STEREO/Waves CDPP data reader [IDL]

1. STEREO/Waves Data.

The format of the data stored at the CDPP (*Plasma Physics Data Center*) is present in the "Archives Données Radio STEREO" (*STEREO/Waves Radio Data Archive Guide at CDPP*) avaliable on the CDPP web page. With the IDL routines, the output of the readers do not respect that data format because of its variable packet length specification, which makes it difficult to treat long series of data. The data chosen structure is thus a series of elementary samples (called 2-antenna samples) containing basics instantaneous measurements on the 2 channels of the recevier. The high resolution data structure is defined in the IDL routine `str_wav_h_res_data_l1__define.pro` (appendix A) and is the following:

```
** Structure STR_WAV_H_RES_DATA_L1, 14 tags, length=60, data length=55:
   AMJ             LONG        : Year-Month-Day (YYYYMMDD)
   NUM             LONG        : sequential number in file (see note [a])
   SWEEP           LONG        : sequential sweep number in file (see note [b])
   SEC             DOUBLE      : second of day
   FREQ            FLOAT       : frequency of analysis (kHz)
   FI              INT         : frequency index (see note [c])
   DT              FLOAT       : effective integration time (ms)
   DF              FLOAT       : effective integration bandwidth (kHz)
   ANT             BYTE        : antenna configuration (see note [d])
   AGC1            FLOAT       : Automatic Gain Control Channel 1 (V^2/Hz)
   AGC2            FLOAT       : Automatic Gain Control Channel 2 (V^2/Hz)
   AUTO1           FLOAT       : Autocorrelation Channel 1 (V^2/Hz)
   AUTO2           FLOAT       : Autocorrelation Channel 2 (V^2/Hz)
   CROSR           FLOAT       : Normalized Cross-correlation (real part)
   CROSI           FLOAT       : Normalized Cross-correlation (imaginary part)
```

Notes:
[a] Each 2-antenna sample has a sequential number in file. It is useful in preparation for the higher level data analysis.
[b] Each 2-antenna sample has a sequential sweep number allowing to identify series of samples recorded during the same frequency sweep.
[c] The frequency index is an integer BNNN describing which frequency channel was used for the measurement. B = 1, 2, 3 or 4 (respectively band LFR-A, LFR-B, LFR-C, HFR)[1], NNN = channel number in selected band B. For bands LFR-A, LFR-B and LFR-C, NNN is within 0 and 15, for band HFR, NNN is within 0 and 318.
[d] Excerpt of the *STEREO/Waves Radio Data Archive Guide at CDPP* :

```
LFA   : 00 (Off/Off), 10 (-Ex/Off),    20 (+Ey/Off),    30 (Ey-Ex/Off).
LFB/C : 00 (Off/Off), 10 (-Ex/Off),    20 (+Ey/Off),    30 (Ey-Ex/Off),
      : 01 (Off/-Ez), 02 (Off/-Ey),    03 (Off/Ey-Ez),
      : 11 (-Ex/-Ez), 12 (-Ex/+Ey),    13 (-Ex/Ey-Ez),  21 (+Ey/-Ez),       31 (Ey-Ex/-Ez),
      :-22 (+Ey/+Ey),-32 (Ey-Ex/+Ey), -23 (+Ey/Ey-Ez), -33 (Ey-Ex/Ey-Ez).
HF1/2 : 00 (Off/Off), 10 (+Ex/Off),    20 (-Ey/Off),    30 (Ex-Ey/Off),
      : 01 (Off/+Ez), 02 (Off/+Ey),    03 (Off/Ez-Ey),
      : 11 (+Ex/+Ez), 12 (+Ex/-Ey),    13 (+Ex/Ez-Ey),  21 (-Ey/+Ez),       31 (Ex-Ey/+Ez).
      :-22 (-Ey/-Ey),-32 (Ex-Ey/-Ey), -23 (-Ey/Ez-Ey), -33 (Ex-Ey/Ez-Ey).
Illegal code (error in data) : 99
```

The antenna code provides the antennas connected to each of the receiver channels.

[1] In older versions of the reader, B could be equal to 5 (for HFR-2), 4 representing HFR-1 only. The current definition and use of B simplifies when HFR-1 and HFR-2 are sweeping over the same frequency range.

