data Point Number', ytitle = 'Eng. ' + tlmtagname(tlmtag)
       endif
       eng_struct.(iengtag(0))  = eng
     endelse
     ; Pass back the structure
     cmd = 'xrs_eng' + vartxt + ' = eng_struct'
     ok = execute(cmd)
   endfor
   ;wait_user
 endfor

  ; end of telem2eng
RETURN
END
```

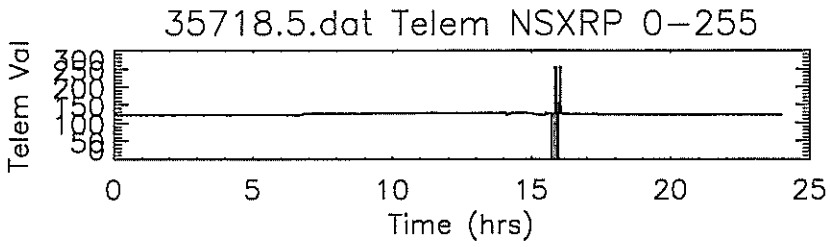
```
FUNCTION xrsi_conv, satnum, tlm_fluxes, xrs_fluxes
;+  *    *                xrsi_conv.c
;
; PURPOSE:
;   Original purpose (L. Matheson's c code) was to decode most of
;   the xrs related spacecraft words and status push data on to
;   averaging.  This version (P.Bornmann's) only converts to x-ray fluxes.
;
;  INPUT
;    satnum  the satelite number (e.g 9 for GOES-9/I)
;
;  OUTPUT:
;    (see COMMON)
;
;  COMMON BLOCKS:
;     xrsi_conv
;
;  REFERENCES:
;       Telemetry Reference is Goes IJK/LM DRL 0-14 Telemetry and
;       Command lists
;       Instrument and spacecraft reference is Goes Program,
;       Spacecraft Operations Handbook, Volume III, Spacecraft
;       Description, DRL 503-02, Goes I-M
;
;       Telemetry conversion coefficients are in Space Operations
;       Handbook, Volume VI, Spacecraft Data, DRL 503-02, Goes I and
;       Goes-I Data and Calibration Report for Space Environment
;       Monitor, DRL 311-01 as well as the Panametrics Calibration
;       Report
;
;  MODIFICATION HISTORY
;      started 5 Aug 94, ldm
;      95 Mar 11  eclipse suppression, housekeeping  ldm
;         May 16  added Goes-9 xrs coefficients
;         Dec 20  corrected error in g9 using g8 kx,
;                 updated kx to new coefficients
;      96 Jan  3  sun lock log entry, new FAC entries
;              8  new FAC for long and short
;      96 Jul 18  fixed logging error in range changes
;      96 aug 20  $ $ PATCH to suppress xrs on Goes-9 during
;         motor saving maneuvers, sas checking
;      97 Feb 24  Goes-10 added, no xrs_pos
;      98 Oct  6  redo startup of xrs, make longer
;      1999-02-12 LDL: add Goes 11, cloned from Goes 8
;      11/24/99 P.L. Bornmann adding explanatory comments
;  Modifications 4/27/00 by P.L. Bornmann
;    converted to IDL
;    pass satelite number
;    removed calls to gi_ave_datum.  This is C++ object code
;      LDL wrote to calculate the average
;    remove test for first call (had used static variable start)
;      since this code will be used to process full array and
;      not called many times.
;    remove discard of flux data during cals, not sun locked,
;      in eclipse, sas angle > 1.2
;    Modifications 5/9/00 by P.L. Bornmann
;      correction factors as array, factor used is based on supplied
;      satellite number
;      removed subcommed extractions that are done by goesi_subcom.c
;      removed conversions done by goesi_vcurve.c
;      removed extractions done by print_xrs.c
;      removed the code that replaced fluxes with no data value when
;      xrs was not in sun-observing mode
;   Modification 5/10/00 by P.L. Bornmann
;      PLB removed the NO_DATA values for times when XRS flux
;      is not valid solar flux, because these values are needed
;      for instrument analysis.
;      removed calls to writei_log
;   Modifications 5/17/00 by P.L. Bornmann
;      preparing to interface with doit4goesi.pro
;      pass values of tlm_fluxes and xrs_fluxes
;
; QUESTIONS:
;  Is g_stat a global variable?  Not defined prior to this.
;  Are the temperature coefficients constant for all s/c, as
;             assumed here.  Probably are if used same thermistors
; USAGE:
;   Must set PREPROCESSOR VALUE FOR GSAT during compile
;
; FUTURE MODIFICATIONS:
;   PLB may want to make the conversion coefficients into
;   arrays that include the spacecraft, rather than create different
;   code for each s/c
;   Case statment might be more efficient because it has breaks
;   Looks like g_stat.x_sh_ra is previous short-wavelength range
;              g_stat.xrs_sgain counter for RC transient
;   upgrade treatment of dwell points
;   get info from Lorne re tmes of noise, if this is not included
;   in the code alread available
;
; CALLS:
;   calc_xrs_coef
;
; ALGORITHM:
; Looks like also set to treat change to calibration as RC transient
;-

    ;******************
    ;*  Declarations  *
    ;******************

    ; The c version had includes for
    ; goesiave.h, Goesi_channels.h, goesiraw.h,
    ; goesi_proto.h, goesi_subcom.h

    ; Factor used to smooth telemetered values to represent the
    ; XRS sensor temperature
    FAC_T     = 0.05

    ; value used to remove RC transient after range change (same value for
    ; short and long channels
    XRS_SW    =  8
    XRS_SW_ON = 13

    ; Factors used to make fluxes agree with those from the
    ; spinning GOES
    satnums    = [   8,    9,   10,   11]
    FAC_LONGS  = [0.70, 0.70, 0.70, 0.70]
    FAC_SHORTS = [0.85, 0.85, 0.85, 0.85]
    ; long_f is not used, were these correction factors used for the spinners?
    ; long_f  = [0.766, 0.808, 0.821, 0.830]
    ; short_f = [0.894, 0.898, 0.940, 0.940]

    ; Determine which correction factor applies based on
    ; supplied satellite number
    ; This assumes only one satnum is supplied
    isat = where(satnums eq satnum, nsat)
    if (nsat le 0) then begin
      print, 'xrsi_conv.pro: invalid satelite number ', satnum
      print, '               valid values are ', satnums
```

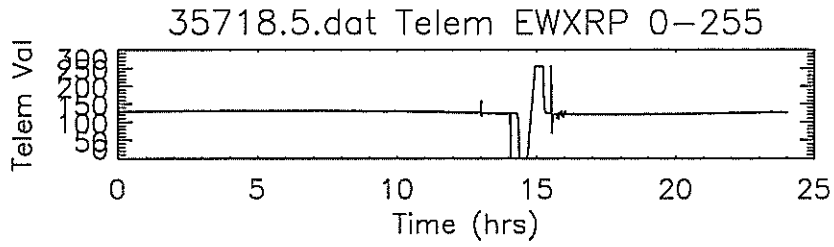
```
      STOP
    endif
    FAC_LONG  = FAC_LONGS (satnums(isat))
    FAC_SHORT = FAC_SHORTS(satnums(isat))

static float xrs_coef[2][2];      ;   first index 0-short, 1-long
                                  ;   second      0-volt offset
                                  ;               1-sensitivity


;    COMMON xrsi_conv, la_xrs_cal , la_xrs_on ,  $
;                      la_xrp_slew,  $
;                      sens_temp, temp_in,  $
;                      kss  , ksl  , temp_coef
;
;    la_xrs_cal = -1;    ;   holds the last cal/data bit
;                        ;   0 is calibrate, 1 is data/off
;    la_xrs_on  = -1;    ;   holds the last on/off bit
;                        ;   0 is on, 1 is off
;    la_xrp_slew  = -1;  ;   0 north, 1 south   ;/
;
;    ;;start = 1;        ;   first-time flag

    sens_temp = 4.

    ; Seems like this quantization level would vary for different instruments -- PLB
    ; but that not important, since this is only used as a lower limit
    kss  = [ 2.238e-9]     ; quantization level of lowest short range
    ksl  = [ 9.314e-9]     ; quantization level of lowest long  range

    ; temperature coefficients and conversion will become external
    ; to this function

    ;flcat longx, shortx
    ;float tdat;            ; variable to hold data temporarily
    ;float xrs_ion;         ;   xrs ion chamber voltage

    ;************************
    ;**  Start Processing  **
    ;************************

;;    ; Not valid data if in dwell mode
;;    if (g_stat.dwell) then RETURN, 0
;;
;;    ; Count-down history counters
;;    if (g_stat.xrs_lgain ge 0) then  g_stat.xrs_lgain = g_stat.xrs_lgain - 1
;;    if (g_stat.xrs_cal   ge 0) then  g_stat.xrs_cal   = g_stat.xrs_cal   - 1
;;    if (g_stat.xrs_off   ge 0) then  g_stat.xrs_off   = g_stat.xrs_off   - 1

    ; Frame number modulo 16, means ignore bits above 16
    frame16 = g_stat.frame & 0x0f
    if (frame16 eq 2) then begin
        ; get sensor temperature and smooth
        tdat = g_raw.raw_w[101]
        ; Average in the change of the new temperature with the
        ; previous average
        sens_temp = sens_temp + FAC_T * (temp_in - sens_temp)
    endif

    ; xrs coarse position (n-s)
    ; PLB NOTES THAT goesi_vcurve DID NOT HAVE
    ; SPACECRAFT-DEPENDENT CONVERSION FACTORS
#if    (GSAT eq 8) then begin
        xrs_pos = 28.37270 - tdat*0.4816012
endif else (if eq 9) then then begin
        xrs_pos = 28.00    - tdat*0.481601
endif else (if eq 10) then then begin
        xrs_pos = 28.00    - tdat*0.481601
endif else (if eq 11) then then begin
        xrs_pos = 28.37270 - tdat*0.4816012
endif

    if (g_stat.frame eq 29) then begin
       ;   get xrs ion chamber voltage
;;  Looks like the bias voltage from goesi_vcuve
;;  Should verify
        xrs_ion = -0.2000 - tdat*0.4000
     endif

    if ((lo_ra ne g_stat.x_lo_ra) or (sh_ra ne g_stat.x_sh_ra) or $
        (frame16 eq 2)) then begin
        ; Get new flux conversion coefficients after range change
        ; for both short and long channels.
        calc_xrs_coef(sens_temp, sh_ra, lo_ra)
    endif

    ;   look at status bits for current xrpe track/slew status
    bits = g_raw.raw_w[52] & 1

;;    if ((la_xrs_cal eq 0) and (g_stat.xrs_cal le XRS_SW)) then begin
;;        g_stat.xrs_cal = XRS_SW
;;
;;    if ((la_xrs_on eq 1) and (g_stat.xrs_off le XRS_SW_ON)) then $
;;        g_stat.xrs_off = XRS_SW_ON
;;
;;    if (g_stat.xrs_off ge 0) then begin
;;        g_stat.stat2 = g_stat.stat2 | 1
;;        RETURN, 0
;;    endif

    ;*****************************
     * Calculate the X-ray Fluxes *
     *****************************
    ;   calculate short flux   ASK LOREN ABOUT THIS
    if ((g_raw.raw_w[56] eq 0) or (g_raw.raw_w[56] eq 255)) then $
        g_stat.stat2 = g_stat.stat2 | 020

    ;   calculate long flux
    if ((g_raw.raw_w[57] eq 0) or (g_raw.raw_w[57] eq 255)) then $
        g_stat.stat2 = g_stat.stat2 | 010


    ; PLB NOTES THAT ALL THE OTHER CAL CURVES (e.g. goesi_vcurve.pro)
    ; USE EXPRESSION val = const + mult * telem, BUT THIS USES
    ; EXPRESSION val = (telem - const ) * mult

    ; Lorne Matheson says (May 2000) that the 0.02
    ; comes from the standard voltage curve (1767) at the
    ; very start of the SOH.  He omitted the constant 0.01
    ; because it is only included in the SOH to shift the
    ; value from that at the start of the range to that of the midpoint
    ; The used of this conversion for XRS fluxes is not documented in
    ; the SOH or the Panametrics calibration reports.
    ; This step converts telemetry to voltage

    ; Convert flux telemetry to engineering voltages
    ;shortx = g_raw.raw_w[56] * 0.02
    ;longx  = g_raw.raw_w[57] * 0.02
    eng_fluxes.shflux = tlm_fluxes.shflux * 0.02
    eng_fluxes.lflux  = tlm_fluxes.lflux  * 0.02
```

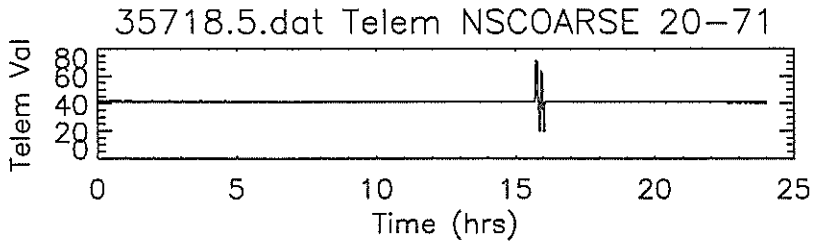
```
    ; Looks like x-ray flux = (voltage - xrs_coef constant)/ xrs_coef multiplier
    ; Convert XRS voltage to x-ray flux
    ;shortx = (shortx - xrs_coef[0,0])* xrs_coef[0,1]
    ; Looks like x-ray flux = (voltage - xrs_coef constant)/ xrs_coef multiplier
    ;longx = (longx - xrs_coef[1,0])* xrs_coef[1,1]
    xrs_fluxes.shflux = (eng_fluxes.shflux - xrs_coef[0,0])* xrs_coef[0,1]
    xrs_fluxes.lflux  = (eng_fluxes.lflux  - xrs_coef[1,0])* xrs_coef[1,1]

    ; Correct for the spacecraft normalization (FAC_SHORT, FAC_LONG)
    ;shortx = FAC_SHORT * shortx
    ;longx  = FAC_LONG  * longx
    xrs_fluxes.shflux = FAC_SHORT * xrs_fluxes.shflux
    xrs_fluxes.lflux  = FAC_LONG  * xrs_fluxes.lflux

    ; minimum flux is .4 of the nominal quantization interval of
    ; the most sensitive range         ?? WHY 40%??
    min_short = 0.4 * kss
    min_long  = 0.4 * ksl

    ; Fluxes truncated to the lowest quantization level
    xrs_fluxes.shflux = xrs_fluxes.shflux > min_short
    xrs_fluxes.lflux  = xrs_fluxes.lflux  > min_long
    ;shortx = shortx > min_short
    ;longx  = longx  > min_long

    ; WHAT ARE THESE???
;;    if (g_stat.xrs_lgain ge 0) then begin
;;        g_stat.stat2 = g_stat.stat2 | 040
;;        longx = NO_DATA
;;    endif
;;    if (g_stat.xrs_sgain ge 0) then begin
;;        g_stat.stat2 = g_stat.stat2 | 0100
;;        shortx = NO_DATA
;;    endif

    ; Times when XRS flux is not valid,
    ; PLB removed the NO_DATA values for these points,
    ; because they are needed for instrument analysis.
    ;    xrs calibrating if (g_stat.xrs_cal ge 0)
    ;   is n-s pointing locked if (la_xrs_sun eq 0)
    ;   are we in eclipse   bits = (int) (g_stat.stat2 & 01000) if (bits ne 0)
    ;   check n-s, e-w sas if ((sas_ns ge 1.2) or (sas_ns le -1.2))
    ;   if (sas_ns ge -4.) then g_stat.stat2 = g_stat.stat2 | 0400
    ;   if ((sas_ew ge 1.2) or (sas_ew le -1.2))
    ;   if (sas_ew ge -4.) then g_stat.stat2 = g_stat.stat2 | 0400

