

SIGPROC–v1.0 : (Pulsar) Signal Processing Programs

Dunc Lorimer — Arecibo Observatory — dunc@naic.edu — September 3, 2001

Summary: The SIGPROC package is a collection of programs written to convert and process fast-sampled pulsar data into a compact and easy-to-use format suitable for off-line analyses for searching, timing and polarimetry applications. This document describes how to install and run the various programs. Several example applications are presented using real and simulated data sets.

Contents

| | | |
|-----------|---|-----------|
| 1 | About SIGPROC | 2 |
| 2 | Installation Procedure | 3 |
| 3 | Header Information and Data format | 3 |
| 4 | Data conversion using filterbank | 5 |
| 5 | Creating mock data sets using fake | 9 |
| 6 | Looking at headers using header | 10 |
| 7 | Looking at data using bandpass and reader | 11 |
| 8 | Data reduction using decimate and dedisperse | 13 |
| 9 | Getting pulse profiles using fold | 15 |
| 10 | Putting it all together: the quicklook data reduction script | 17 |
| 11 | Plans for Future Work | 19 |
| A | Monitoring programs using the Tk monitor widget | 19 |
| B | Running TEMPO to generate polynomial coefficients | 20 |
| C | The EPN Data format | 21 |

1 About SIGPROC

SIGPROC is a package designed to standardize the initial analysis of the many types of fast-sampled pulsar data being collected using machines at Arecibo and other observatories. Currently recognized machines are the Wide Band Arecibo Pulsar Processor (WAPP), the Penn State Pulsar Machine (PSPM), the Arecibo Observatory Fourier Transform Machine (AOFTM), the Berkeley Pulsar Processors (BPP) and the filterbank at the Ooty radio telescope (OOTY). The SIGPROC tools should help users look at their data quickly, without the need to write (yet) another routine to read data or worry about big/little endian compatibility (byte swapping is handled automatically). The current suite of programs is:

filterbank - convert raw pulsar-machine data to filterbank format: a stream of n-bit numbers corresponding to multiple polarization and/or frequency channels.

fake - produce fake filterbank format data containing periodic signals immersed in Gaussian noise for testing and calibration of downstream programs.

decimate - add together frequency channels and/or time samples of incoming filterbank data to reduce the time and/or frequency resolution (useful for quick-look purposes).

dedisperse - correct incoming filterbank data for interstellar dispersion, writing the output time series as one or more dedispersed sub-bands.

fold - fold incoming filterbank or time series data modulo a pulse period. Pulses are output in ASCII or EPN format (§C). An Expect script to generate polynomial coefficients is also available.

bandpass - write out the mean bandpass to an ASCII file.

header - read the header of filterbank or time series data and display in human-readable form.

reader - read the filterbank or time series data and display in human-readable form.

quicklook - csh script to perform a quick analysis of total-power filterbank data on a known pulsar.

monitor - wish script to monitor programs running in a given directory using a Tk pop-up widget.

All of the programs within SIGPROC are written in C and can be run from the UNIX command-line. Use is made of standard input and output streams so that piping between programs is possible to “glue” together various tasks. As an example, the following pipeline:

```
% filterbank B0823+26.pspm | dedisperse -d 19 -s 4 | fold -p polyco.dat > B0823+26.epn
```

will read in and dedisperse raw PSPM data into four subbands which are then folded modulo the pulse period based on a set of polynomial coefficients generated by TEMPO stored in the file `polyco.dat`. The folded profiles for each band are written in EPN format¹ to the file `B0823+26.epn`. The raw data used in this example, and other files, are available for download from the Arecibo pulsar home page (**todo!**).

A detailed description of these programs and scripts is given in the remainder of this document which is structured as follows: in §2 we describe how to install SIGPROC; §3 describes the filterbank data and

¹Those who cringe at the words “EPN” and “format” will be relieved to know that a command-line option in `fold` to output plain old ASCII format for the folded profiles is also available!

header format used by all the programs; producing real and fake filterbank data is described in §4 and §5 respectively; programs to look at the headers and raw data are discussed in §6 and §7; data reduction tasks (decimation and dedispersion) are described in §8; folding filterbank data to produce pulse profiles is described in 9; a script do quick data analyses is presented in §10 respectively; plans for future work are outlined in §11. Supplementary information is given in the appendices on: monitoring the programs (§A); a standalone script to generate `polyco.dat` files using TEMPO (§B); the EPN data format (§C);

2 Installation Procedure

SIGPROC has so far been successfully installed for use on Solaris, Linux and HP-UX. At Arecibo, Solaris and Linux executables can be found under `~pulsar/bin/solaris` and `~pulsar/bin/linux` respectively. ANSII C was (hopefully!) adhered to fairly closely during writing of the programs so that installation on other operating systems should also be possible. Before starting, install the `fftw` libraries, available free of charge from <http://www.fftw.org>. Once this package is on your system, proceed with the following:

0: Download the package from <http://www.naic.edu/~pulsar/download/sigproc-1.0.tar.gz>

1: Unpack the gzip-compressed tar file via the command `gunzip sigproc-1.0.tar.gz` and extract the contents using the `tar` program: `tar xf sigproc-1.0.tar`

2: Before compiling the software, check that the environment variable `OSTYPE` is defined on your system via the command `printenv OSTYPE` if this is not set already, edit your shell setup file to include a line of the form: `setenv OSTYPE osname`, where `osname` may be either `solaris`, `linux` or `hpux`.