Averaged data on 60s intervals are also available. The average data structure is defined in the IDL routine `str_wav_average_data_l1__define.pro` (see appendix A) and is the following :

```
** Structure STR_WAV_AVERAGE_DATA_L1, 6 tags, length=28, data length=28:
   AMJ             LONG        : Year-Month-Day (YYYYMMDD)
   NUM             LONG        : sequential number in file (see note [a])
   SWEEP           LONG        : sequential sweep number in file (see note [b])
   SEC             DOUBLE      : second of day
   FREQ            FLOAT       : frequency of analysis (kHz)
   FLUX            FLOAT       : flux (SFU)
```

Notes :
[a] Each 2-antenna sample has a sequential number in file. It is useful in preparation for the higher level data analysis (this sequential number is different than the one in high resolution data).
[b] Each 2-antenna sample has a sequential sweep number allowing to identify series of samples recorded during the same frequency sweep (this sequential sweep number is different than the one in high resolution data).

2. Routines and Calling Sequence.

A set of six routines is necessary to read the CDPP data files. These routines are given in the appendix A.

1. Calling sequence for high resolution data:

```
IDL> file='/Volumes/data/CDPP_data/STR_WAV/PRE/H_RES/STB_WAV_LFR_20070124.B3E'
IDL> READ_STR_WAV_H_RES_DATA,file,data
% Compiled module: READ_STR_WAV_H_RES_DATA.
% Compiled module: STR_WAV_H_RES_HEADER__DEFINE.
% Compiled module: STR_WAV_H_RES_DATA_L1__DEFINE.
% READ_STR_WAV_H_RES_DATA:         6063 records read in        1.3386581 seconds.
% READ_STR_WAV_H_RES_DATA:        97008 samples stored.
IDL> help, data
DATA            STRUCT    = -> STR_WAV_H_RES_DATA_L1 Array[97008]
IDL>
```

If you want the auto-correlation data to be in dB, you can either use the former calling sequence and then convert the `data.auto1` and `data.auto2` into dB afterwards, or use the `/autodb` keyword as follows:

```
IDL> file='/Volumes/data/CDPP_data/STR_WAV/PRE/H_RES/STB_WAV_LFR_20070124.B3E'
IDL> READ_STR_WAV_H_RES_DATA,file,data,/autodb
% Compiled module: READ_STR_WAV_H_RES_DATA.
% Compiled module: STR_WAV_H_RES_HEADER__DEFINE.
% Compiled module: STR_WAV_H_RES_DATA_L1__DEFINE.
% READ_STR_WAV_H_RES_DATA:         6063 records read in        1.3386581 seconds.
% READ_STR_WAV_H_RES_DATA:        97008 samples stored.
IDL> help, data
DATA            STRUCT    = -> STR_WAV_H_RES_DATA_L1 Array[97008]
IDL>
```

2. Calling sequence for averaged data:

```
IDL> file='/Volumes/data/CDPP_data/STR_WAV/PRE/AVERAGE/STB_WAV_LFR_60s_20070124.B3E'
IDL> READ_STR_WAV_AVERAGE_DATA,file,data
% Compiled module: READ_STR_WAV_AVERAGE_DATA.
% Compiled module: STR_WAV_AVERAGE_HEADER__DEFINE.
% Compiled module: STR_WAV_AVERAGE_DATA_L1__DEFINE.
% READ_STR_WAV_AVERAGE_DATA:         1440 records read in       0.15589499 seconds.
% READ_STR_WAV_AVERAGE_DATA:        69120 samples stored.
IDL> help, data
DATA            STRUCT    = -> STR_WAV_AVERAGE_DATA_L1 Array[69120]
IDL>
```

3. Using the data:

Once the data is loaded, here are some examples given to get you familiar with the syntax to be used with the high resolution data. The data should be loaded as follows:

```
IDL> file='/Volumes/data/CDPP_data/STR_WAV/PRE/H_RES/STB_WAV_LFR_20070124.B3E'
IDL> READ_STR_WAV_H_RES_DATA,file,data
```