RETURN, 0
END
```

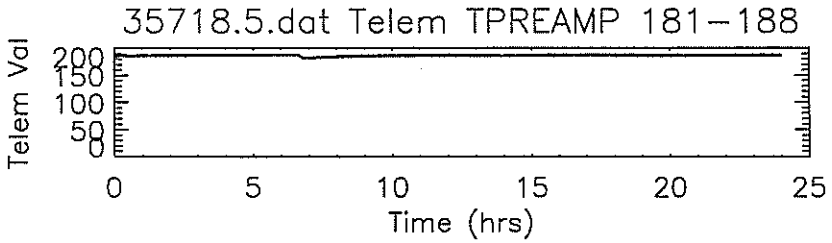

35718.5.dat Telem SADA1 0-253
Telem Val
0
5
10
15
20
25
Time (hrs)

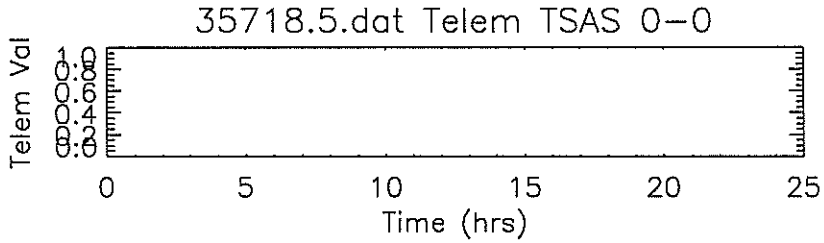

35718.5.dat Telem SADA2 0-252
Telem Val
0
5
10
15
20
25
Time (hrs)

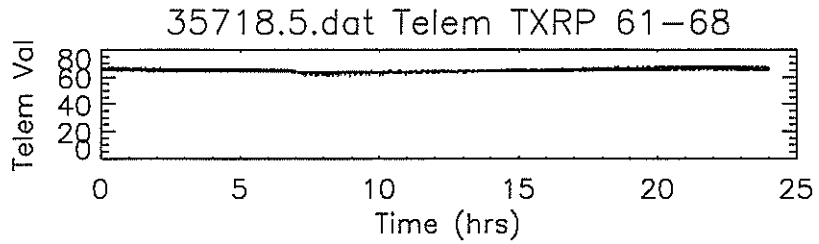

35718.5.dat Telem NSXRP 0-255
Telem Val
0
5
10
15
20
25
Time (hrs)

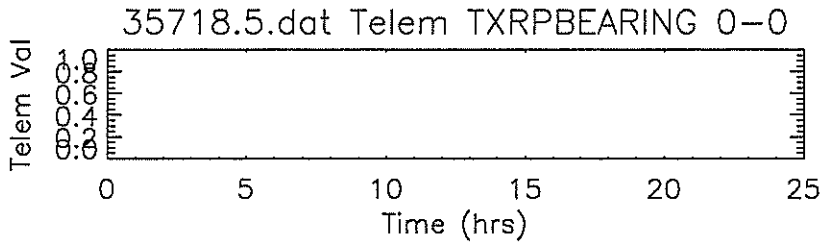

35718.5.dat Telem EWXRP 0-255
Telem Val
0
5
10
15
20
25
Time (hrs)

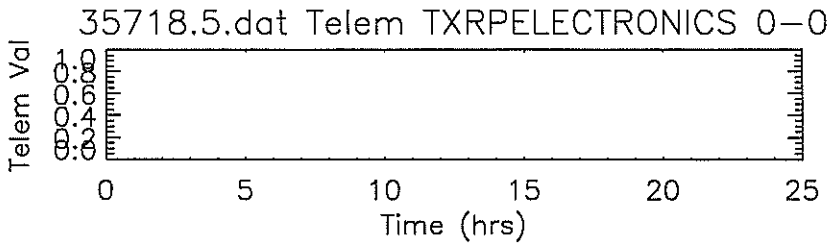

35718.5.dat Telem NSCOARSE 20-71
Telem Val
80
60
40
20
0
0
5
10
15
20
25
Time (hrs)

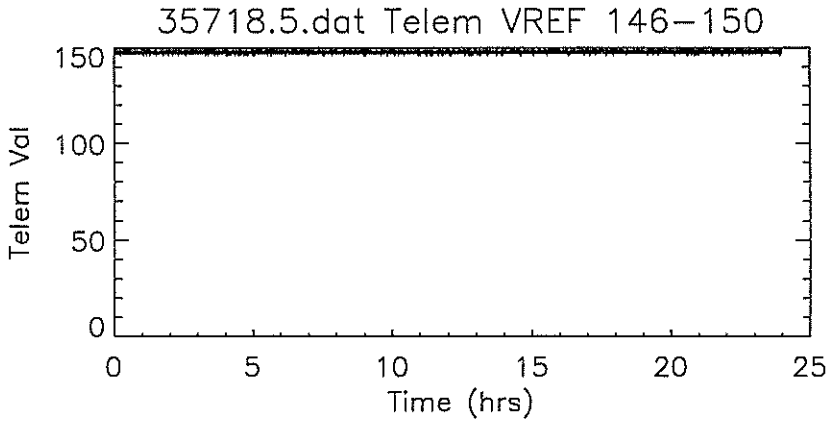

35718.5.dat Telem TPREAMP 181–188
Telem Val
200
150
100
50
0
0
5
10
15
20
25
Time (hrs)

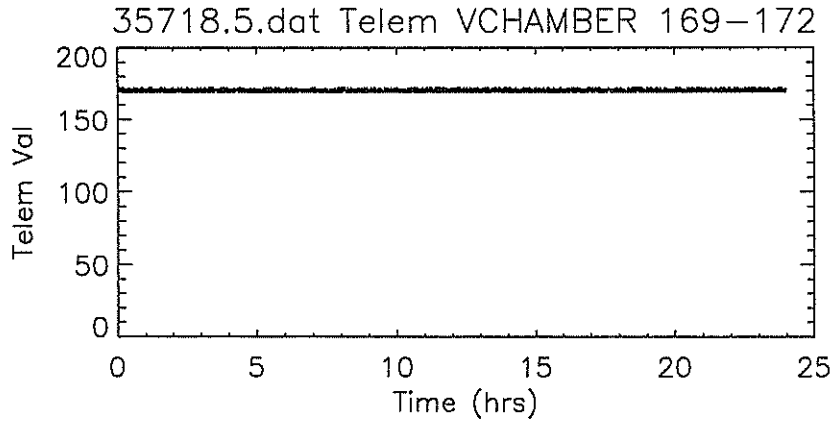

35718.5.dat Telem TSAS 0–0
Telem Val
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
Time (hrs)

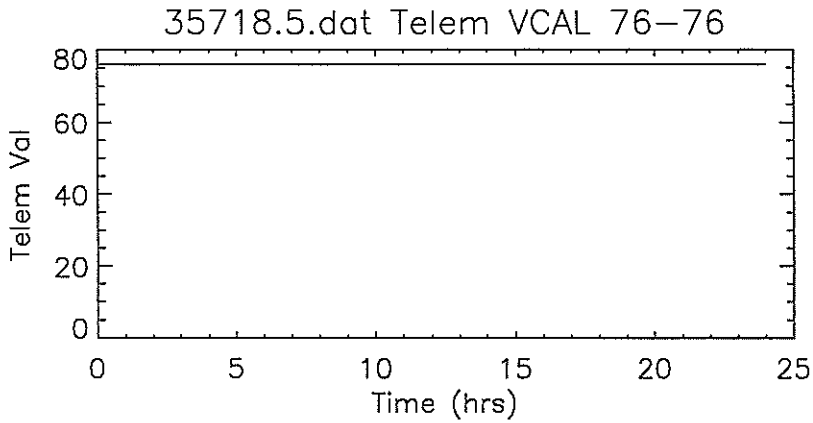

35718.5.dat Telem TXRP 61–68
Telem Val
80
60
40
20
0
0
5
10
15
20
25
Time (hrs)

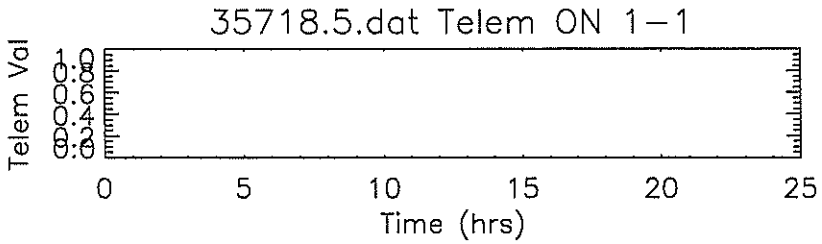

35718.5.dat Telem TXRPBEARING 0–0
Telem Val
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
Time (hrs)

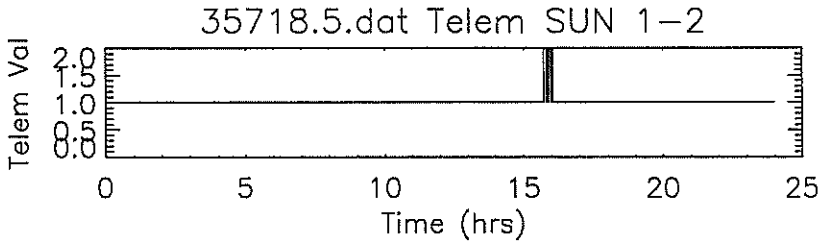

35718.5.dat Telem TXRPELECTRONICS 0–0
Telem Val
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
Time (hrs)

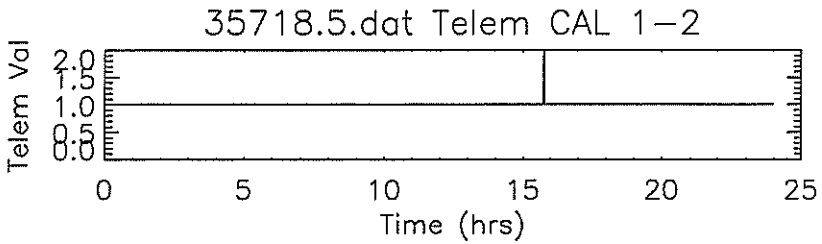

35718.5.dat Telem VREF 146−150
Telem Val
150
100
50
0
0
5
10
15
20
25
Time (hrs)

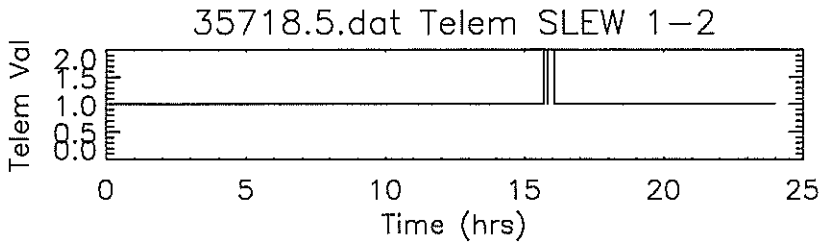

35718.5.dat Telem VCHAMBER 169−172
Telem Val
200
150
100
50
0
0
5
10
15
20
25
Time (hrs)

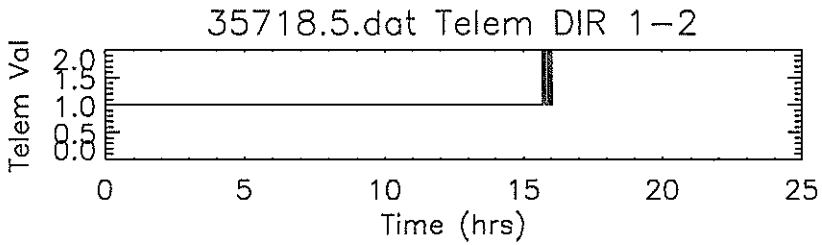

35718.5.dat Telem VCAL 76−76
Telem Val
80
60
40
20
0
0
5
10
15
20
25
Time (hrs)

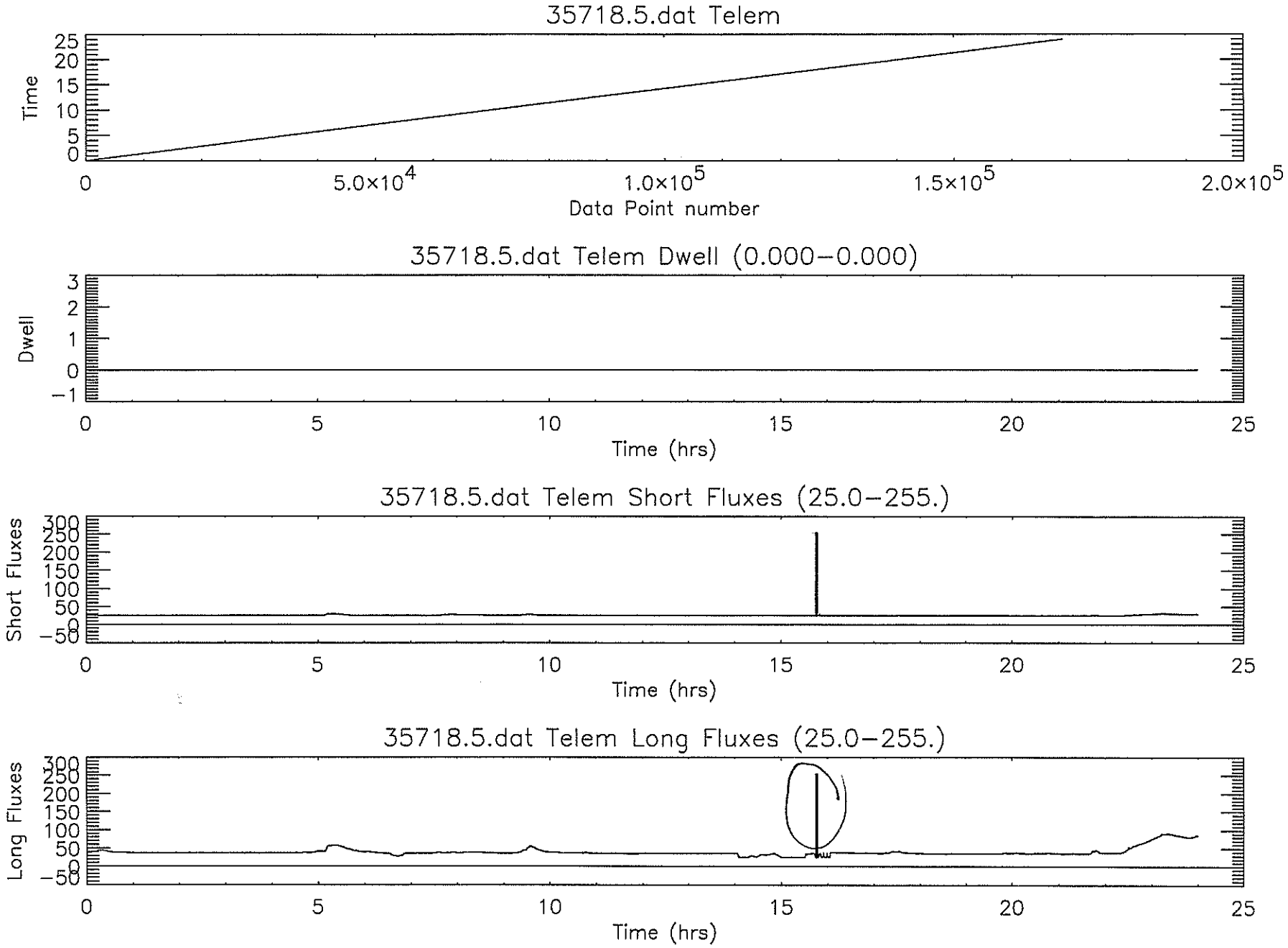

35718.5.dat Telem ON 1−1
Telem Val
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
Time (hrs)

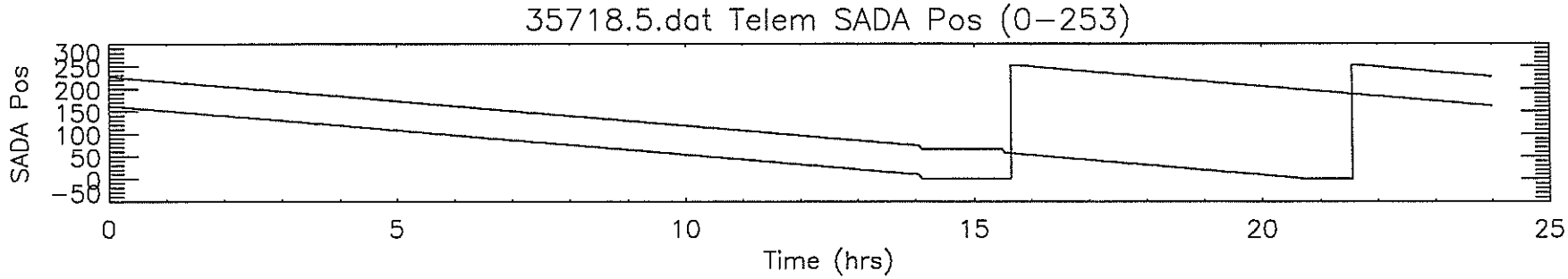

35718.5.dat Telem SUN 1−2
Telem Val
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
Time (hrs)

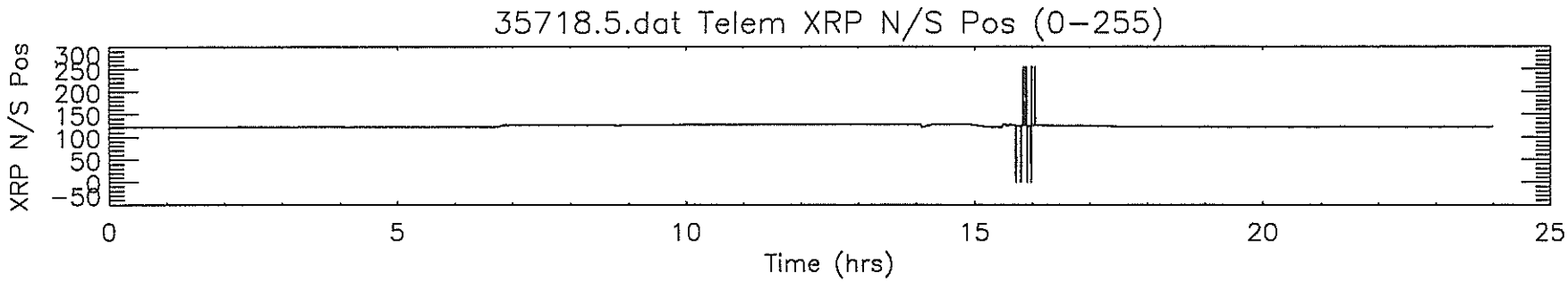

35718.5.dat Telem CAL 1−2
Telem Val
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
Time (hrs)

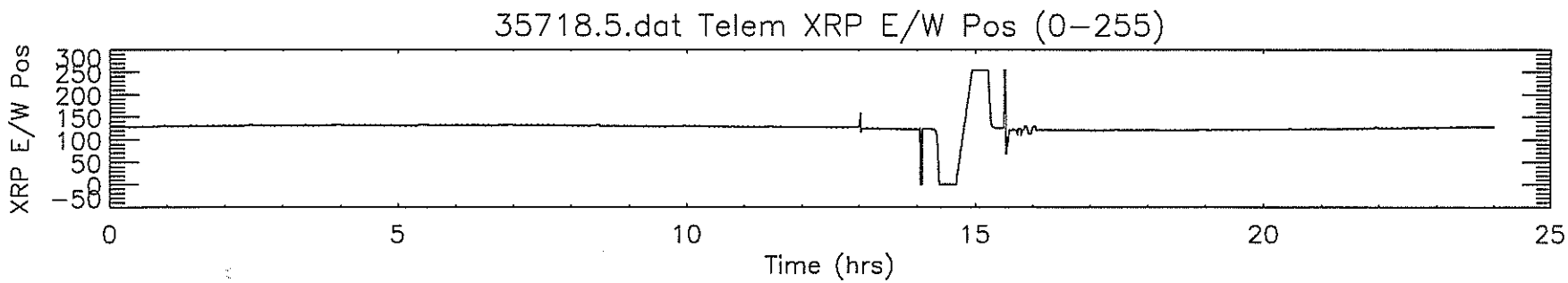

35718.5.dat Telem SLEW 1−2
Telem Val
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
Time (hrs)

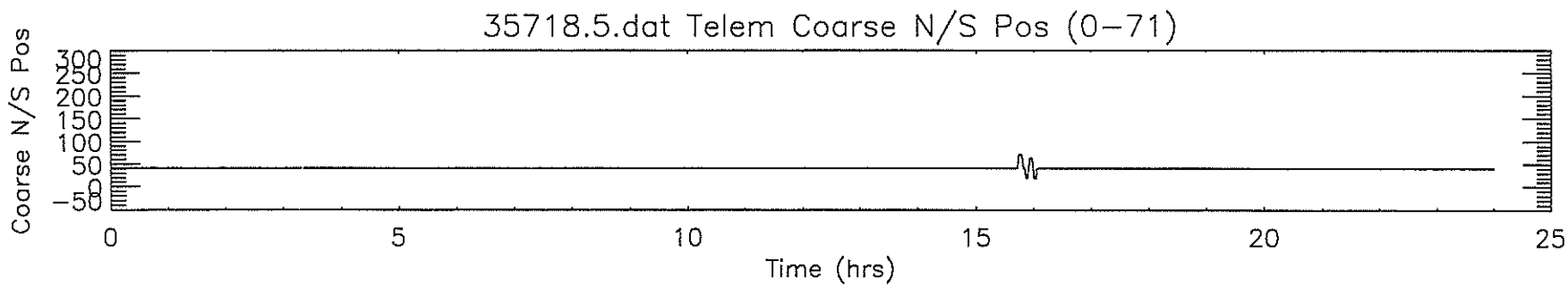

35718.5.dat Telem DIR 1−2
Telem Val
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
Time (hrs)

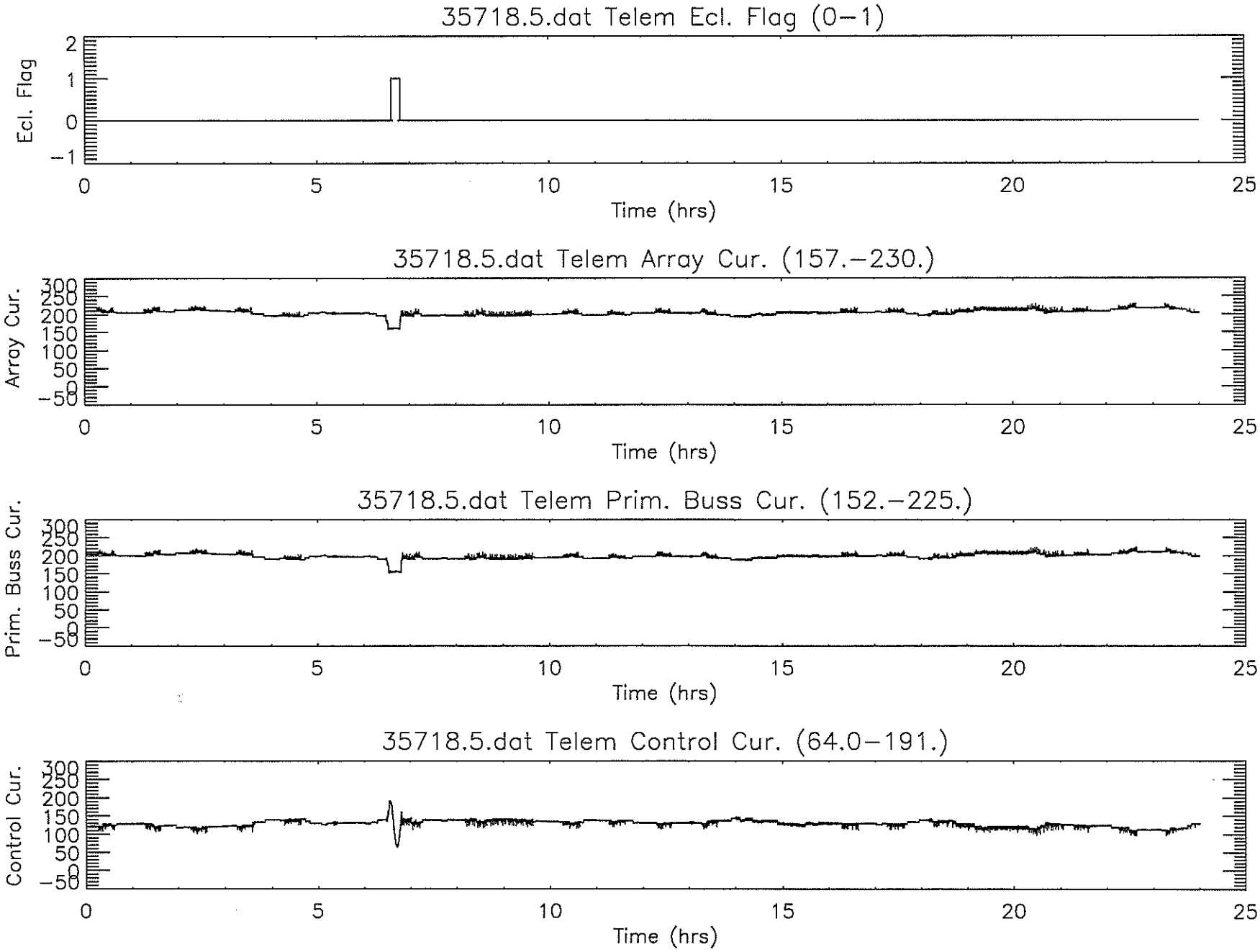

35718.5.dat Telem
Time
25
20
15
10
5
0
0
5.0×10^4
1.0×10^5
1.5×10^5
2.0×10^5
Data Point number
35718.5.dat Telem Dwell (0.000–0.000)
Dwell
3
2
1
0
−1
0
5
10
15
20
25
Time (hrs)
35718.5.dat Telem Short Fluxes (25.0–255.)
Short Fluxes
300
250
200
150
100
50
0
−50
0
5
10
15
20
25
Time (hrs)
35718.5.dat Telem Long Fluxes (25.0–255.)
Long Fluxes
300
250
200
150
100
50
0
−50
0
5
10
15
20
25
Time (hrs)

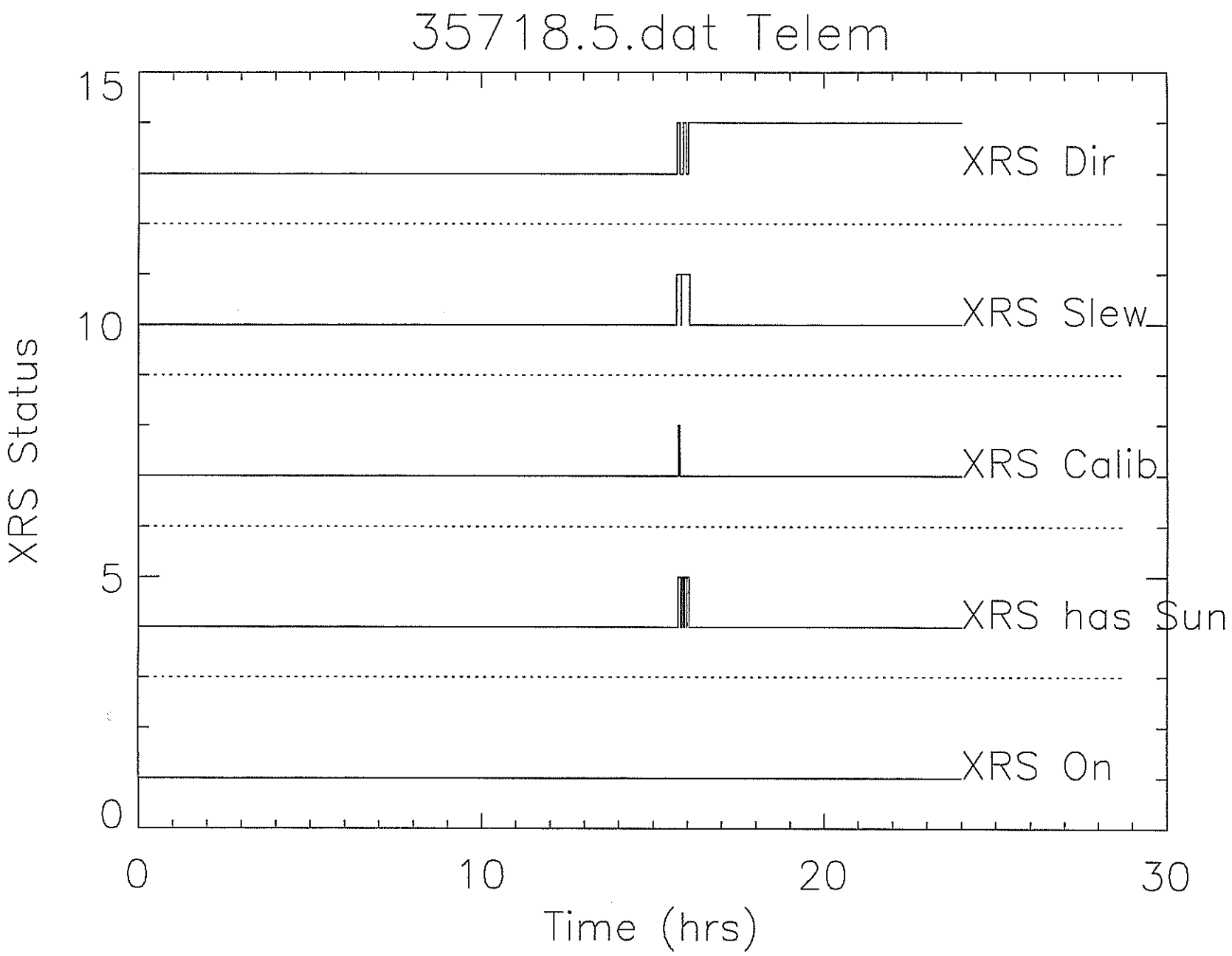