1) plotting the autocorrelation on channel 1 with respect to time (in hour) for all frequencies, with a log scale on abscissa (see output in figure 1).

```
IDL> plot_io,data.sec/3600.d0,data.auto1,psym=3,xr=[0,24],xs=1,xtit='Hour of day', $
IDL> ytit='Auto Channel 1 (V2/Hz)',tit='STEREO-B, 2007-01-24'
```

2) plotting the autocorrelation on channel 1 with respect to time (in hour) for frequency index 3012 (band C, channel #12 [118.146 kHz]), with a log scale on abscissa (see output in figure 2).

```
IDL> ww = where(data.fi eq 3012)
IDL> plot_io,data(ww).sec/3600.d0,data(ww).auto1,psym=3,xr=[0,24],xs=1, $
IDL> xtit='Hour of day',ytit='Auto Channel 1 (V2/Hz)', $
IDL> tit='STEREO-B, 2007-01-24, F=118.146 kHz'
```

3) plotting the autocorrelation on channel 1 spectrum for sweep number 1000 (see output in figure 3).

```
IDL> ww = where(data.sweep eq 1000)
IDL> plot_oo,data(ww).freq,data(ww).auto1,psym=3,xs=3, $
IDL> xtit='Frequency (kHz)', ytit='Auto Channel 1 (V2/Hz)', $
IDL> tit='STEREO-B, 2007-01-24, sweep # 1000'
```