35718.5.dat Telem SADA Pos (0-253)
SADA Pos
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

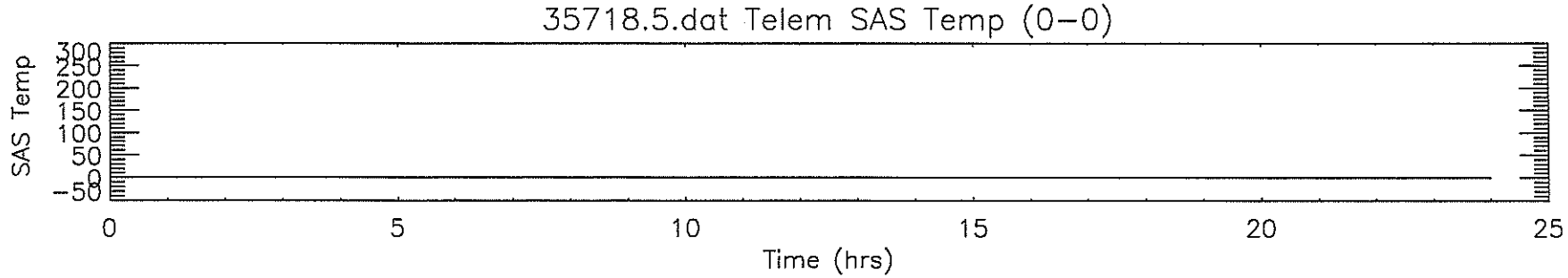

35718.5.dat Telem XRP N/S Pos (0-255)
XRP N/S Pos
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

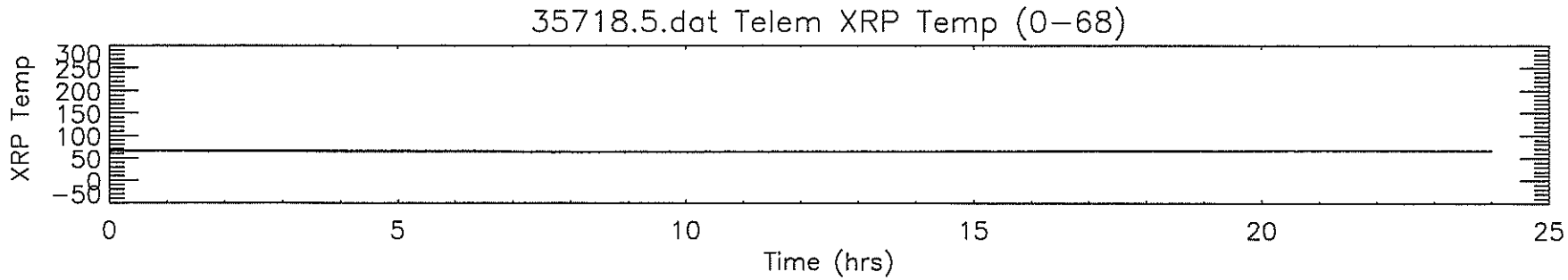

35718.5.dat Telem XRP E/W Pos (0-255)
XRP E/W Pos
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

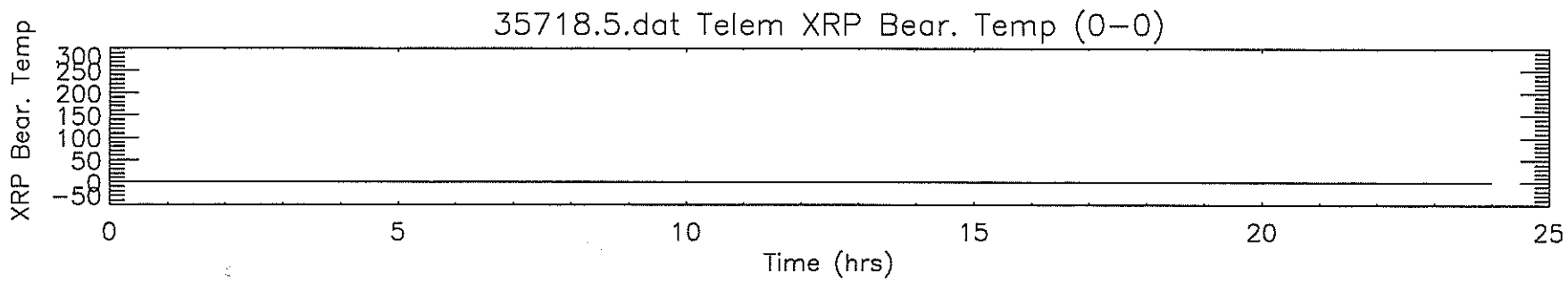

35718.5.dat Telem Coarse N/S Pos (0-71)
Coarse N/S Pos
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

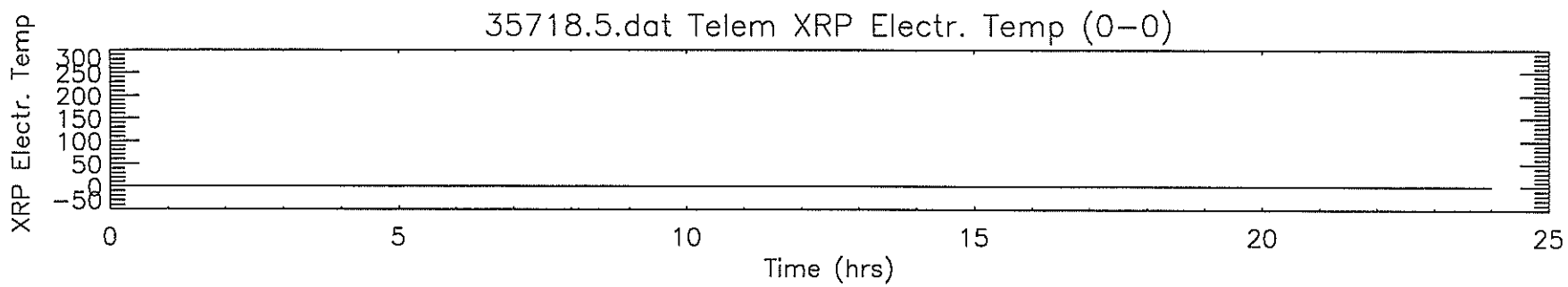

35718.5.dat Telem Ecl. Flag (0−1)
Ecl. Flag
2
1
0
−1
0
5
10
15
20
25
Time (hrs)
35718.5.dat Telem Array Cur. (157.−230.)
Array Cur.
300
250
200
150
100
50
0
−50
0
5
10
15
20
25
Time (hrs)
35718.5.dat Telem Prim. Buss Cur. (152.−225.)
Prim. Buss Cur.
300
250
200
150
100
50
0
−50
0
5
10
15
20
25
Time (hrs)
35718.5.dat Telem Control Cur. (64.0−191.)
Control Cur.
300
250
200
150
100
50
0
−50
0
5
10
15
20
25
Time (hrs)

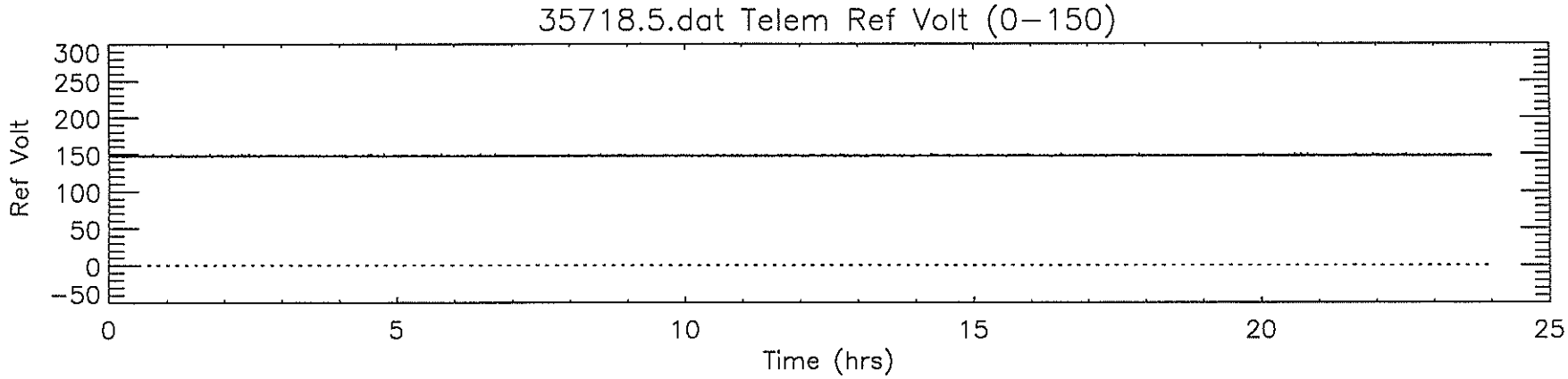

35718.5.dat Telem
XRS Status
15
10
5
0
XRS Dir
XRS Slew
XRS Calib
XRS has Sun
XRS On
0
10
20
30
Time (hrs)

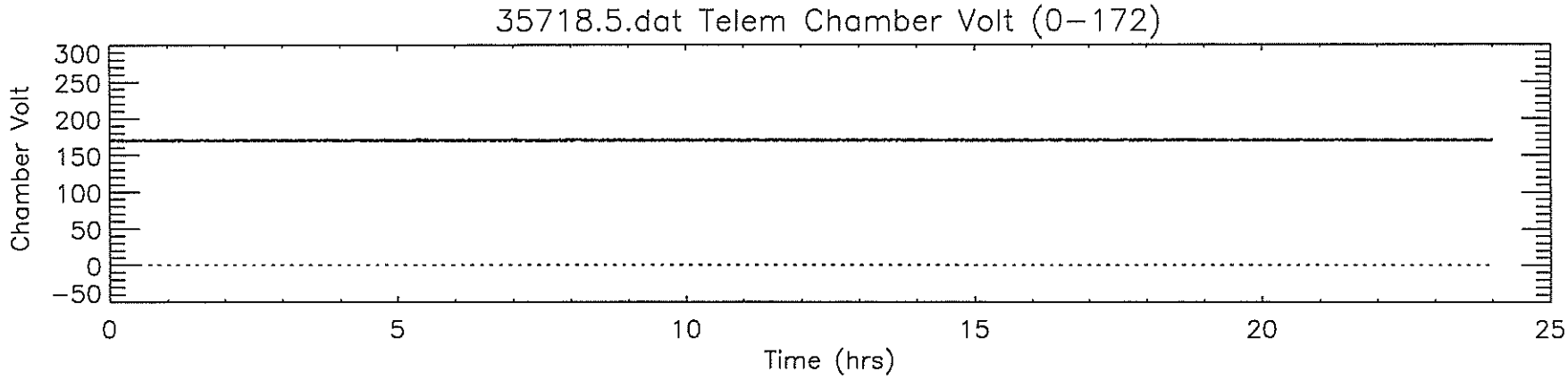

35718.5.dat Telem SAS Temp (0-0)
SAS Temp
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

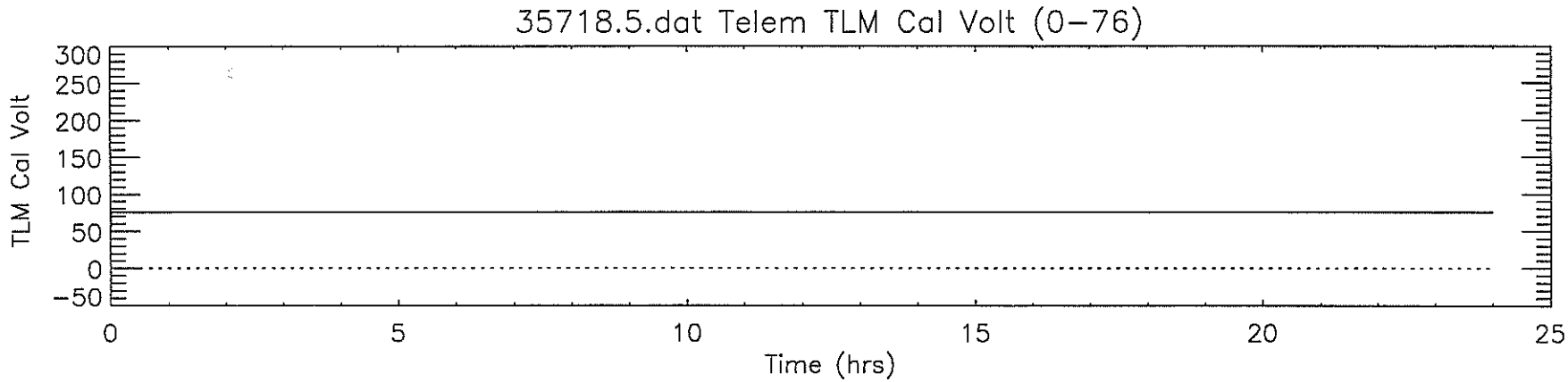

35718.5.dat Telem XRP Temp (0-68)
XRP Temp
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

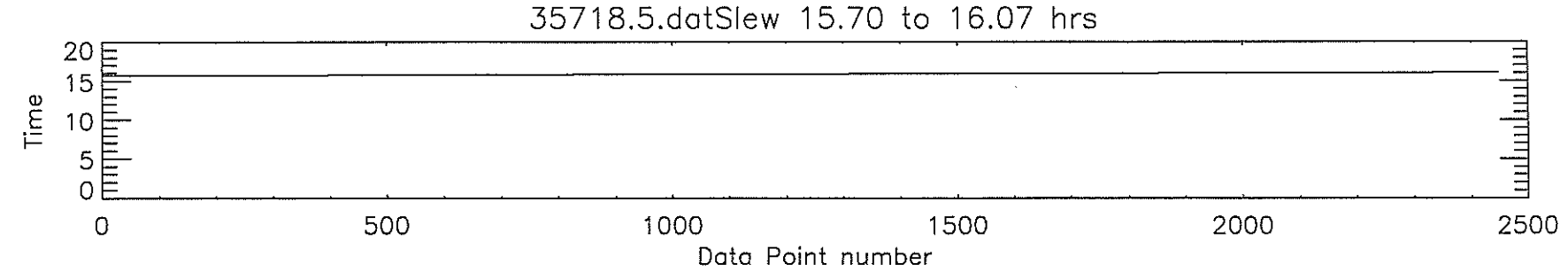

35718.5.dat Telem XRP Bear. Temp (0-0)
XRP Bear. Temp
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

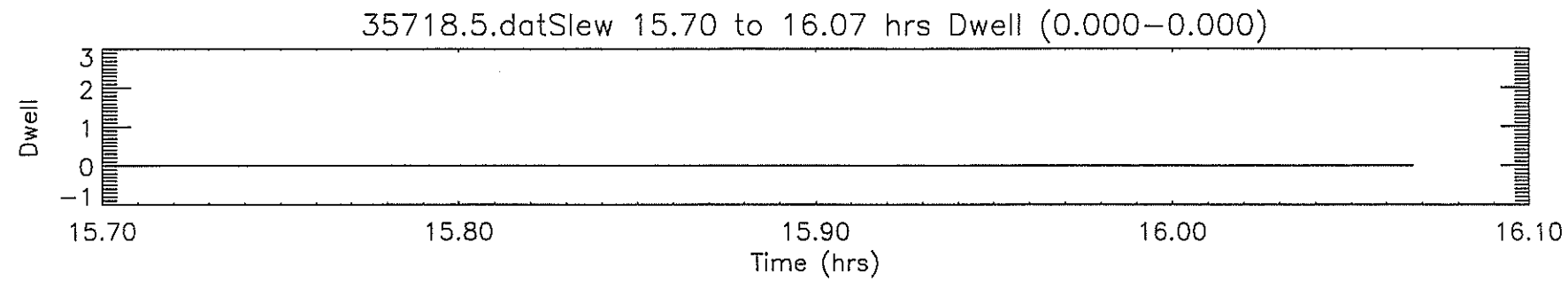

35718.5.dat Telem XRP Electr. Temp (0-0)
XRP Electr. Temp
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

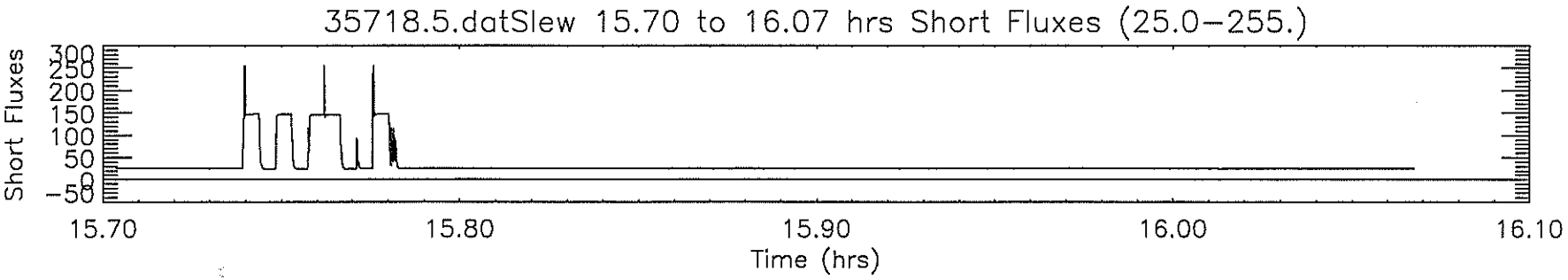

35718.5.dat Telem Ref Volt (0-150)
Ref Volt
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

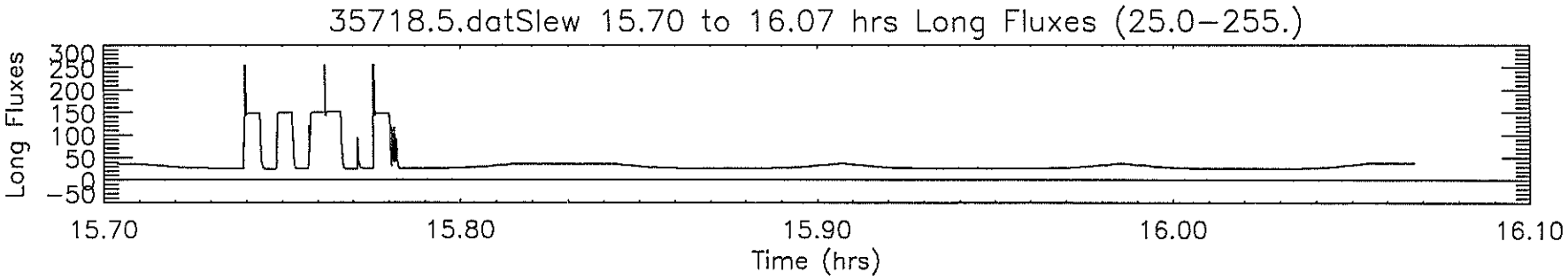

35718.5.dat Telem Chamber Volt (0-172)
Chamber Volt
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

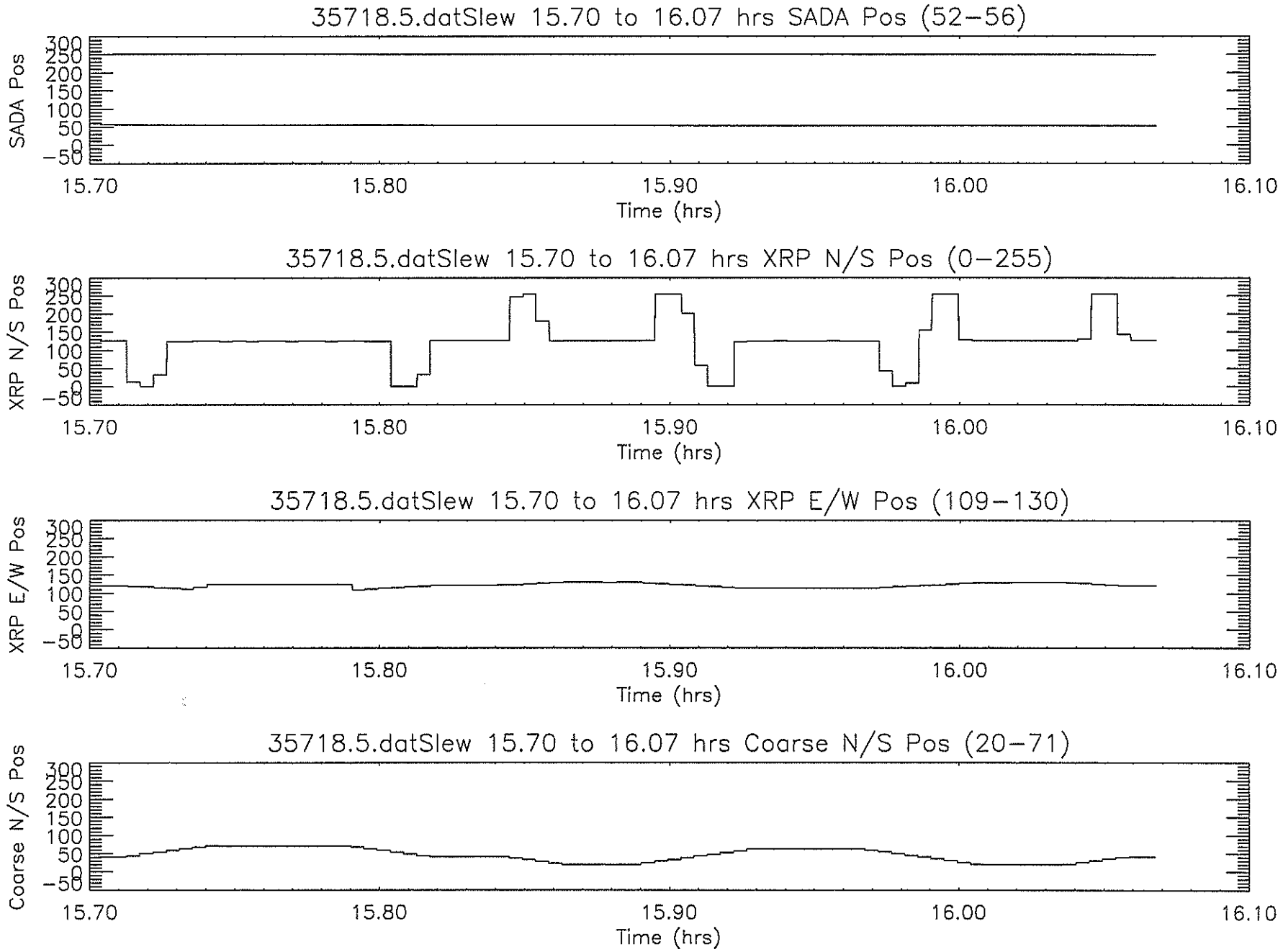

35718.5.dat Telem TLM Cal Volt (0-76)
TLM Cal Volt
300
250
200
150
100
50
0
-50
0
5
10
15
20
25
Time (hrs)

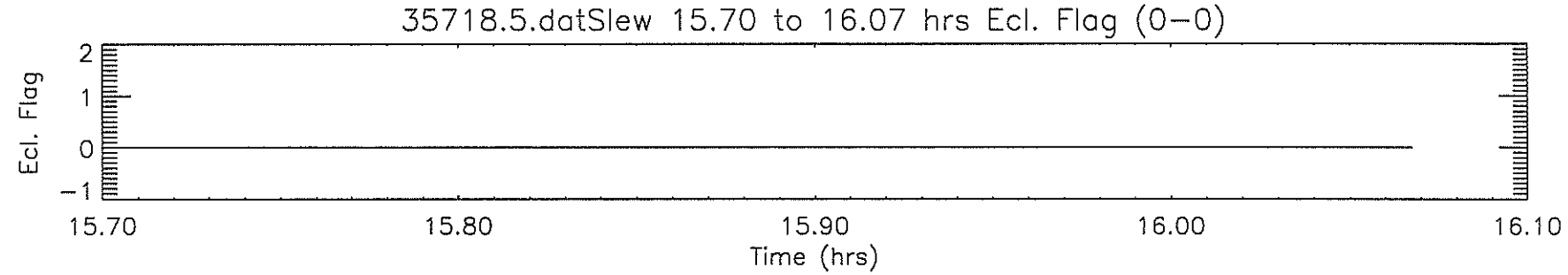

35718.5.datSlew 15.70 to 16.07 hrs
Time
20
15
10
5
0
0
500
1000
1500
2000
2500
Data Point number

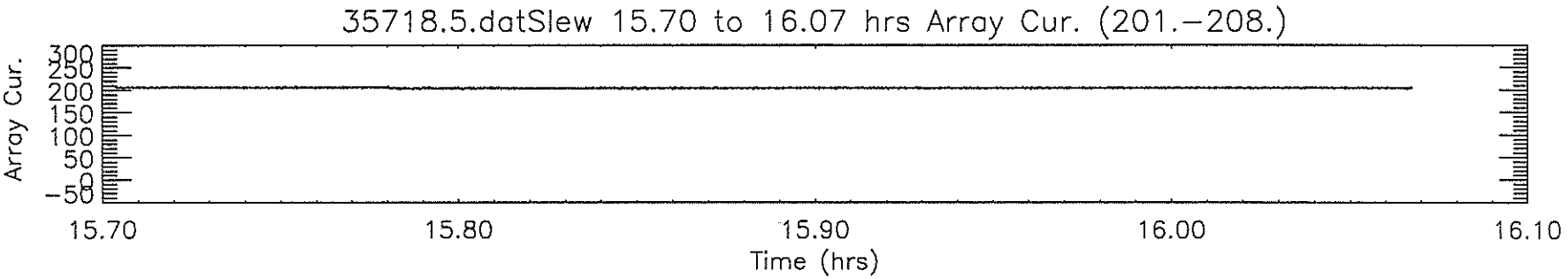

35718.5.datSlew 15.70 to 16.07 hrs Dwell (0.000-0.000)
Dwell
3
2
1
0
-1
15.70
15.80
15.90
16.00
16.10
Time (hrs)

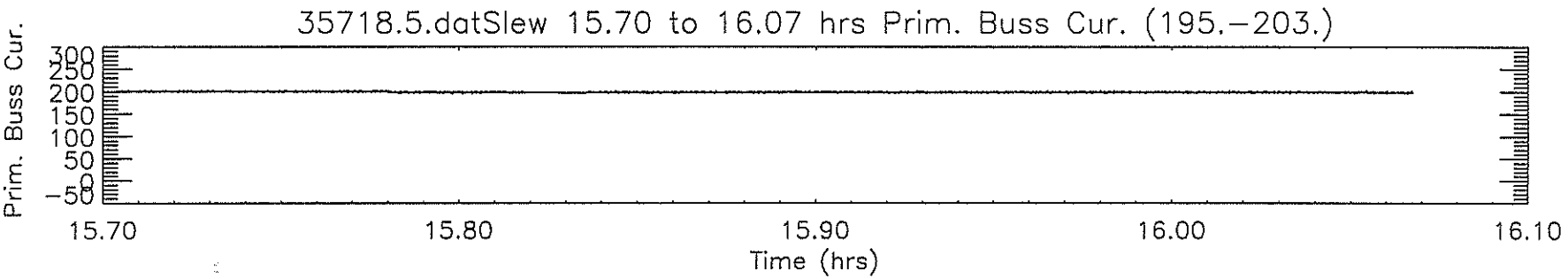

35718.5.datSlew 15.70 to 16.07 hrs Short Fluxes (25.0-255.)
Short Fluxes
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

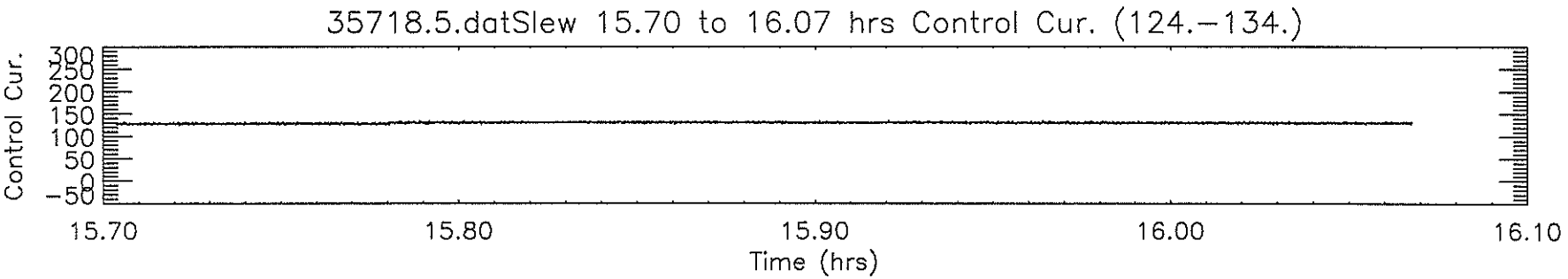

35718.5.datSlew 15.70 to 16.07 hrs Long Fluxes (25.0-255.)
Long Fluxes
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

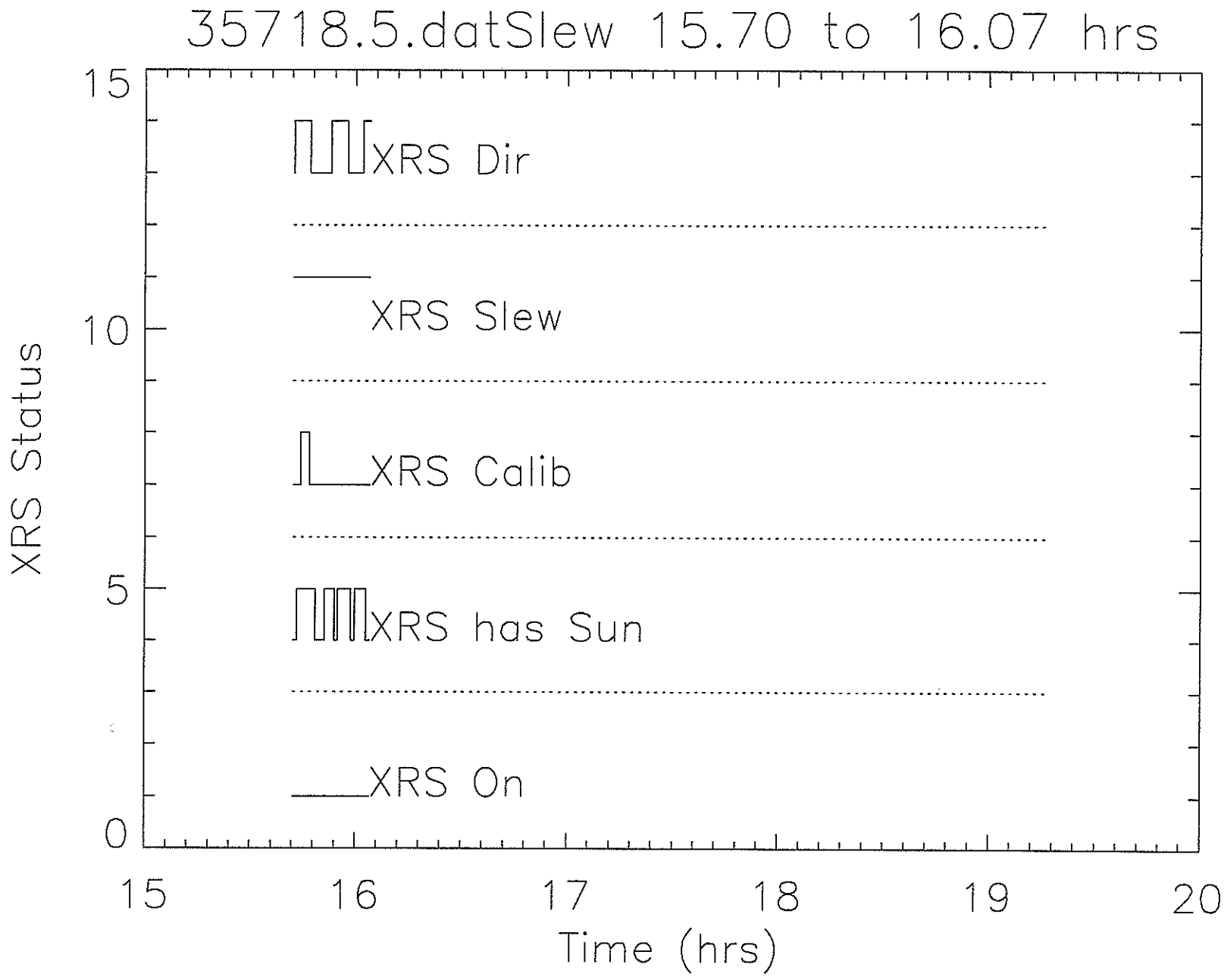

35718.5.datSlew 15.70 to 16.07 hrs SADA Pos (52–56)
SADA Pos
300
250
200
150
100
50
0
−50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs XRP N/S Pos (0–255)
XRP N/S Pos
300
250
200
150
100
50
0
−50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs XRP E/W Pos (109–130)
XRP E/W Pos
300
250
200
150
100
50
0
−50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs Coarse N/S Pos (20–71)
Coarse N/S Pos
300
250
200
150
100
50
0
−50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

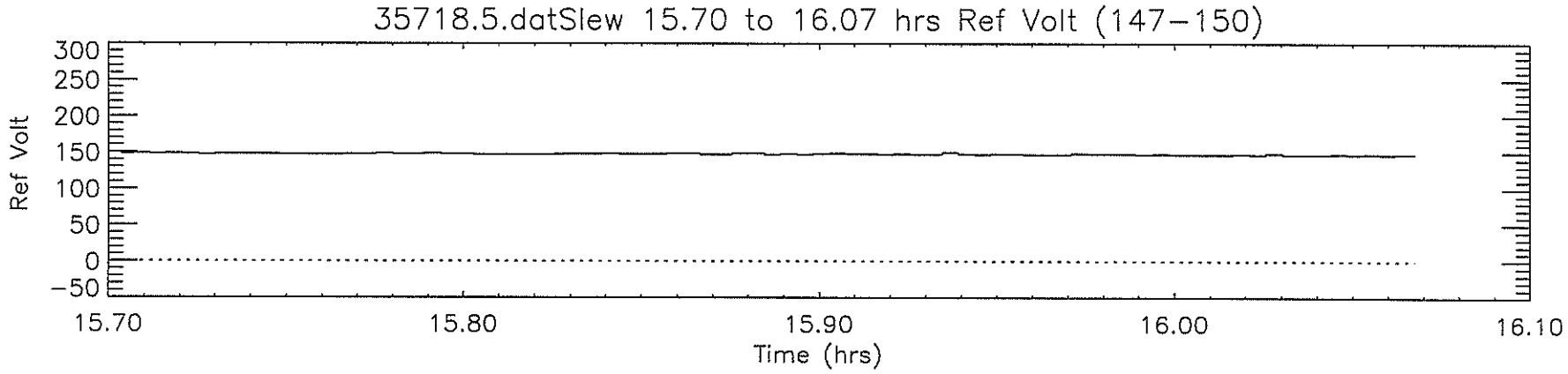

35718.5.datSlew 15.70 to 16.07 hrs Ecl. Flag (0-0)
Ecl. Flag
2
1
0
-1
15.70
15.80
15.90
16.00
16.10
Time (hrs)

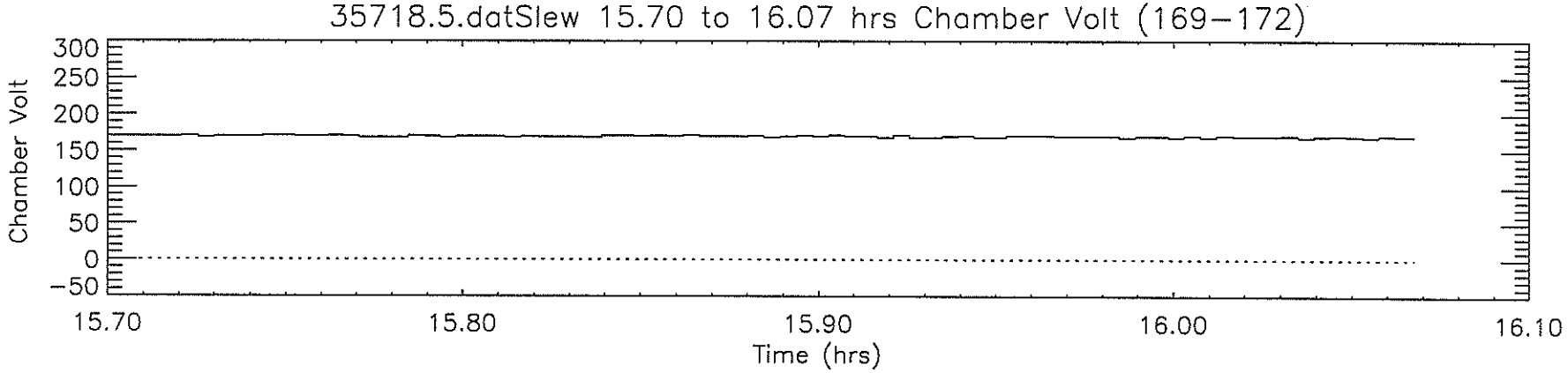

35718.5.datSlew 15.70 to 16.07 hrs Array Cur. (201.-208.)
Array Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

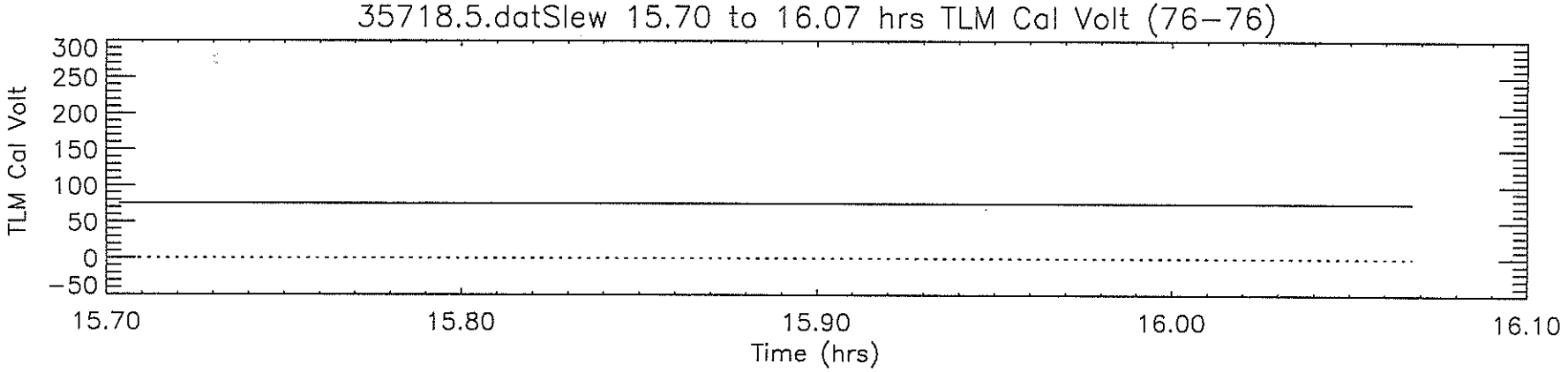

35718.5.datSlew 15.70 to 16.07 hrs Prim. Buss Cur. (195.-203.)
Prim. Buss Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

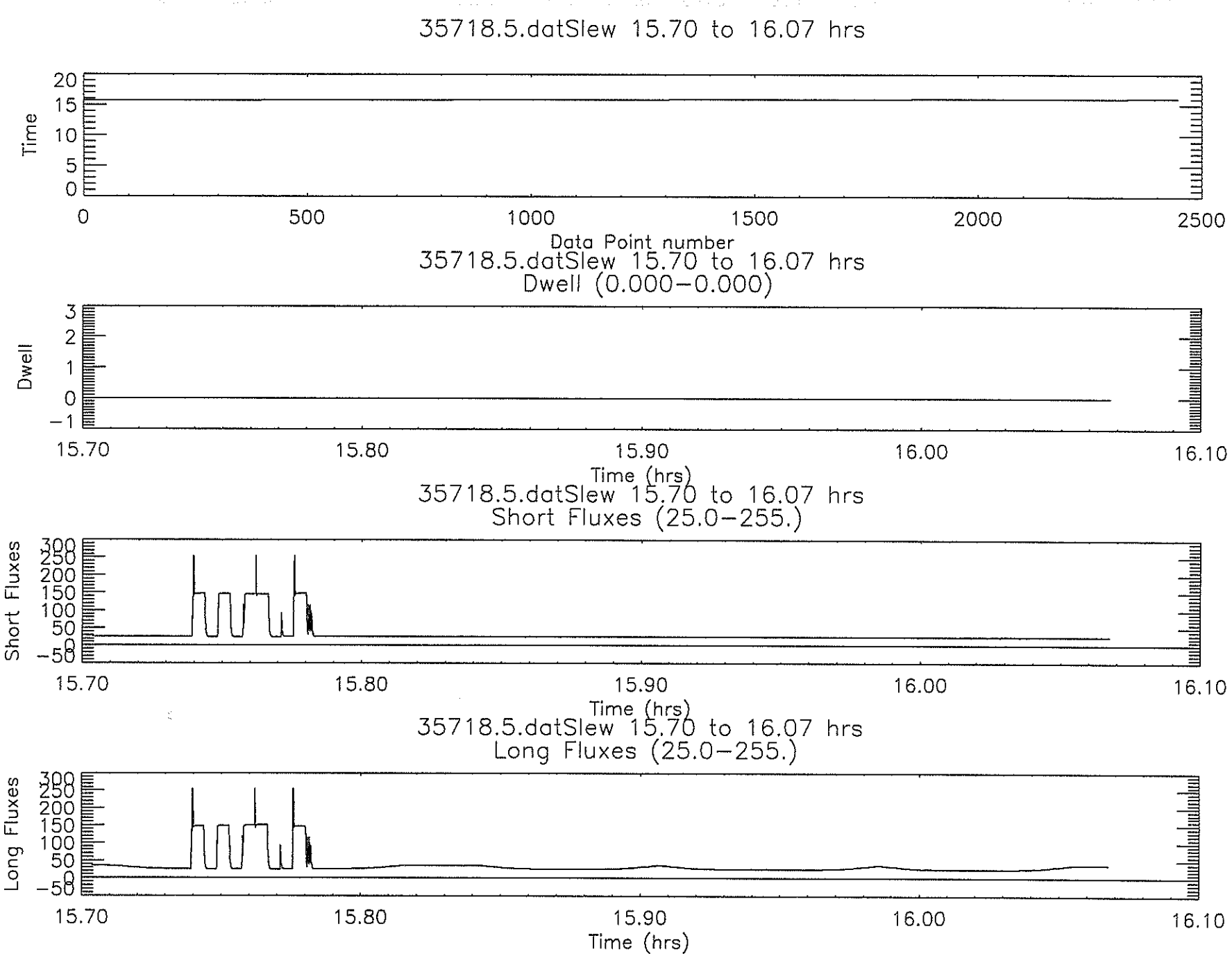

35718.5.datSlew 15.70 to 16.07 hrs Control Cur. (124.-134.)
Control Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

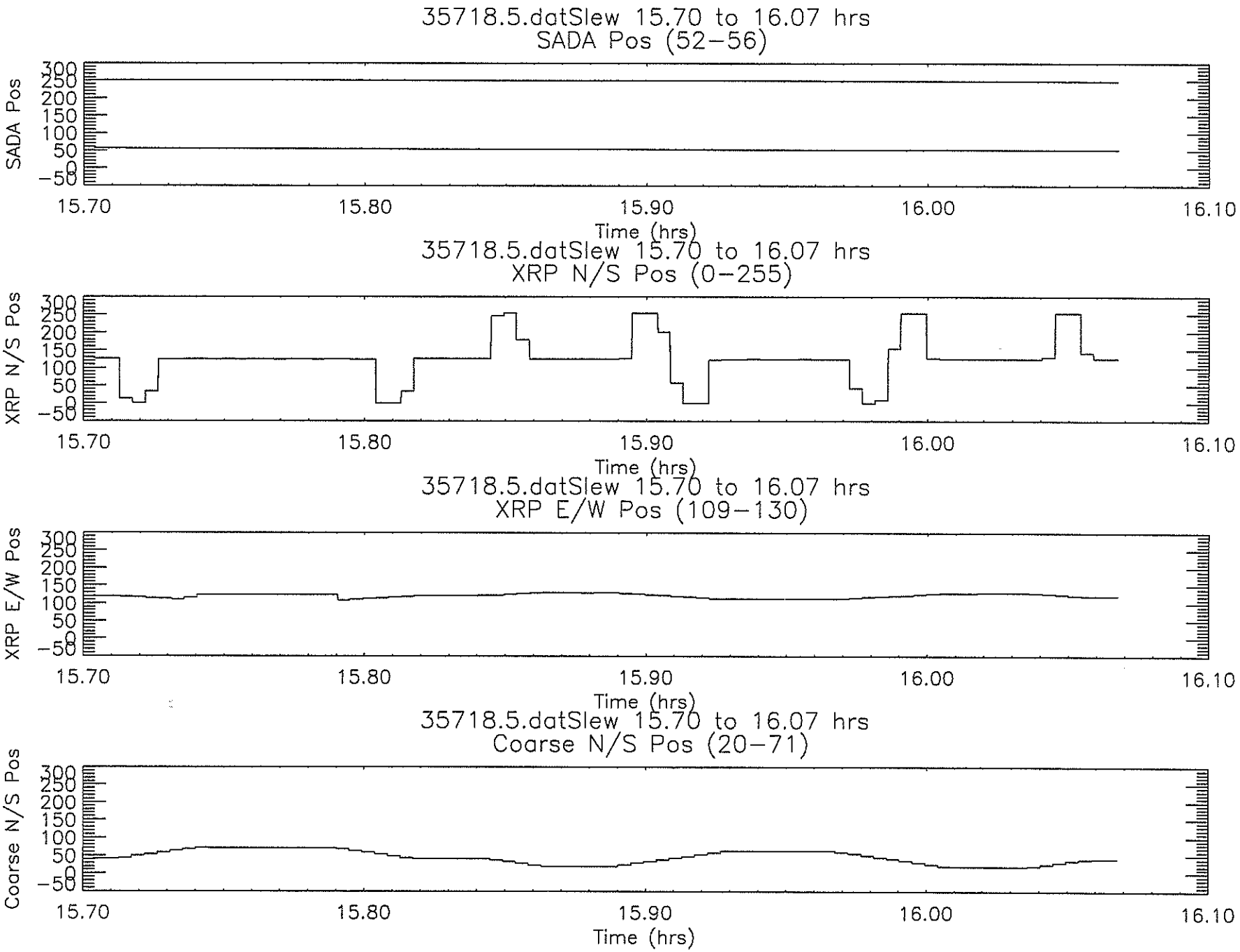

35718.5.datSlew 15.70 to 16.07 hrs
XRS Status
15
10
5
0
XRS Dir
XRS Slew
XRS Calib
XRS has Sun
XRS On
15
16
17
18
19
20
Time (hrs)

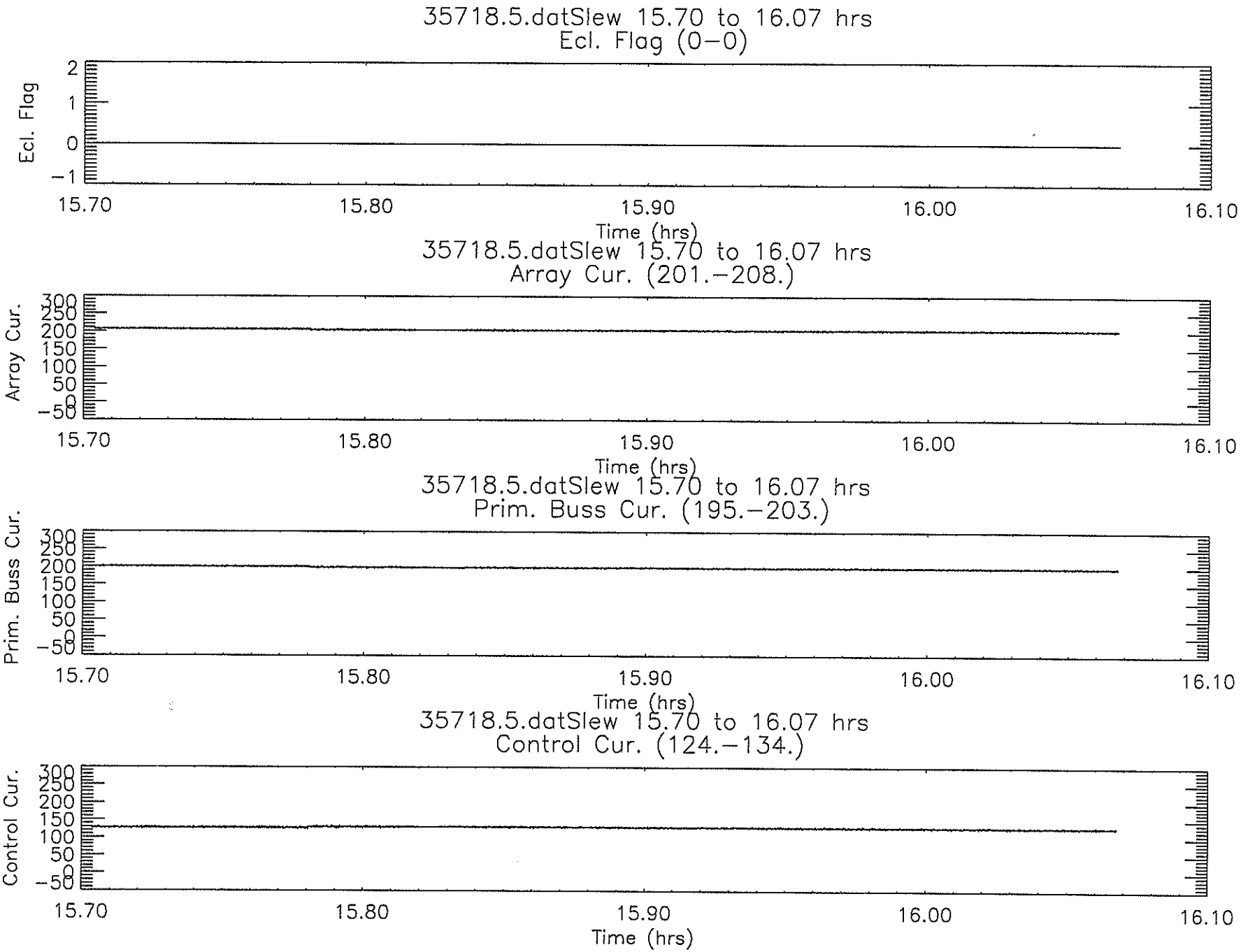

35718.5.datSlew 15.70 to 16.07 hrs Ref Volt (147–150)
Ref Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

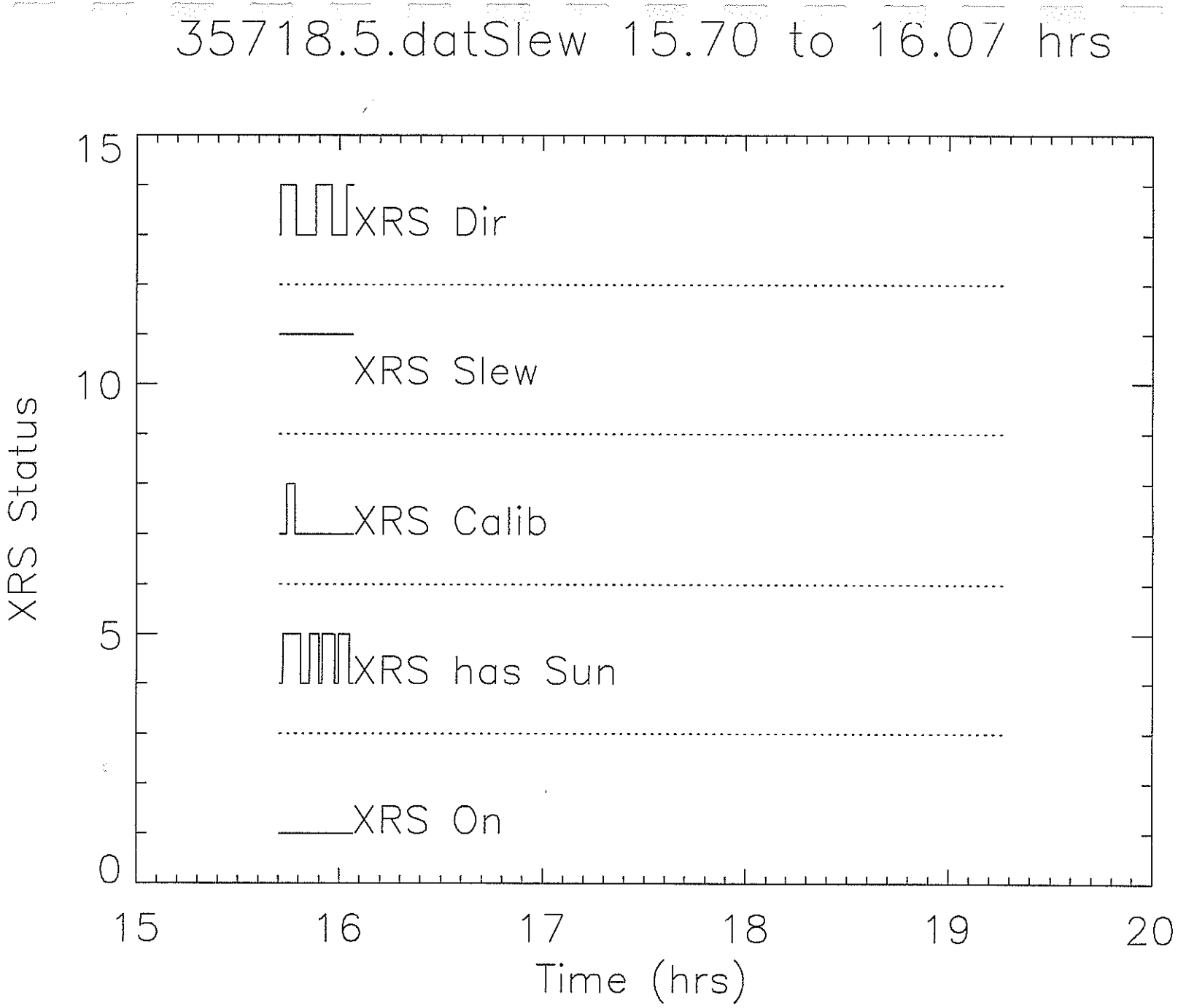

35718.5.datSlew 15.70 to 16.07 hrs Chamber Volt (169–172)
Chamber Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

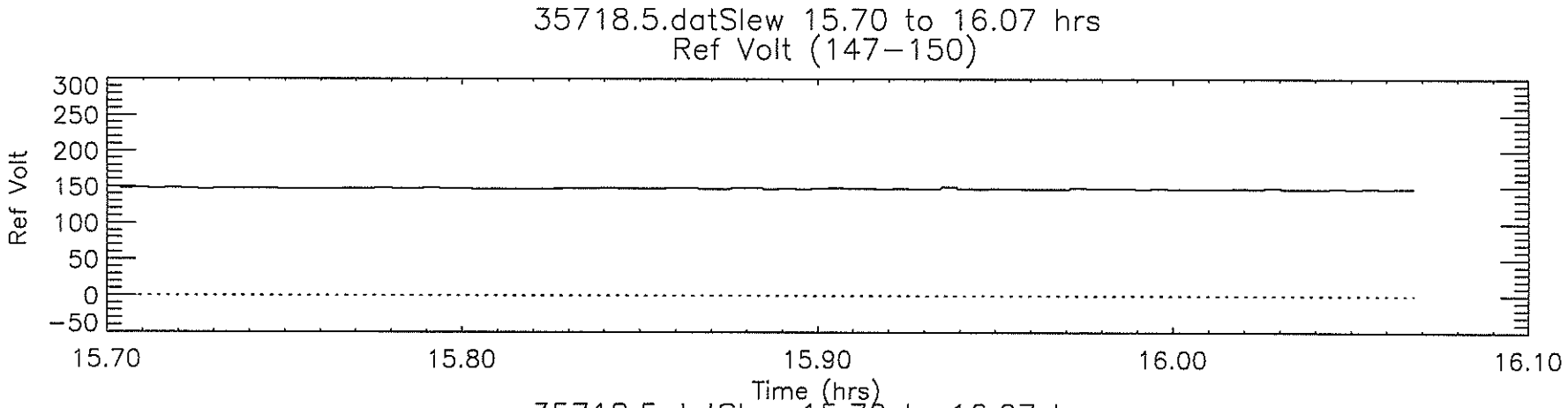

35718.5.datSlew 15.70 to 16.07 hrs TLM Cal Volt (76–76)
TLM Cal Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

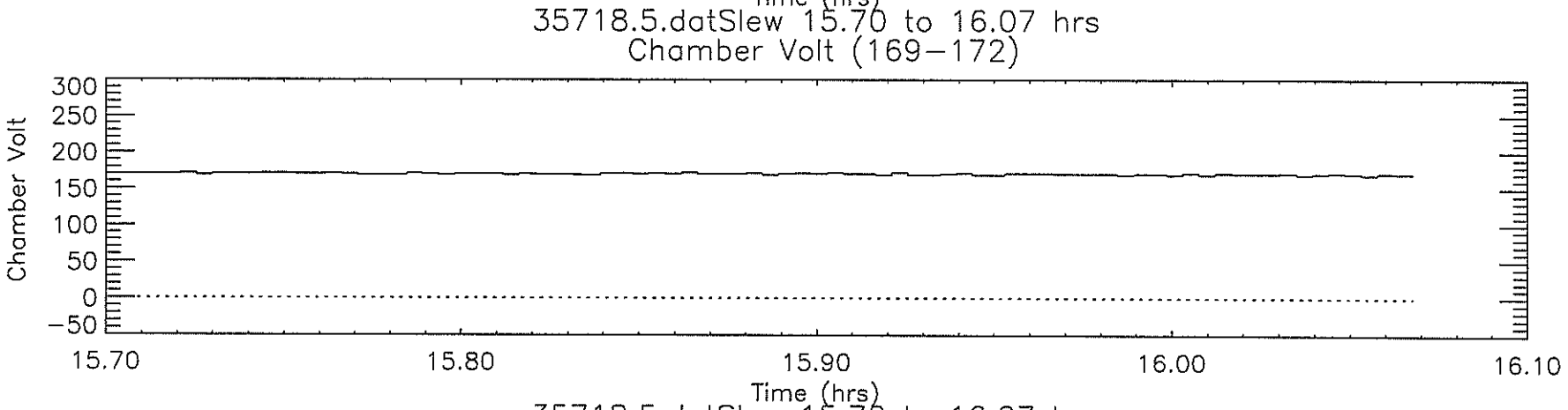

35718.5.datSlew 15.70 to 16.07 hrs
Time
20
15
10
5
0
0
500
1000
1500
2000
2500
Data Point number
35718.5.datSlew 15.70 to 16.07 hrs
Dwell (0.000-0.000)
Dwell
3
2
1
0
-1
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Short Fluxes (25.0-255.)
Short Fluxes
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Long Fluxes (25.0-255.)
Long Fluxes
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

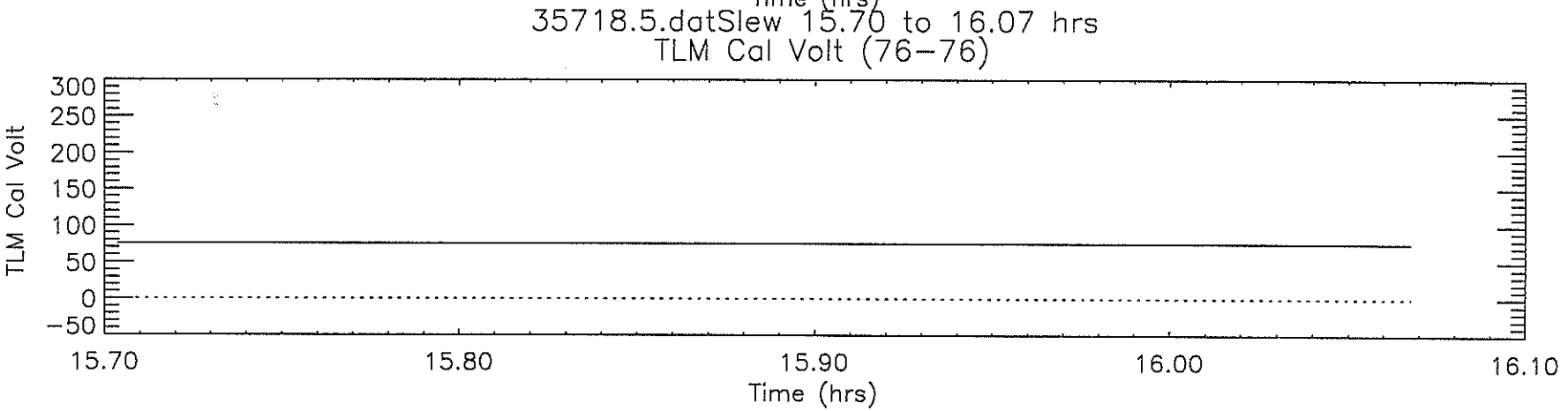

35718.5.datSlew 15.70 to 16.07 hrs
SADA Pos (52–56)
SADA Pos
300
250
200
150
100
50
0
−50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
XRP N/S Pos (0–255)
XRP N/S Pos
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
XRP E/W Pos (109–130)
XRP E/W Pos
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Coarse N/S Pos (20–71)
Coarse N/S Pos
Time (hrs)


35718.5.datSlew 15.70 to 16.07 hrs
Ecl. Flag (0-0)
Ecl. Flag
2
1
0
-1
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Array Cur. (201.-208.)
Array Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Prim. Buss Cur. (195.-203.)
Prim. Buss Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)