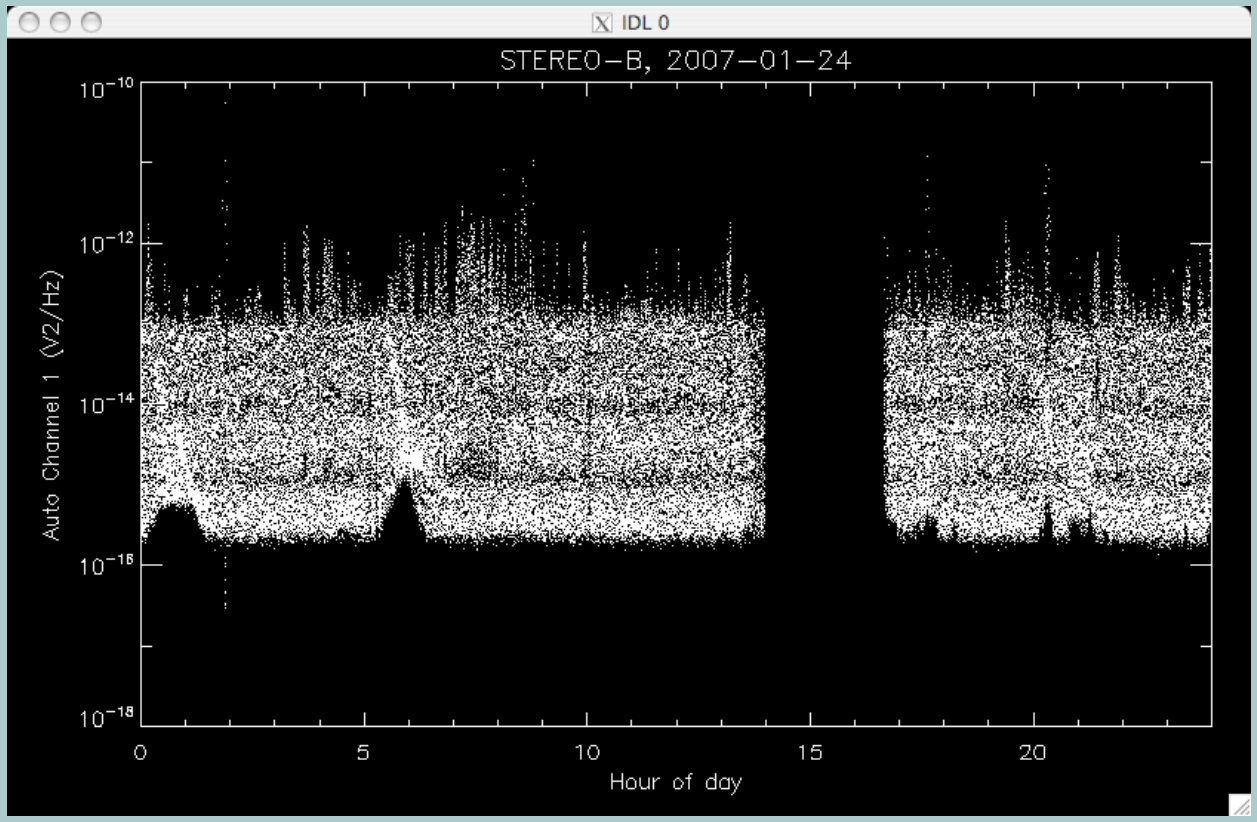

Figure 1. Graphical output for example 1.

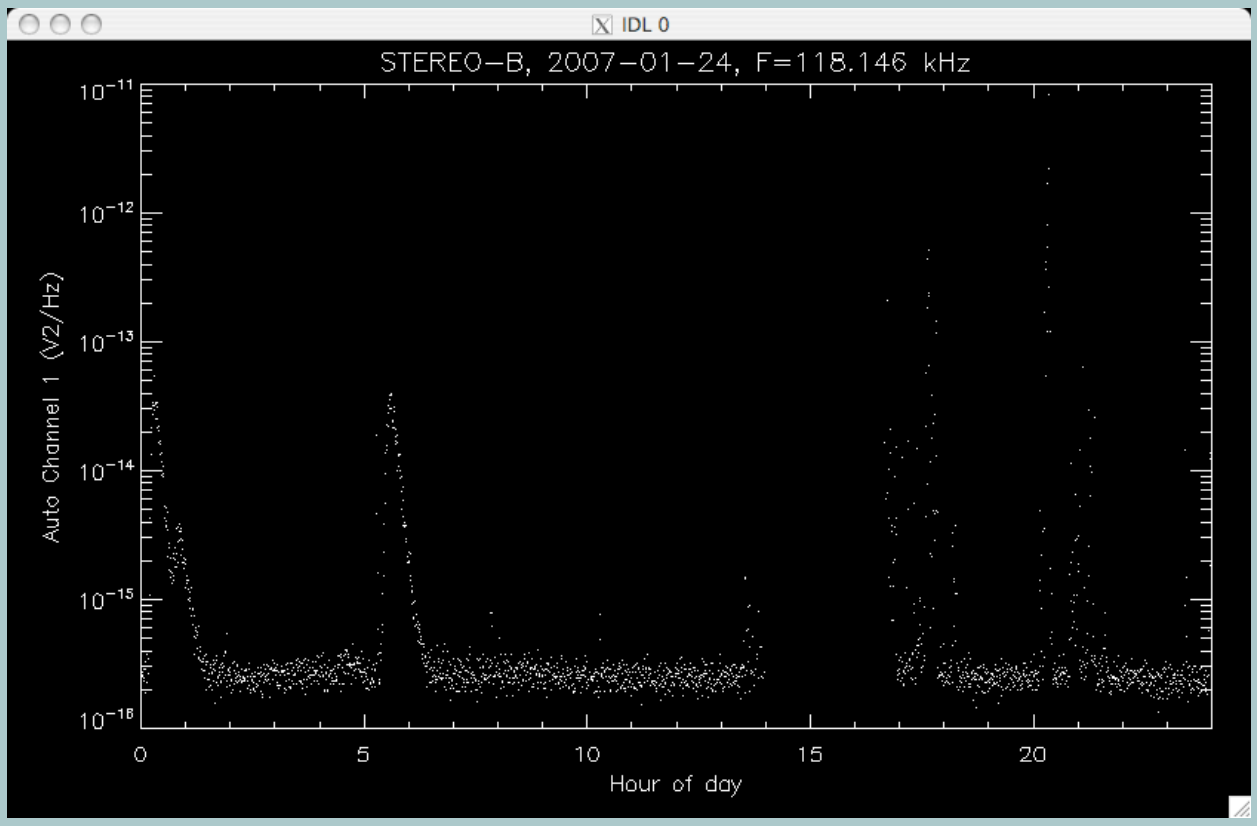

Figure 2. Graphical output for example 2.