35718.5.datSlew 15.70 to 16.07 hrs
Control Cur. (124.-134.)
Control Cur.
300
250
200
150
100
50
0
-50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

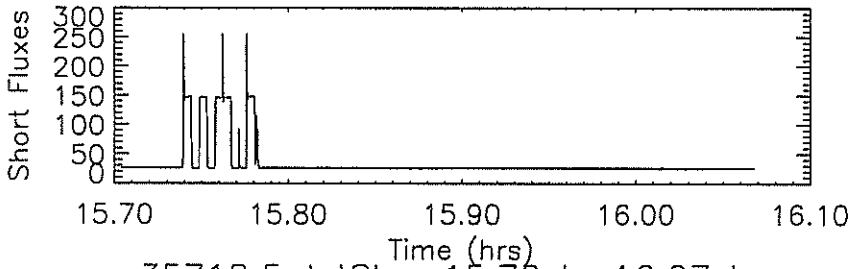

35718.5.datSlew 15.70 to 16.07 hrs
XRS Status
15
10
5
0
XRS Dir
XRS Slew
XRS Calib
XRS has Sun
XRS On
15
16
17
18
19
20
Time (hrs)


35718.5.datSlew 15.70 to 16.07 hrs
Ref Volt (147–150)
Ref Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

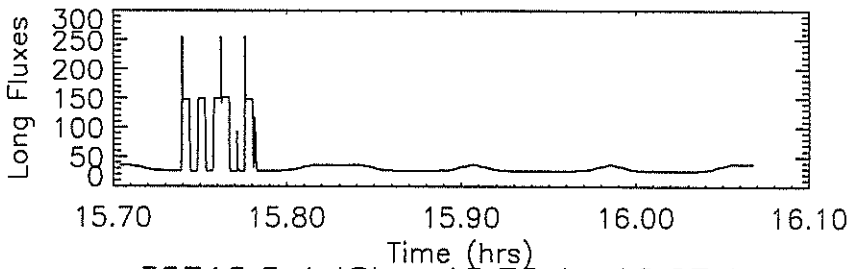

35718.5.datSlew 15.70 to 16.07 hrs
Chamber Volt (169–172)
Chamber Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)


35718.5.datSlew 15.70 to 16.07 hrs
TLM Cal Volt (76–76)
TLM Cal Volt
300
250
200
150
100
50
0
–50
15.70
15.80
15.90
16.00
16.10
Time (hrs)

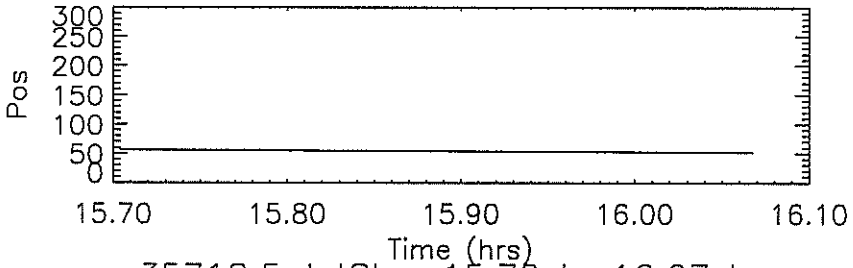
35718.5.datSlew 15.70 to 16.07 hrs



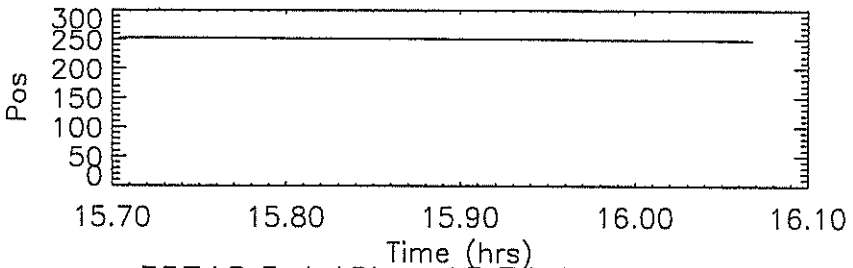

35718.5.datSlew 15.70 to 16.07 hrs



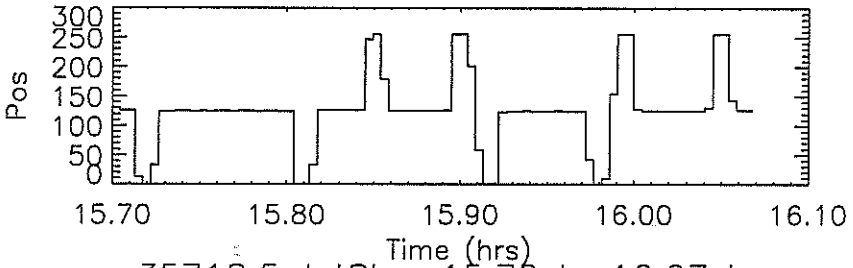

35718.5.datSlew 15.70 to 16.07 hrs
SADA 1 Pos



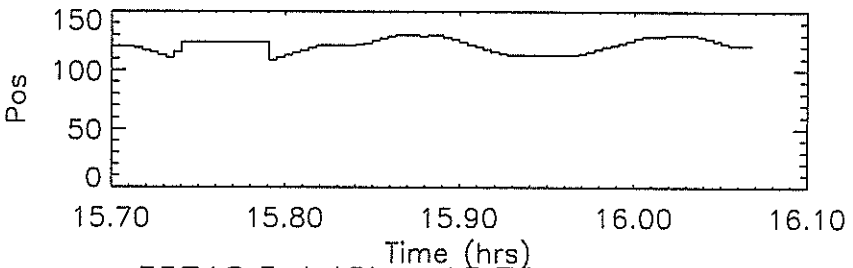

35718.5.datSlew 15.70 to 16.07 hrs
SADA 2 Pos



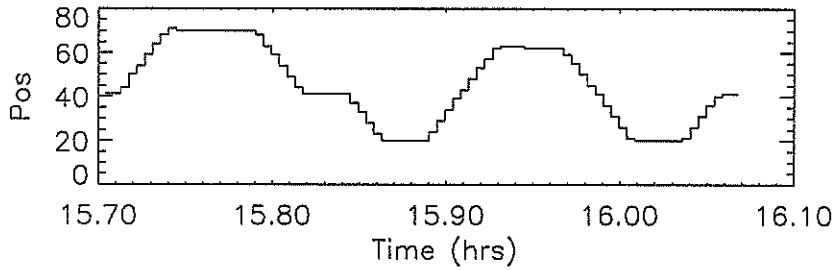

35718.5.datSlew 15.70 to 16.07 hrs
XRP N/S Pos



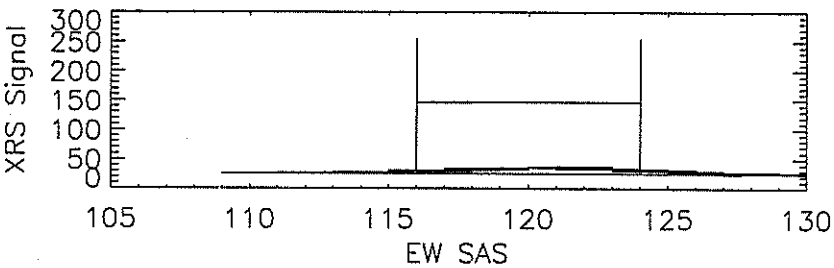

35718.5.datSlew 15.70 to 16.07 hrs
XRP E/W Pos

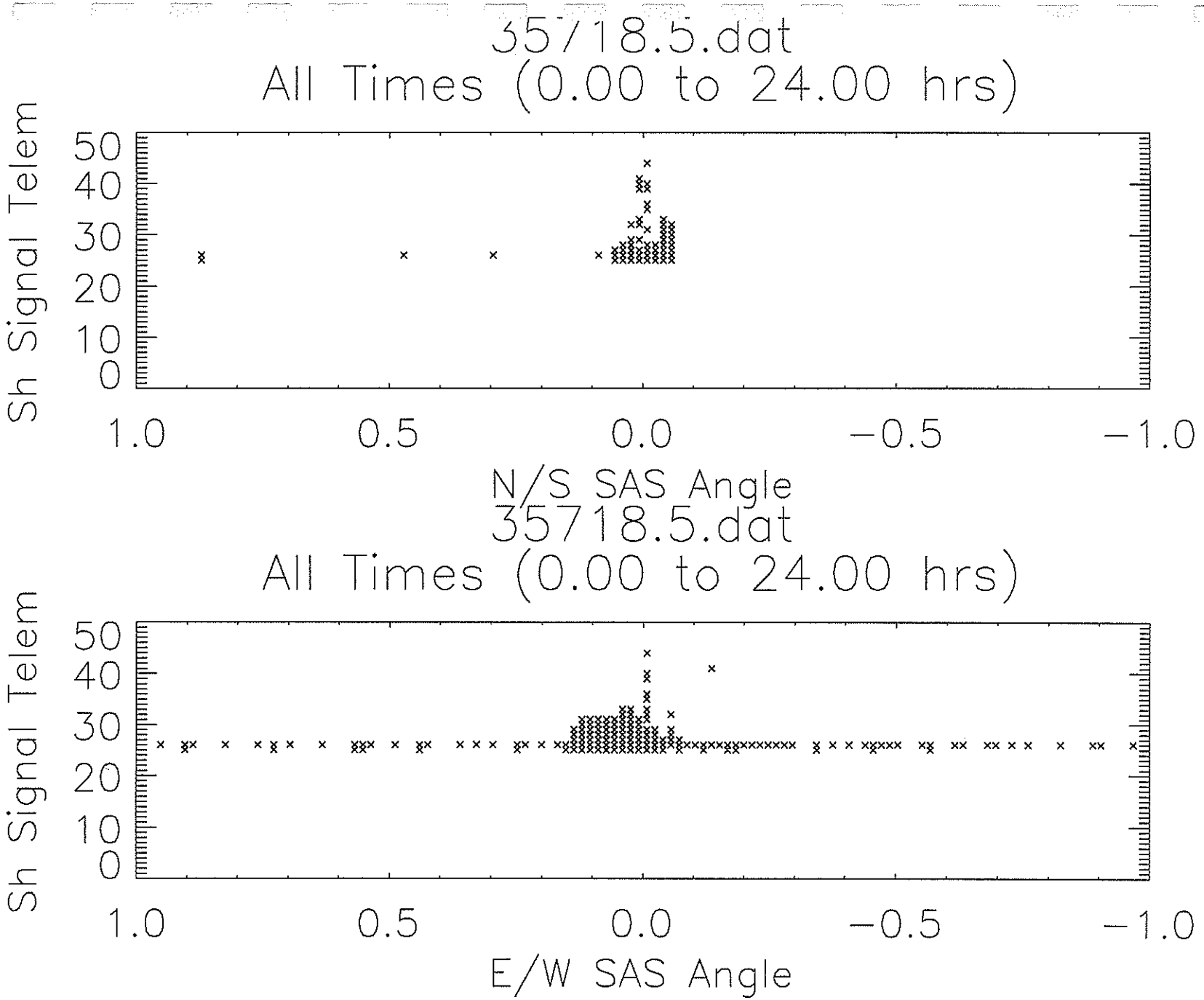


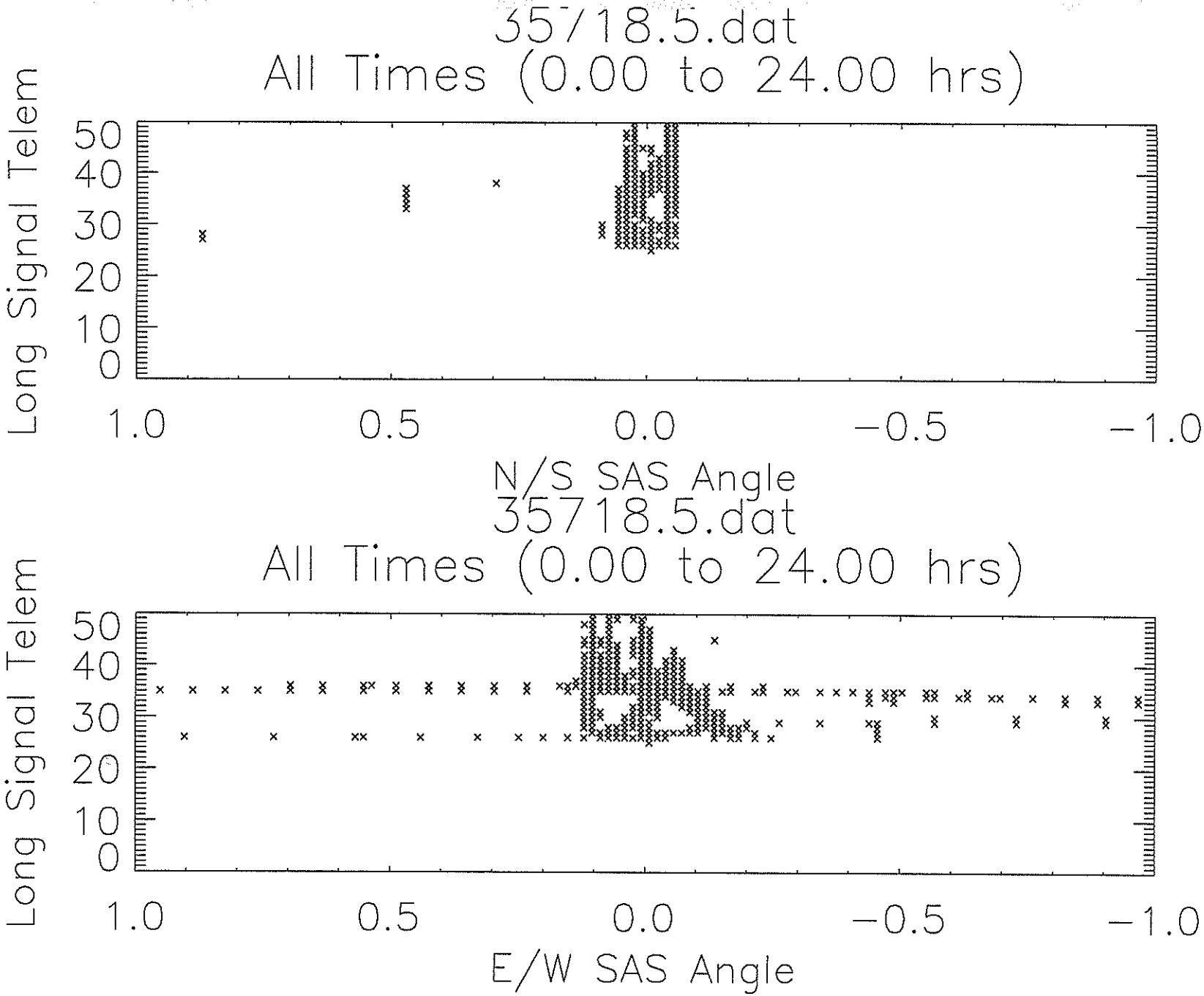

35718.5.datSlew 15.70 to 16.07 hrs
Coarse N/S Pos

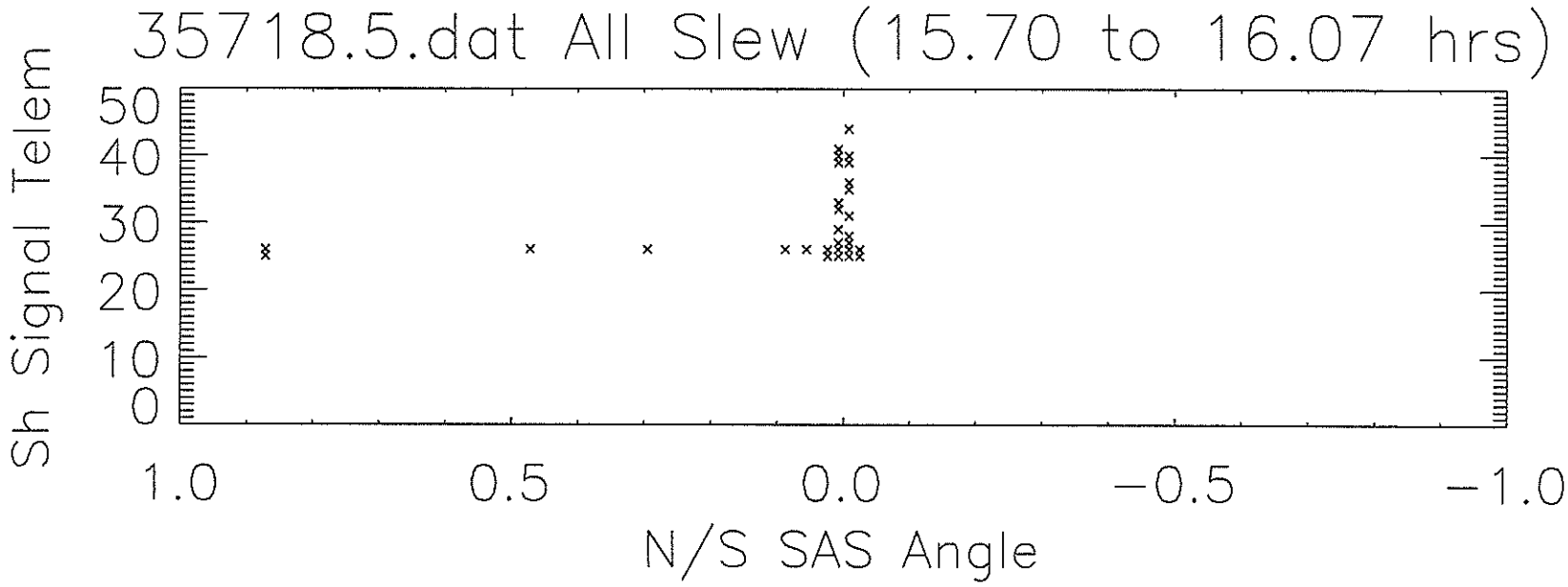


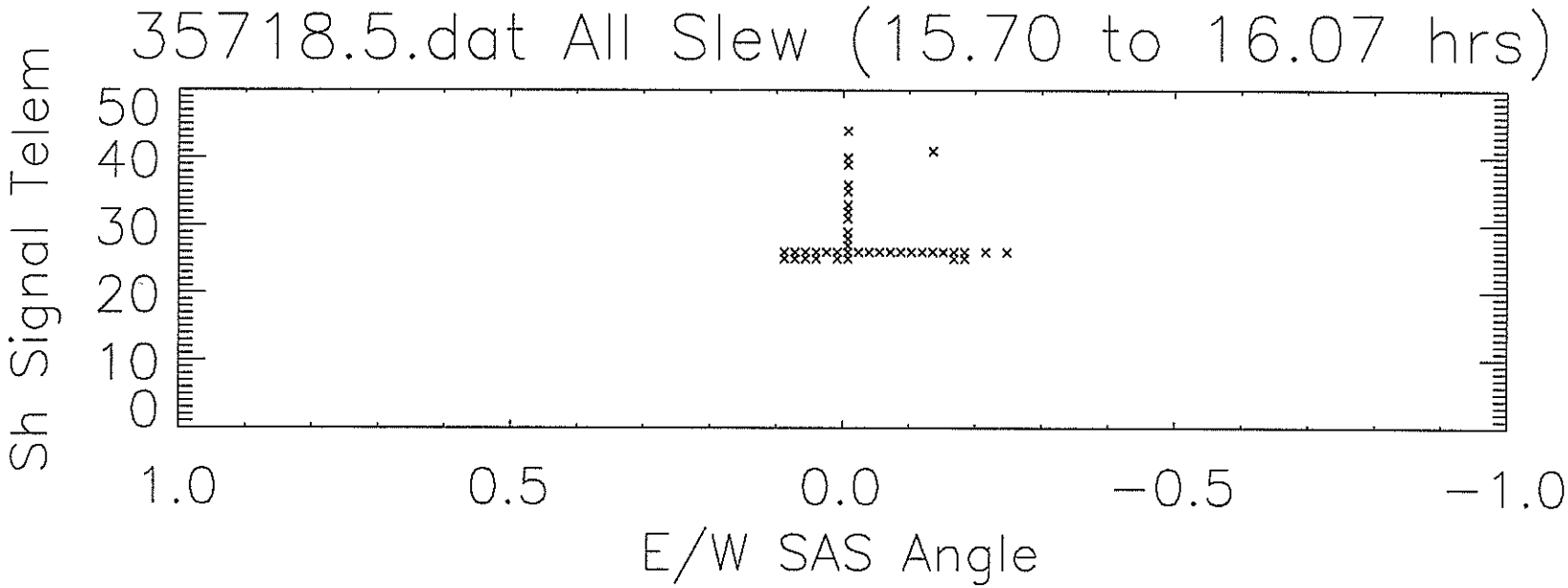

35718.5.datSlew 15.70 to 16.07 hrs

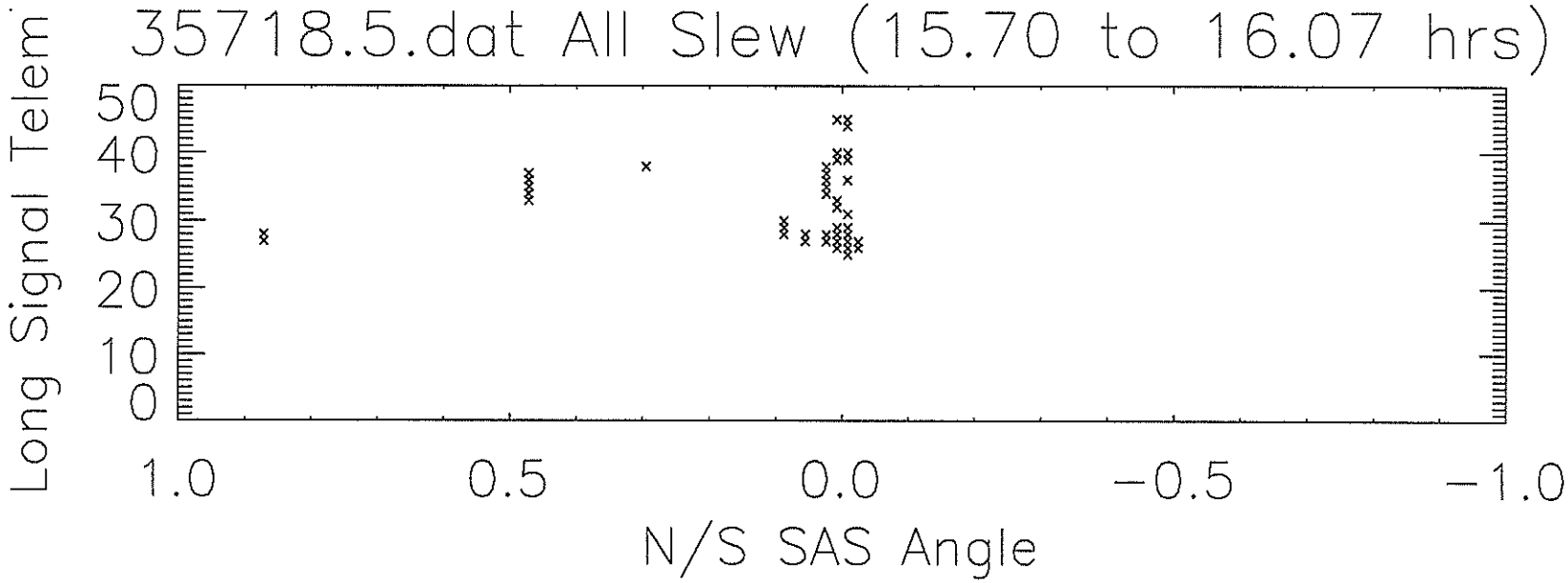


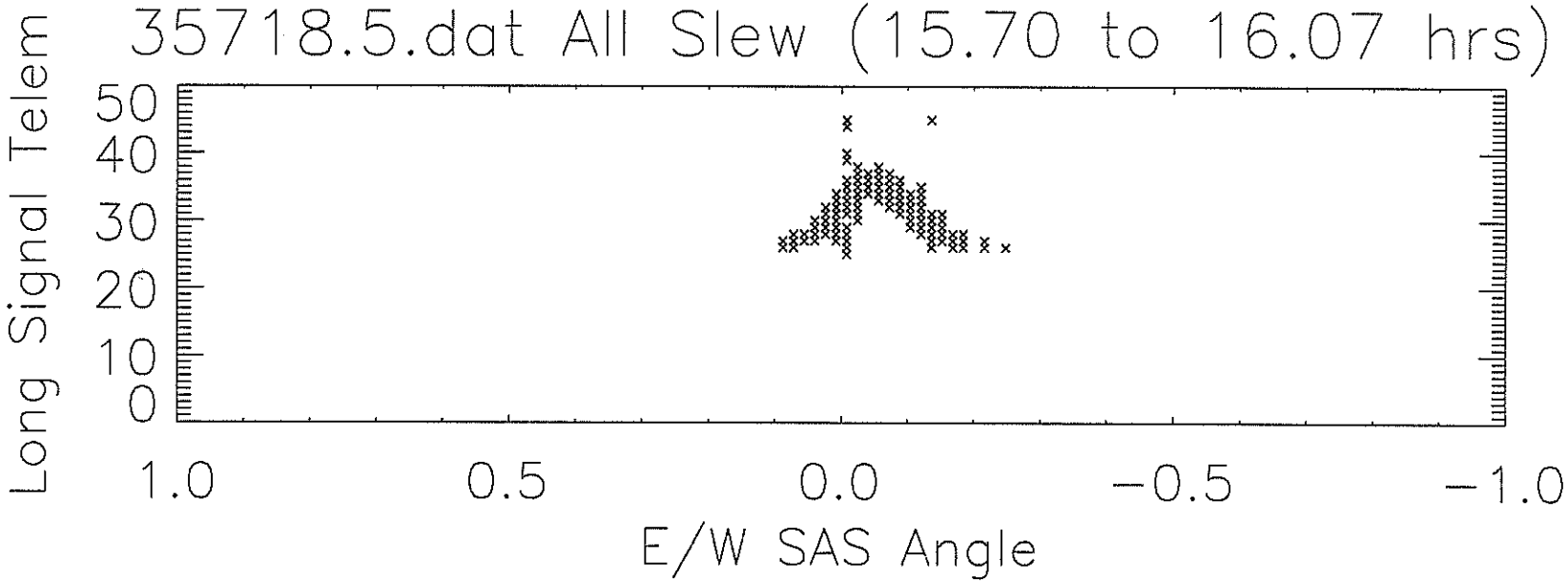

35718.5.dat
All Times (0.00 to 24.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle
35718.5.dat
All Times (0.00 to 24.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

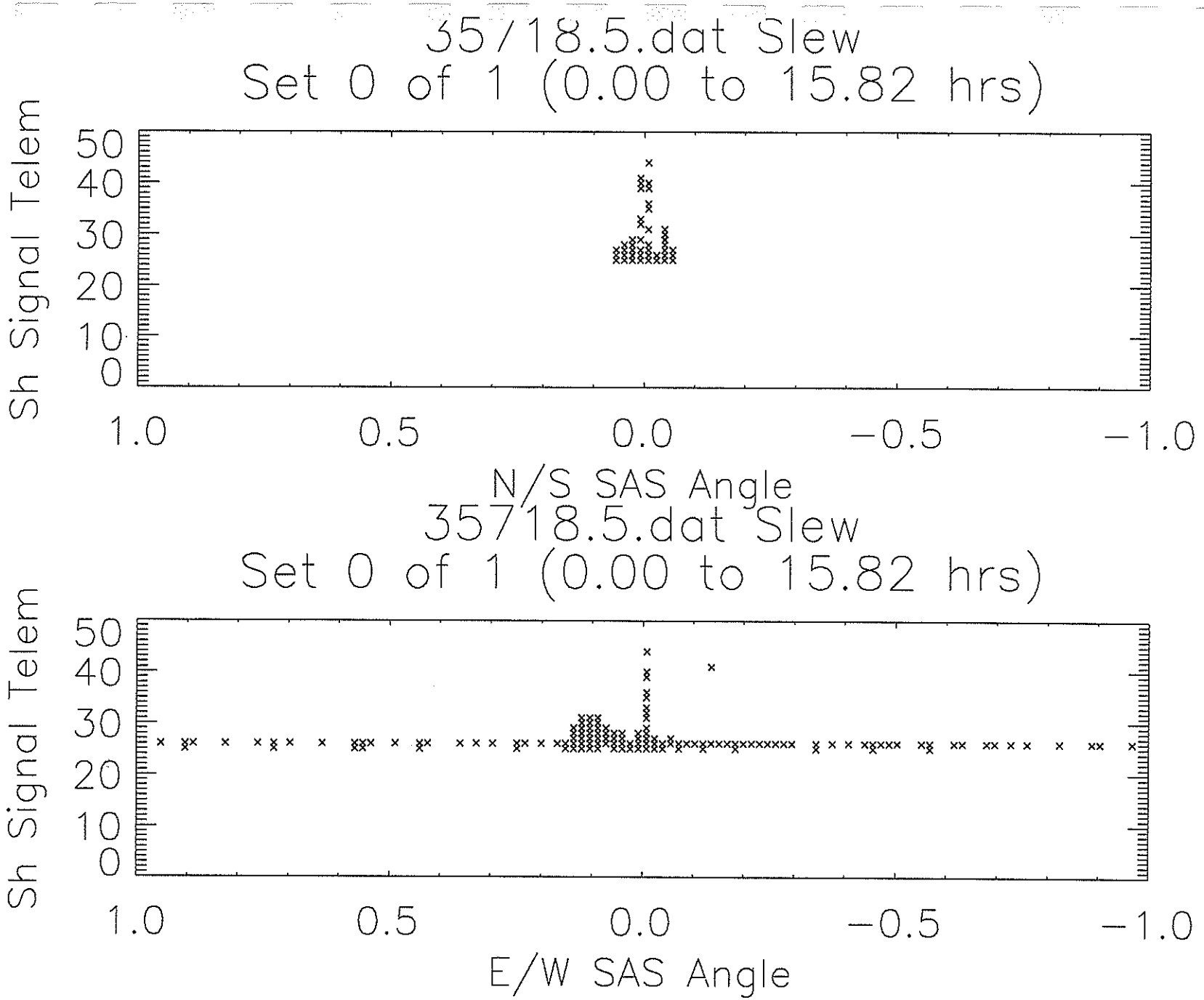

35718.5.dat
All Times (0.00 to 24.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle
35718.5.dat
All Times (0.00 to 24.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

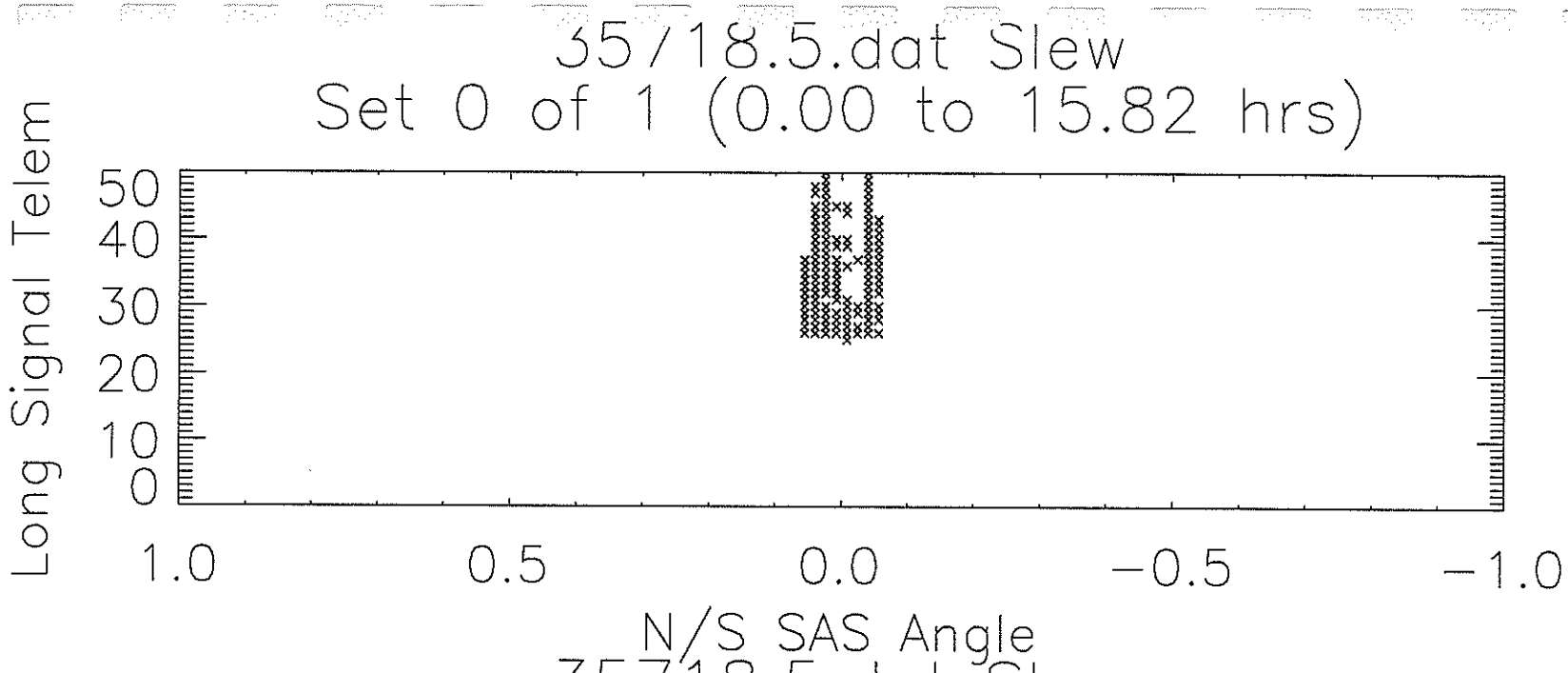

35718.5.dat All Slew (15.70 to 16.07 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

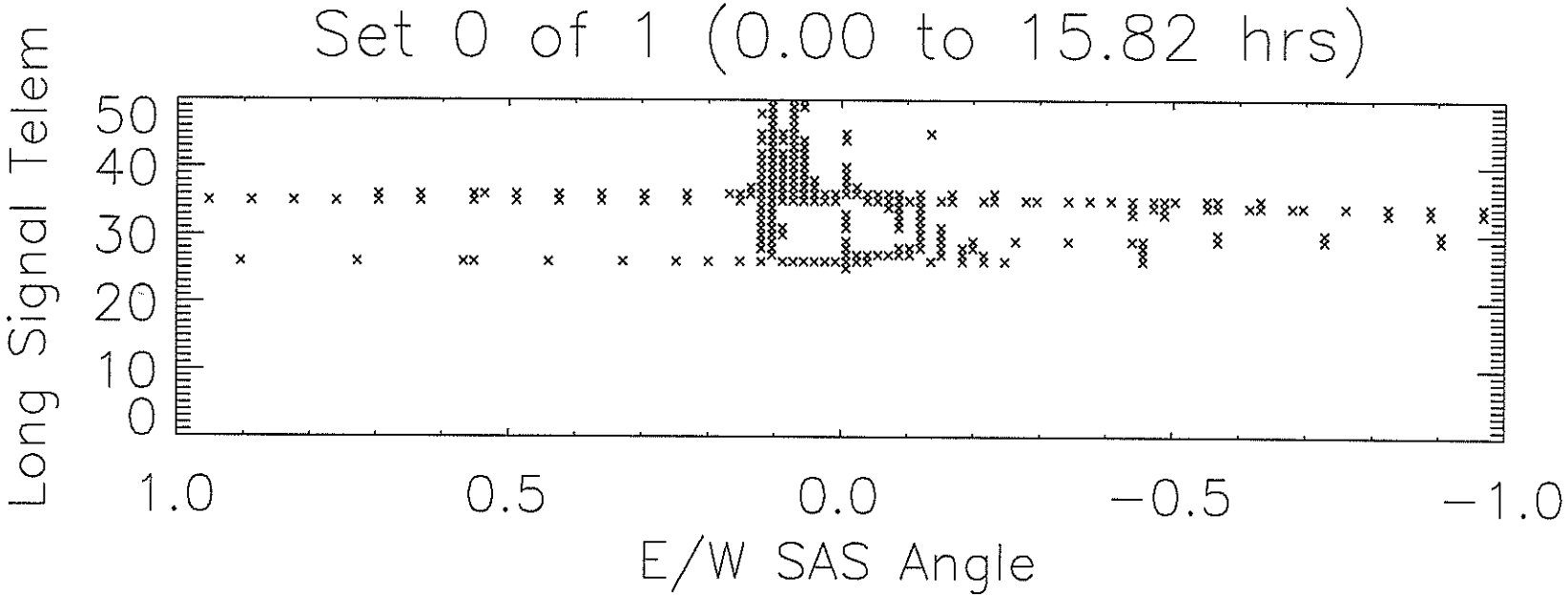

35718.5.dat All Slew (15.70 to 16.07 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

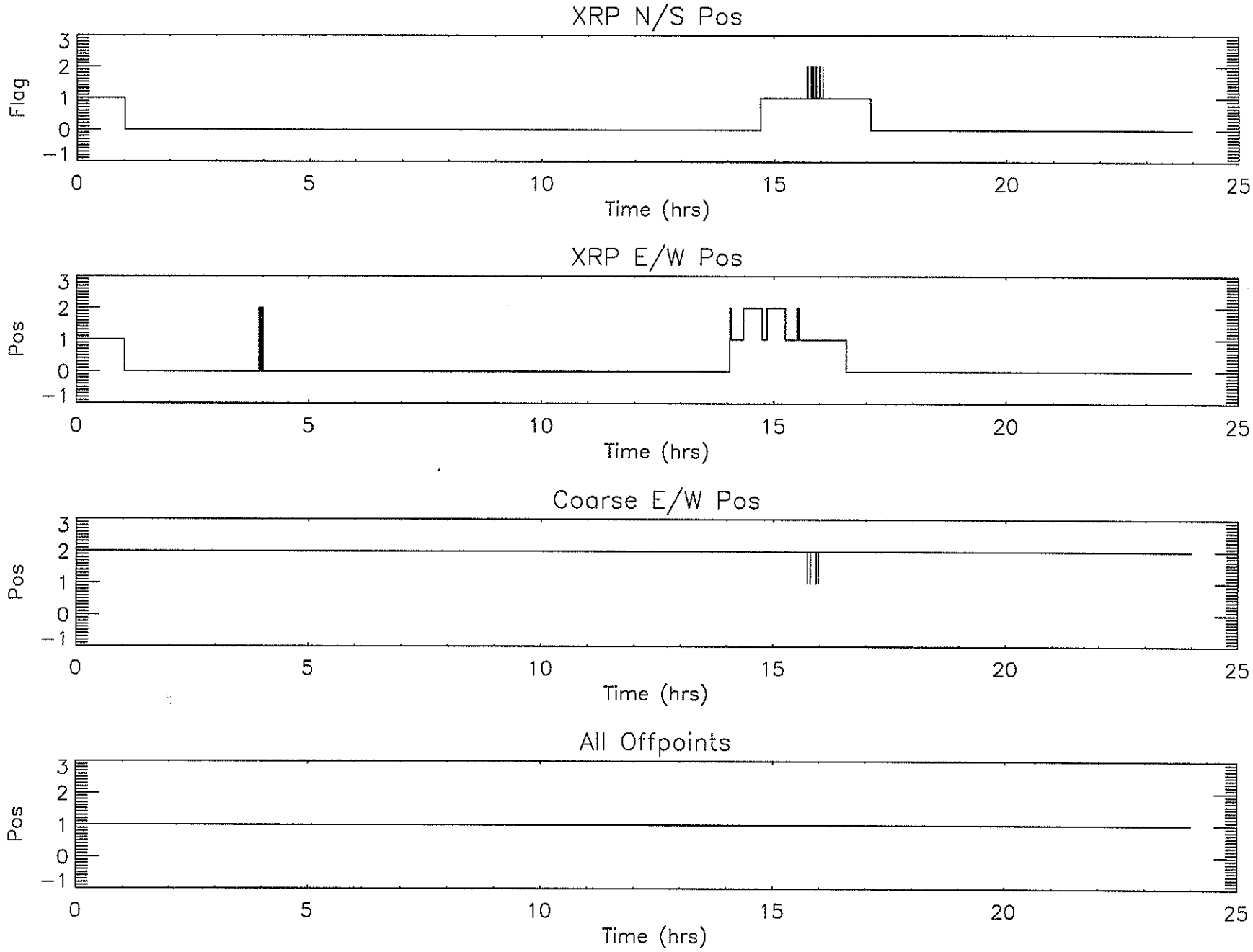

35718.5.dat All Slew (15.70 to 16.07 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

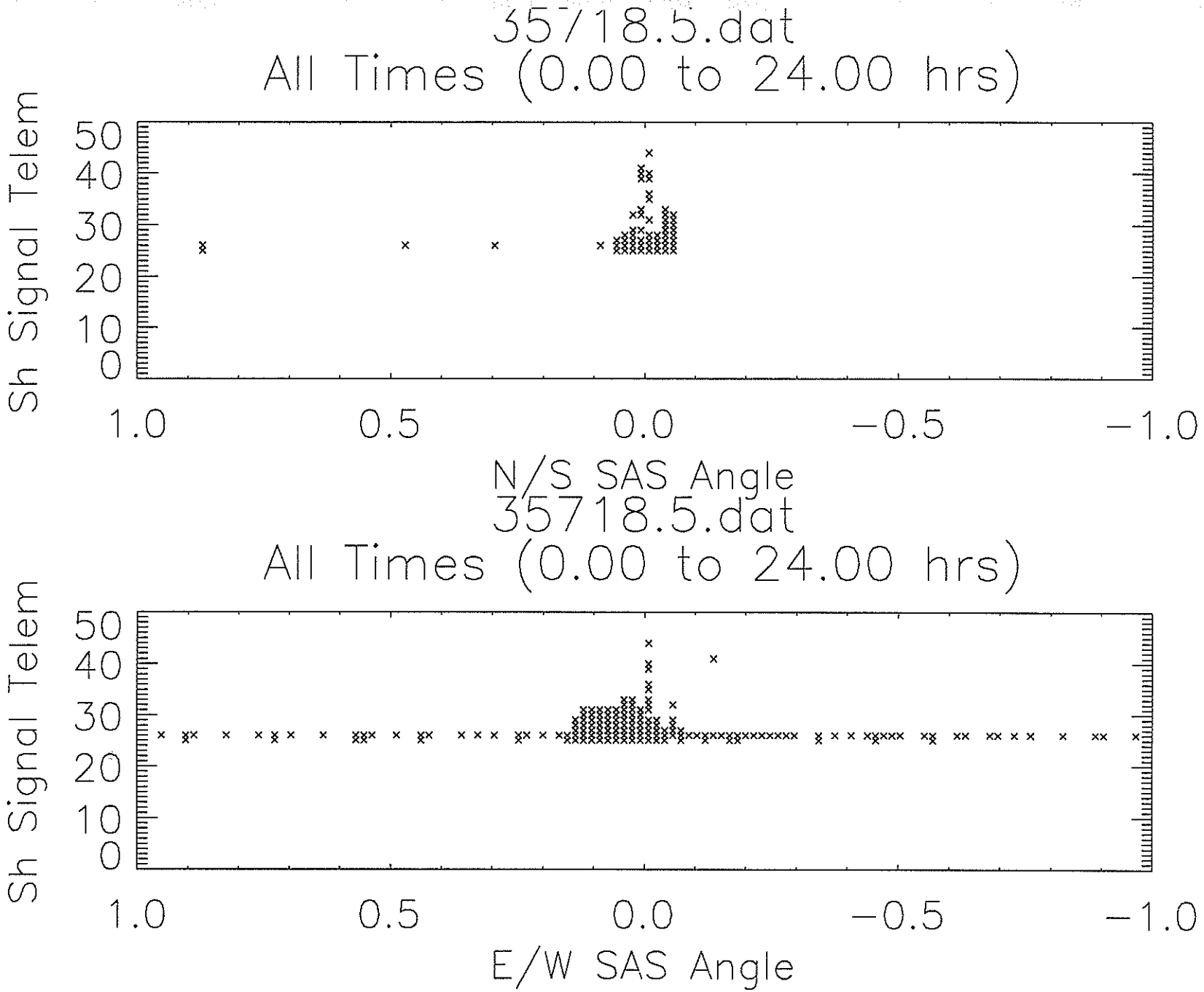

35718.5.dat All Slew (15.70 to 16.07 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

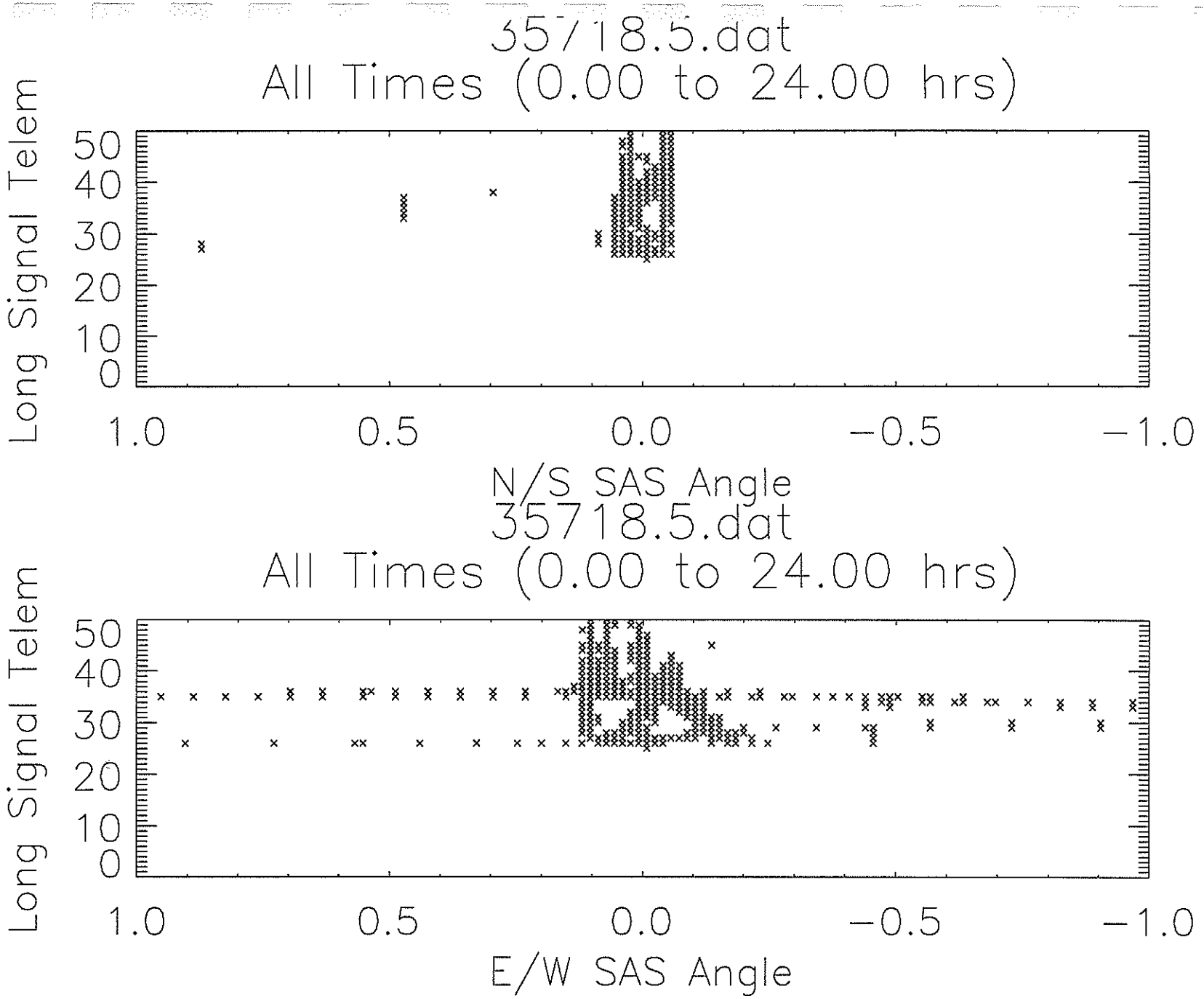

35718.5.dat Slew
Set 0 of 1 (0.00 to 15.82 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle
35718.5.dat Slew
Set 0 of 1 (0.00 to 15.82 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

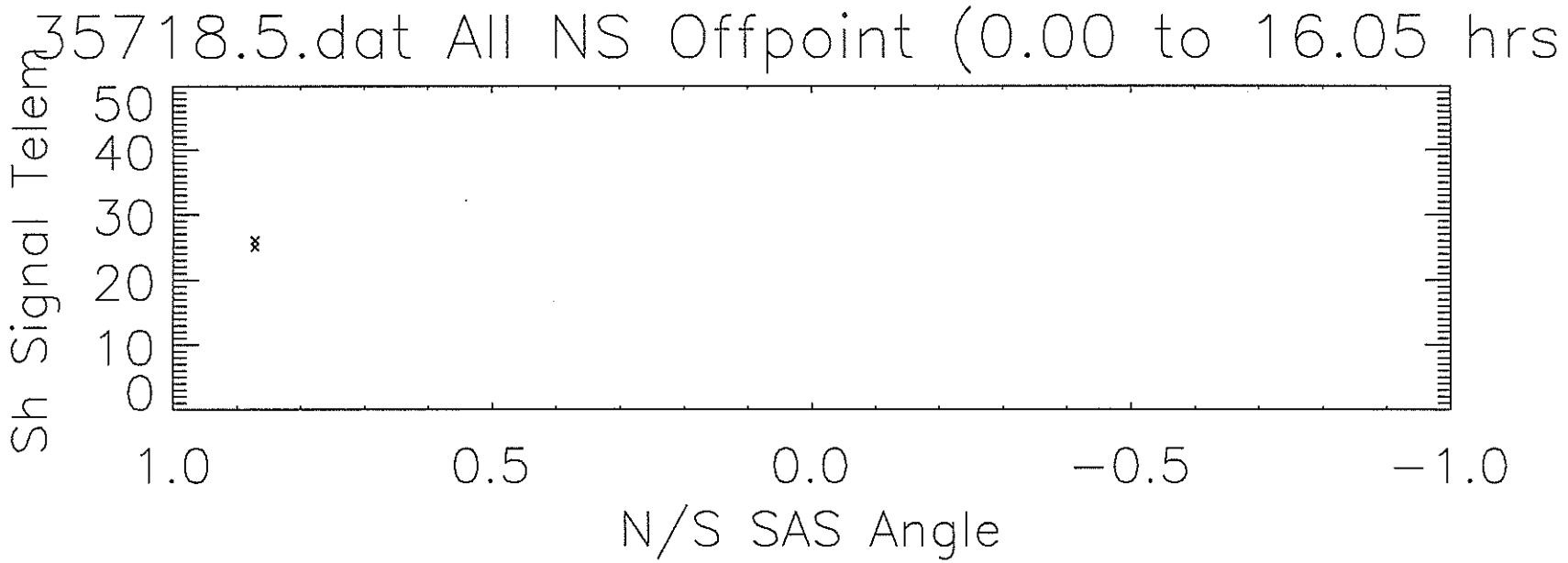

35718.5.dat Slew
Set 0 of 1 (0.00 to 15.82 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

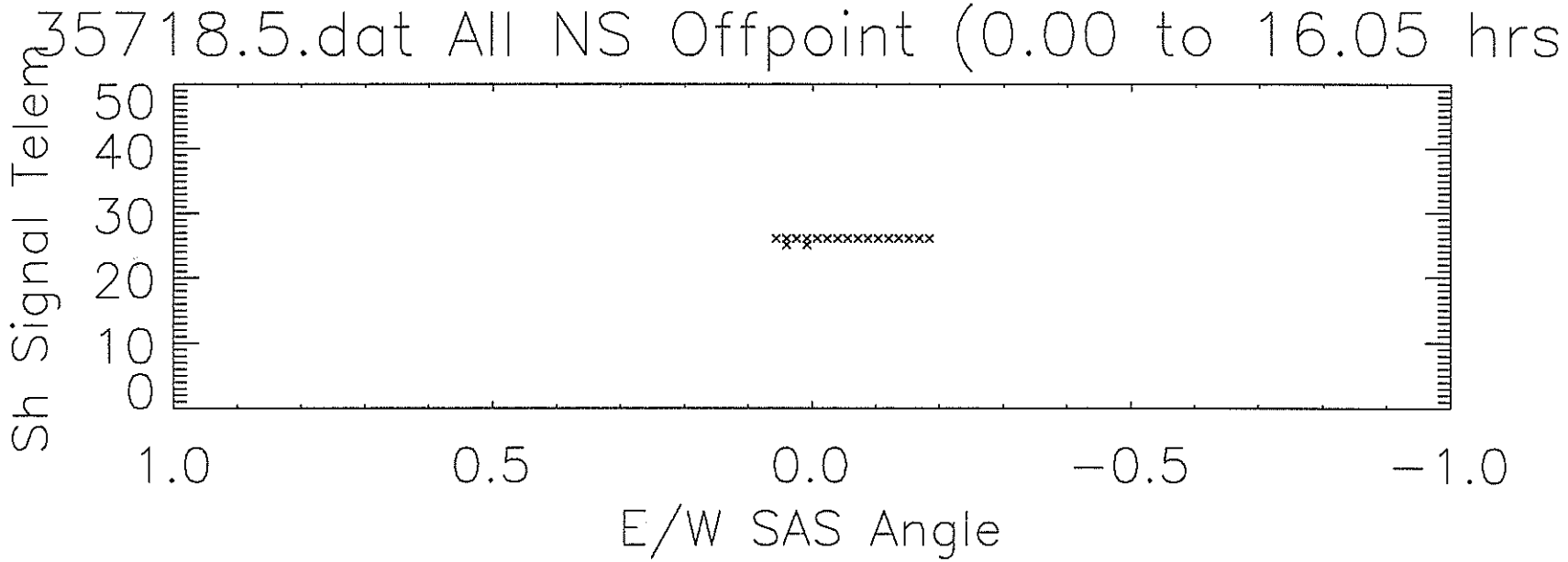

35718.5.dat Slew
Set 0 of 1 (0.00 to 15.82 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

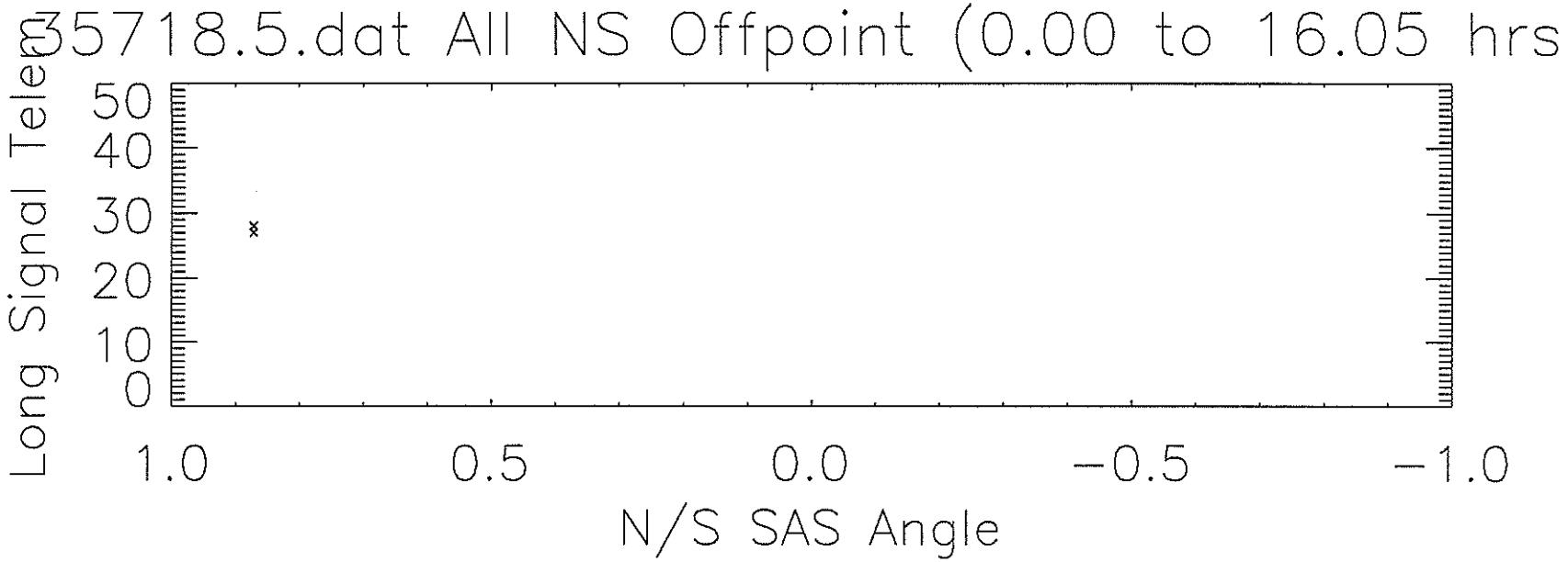

XRP N/S Pos
Flag
3
2
1
0
−1
0
5
10
15
20
25
Time (hrs)
XRP E/W Pos
Pos
3
2
1
0
−1
0
5
10
15
20
25
Time (hrs)
Coarse E/W Pos
Pos
3
2
1
0
−1
0
5
10
15
20
25
Time (hrs)
All Offpoints
Pos
3
2
1
0
−1
0
5
10
15
20
25
Time (hrs)

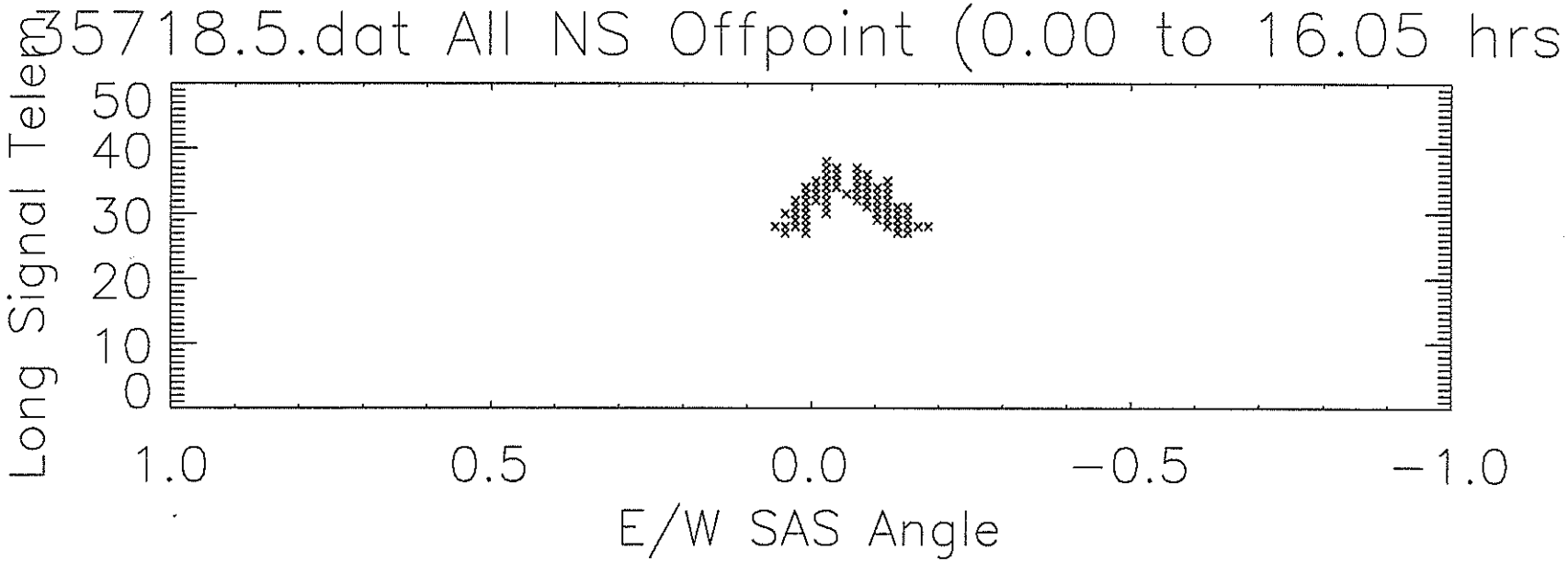

35718.5.dat
All Times (0.00 to 24.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle
35718.5.dat
All Times (0.00 to 24.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

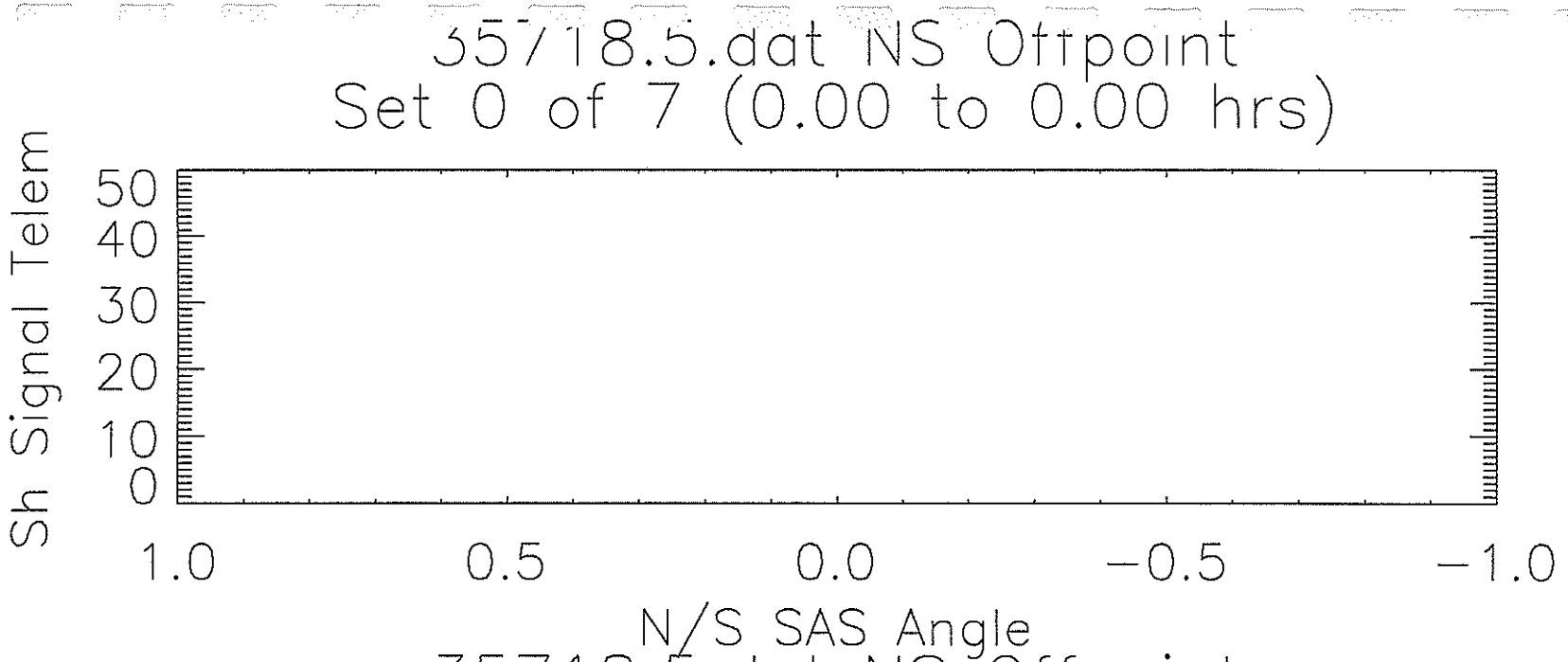

35718.5.dat
All Times (0.00 to 24.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle
35718.5.dat
All Times (0.00 to 24.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

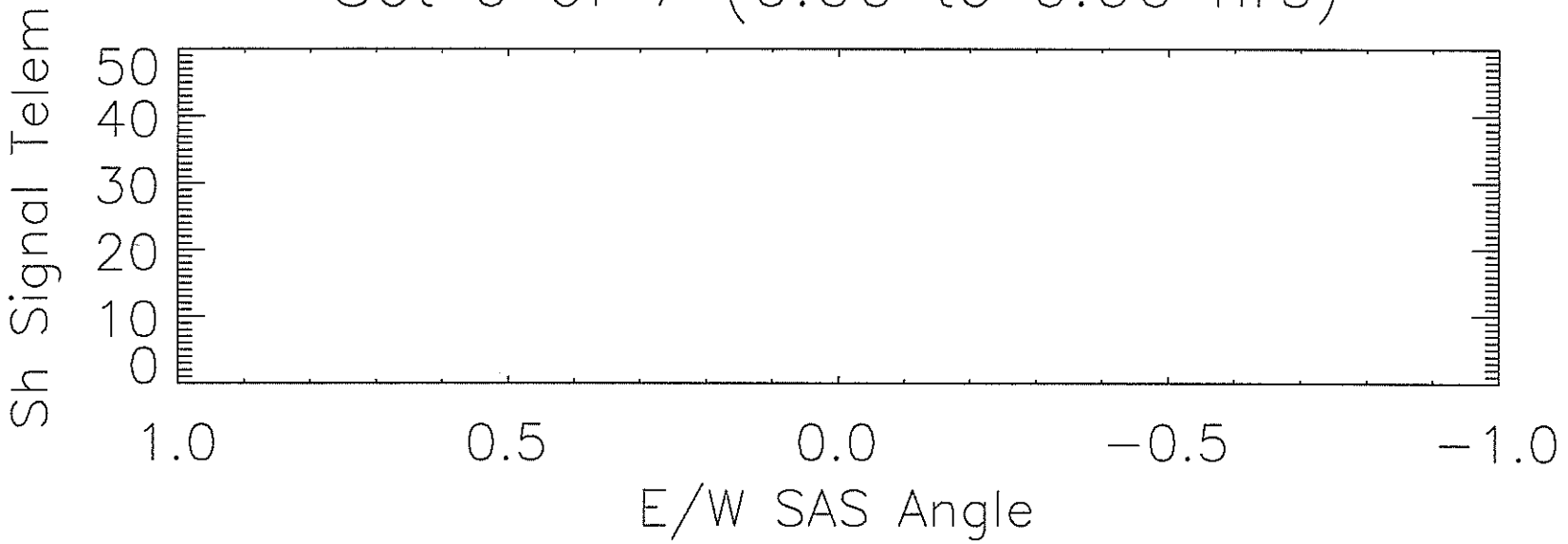

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

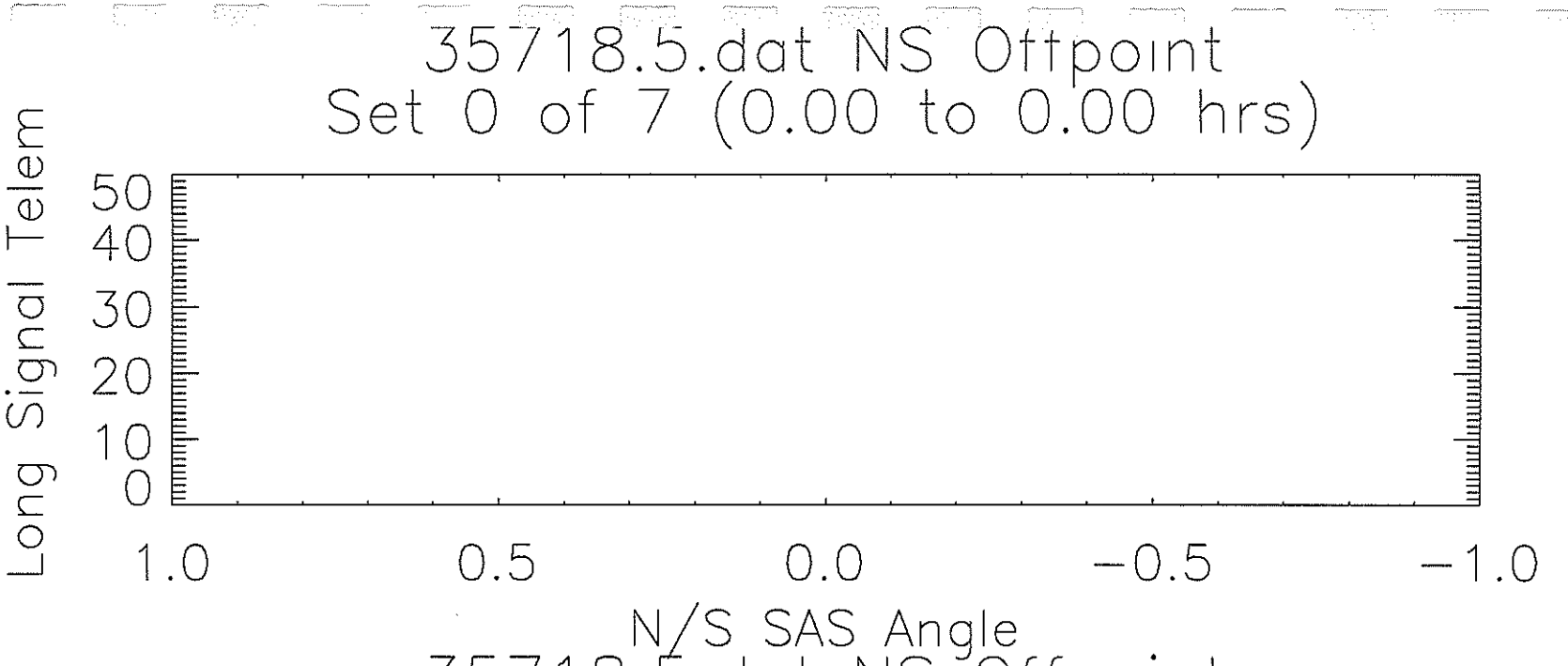

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

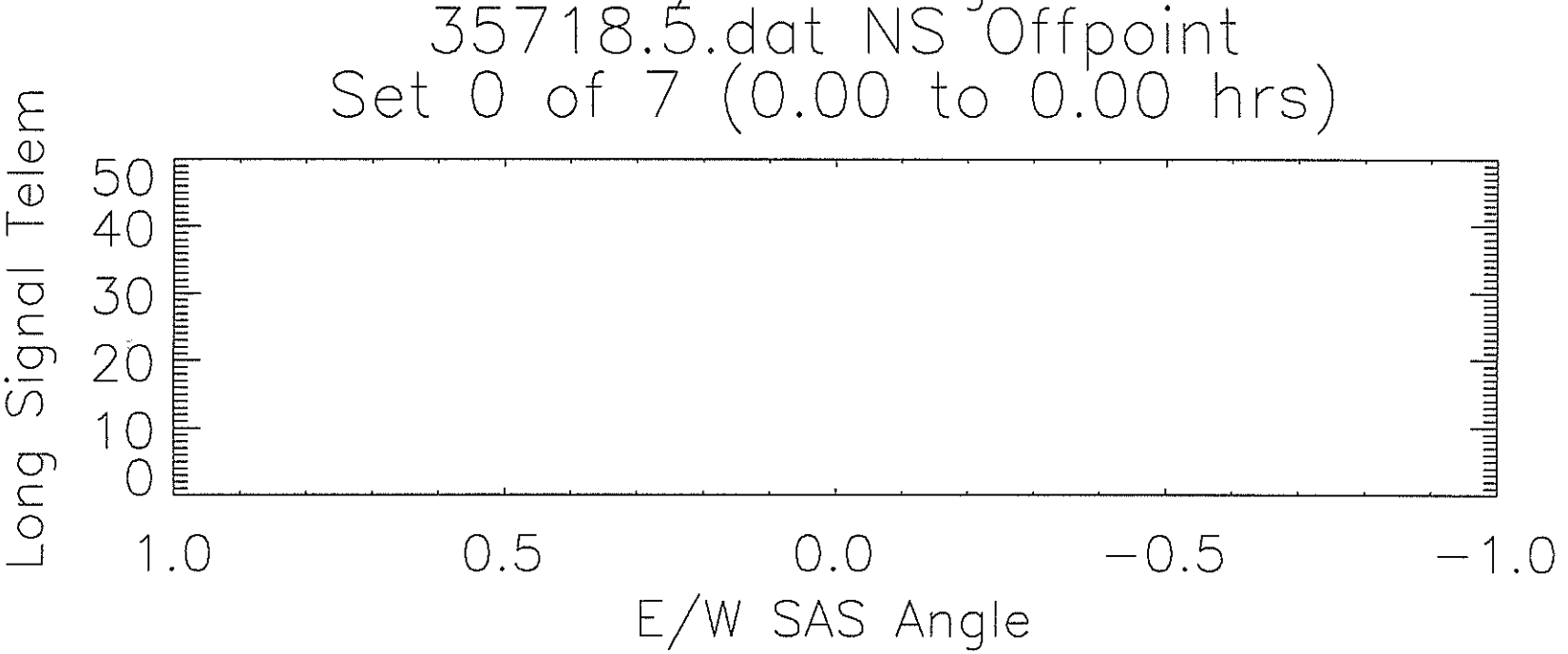

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

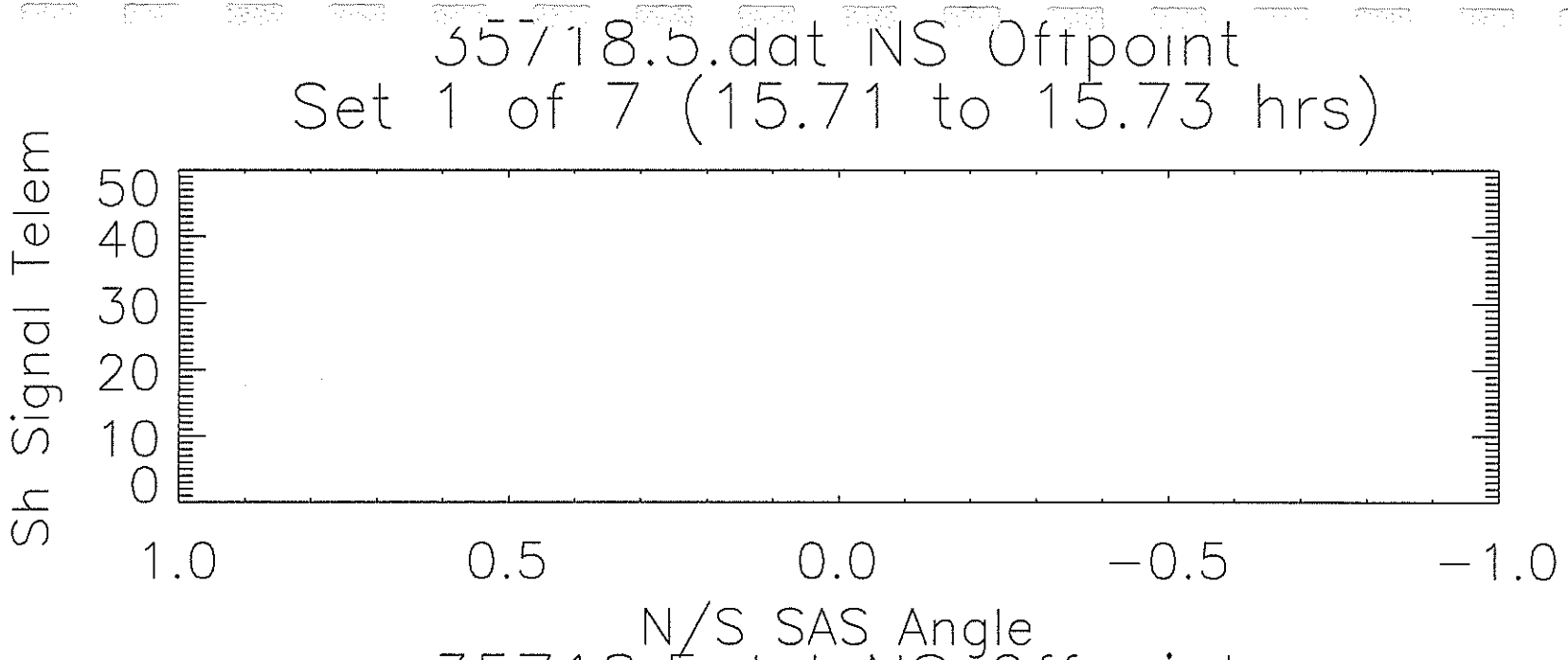

35718.5.dat All NS Offpoint (0.00 to 16.05 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

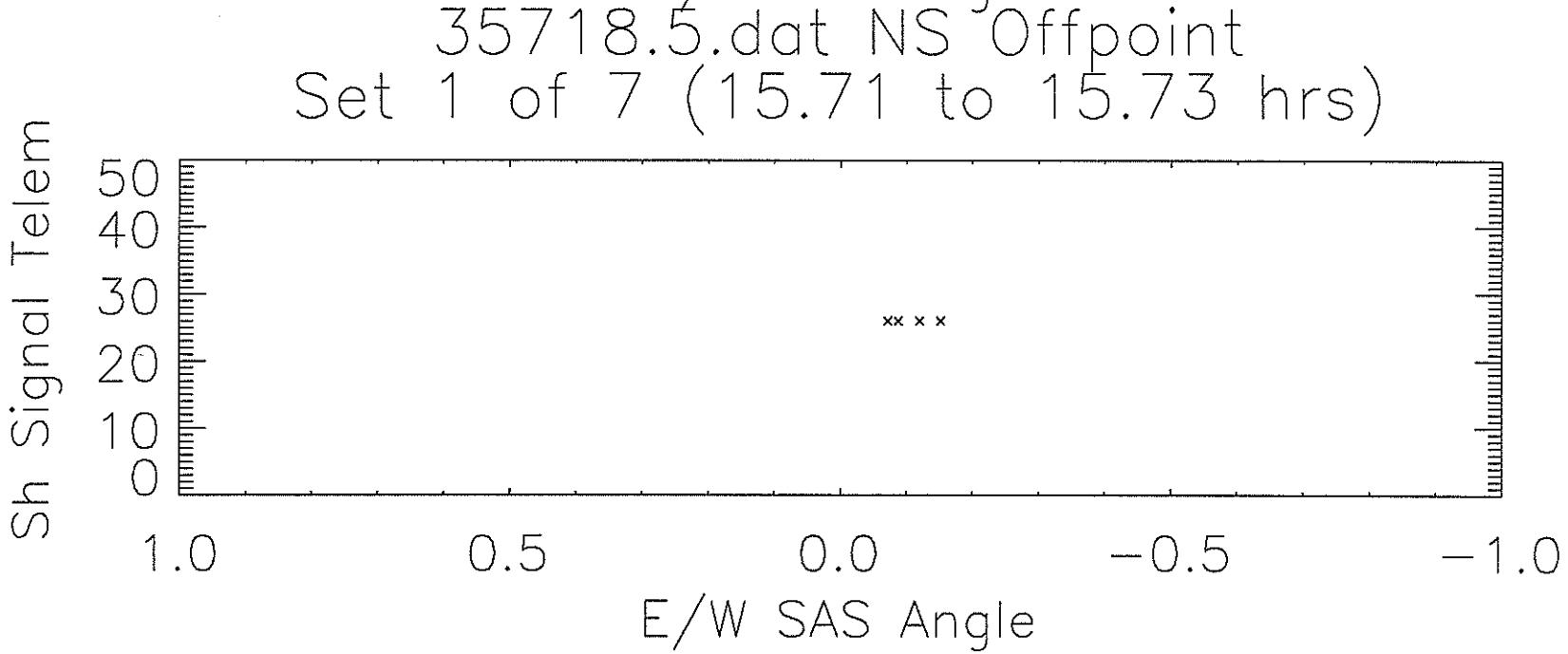

35718.5.dat NS Offpoint
Set 0 of 7 (0.00 to 0.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

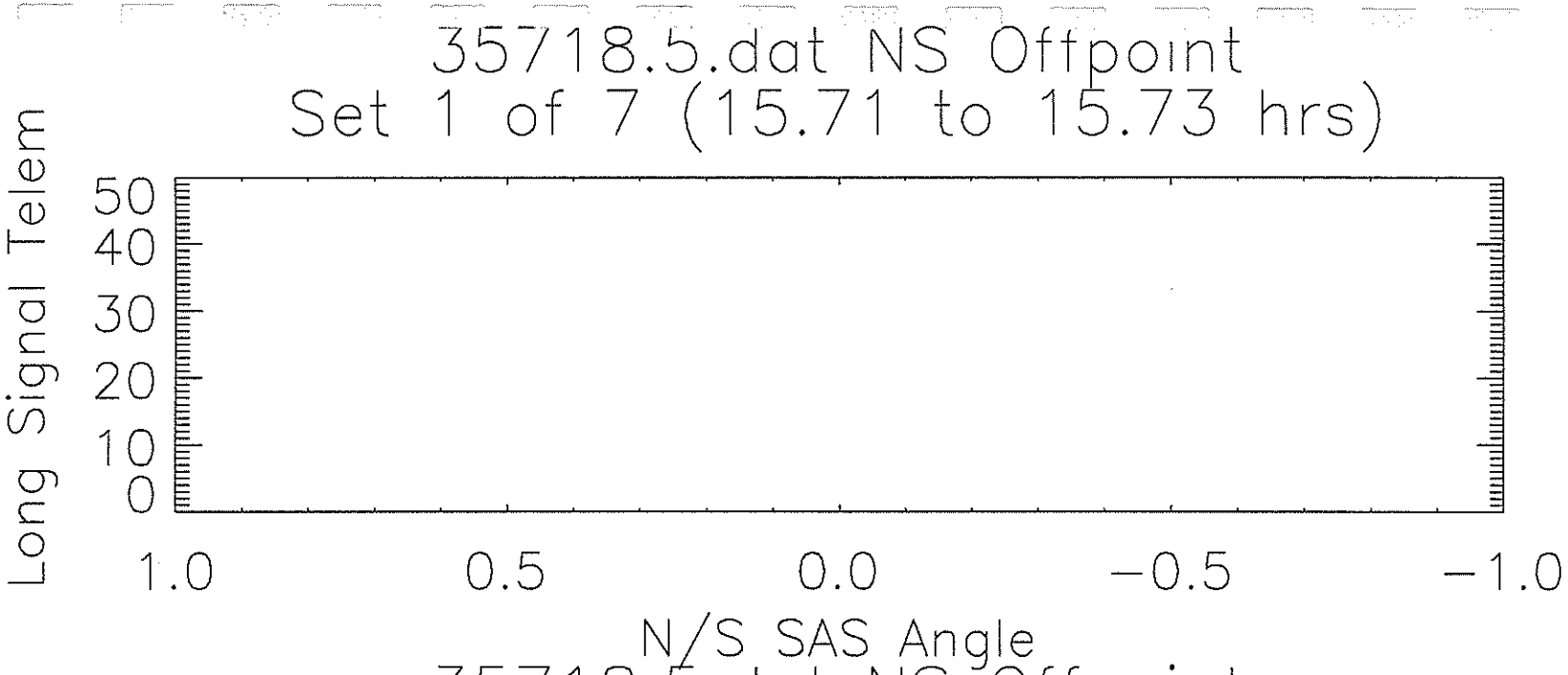

35718.5.dat NS Offpoint
Set 0 of 7 (0.00 to 0.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

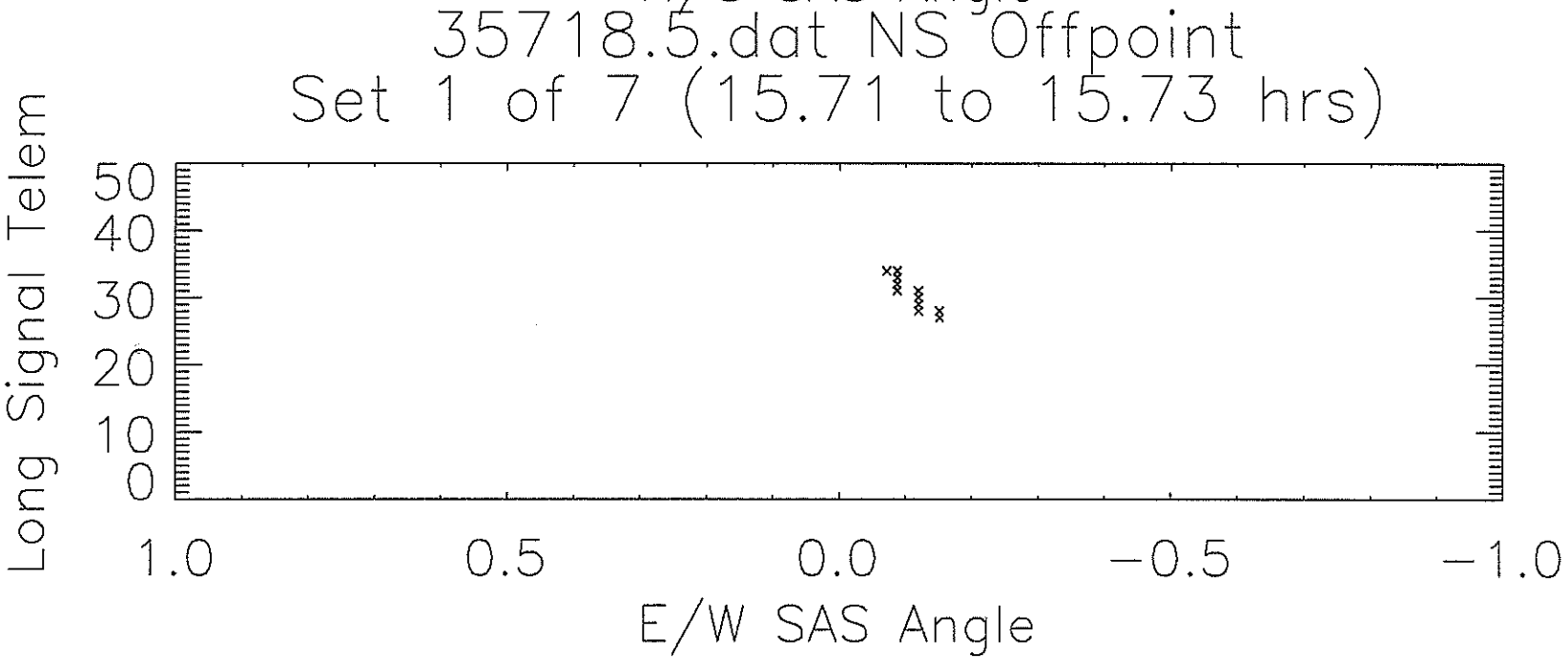

35718.5.dat NS Offpoint
Set 0 of 7 (0.00 to 0.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

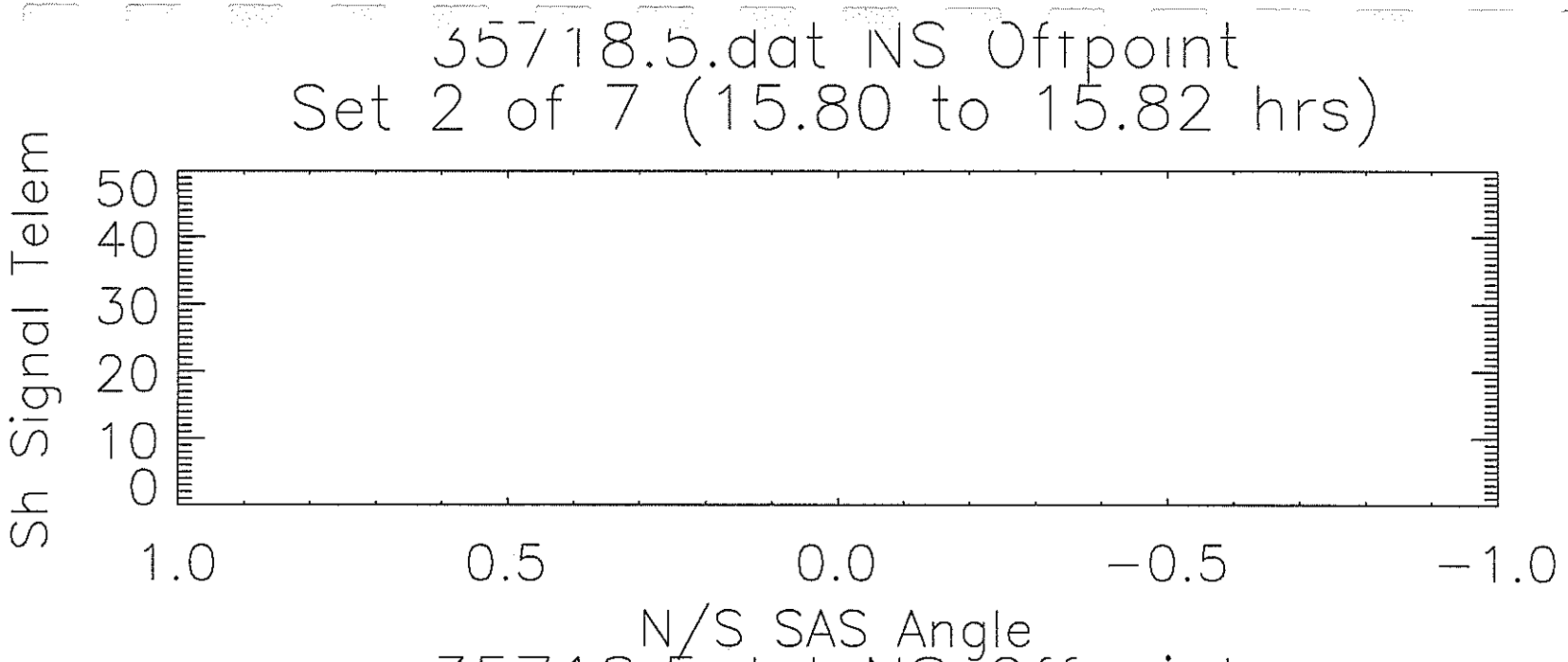

35718.5.dat NS Offpoint
Set 0 of 7 (0.00 to 0.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

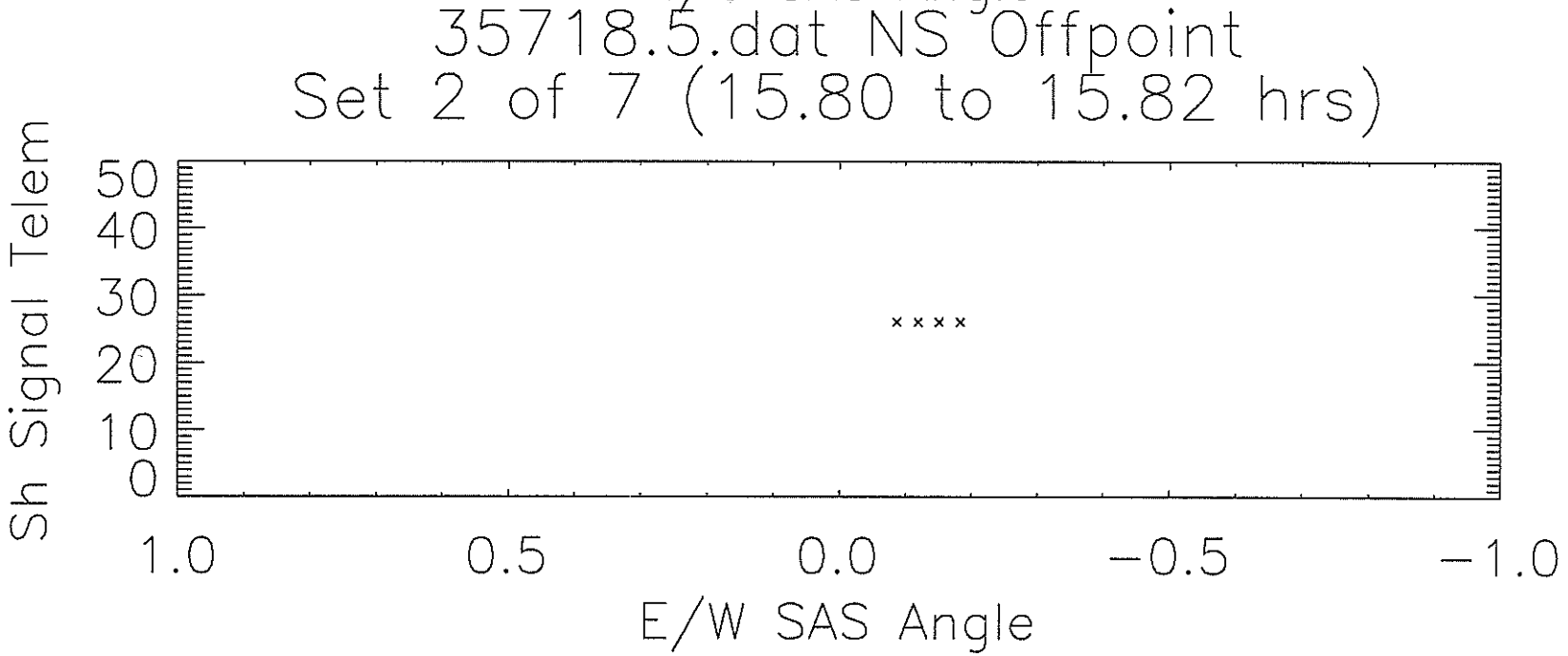

35718.5.dat NS Offpoint
Set 1 of 7 (15.71 to 15.73 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

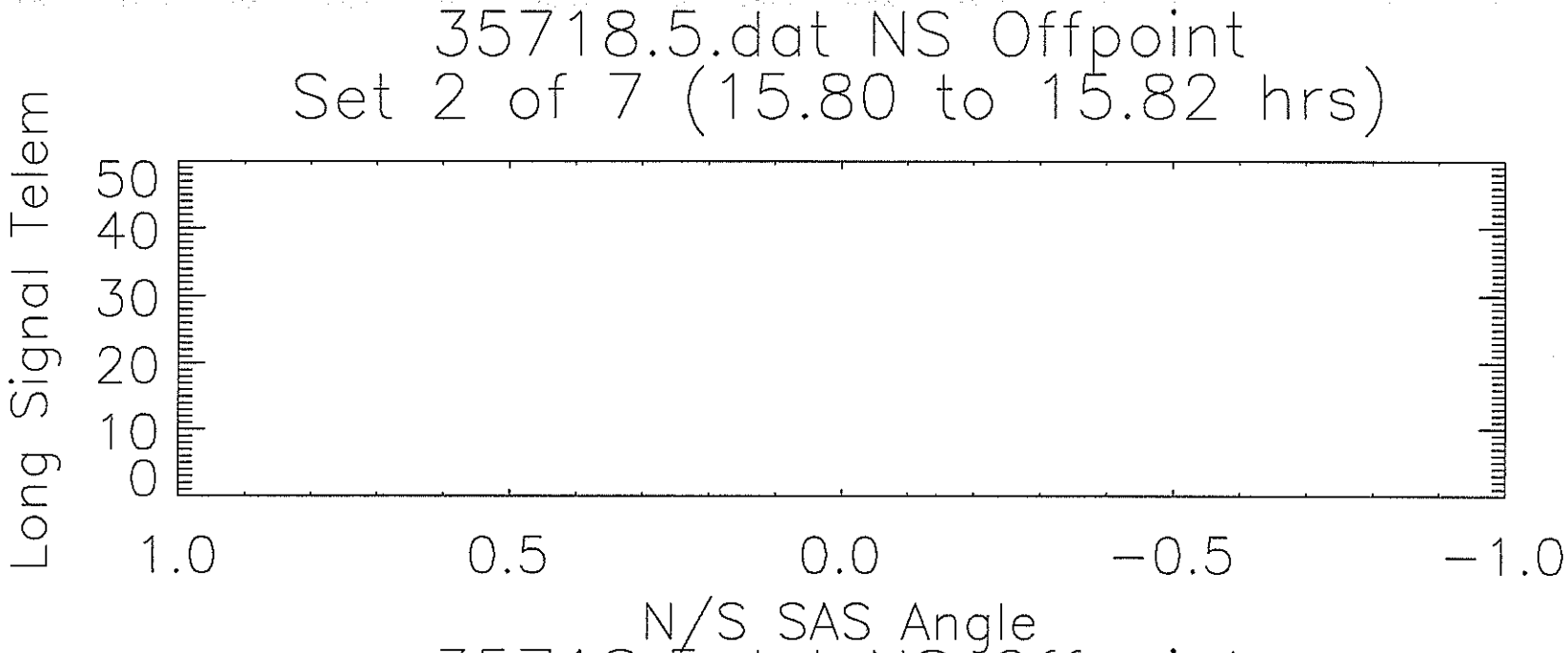

35718.5.dat NS Offpoint
Set 1 of 7 (15.71 to 15.73 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

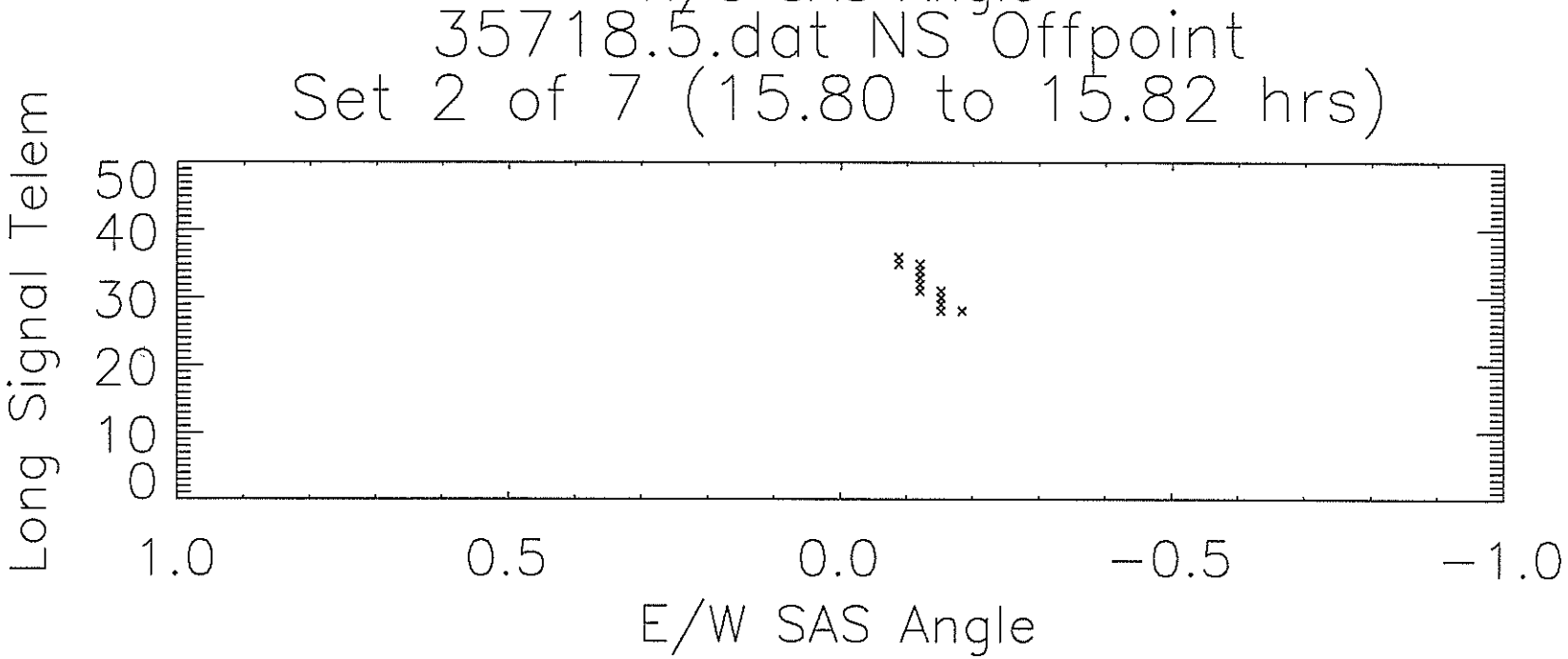

35718.5.dat NS Offpoint
Set 1 of 7 (15.71 to 15.73 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

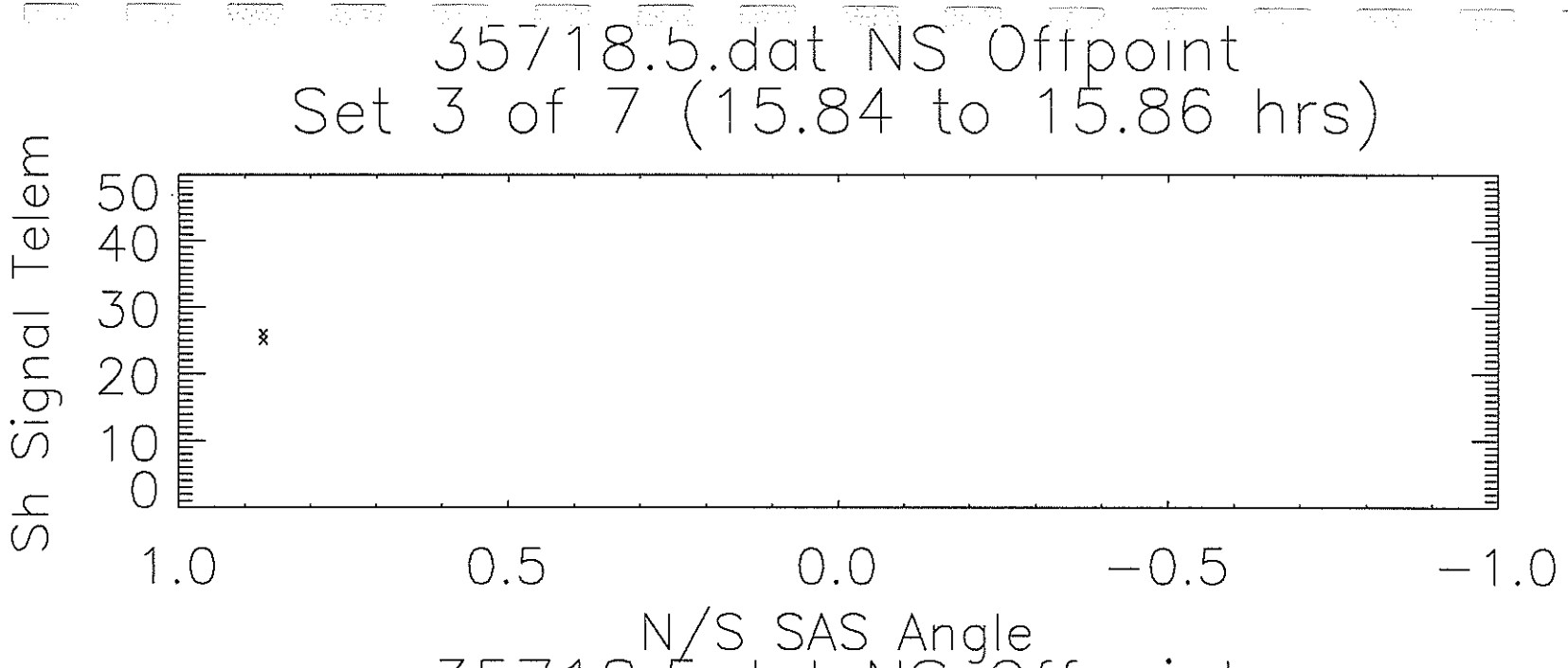

35718.5.dat NS Offpoint
Set 1 of 7 (15.71 to 15.73 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

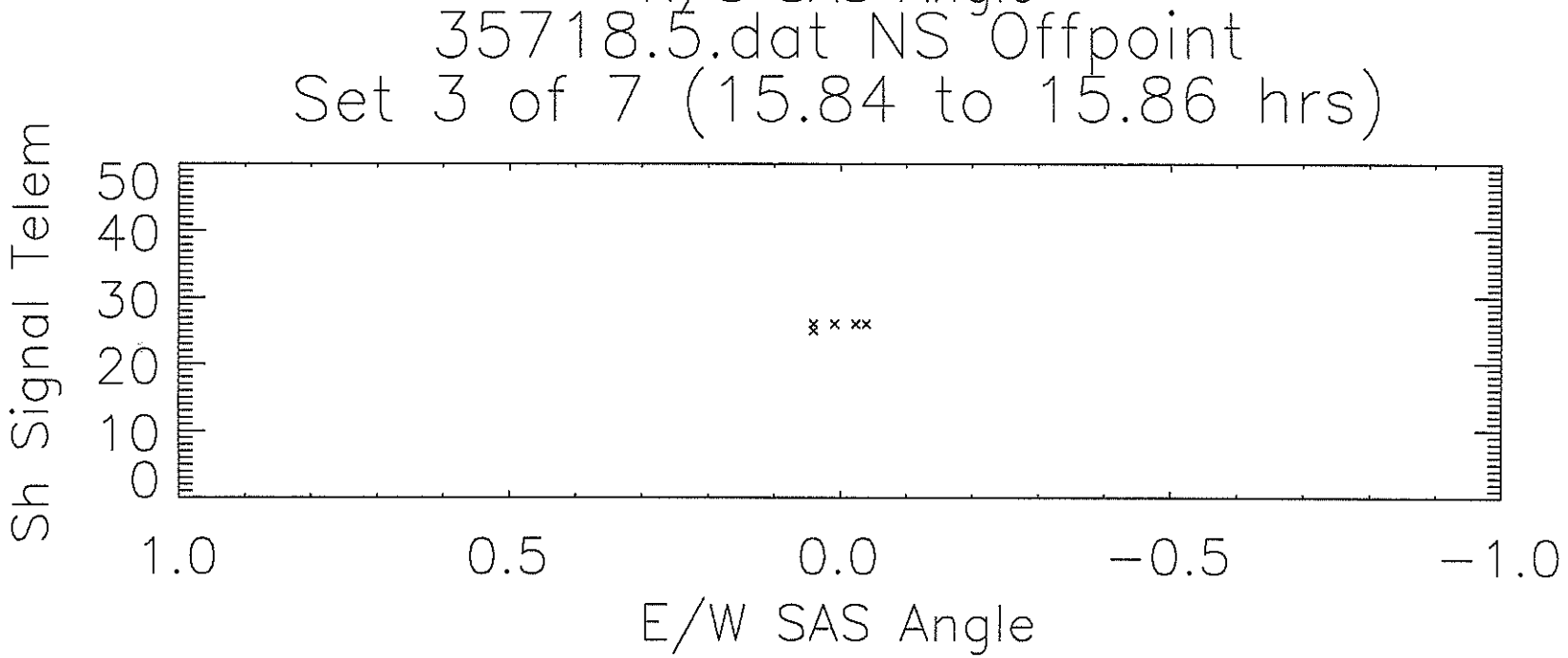

35718.5.dat NS Offpoint
Set 2 of 7 (15.80 to 15.82 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

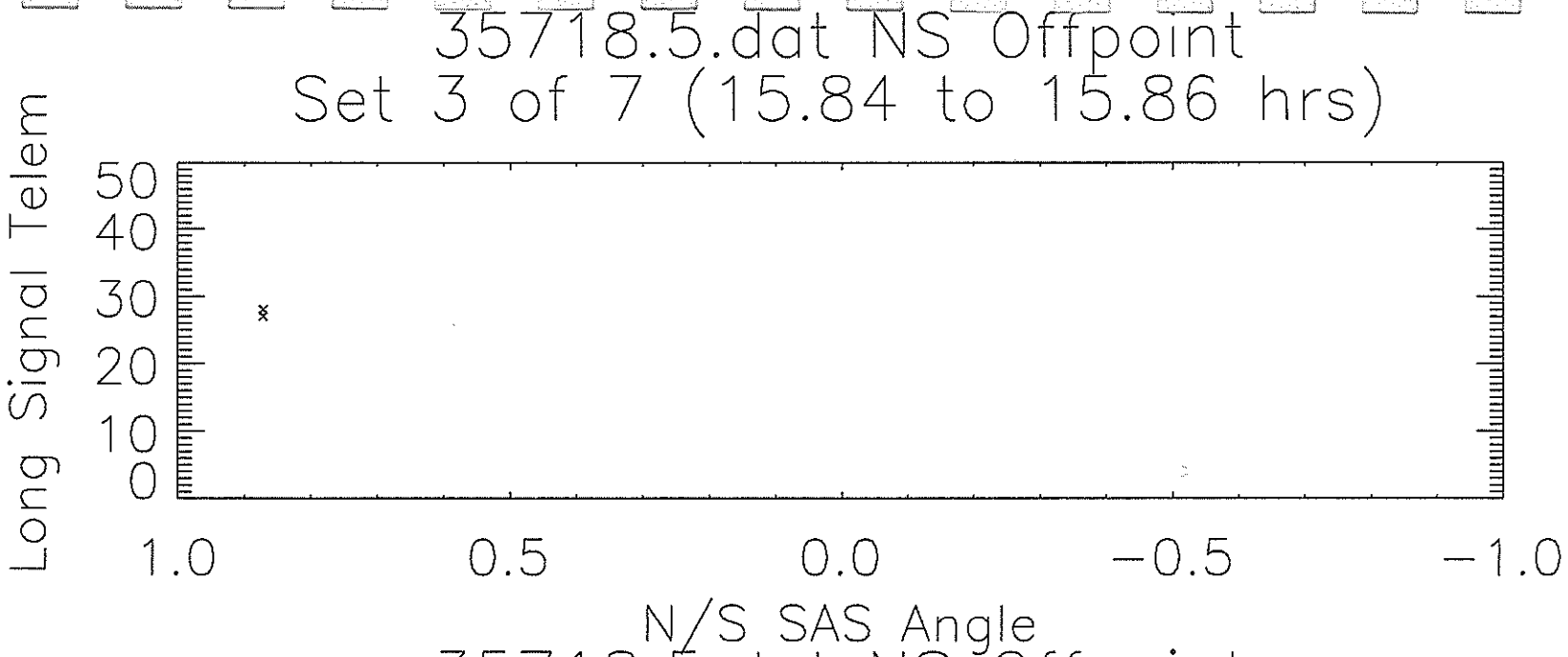

35718.5.dat NS Offpoint
Set 2 of 7 (15.80 to 15.82 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

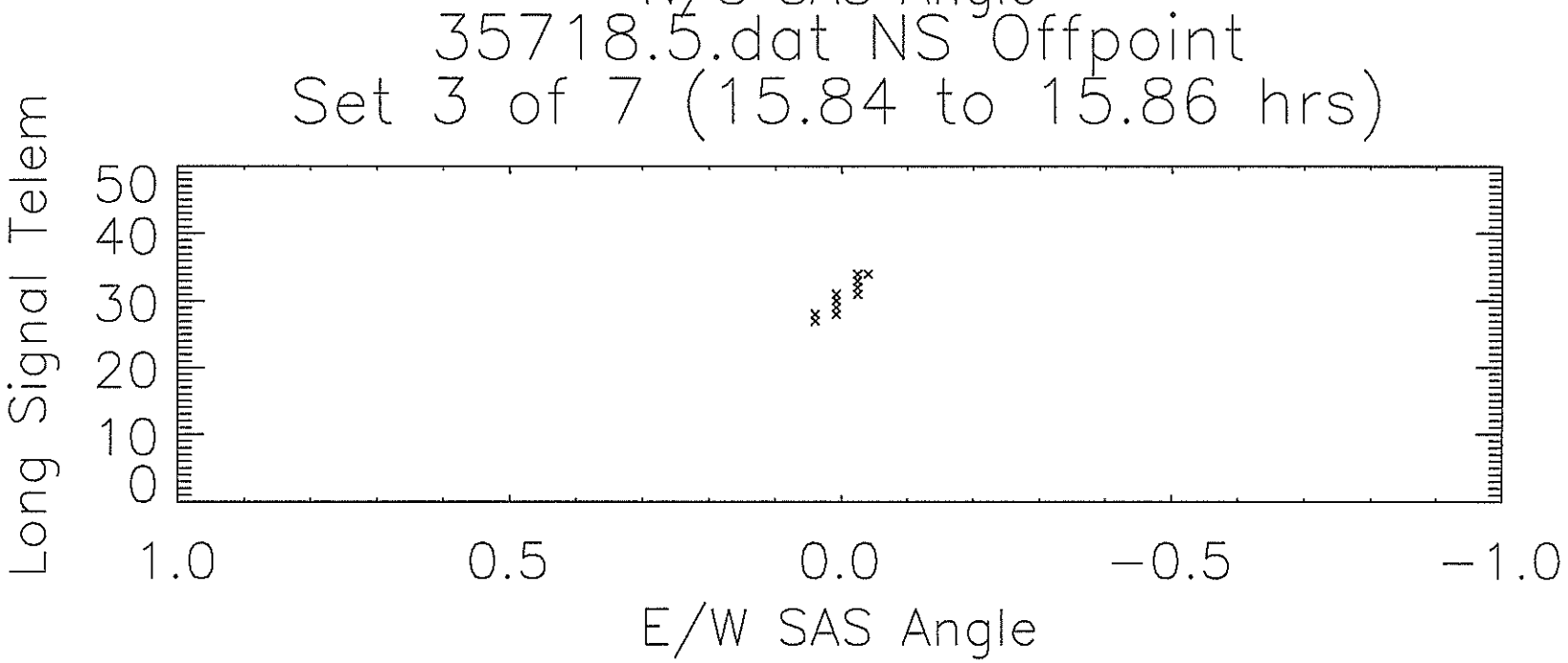

35718.5.dat NS Offpoint
Set 2 of 7 (15.80 to 15.82 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