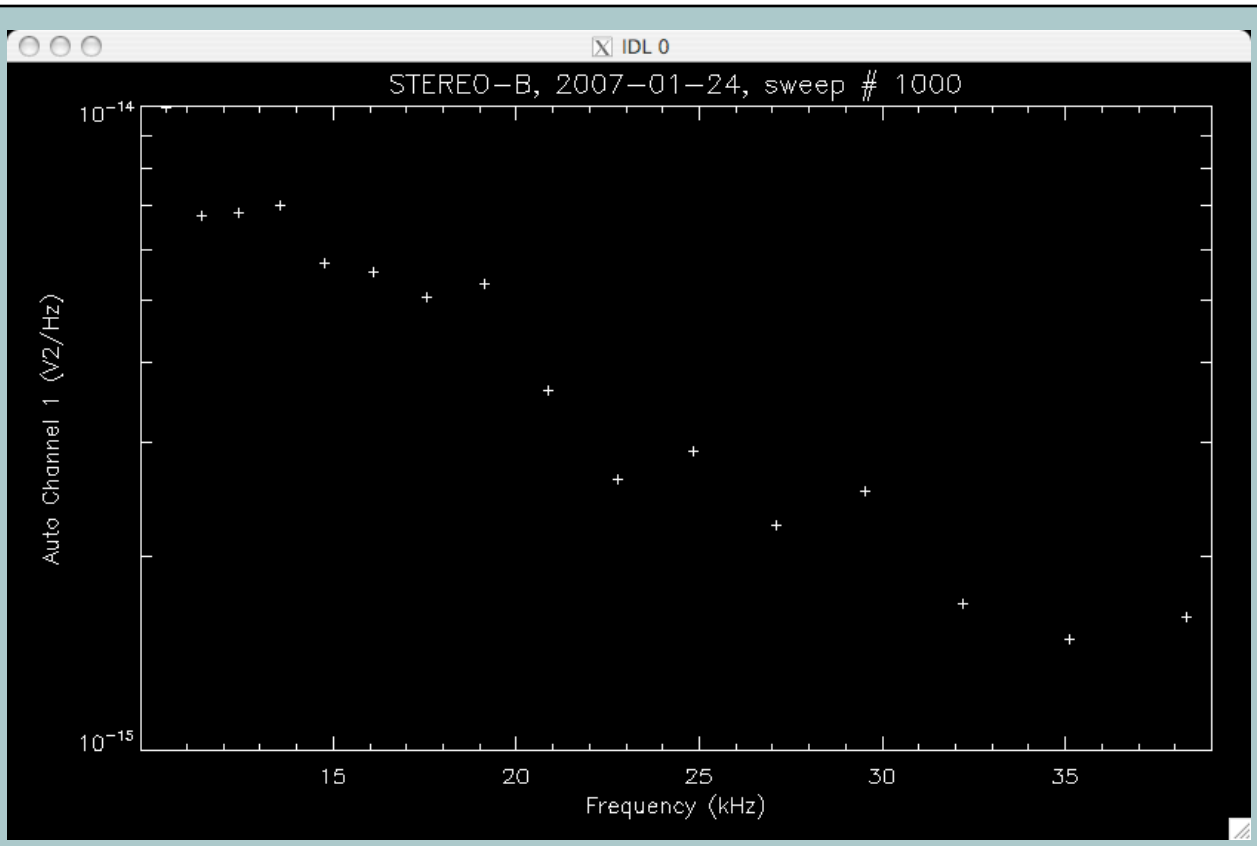

Figure 3. Graphical output for example 3.

Appendix

3. Sources of IDL Routines.

1. STR_WAV_H_RES_HEADER__DEFINE.PRO

```
PRO STR_WAV_H_RES_HEADER__DEFINE

tmp = {STR_WAV_H_RES_HEADER, irad:0, ITCDS:[0l,0l], jusecy:0l, Jamjcy:[0,0,0], $
 Jhmscy:[0,0,0], Sfract:0., Msti:0., Npalcy:0, Nfrpal:0, Nfreq:0, Nvoie:0, $
 Iant12:[0,0,0], Nconfig:0 , Ncag2:0, Nauto2:0, LoopA:0, LoopC:0}

END
```