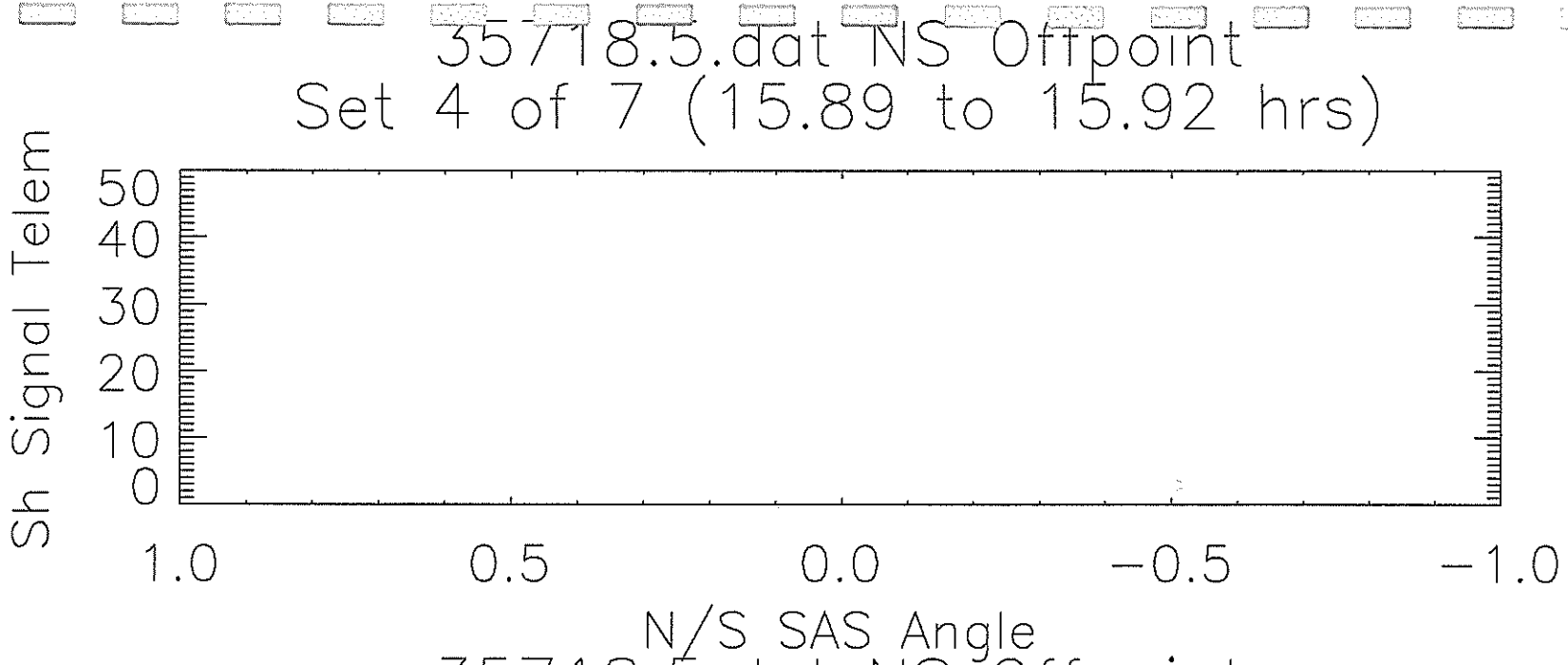

35718.5.dat NS Offpoint
Set 2 of 7 (15.80 to 15.82 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

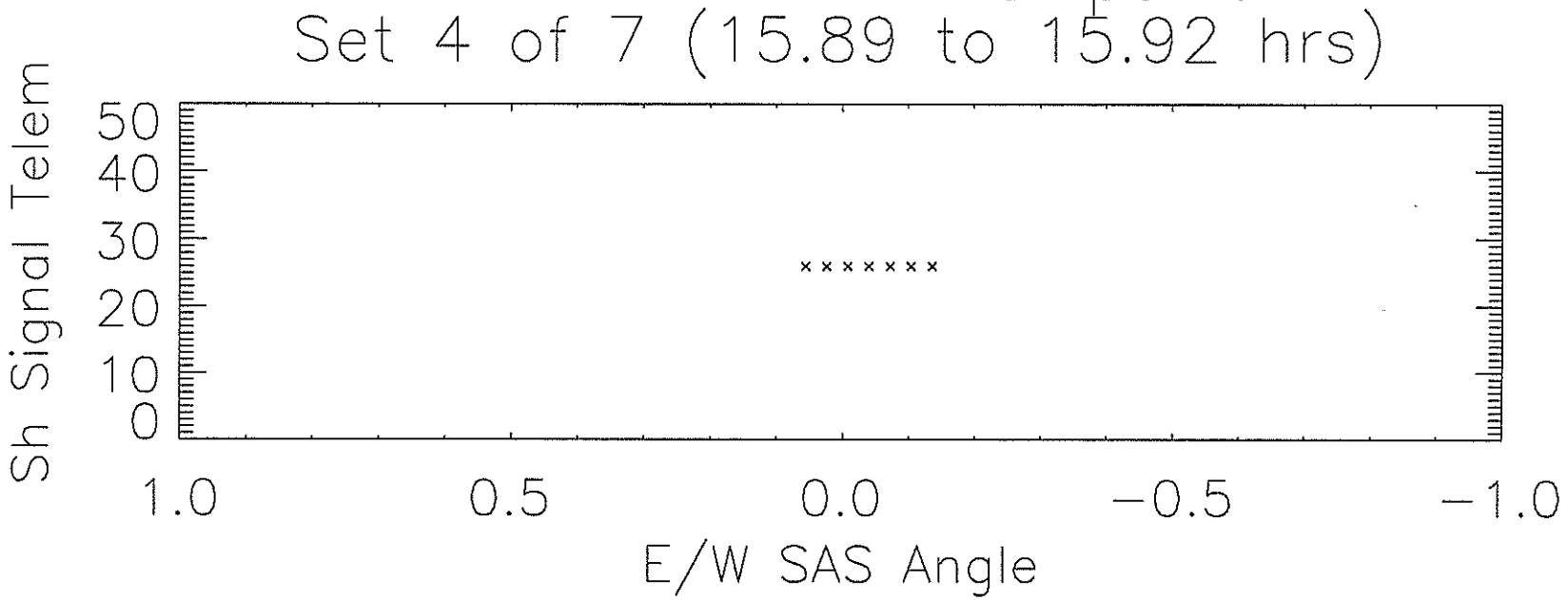

35718.5.dat NS Offpoint
Set 3 of 7 (15.84 to 15.86 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

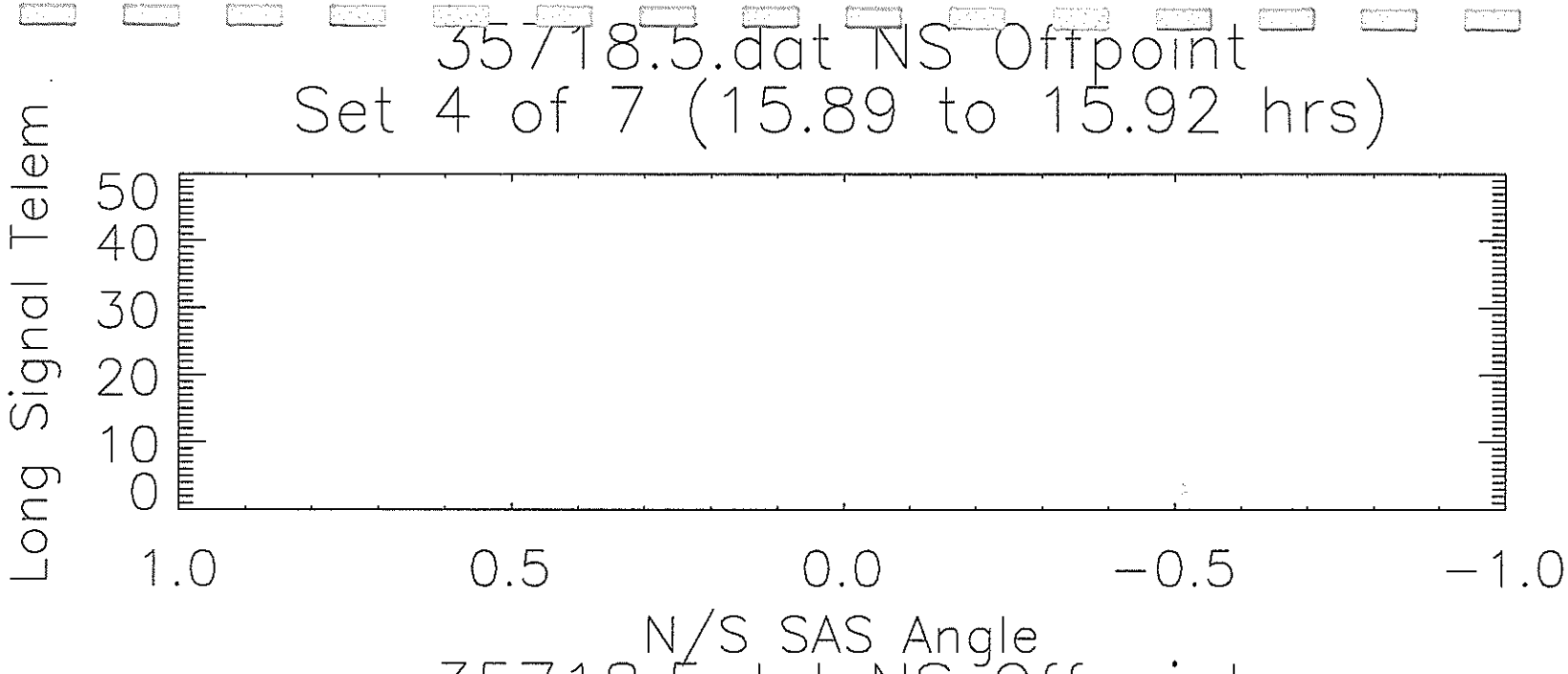

35718.5.dat NS Offpoint
Set 3 of 7 (15.84 to 15.86 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

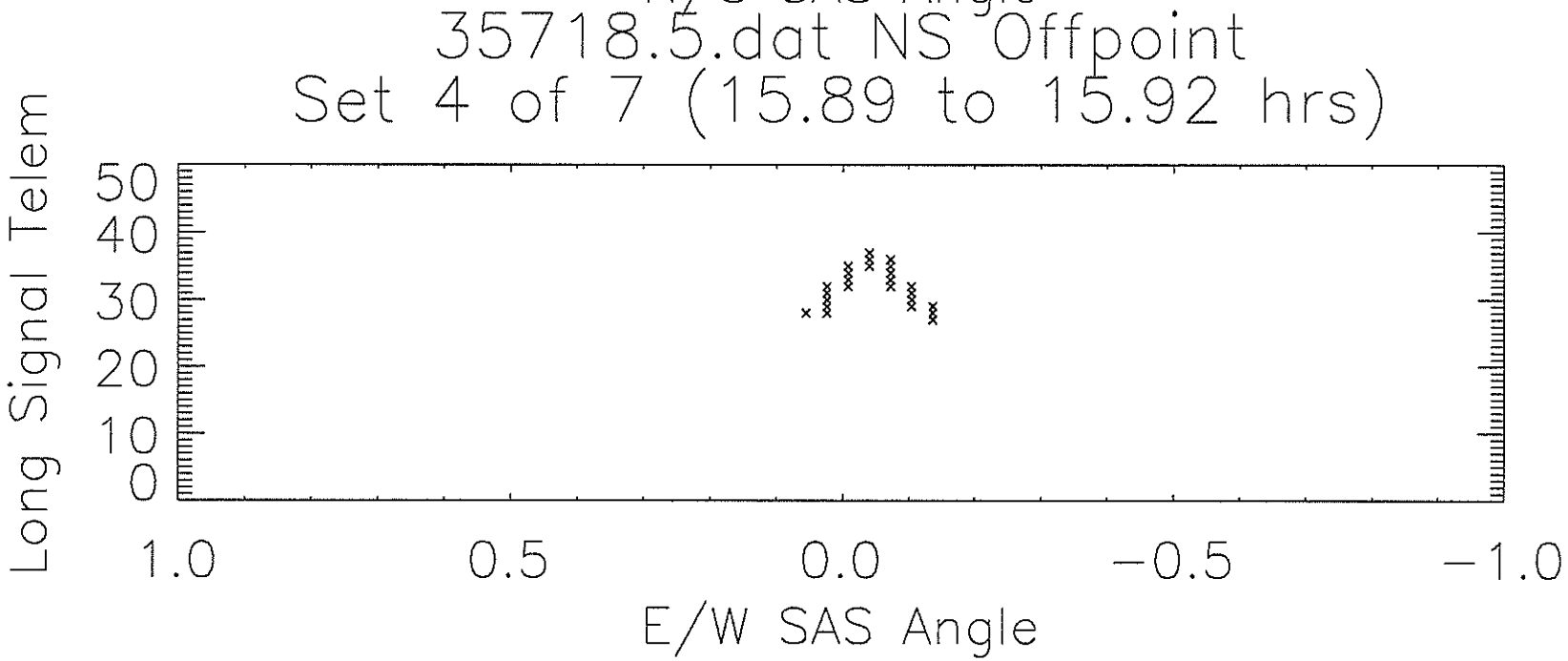

35718.5.dat NS Offpoint
Set 3 of 7 (15.84 to 15.86 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

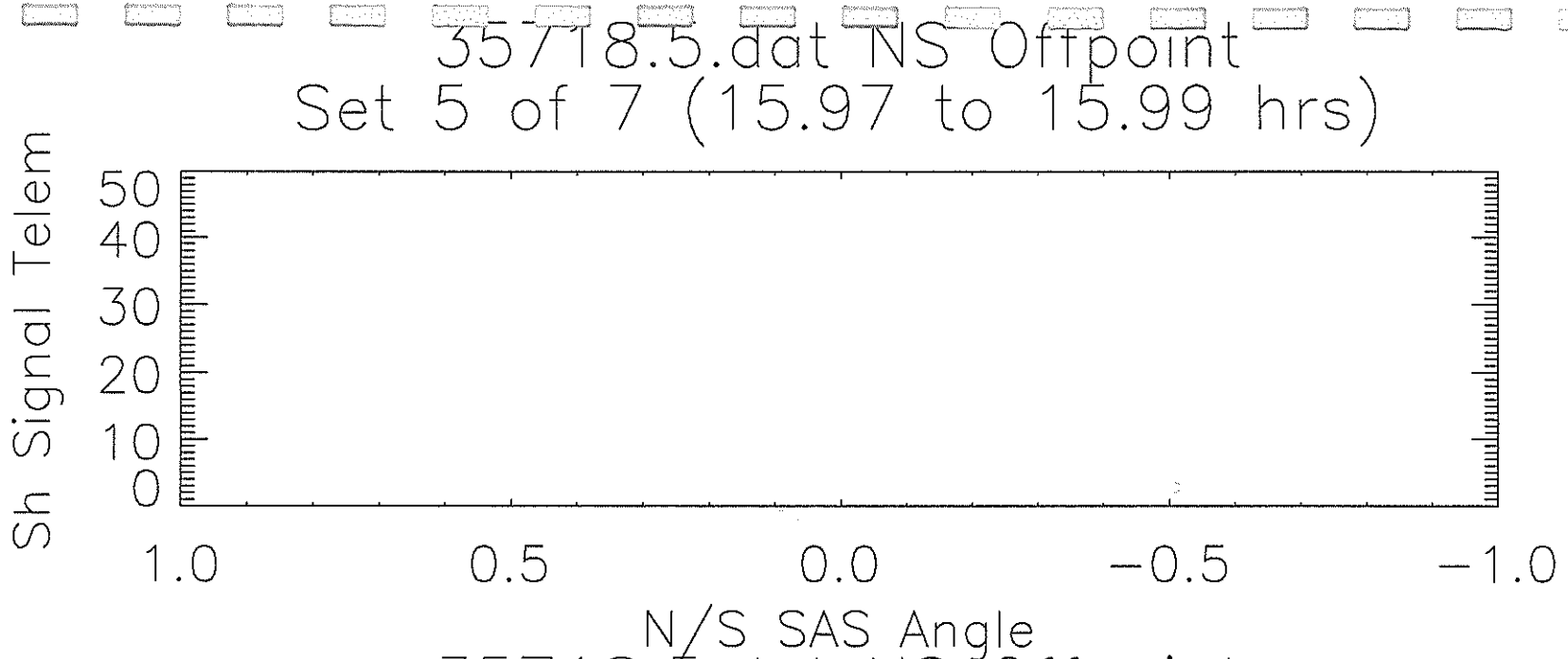

35718.5.dat NS Offpoint
Set 3 of 7 (15.84 to 15.86 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

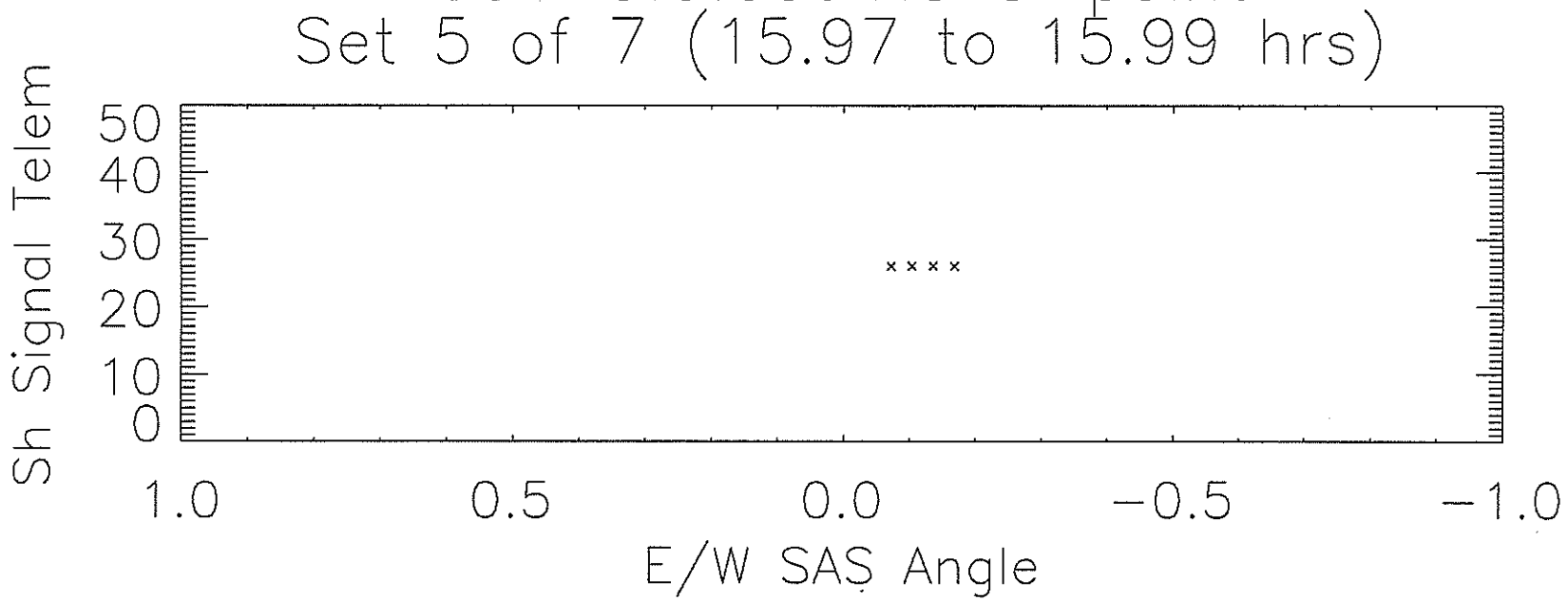

35718.5.dat NS Offpoint
Set 4 of 7 (15.89 to 15.92 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

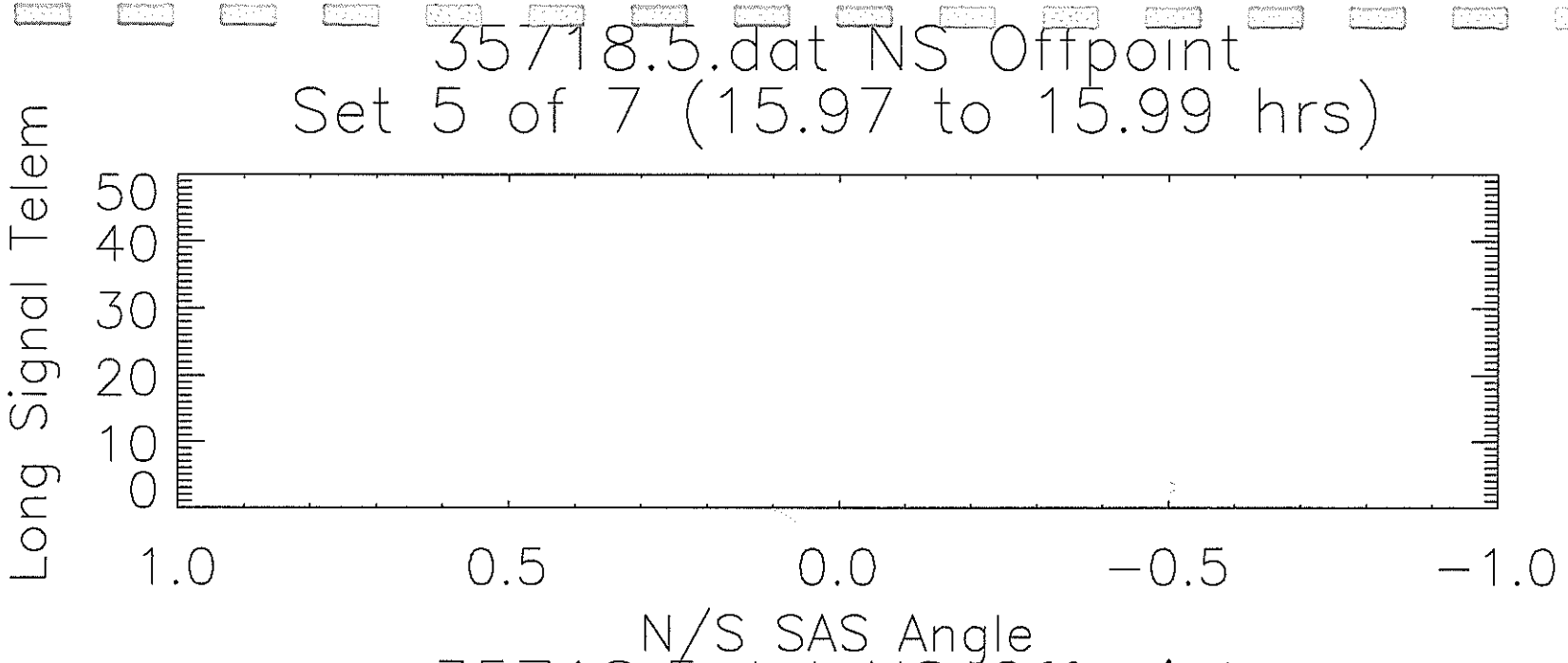

35718.5.dat NS Offpoint
Set 4 of 7 (15.89 to 15.92 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

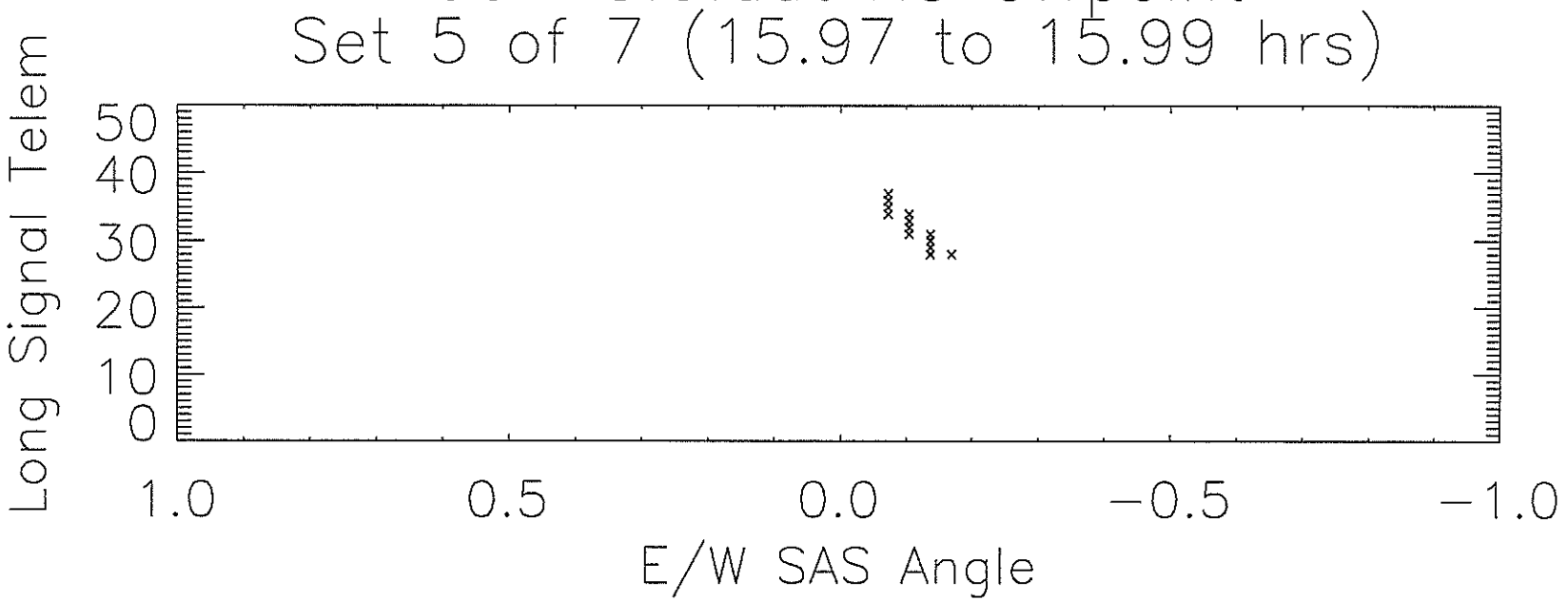

35718.5.dat NS Offpoint
Set 4 of 7 (15.89 to 15.92 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

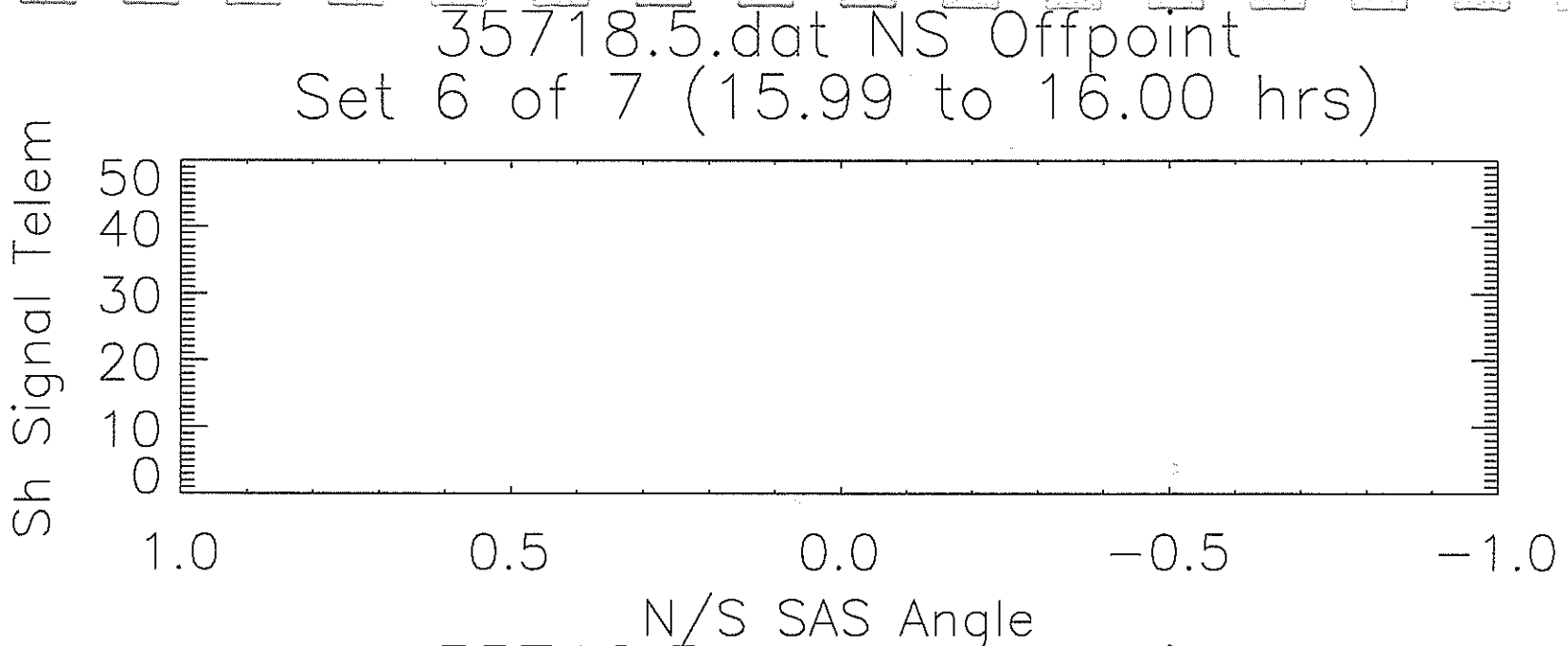

35718.5.dat NS Offpoint
Set 4 of 7 (15.89 to 15.92 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

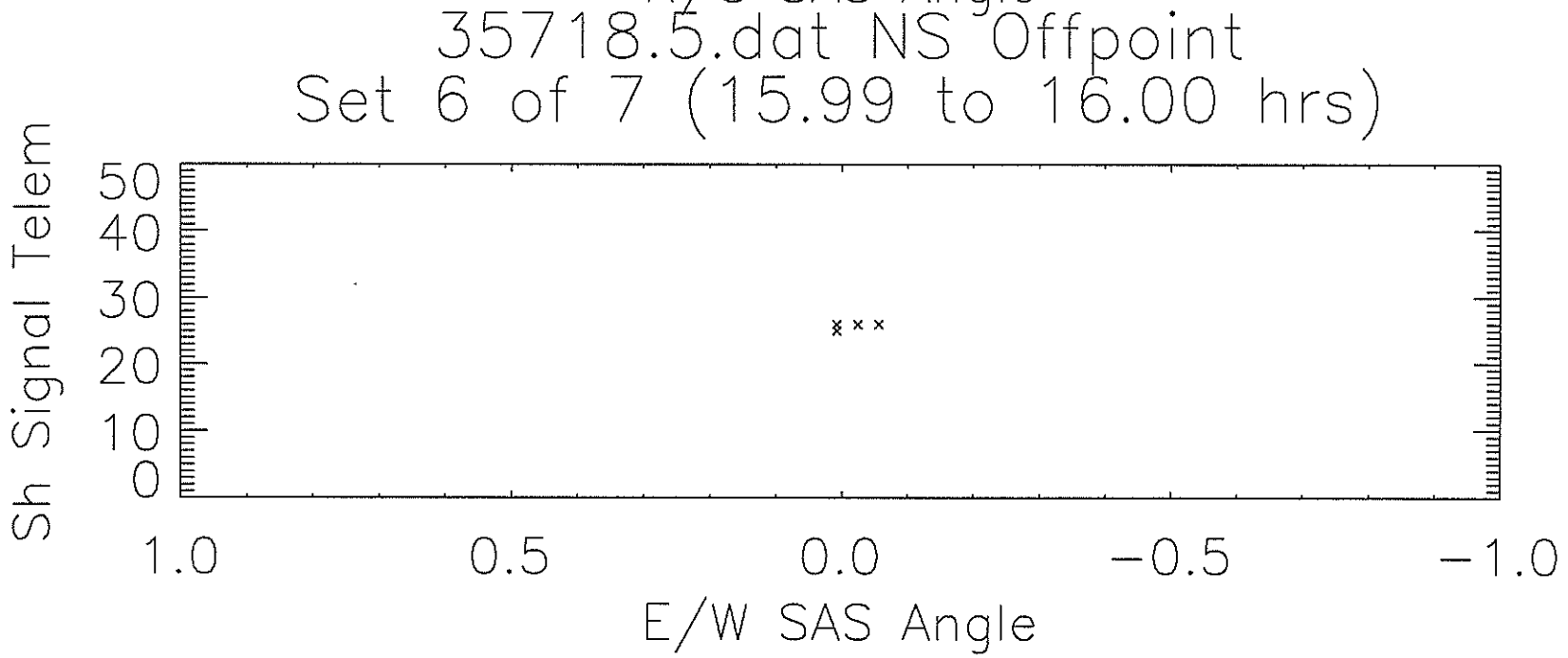

35718.5.dat NS Offpoint
Set 5 of 7 (15.97 to 15.99 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

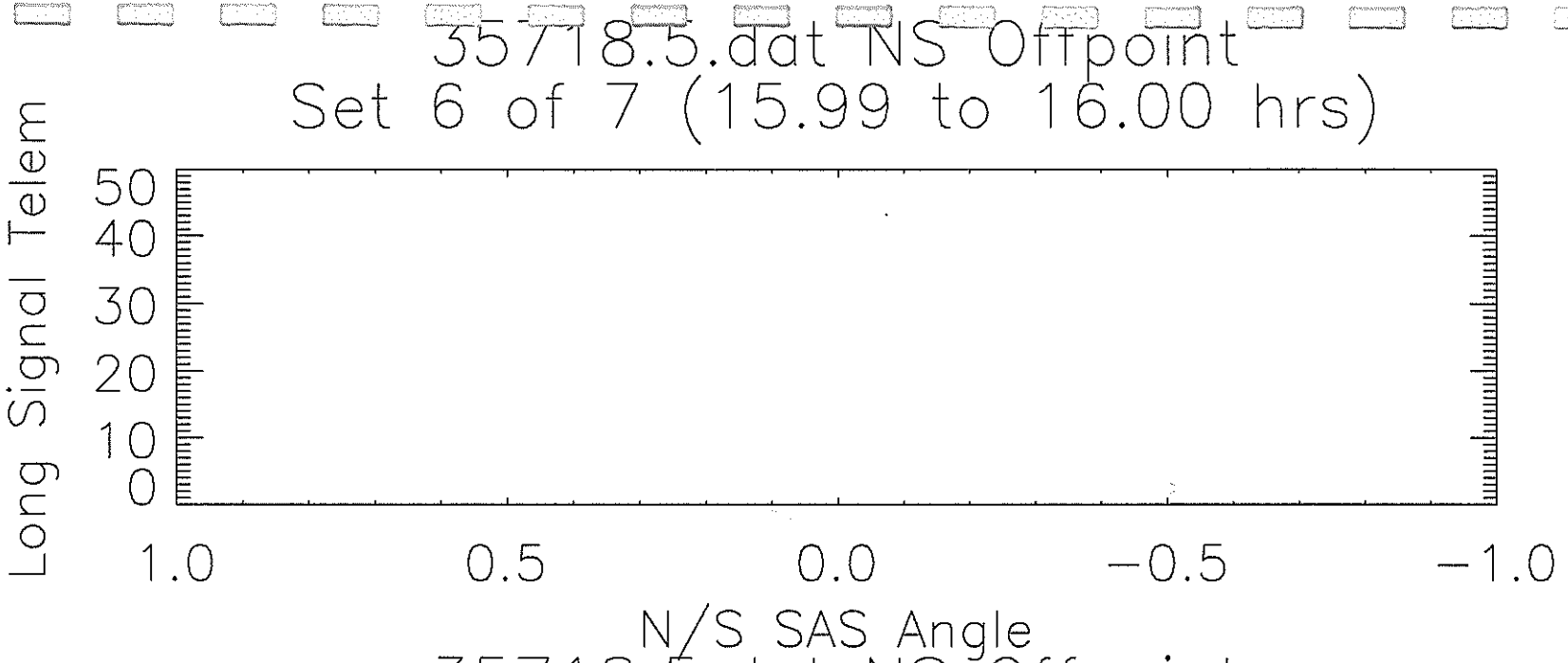

35718.5.dat NS Offpoint
Set 5 of 7 (15.97 to 15.99 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

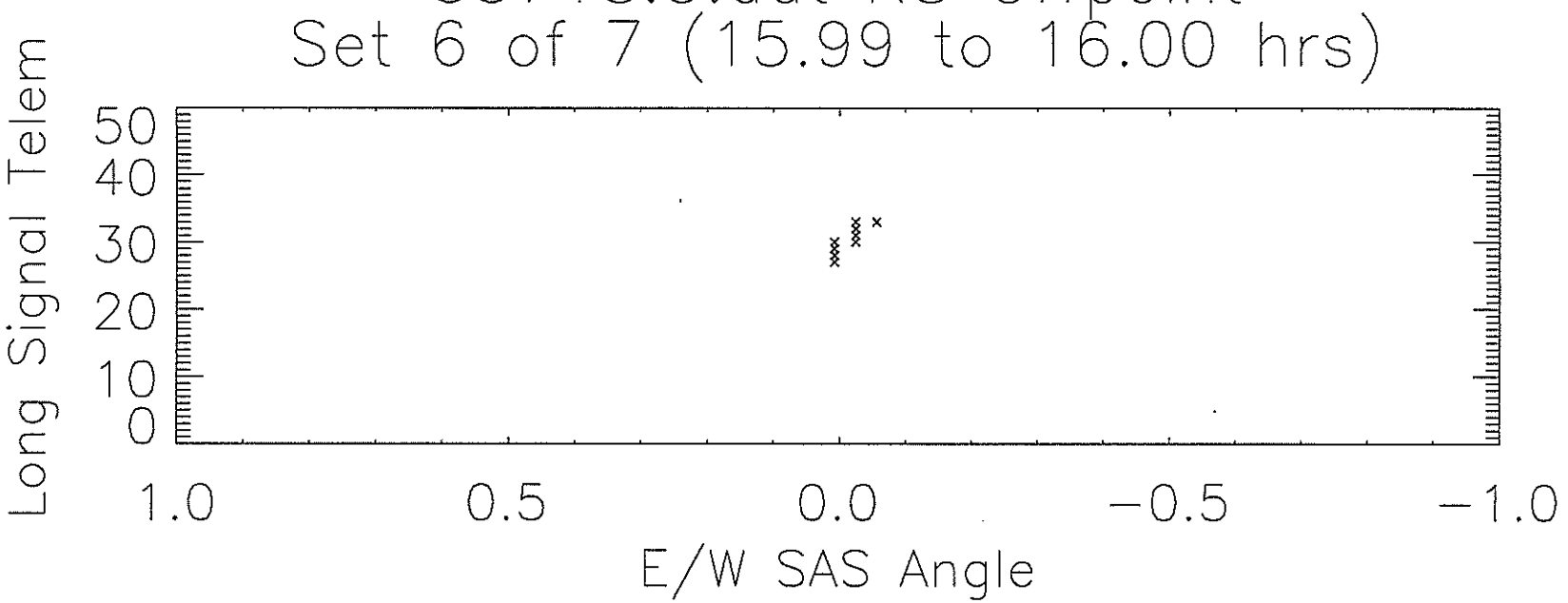

35718.5.dat NS Offpoint
Set 5 of 7 (15.97 to 15.99 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

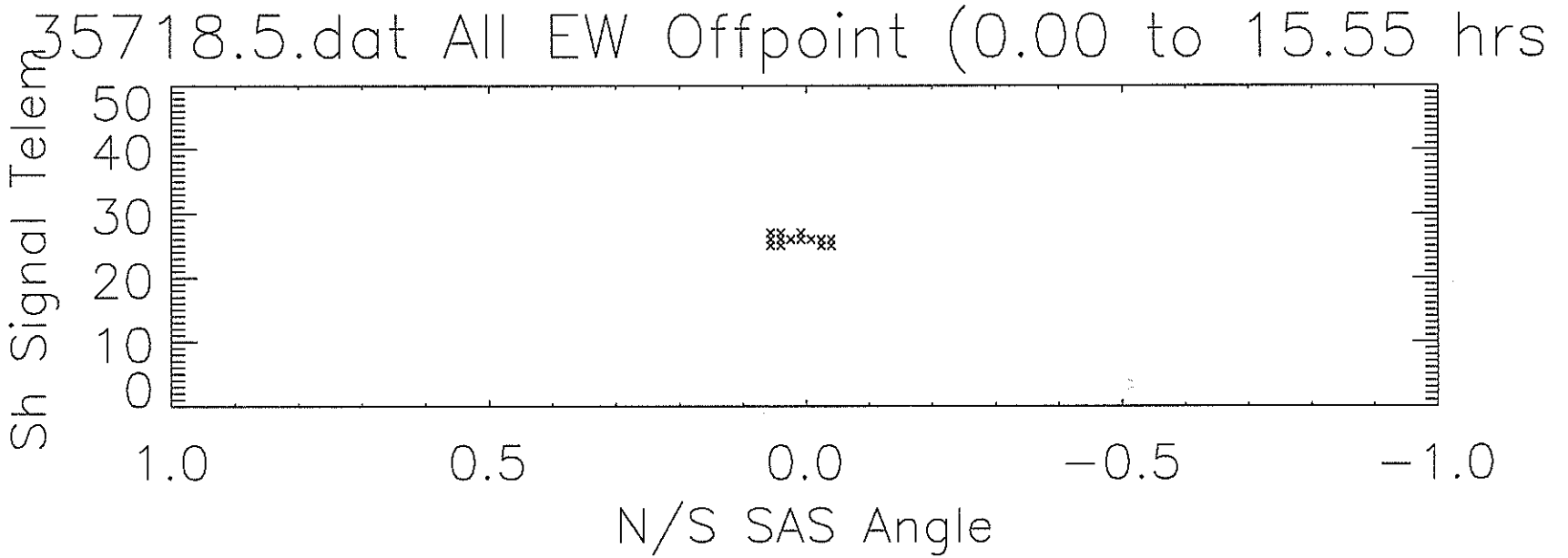

35718.5.dat NS Offpoint
Set 5 of 7 (15.97 to 15.99 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

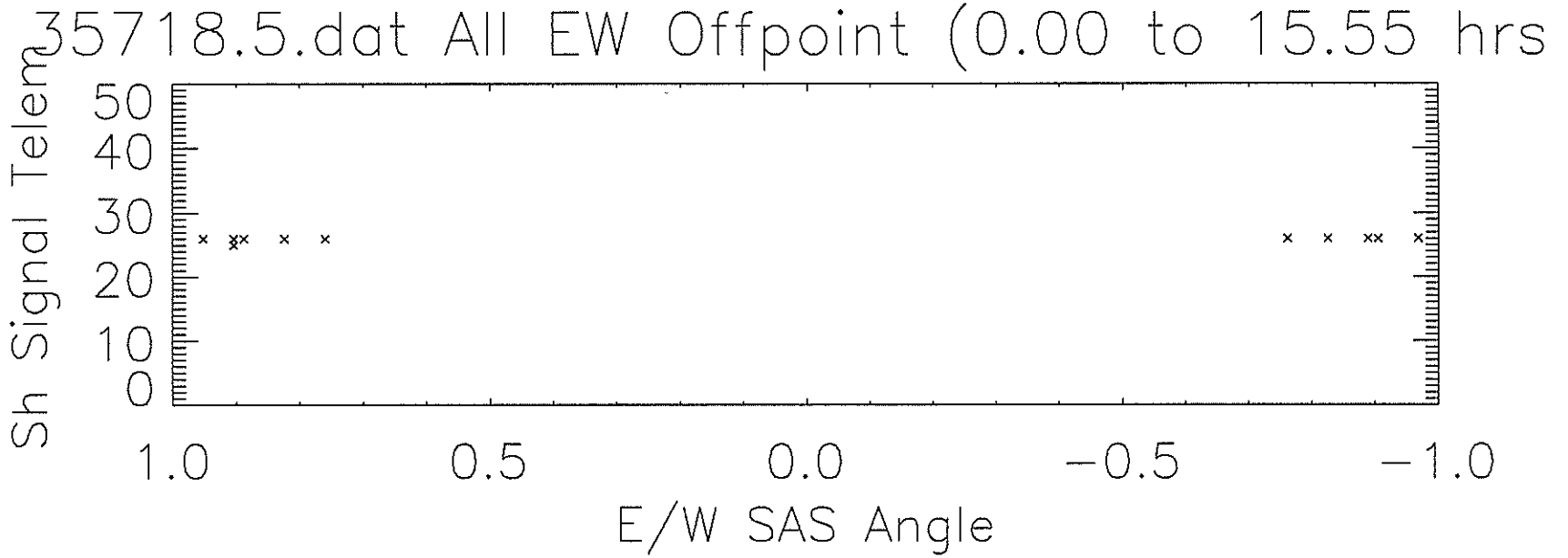

35718.5.dat NS Offpoint
Set 6 of 7 (15.99 to 16.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

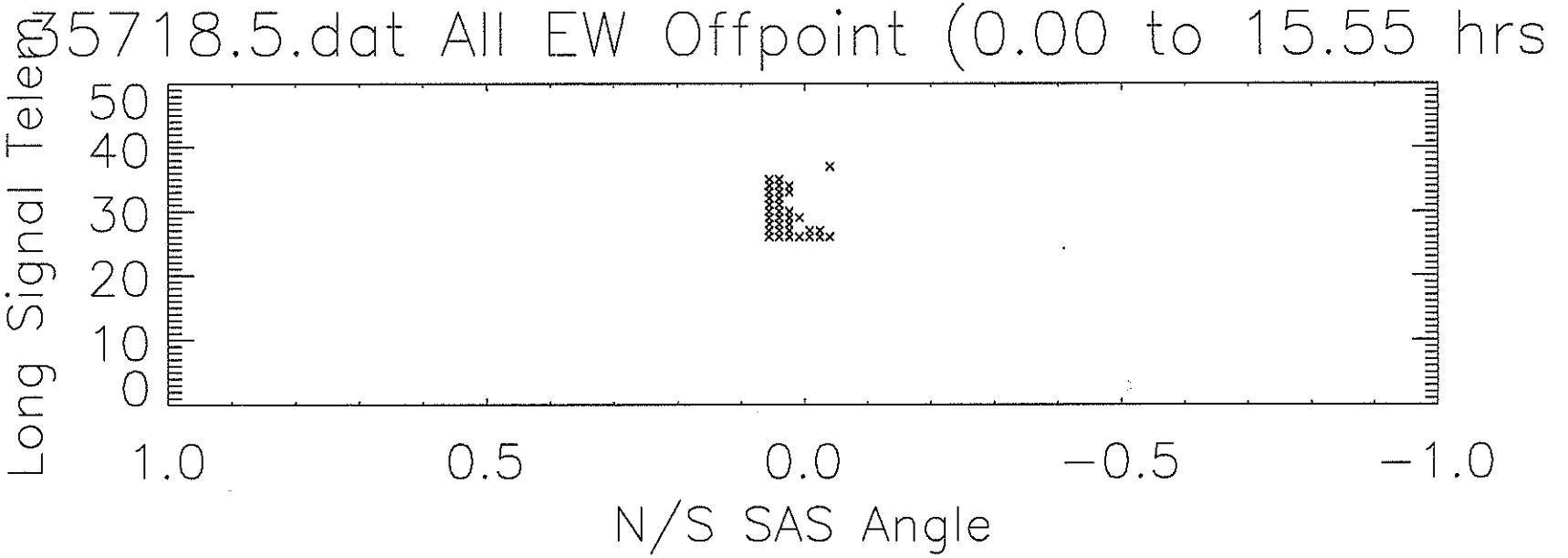

35718.5.dat NS Offpoint
Set 6 of 7 (15.99 to 16.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

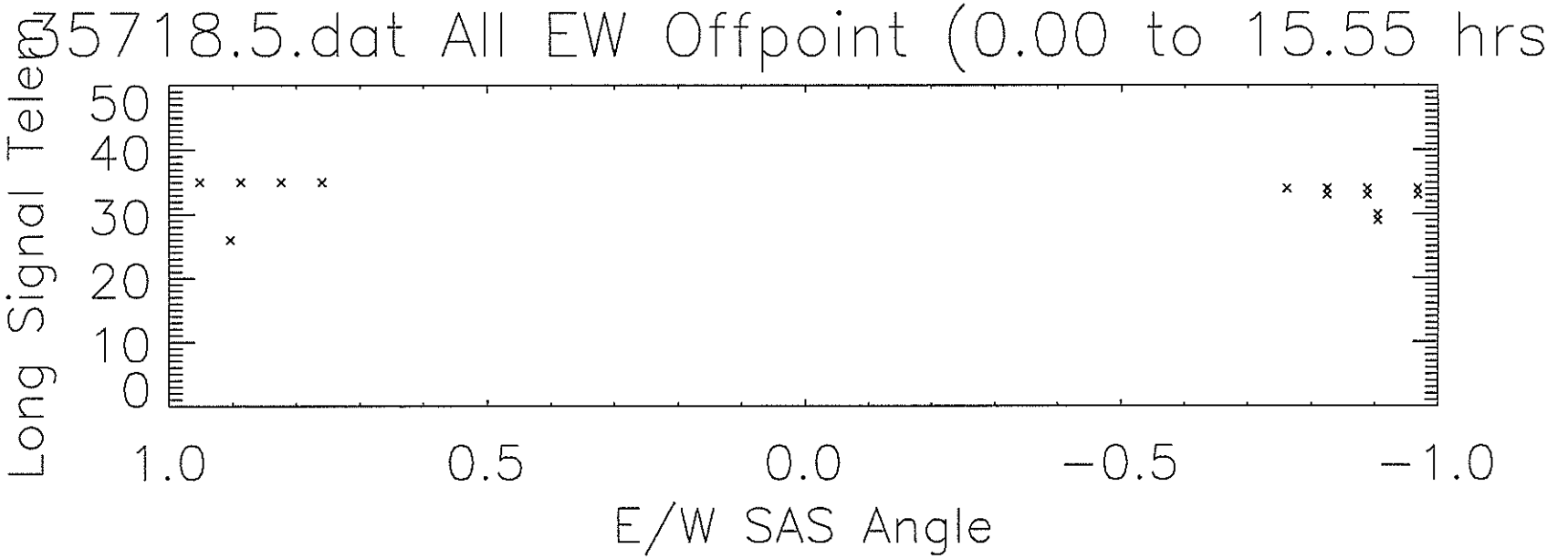

35718.5.dat NS Offpoint
Set 6 of 7 (15.99 to 16.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

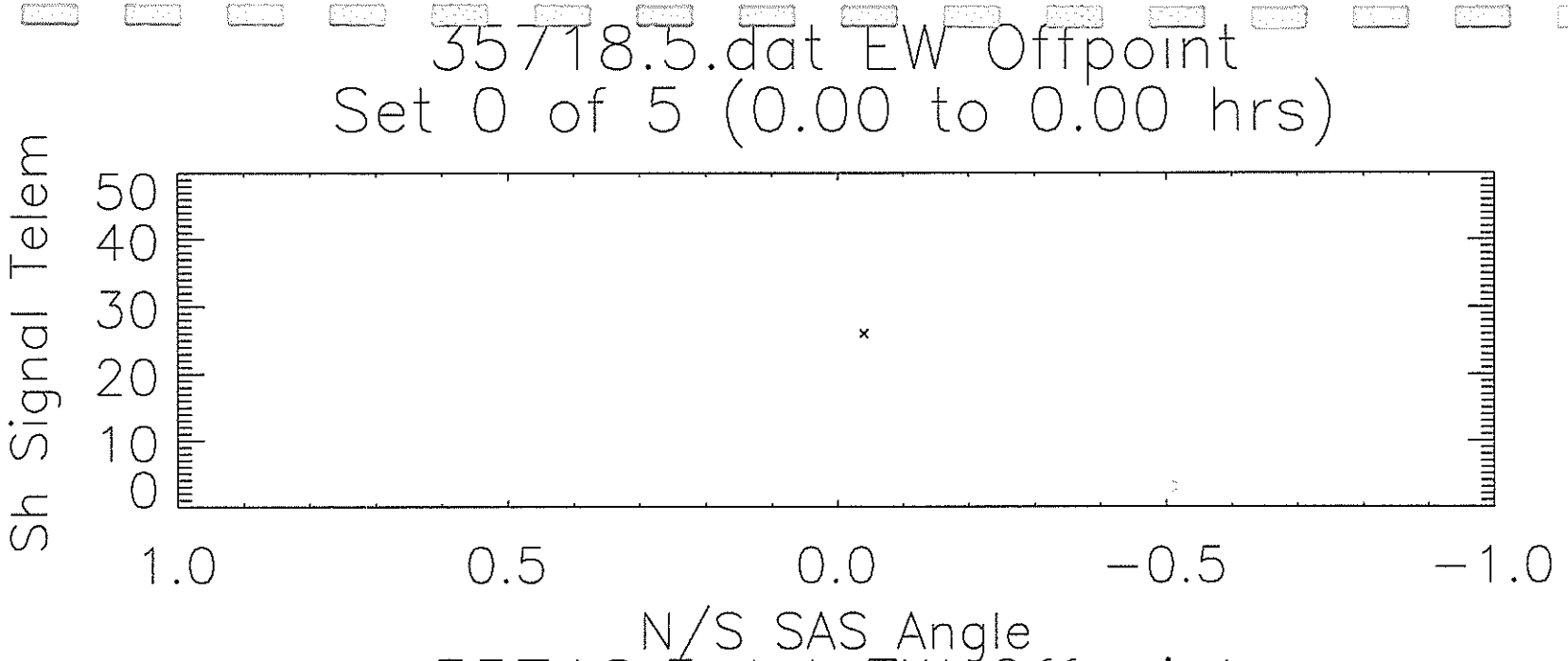

35718.5.dat NS Offpoint
Set 6 of 7 (15.99 to 16.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

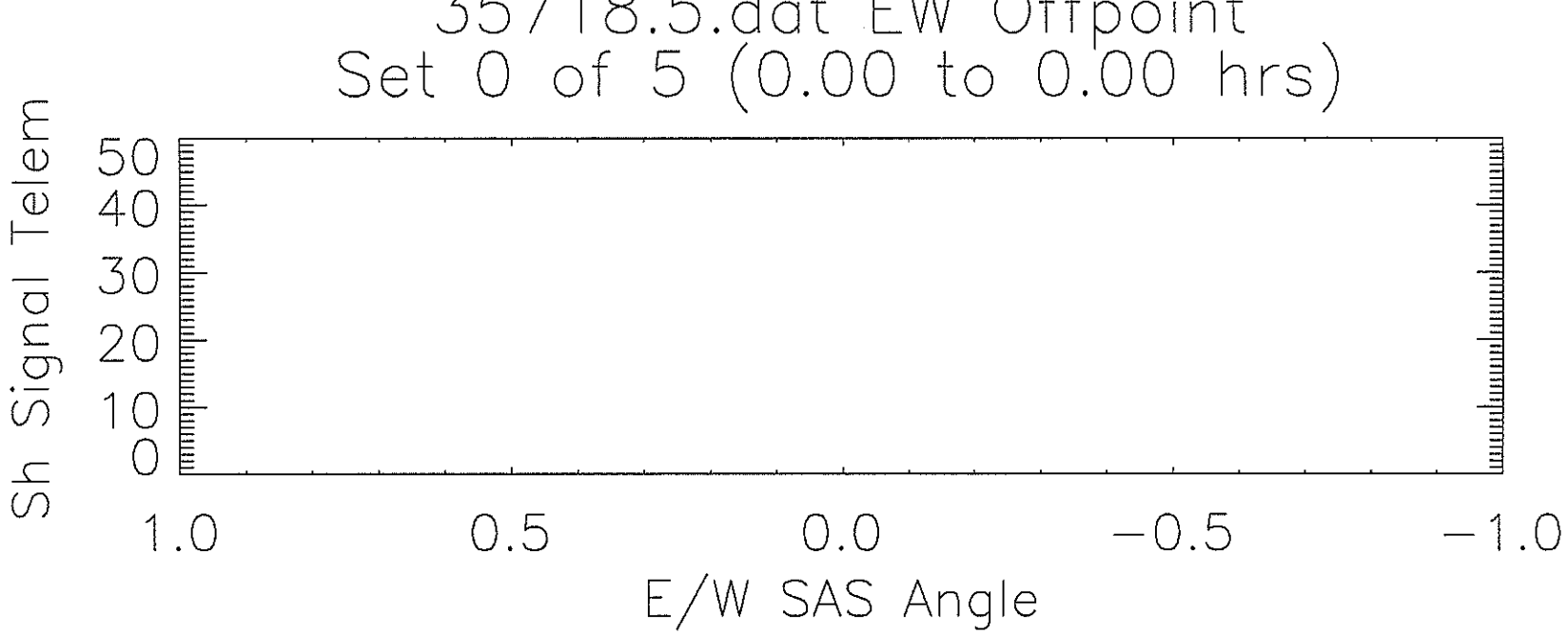

35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

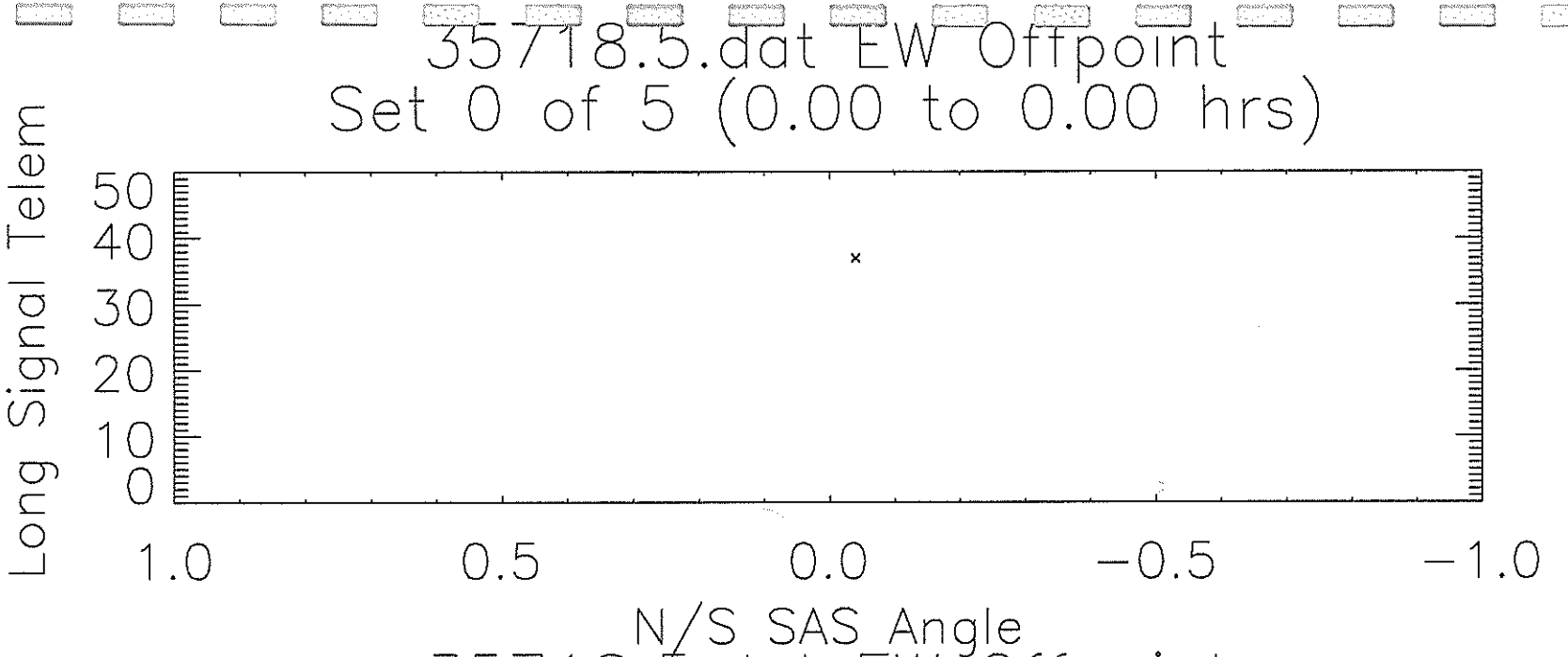

35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

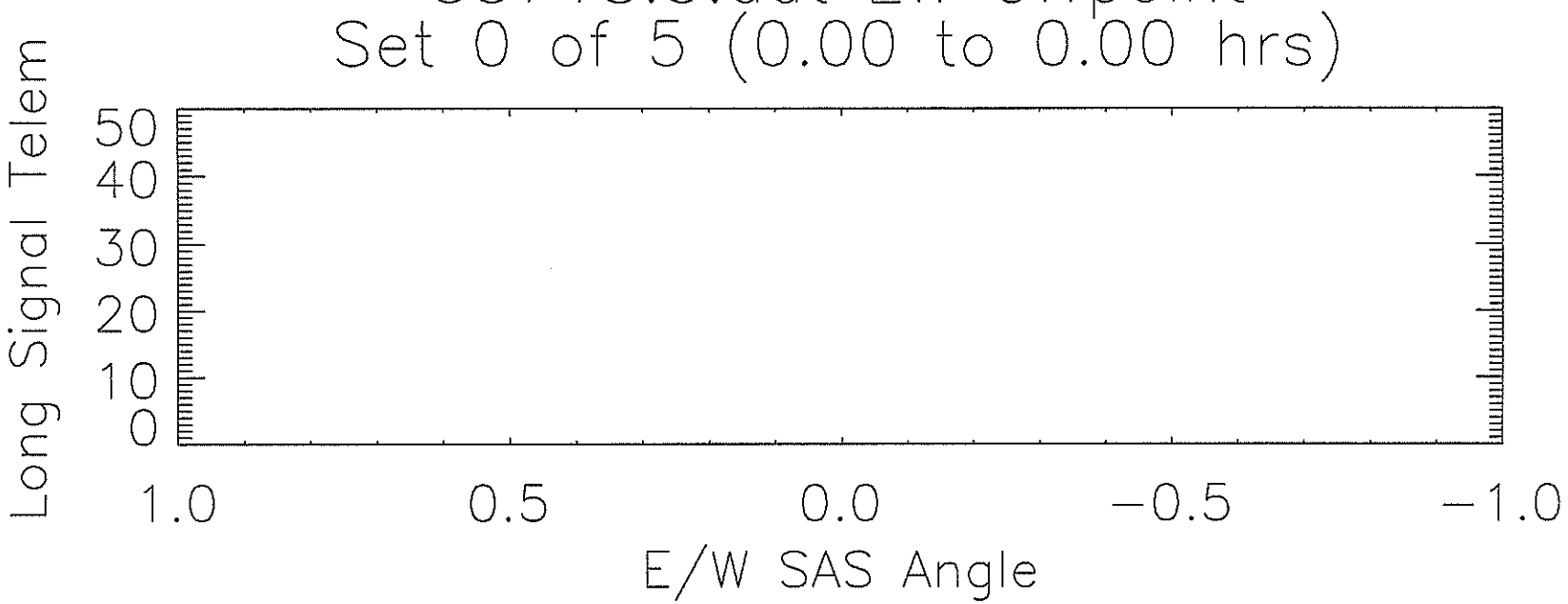

35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

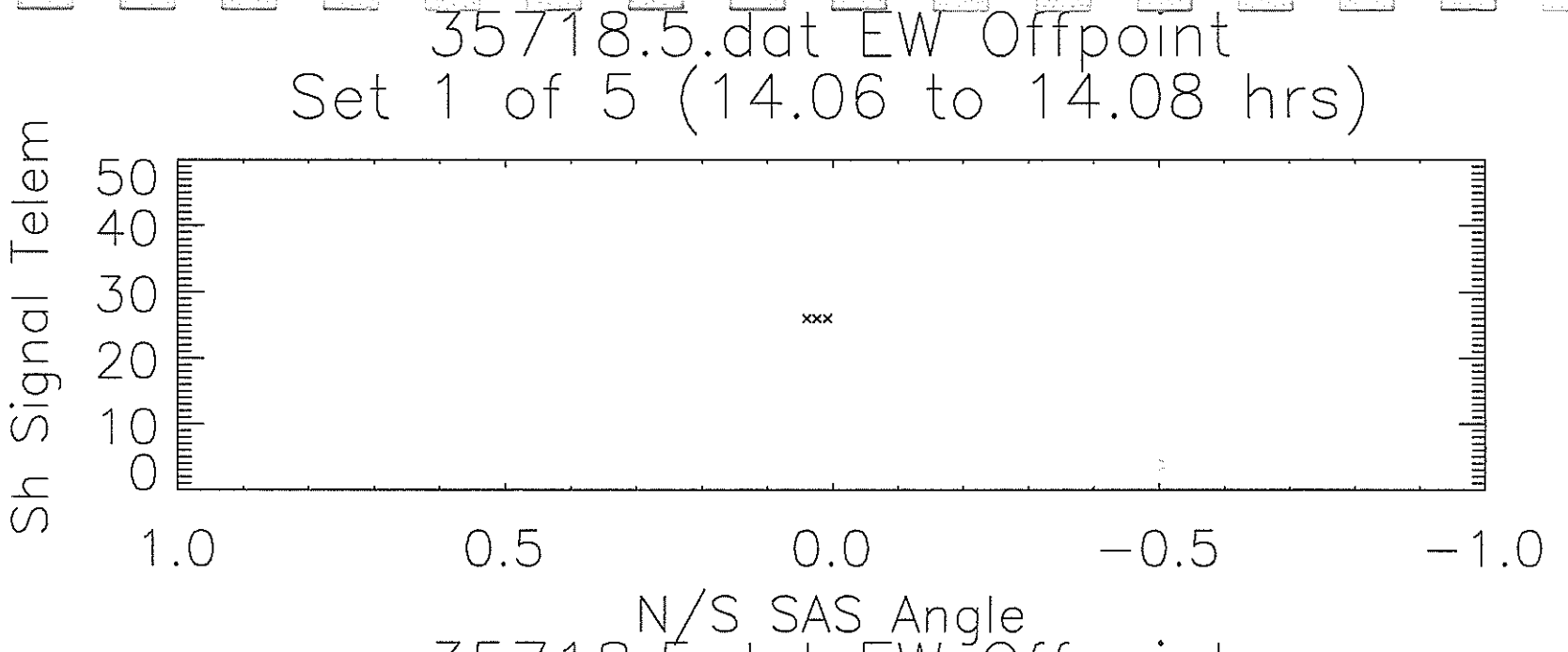

35718.5.dat All EW Offpoint (0.00 to 15.55 hrs
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

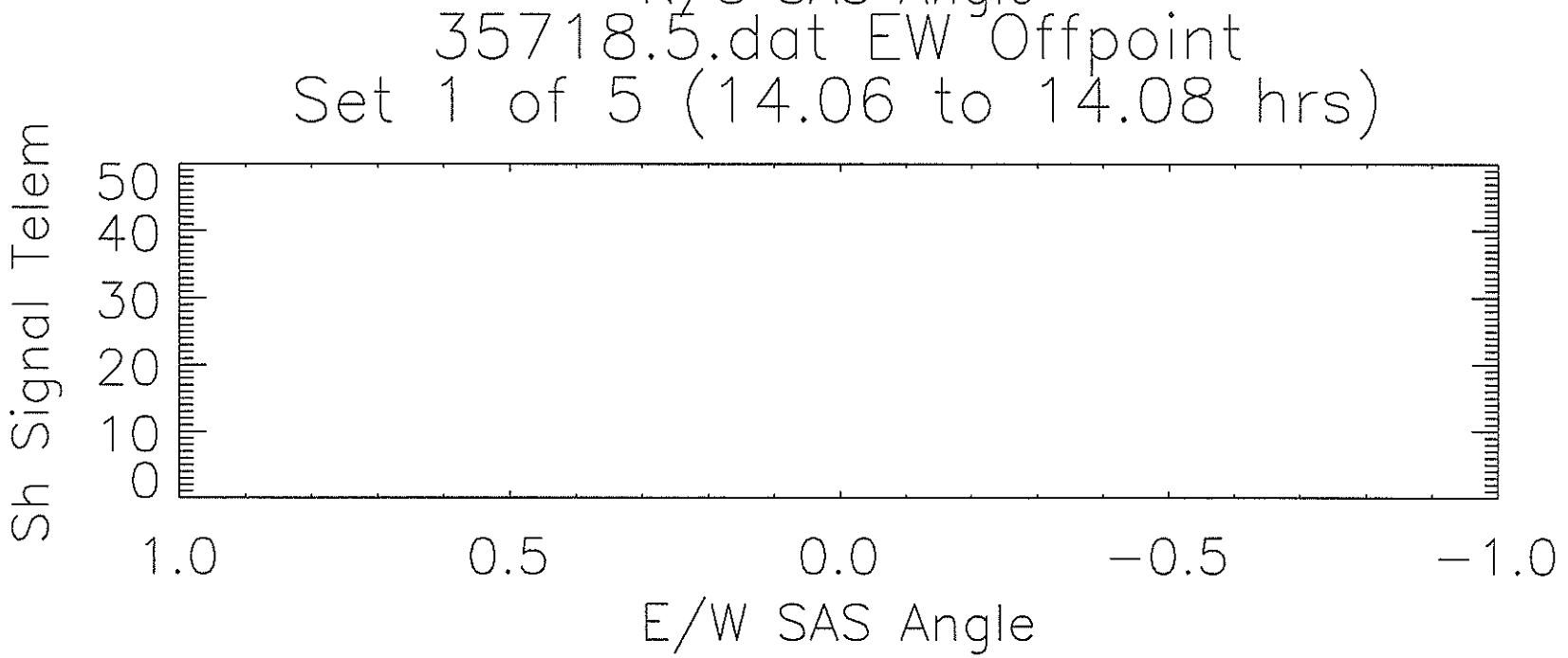

35718.5.dat EW Offpoint
Set 0 of 5 (0.00 to 0.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

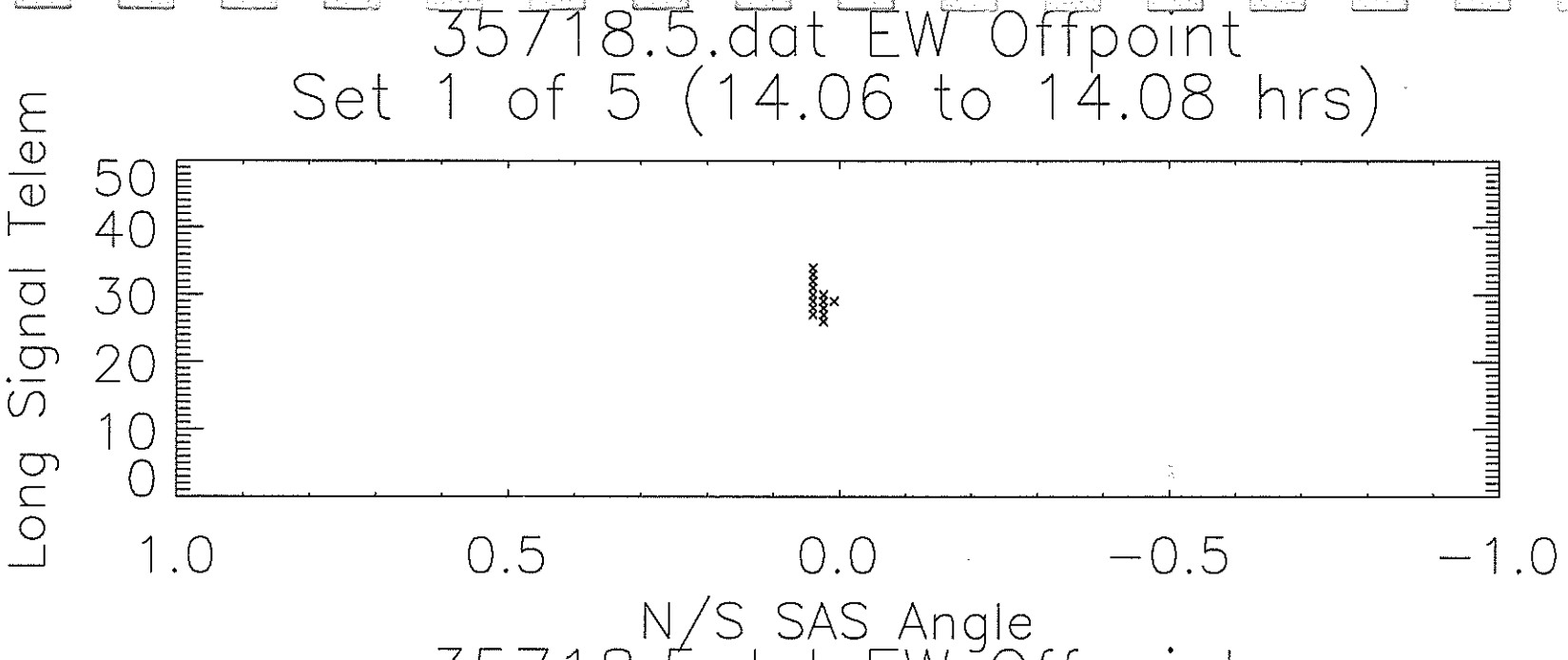

35718.5.dat EW Offpoint
Set 0 of 5 (0.00 to 0.00 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

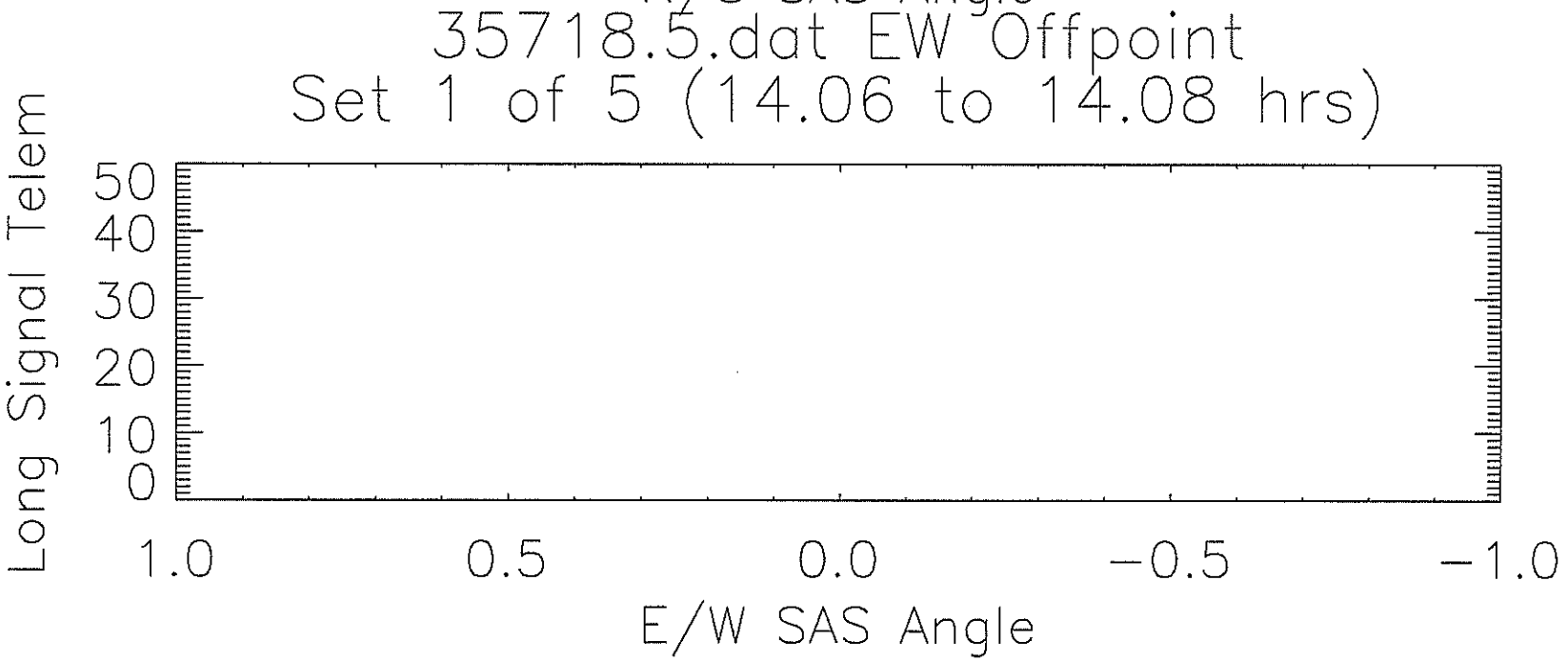

35718.5.dat EW Offpoint
Set 0 of 5 (0.00 to 0.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

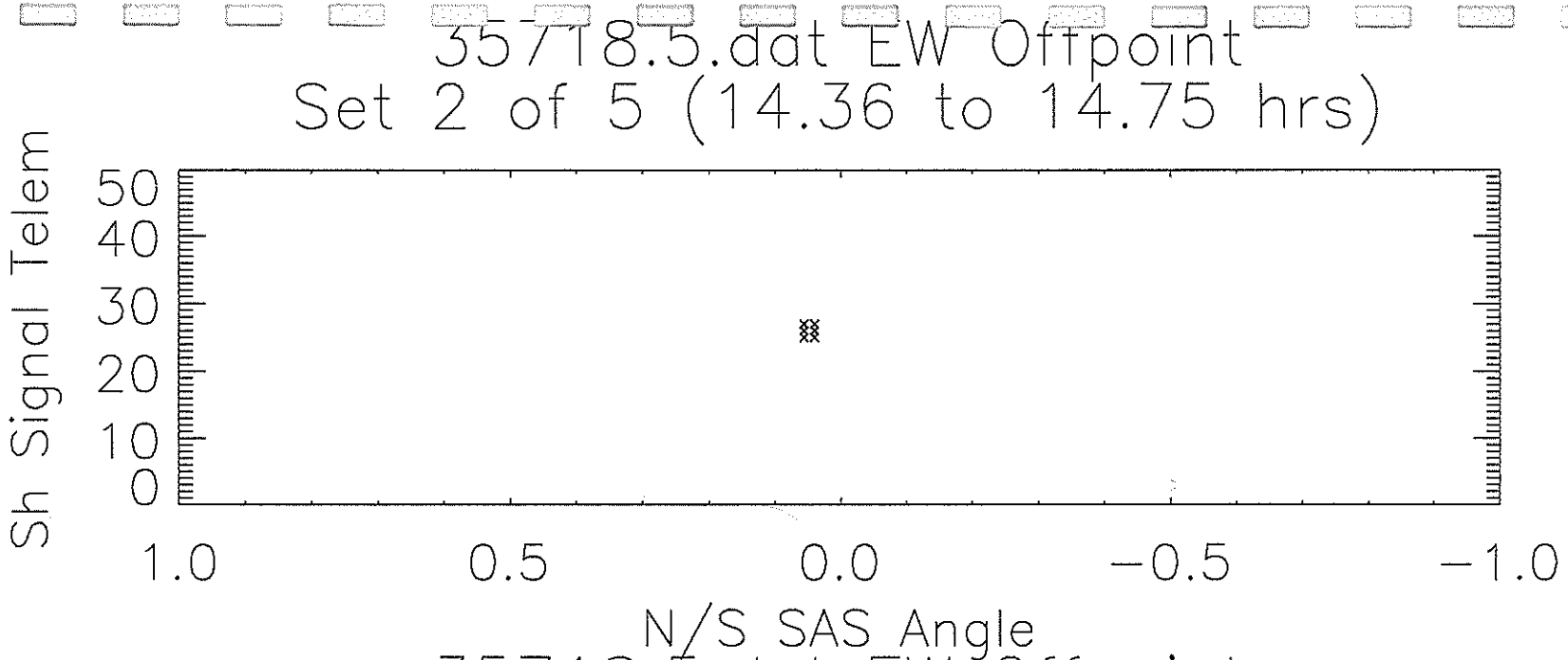

35718.5.dat EW Offpoint
Set 0 of 5 (0.00 to 0.00 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

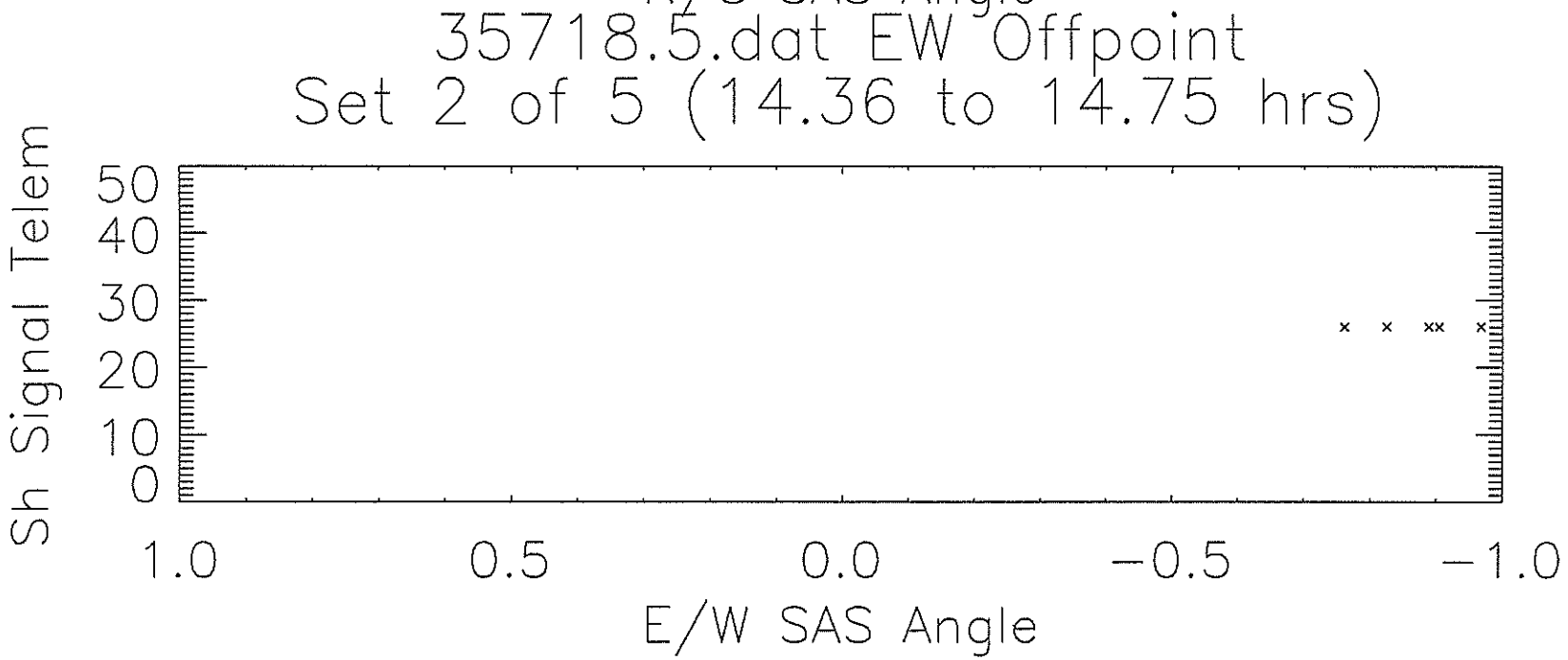

35718.5.dat EW Offpoint
Set 1 of 5 (14.06 to 14.08 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

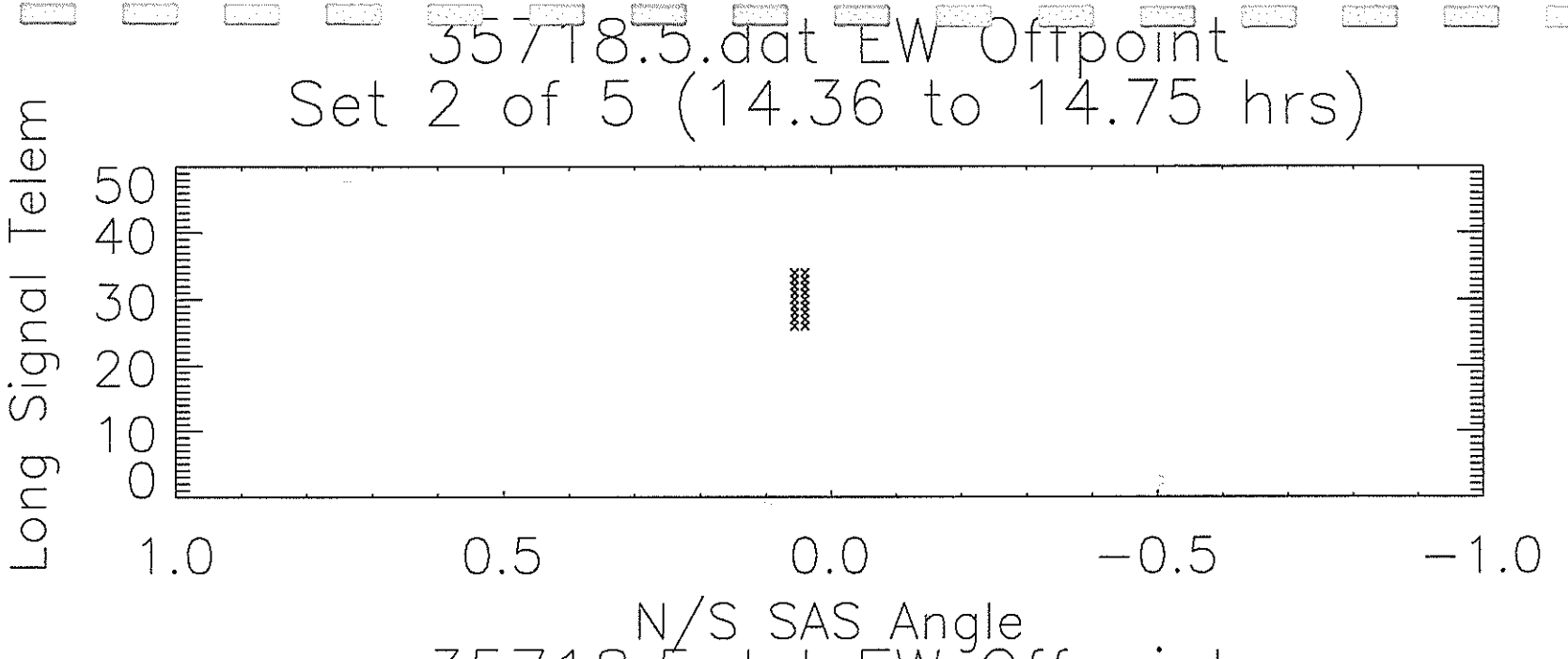

35718.5.dat EW Offpoint
Set 1 of 5 (14.06 to 14.08 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

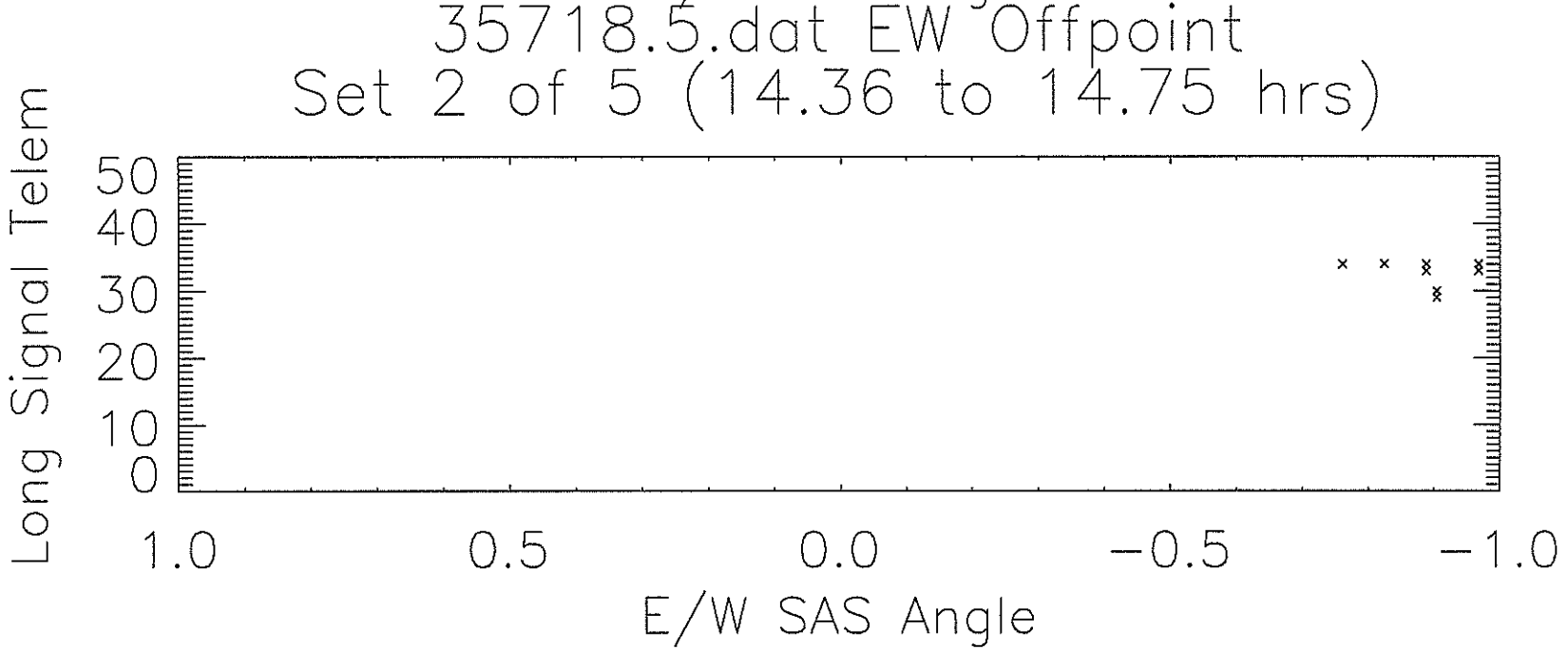

35718.5.dat EW Offpoint
Set 1 of 5 (14.06 to 14.08 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

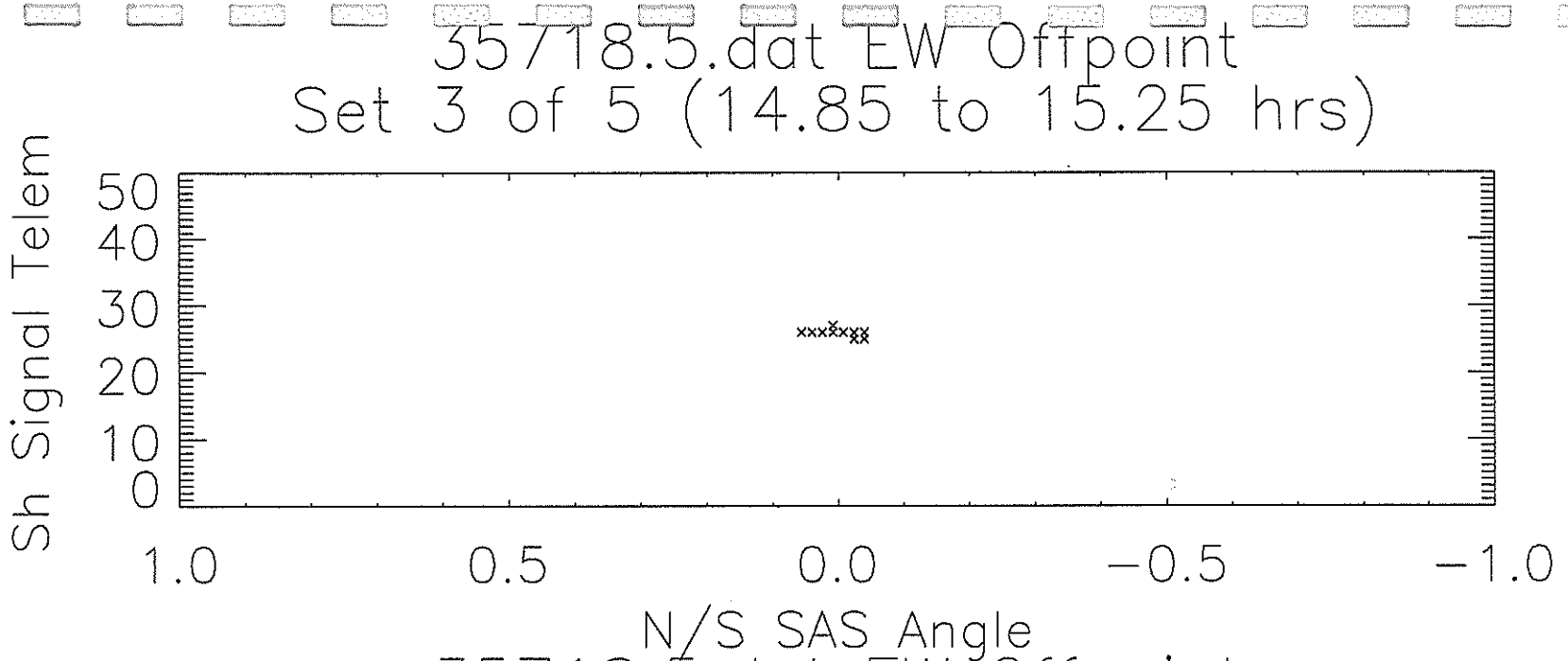

35718.5.dat EW Offpoint
Set 1 of 5 (14.06 to 14.08 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

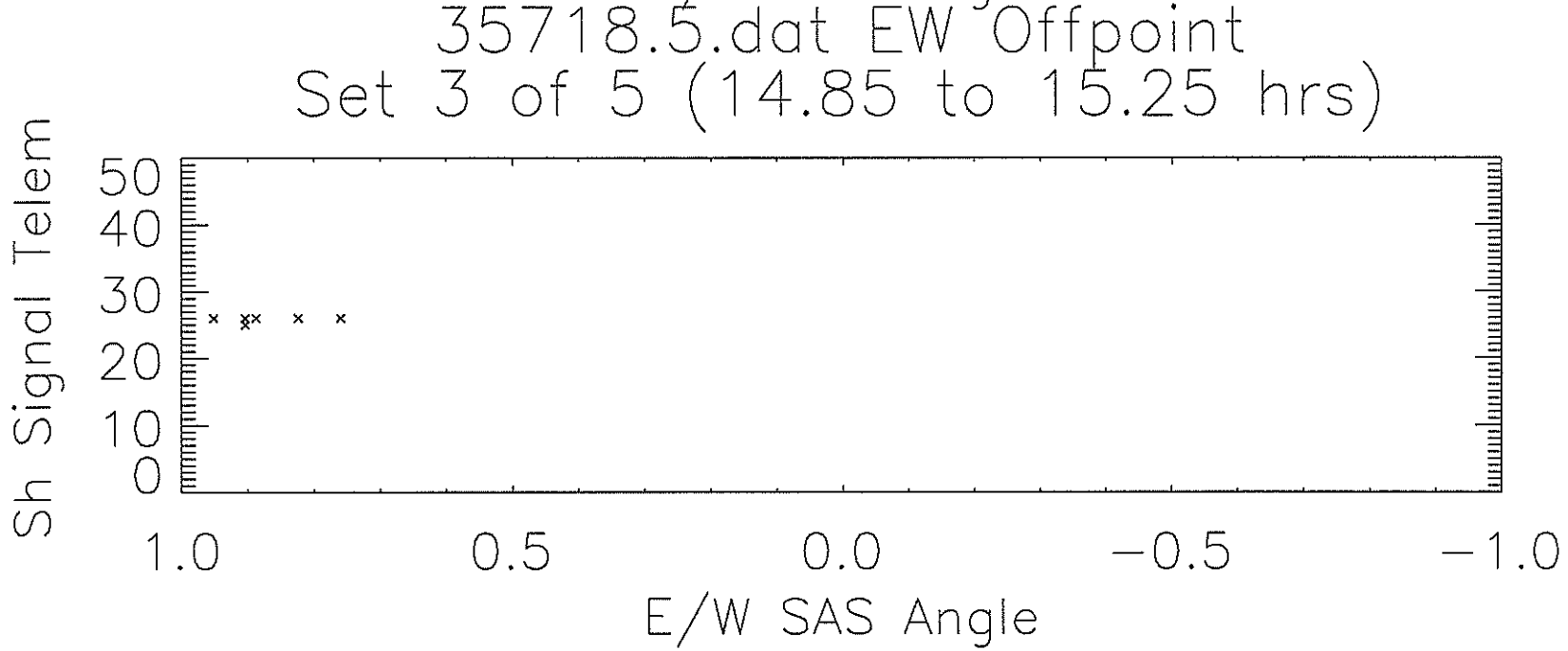

35718.5.dat EW Offpoint
Set 2 of 5 (14.36 to 14.75 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

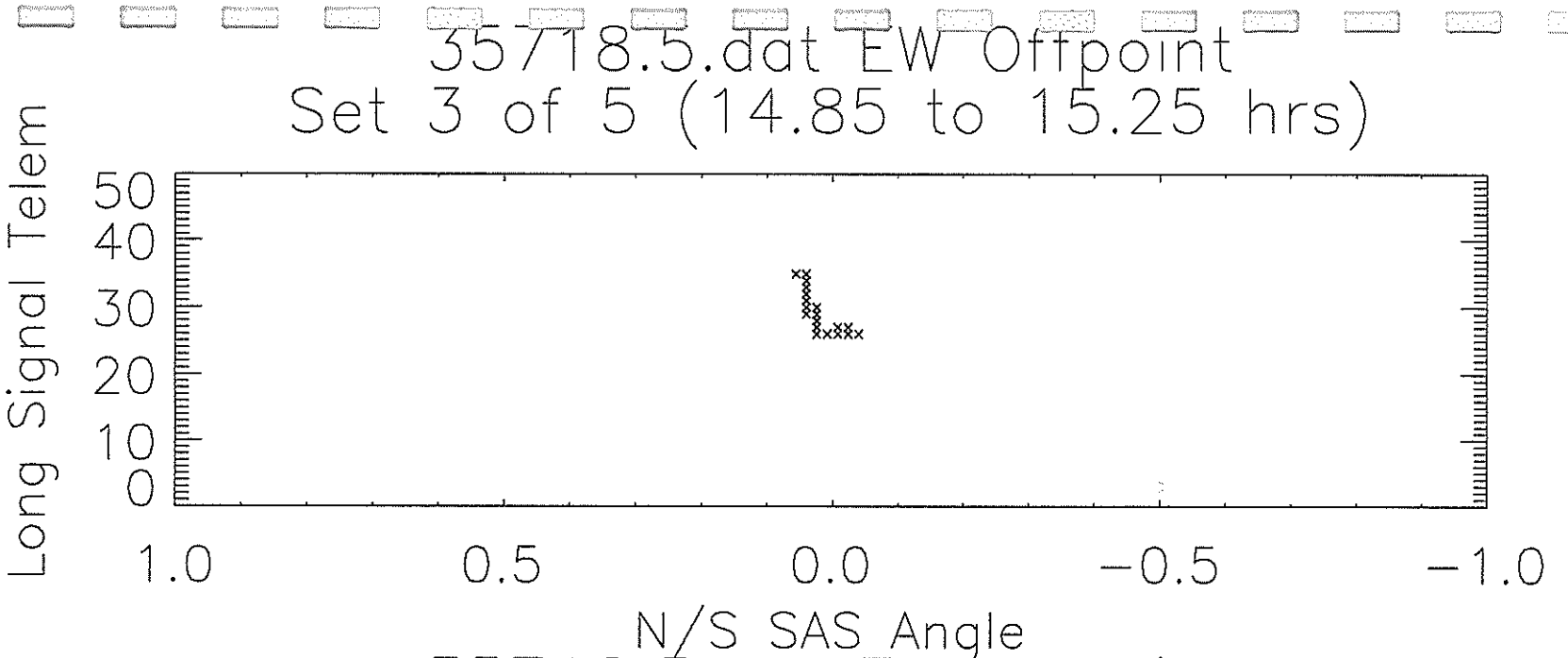

35718.5.dat EW Offpoint
Set 2 of 5 (14.36 to 14.75 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

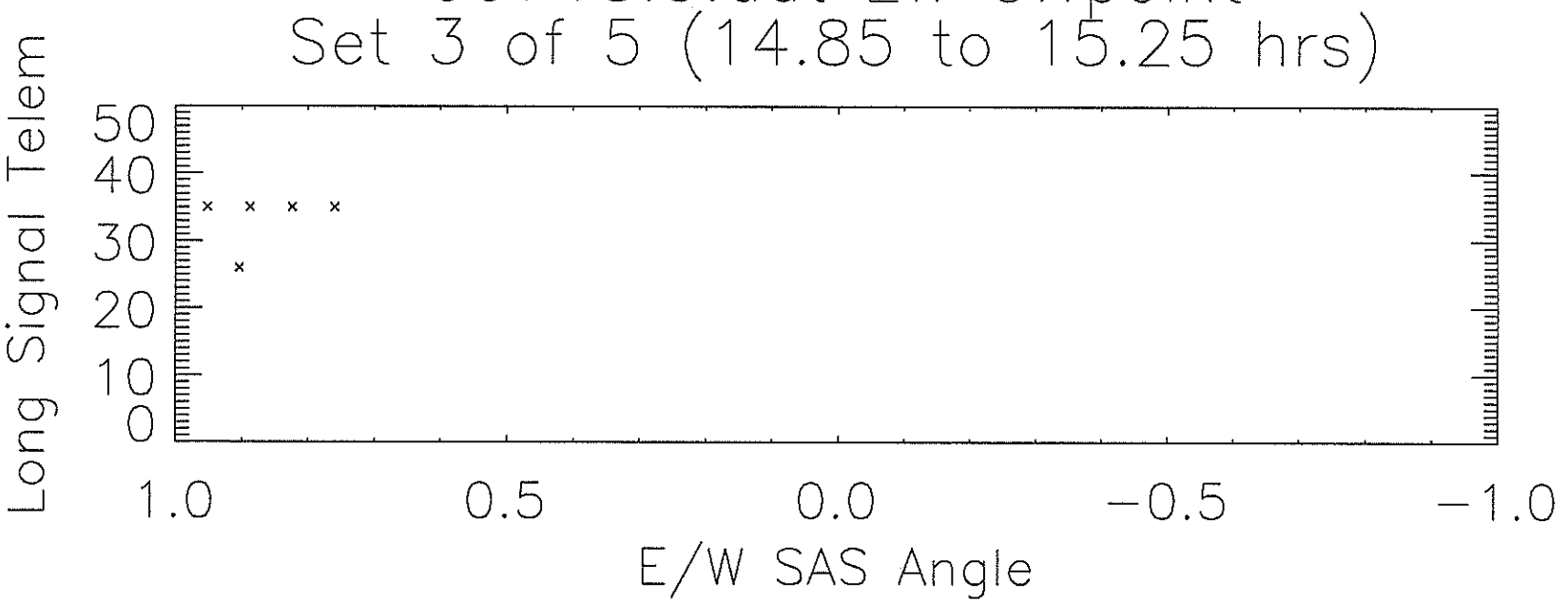

35718.5.dat EW Offpoint
Set 2 of 5 (14.36 to 14.75 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

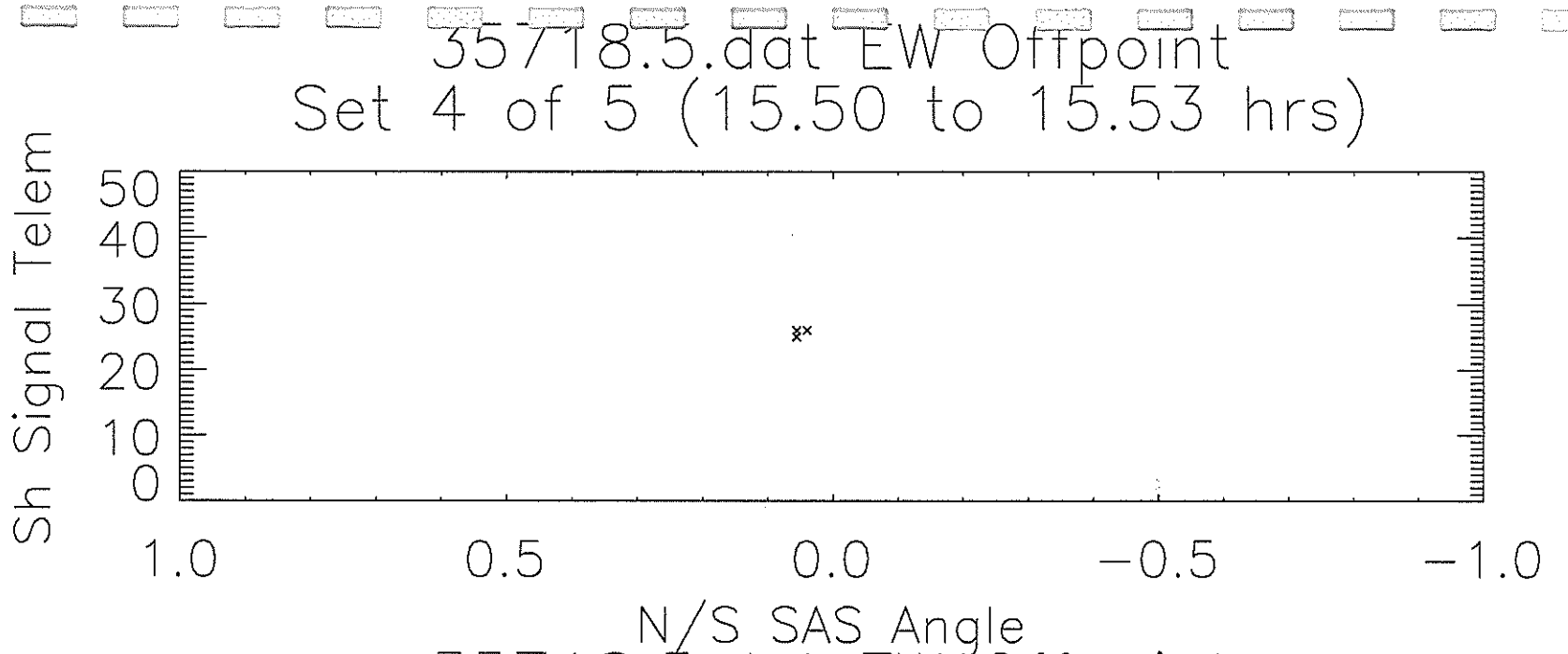

35718.5.dat EW Offpoint
Set 2 of 5 (14.36 to 14.75 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

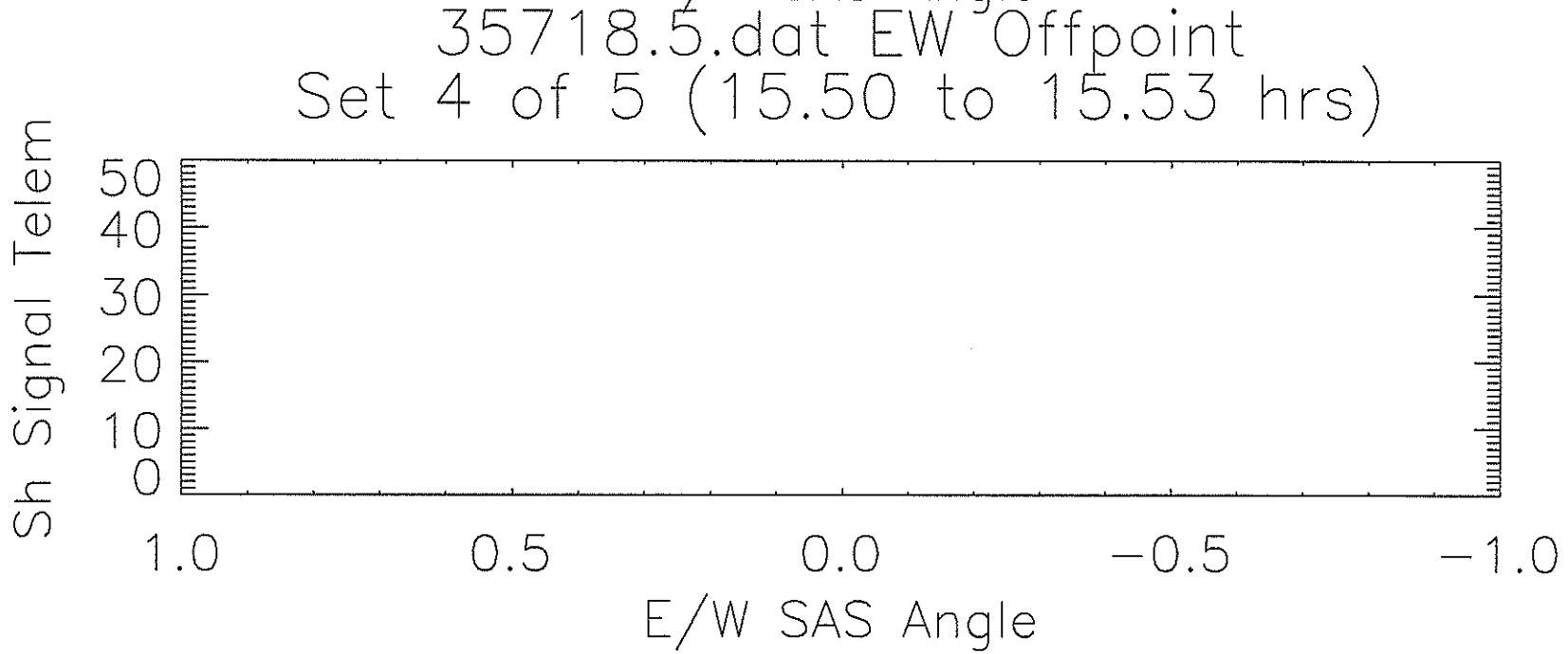

35718.5.dat EW Offpoint
Set 3 of 5 (14.85 to 15.25 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle

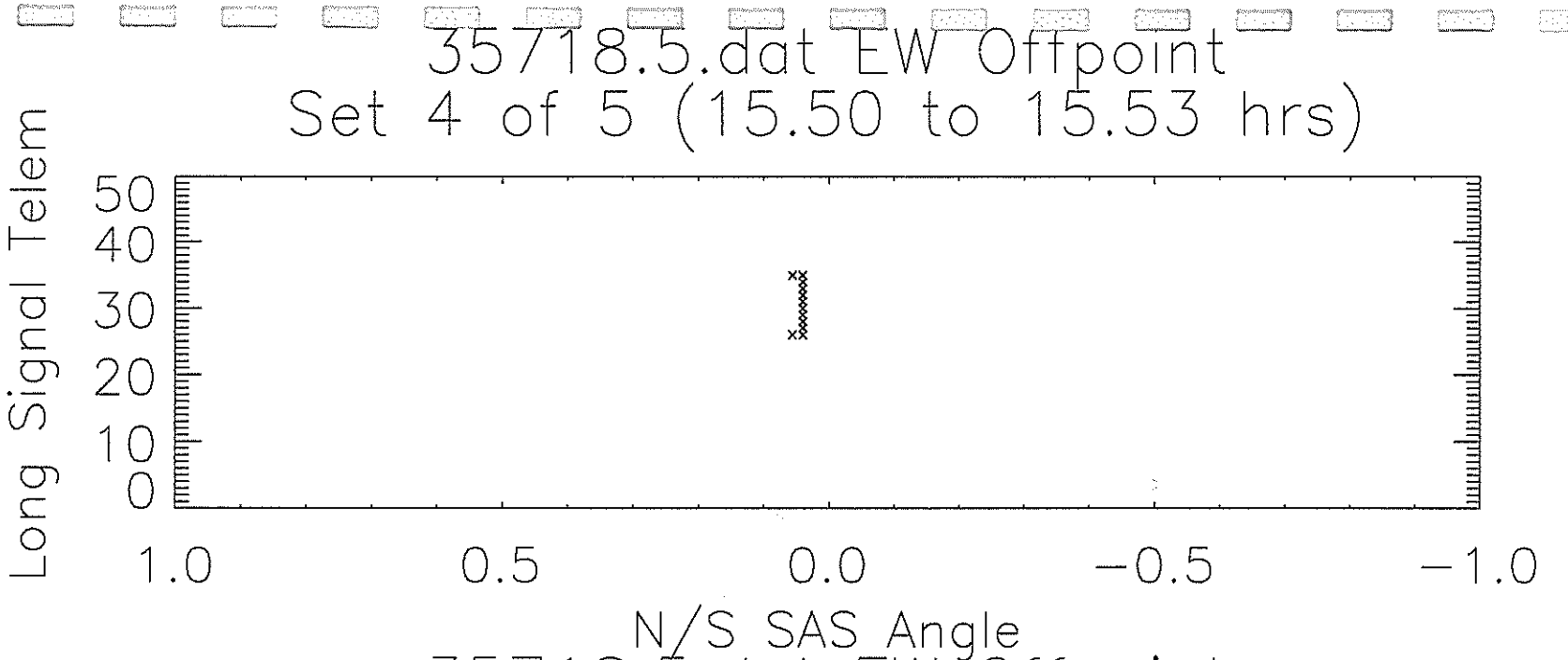

35718.5.dat EW Offpoint
Set 3 of 5 (14.85 to 15.25 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

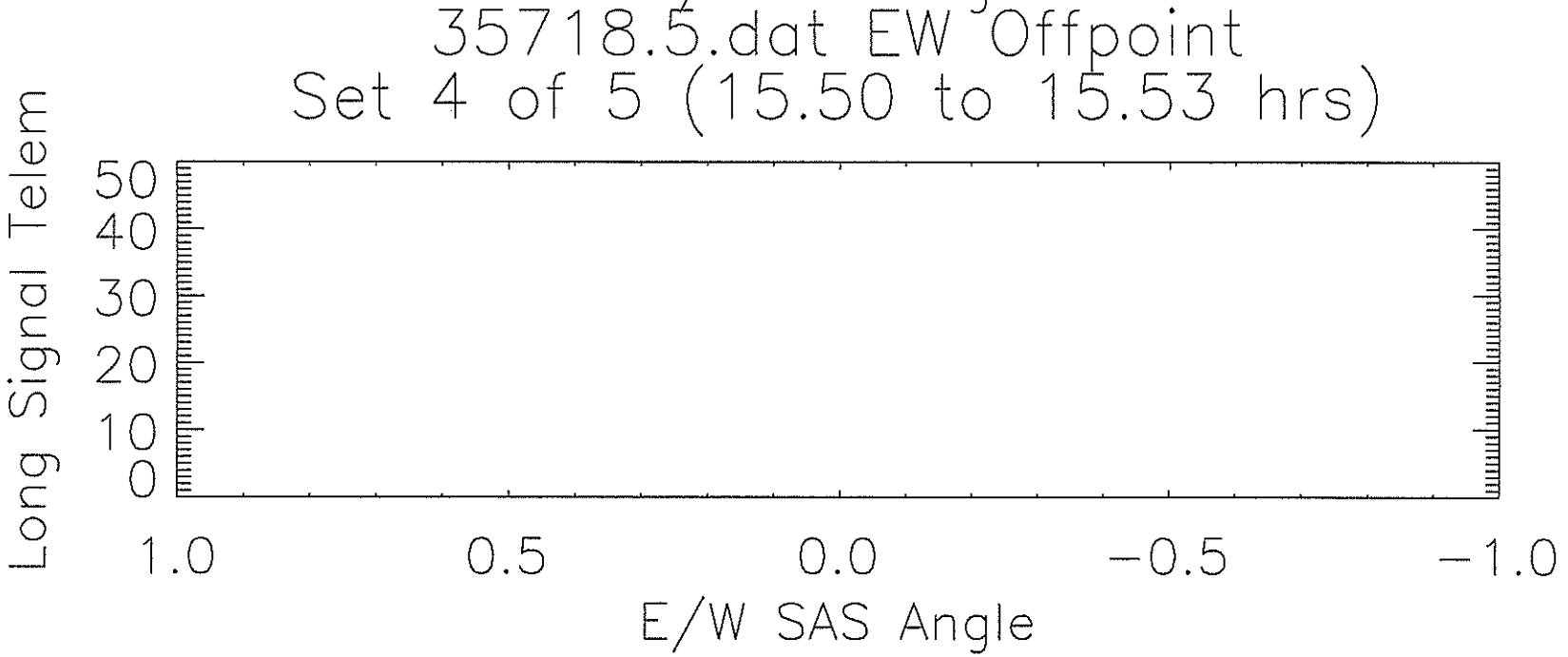

35718.5.dat EW Offpoint
Set 3 of 5 (14.85 to 15.25 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat EW Offpoint
Set 3 of 5 (14.85 to 15.25 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle


35718.5.dat EW Offpoint
Set 4 of 5 (15.50 to 15.53 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat EW Offpoint
Set 4 of 5 (15.50 to 15.53 hrs)
Sh Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle


35718.5.dat EW Offpoint
Set 4 of 5 (15.50 to 15.53 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
N/S SAS Angle


35718.5.dat EW Offpoint
Set 4 of 5 (15.50 to 15.53 hrs)
Long Signal Telem
50
40
30
20
10
0
1.0
0.5
0.0
−0.5
−1.0
E/W SAS Angle

35718.5.datSlew 15.70 to 16.07 hrs