2. STR_WAV_H_RES_DATA_L1__DEFINE.PRO

```
PRO STR_WAV_H_RES_DATA_L1__DEFINE

tmp = {STR_WAV_H_RES_DATA_L1, amj:0l, num:0l, sec:0.d0, freq:0., $
 fi:0, sweep:0l, dt:0., df:0., ant: 0b, $
 agc1:0., agc2:0., auto1:0., auto2:0., crosR:0., crosI:0.}

END
```

3. READ_STR_WAV_H_RES_DATA.PRO

```
; file = '../STR_WAV/PRE/H_RES/STA_WAV_LFR_20061121.B3E'

PRO READ_STR_WAV_H_RES_DATA, file, data_l1, autodb=autodb

; READING FILE IN 2 PASS
; 1st pass = scanning for number of records and total number of samples.
; 2nd pass = loading data into data_l1

time0 = systime(/sec)

; 1st PASS

nrecords = 0l
nsamples = 0l
record_size0 = 0l
record_size1 = 0l
record_size2 = 0l
header  = {STR_WAV_H_RES_HEADER}

openr,lun,file,/get_lun,/swap_if_little_endian
while ~eof(lun) do begin
  nrecords += 1l
  readu,lun,record_size0
  ptr0 = (fstat(lun)).cur_ptr
  readu,lun,header
  nsamples += header.nfreq*header.nconfig

  point_lun,lun,ptr0+record_size0
  ptr1 = (fstat(lun)).cur_ptr
  readu,lun,record_size1
  record_size2 = ptr1-ptr0
  if record_size1 ne record_size0 or record_size1 ne record_size2 then $
    message,'WARNING Checksum failed for record #'+string(nrecords)
endwhile
close,lun
free_lun,lun

; 2nd PASS

nrecords = 0l
record_size0 = 0l
record_size1 = 0l
record_size2 = 0l
```

```
data_l1 = replicate({STR_WAV_H_RES_DATA_L1},nsamples)
start_sample_index = 0l

openr,lun,file,/get_lun,/swap_if_little_endian
while ~eof(lun) do begin

  nrecords += 1l
  readu,lun,record_size0
  ptr0 = (fstat(lun)).cur_ptr
  readu,lun,header

  PalkHz  = fltarr(header.nfreq)
  readu,lun, PalkHz

  Paltime = fltarr(header.npalcy,header.nconfig)
  readu,lun, Paltime

  Cag1    = fltarr(header.npalcy,header.nconfig)
  readu,lun, Cag1

  if header.ncag2 ne 0 then begin
    Cag2    = fltarr(header.ncag2,header.nconfig)
    readu,lun, Cag2
  endif

  if header.LoopA eq 0 then header.LoopA = 1

  Auto1   = fltarr(header.nfreq,header.LoopA)
  readu,lun, Auto1

  if header.nauto2 ne 0 then begin
    Auto2   = fltarr(header.nauto2,header.LoopA)
    readu,lun, Auto2
  endif

  if header.LoopC ne 0 then begin
    CrosR   = fltarr(header.nfreq,header.LoopC)
    readu,lun, CrosR
    CrosI   = fltarr(header.nfreq,header.LoopC)
    readu,lun, CrosI
  endif

  ksamples = header.nconfig*header.nfreq

  data_amj   = header.Jamjcy(0)*10000l+header.Jamjcy(1)*100l+header.Jamjcy(2)
  data_sec   = rebin([double(header.Jhmscy(0)*3600l+header.Jhmscy(1)*60l+header.Jhmscy(2))+ $
               double(header.sfract)],ksamples)+double(rebin(paltime,ksamples))
  data_freq  = reform(rebin(reform(PalkHz,1,header.nfreq),header.nconfig,header.nfreq),$
               ksamples)
  data_dt    = header.msti
  data_ant   = reform(rebin(reform(header.iant12(0:header.nconfig-1),header.nconfig,1), $
               header.nconfig,header.nfreq),ksamples)
  data_agc1  = reform(transpose(rebin(reform(cag1,header.npalcy,header.nconfig), $
               header.nfreq,header.nconfig)),ksamples)
  if header.ncag2 ne 0 then begin
    data_agc2  = reform(transpose(rebin(reform(cag2,header.npalcy,header.nconfig), $
                 header.nfreq,header.nconfig)),ksamples)
  endif else begin
    data_agc2  = 0
  endelse
  if header.loopA ne 0 then begin
    data_auto1 = reform(transpose(rebin(reform(auto1,header.nfreq,header.nconfig), $
                 header.nfreq,header.nconfig)),ksamples)
    if header.nauto2 ne 0 then begin
      data_auto2 = reform(transpose(rebin(reform(auto2,header.nfreq,header.nconfig), $
                   header.nfreq,header.nconfig)),ksamples)
    endif else begin
      data_auto2 = 0
    endelse
  endif else begin
```

```
    data_auto1 = 0
    data_auto2 = 0
  endelse
  if header.LoopC ne 0 then begin
    data_crosR = reform(transpose(rebin(reform(CrosR,header.nfreq,header.nconfig), $
                 header.nfreq,header.nconfig)),ksamples)
    data_crosI = reform(transpose(rebin(reform(CrosI,header.nfreq,header.nconfig), $
                 header.nfreq,header.nconfig)),ksamples)
  endif else begin
    data_crosR = 0
    data_crosI = 0
  endelse

  if (header.irad mod 10) le 3 then begin
    data_fi = reform(rebin(reform(indgen(16),1,16),header.nconfig,16),ksamples) $
              + 1000*(header.irad mod 10)
  endif
  if (header.irad mod 10) gt 3 then data_fi = fix(data_freq-125.)/50 + 4000

  data_sweep = intarr(ksamples)+(nrecords-1l)

  fband = [0.225,0.900,3.600,12.50]
  data_df = fband(data_fi/1000-1)

  data_l1(start_sample_index:start_sample_index+ksamples-1).amj   = data_amj
  data_l1(start_sample_index:start_sample_index+ksamples-1).sec   = data_sec
  data_l1(start_sample_index:start_sample_index+ksamples-1).freq  = data_freq
  data_l1(start_sample_index:start_sample_index+ksamples-1).fi    = data_fi
  data_l1(start_sample_index:start_sample_index+ksamples-1).sweep = data_sweep
  data_l1(start_sample_index:start_sample_index+ksamples-1).dt    = data_dt
  data_l1(start_sample_index:start_sample_index+ksamples-1).df    = data_df
  data_l1(start_sample_index:start_sample_index+ksamples-1).ant   = data_ant
  data_l1(start_sample_index:start_sample_index+ksamples-1).agc1  = data_agc1
  data_l1(start_sample_index:start_sample_index+ksamples-1).agc2  = data_agc2
  data_l1(start_sample_index:start_sample_index+ksamples-1).auto1 = data_auto1
  data_l1(start_sample_index:start_sample_index+ksamples-1).auto2 = data_auto2
  data_l1(start_sample_index:start_sample_index+ksamples-1).crosR = data_crosR
  data_l1(start_sample_index:start_sample_index+ksamples-1).crosI = data_crosI

  start_sample_index += ksamples

  ptr1 = (fstat(lun)).cur_ptr
  readu,lun,record_size1

  record_size2 = ptr1-ptr0
  if record_size1 ne record_size0 or record_size1 ne record_size2 then $
    message,'WARNING Checksum failed for record #'+string(nrecords)

endwhile

data_l1.num = lindgen(nsamples)

if keyword_set(autodb) then begin
  data_l1.agc1  = 10.*alog10(data_l1.agc1)
  data_l1.agc2  = 10.*alog10(data_l1.agc2)
  data_l1.auto1 = 10.*alog10(data_l1.auto1)
  data_l1.auto2 = 10.*alog10(data_l1.auto2)
endif

time1 = systime(/sec)
message,/info,string(nrecords)+' records read in '+string(time1-time0)+' seconds.'
message,/info,string(nsamples)+' samples stored.'


close,lun
free_lun,lun

END
```

4. STR_WAV_AVERAGE_HEADER__DEFINE.PRO

```
PRO STR_WAV_AVERAGE_HEADER__DEFINE

tmp = {STR_WAV_AVERAGE_HEADER, irad:0, ITCDS:[0l,0l], jusecy:0l, Jamjcy:[0,0,0], $
 Jhmscy:[0,0,0], Rua:0., Hlat:0., Hlon:0., Moysec:0, Nfreq:0}

END
```

5. STR_WAV_AVERAGE_DATA_L1__DEFINE.PRO

```
PRO STR_WAV_AVERAGE_DATA_L1__DEFINE

tmp = {STR_WAV_AVERAGE_DATA_L1, amj:0l, num:0l,  sweep:0l, sec:0.d0, freq:0., $
 flux:0.}

END
```

6. READ_STR_WAV_AVERAGE_DATA.PRO

```
PRO READ_STR_WAV_AVERAGE_DATA, file, data_l1,debug=debug

; READING FILE IN 2 PASS
; 1st pass = scanning for number of records and total number of samples.
; 2nd pass = loading data into data_l1

time0 = systime(/sec)

; 1st PASS

nrecords = 0l
nsamples = 0l
record_size0 = 0l
record_size1 = 0l
record_size2 = 0l
header  = {STR_WAV_AVERAGE_HEADER}

openr,lun,file,/get_lun,/swap_if_little_endian
while ~eof(lun) do begin
  nrecords += 1l
  readu,lun,record_size0
  ptr0 = (fstat(lun)).cur_ptr
  readu,lun,header
  nsamples += header.nfreq

  point_lun,lun,ptr0+record_size0
  ptr1 = (fstat(lun)).cur_ptr
  readu,lun,record_size1
  record_size2 = ptr1-ptr0
  if record_size1 ne record_size0 or record_size1 ne record_size2 then $
    message,'WARNING Checksum failed for record #'+string(nrecords)
endwhile
close,lun
free_lun,lun

; 2nd PASS

nrecords = 0l
record_size0 = 0l
record_size1 = 0l
record_size2 = 0l
data_l1 = replicate({STR_WAV_AVERAGE_DATA_L1},nsamples)
start_sample_index = 0l

openr,lun,file,/get_lun,/swap_if_little_endian
while ~eof(lun) do begin

  nrecords += 1l
  readu,lun,record_size0
  ptr0 = (fstat(lun)).cur_ptr
  readu,lun,header
```

```
  if header.nfreq gt 0 then begin
    Freq  = fltarr(header.nfreq)
    readu,lun, Freq

    Flux = fltarr(header.nfreq)
    readu,lun, Flux

    ksamples = header.nfreq

    data_amj   = header.Jamjcy(0)*10000l+header.Jamjcy(1)*100l+header.Jamjcy(2)
    data_sec   = double(header.Jhmscy(0)*3600l+header.Jhmscy(1)*60l+header.Jhmscy(2))
    data_sweep = intarr(ksamples)+(nrecords-1l)
    data_freq  = Freq
    data_flux  = Flux

    data_l1(start_sample_index:start_sample_index+ksamples-1).amj   = data_amj
    data_l1(start_sample_index:start_sample_index+ksamples-1).sec   = data_sec
    data_l1(start_sample_index:start_sample_index+ksamples-1).sweep = data_sweep
    data_l1(start_sample_index:start_sample_index+ksamples-1).freq  = data_freq
    data_l1(start_sample_index:start_sample_index+ksamples-1).flux  = data_flux

    start_sample_index += ksamples

  endif

  ptr1 = (fstat(lun)).cur_ptr
  readu,lun,record_size1

  record_size2 = ptr1-ptr0
  if record_size1 ne record_size0 or record_size1 ne record_size2 then $
    message,'WARNING Checksum failed for record #'+string(nrecords)

  if keyword_set(debug) then stop
endwhile

data_l1.num = lindgen(nsamples)

time1 = systime(/sec)
message,/info,string(nrecords)+' records read in '+string(time1-time0)+' seconds.'
message,/info,string(nsamples)+' samples stored.'


close,lun
free_lun,lun

END
```

4. Historique des Corrections

V1.4 Correction d'une erreur qui empêchaient le chargement des fichiers contenant des mesures sur plus d'un couple d'antenne (modes DF1 et DF2, voir documentation S/Waves).