3: The contents of the tar file will be distributed in the directory `sigproc-1.0/`. Go into this directory and copy the file `makefile.template` to either: `makefile.solaris`, `makefile.linux` or `makefile.hpux` depending on your operating system. If compiling on more than one system, create another `makefile.osname` and compile from the same source code directory. Only one copy of the source code is required.

4: Edit the three lines in the makefile(s) to specify the choice of compiler (it is recommended to use the gnu-C compiler with full optimization: `CC = gcc -O2`), the location of the `fftw` libraries (an entry of the form: `FFT = -L/home/pulsar/lib/ -lfftw -lfftw`), and the path where the executable files will be placed (the `sigproc-1.0` directory is used as a default — i.e. `BIN = ./`).

5: Having edited the makefile(s), type `make` in the directory and let the compiler go to work.

Three other software packages are desirable, but not absolutely necessary. To create files containing polynomial coefficients for high-precision folding, install the TEMPO software package which is freely available from the Princeton pulsar website (<http://pulsar.princeton.edu>). To monitor the programs using a Tk pop-up widget make sure that the `wish` shell is in your path (we recommend use of Tcl/Tk version 8.0 or higher). This is freely available from <http://www.scriptics.com>. For making diagnostic plots you will need to compile the `quickplot` Fortran program which requires the PGPLOT graphics package available from <http://www.astro.caltech.edu/~tjp/pgplot>. Edit the `makefile` to give the appropriate path to PGPLOT on your system before typing `make quickplot`.

3 Header Information and Data format

Before describing the programs in detail, some description of the header and data formats used within SIGPROC is appropriate for those wishing to read the data into other programs. The `filterbank`

program (see §4) reads in the raw data files produced by the machine, dealing with the header information contained in the files and the (usually non-trivial) channel ordering of the samples. `filterbank` outputs the data in the following way:

```
HEADER_START stream_of_header_parameters HEADER_END stream_of_data_values
```

The `HEADER_START` and `HEADER_END` character strings signal the start and finish of a stream of header parameters that describe the data. The default is to include these at the beginning of the data file. We recognize that some users will prefer not to have to deal with the header in this way. For these users, `filterbank` has a `-headerfile` command-line option to pipe the header into a separate ASCII file (this is described along with the other command-line options later on).

The header variables have been restricted to key parameters for ease of use. Currently these are:

- `telescope_id` (int): 0=fake data; 1=Arecibo; 2=Ooty... others to be added
- `machine_id` (int): 0=FAKE; 1=PSPM; 2=WAPP; 3=OOTY... others to be added
- `data_type` (int): 1=filterbank; 2=time series... others to be added
- `rawdatafile` (char []): the name of the original data file
- `tstart` (double): time stamp (MJD) of first sample
- `tsamp` (double): time interval between samples (s)
- `nbits` (int): number of bits per time sample
- `fch1` (double): centre frequency (MHz) of first filterbank channel
- `foff` (double): filterbank channel bandwidth (MHz)
- `nchans` (int): number of filterbank channels
- `nifs` (int): number of separate IF channels
- `refdm` (double): reference dispersion measure ($\text{cm}^{-3} \text{ pc}$)

A given header stream will contain most, but not necessarily all, of the above variables.

In the general case, the data consists of `nifs` polarization channels of `nchans` frequency channels of `nbit` numbers. The data stream following the header can then be thought of as 1-D array of N elements with indices running between 0 and $N - 1$, where

$$N = \text{nifs} \times \text{nchans} \times \text{nsamples},$$

and `nsamples` is the observation time divided by `tsamp`. Thus, for a given IF channel $i = (0, 1, 2, 3)$ and frequency channel $c = (0 \dots \text{nchans} - 1)$, the array index for sample $s = (0, 1, 2 \dots)$ is

$$s \times \text{nifs} \times \text{nchans} + i \times \text{nchans} + c.$$

The sky frequency of channel c is then simply

$$\text{fch1} + c \times \text{foff}.$$

We follow the convention of assigning a negative frequency to `foff` in the headers to signify that the highest frequency channel is `fch1`. Currently, all filterbank data is written out in this order.

4 Data conversion using filterbank

The interface between the raw data and the rest of the SIGPROC package is the `filterbank` program. As with all the programs on-line help is obtained by typing the name of the program followed by `help`:

```
% filterbank help
```

```
filterbank - convert raw pulsar-machine data to filterbank format
```

```
usage: filterbank <rawdatafile> -{options}
```

```
rawdatafile - raw data file (recognized machines: WAPP, PSPM, OOTY)
```

```
options:
```

```
-o filename - output file containing filterbank data (def=stdout)
-s skiptime - skip the first skiptime (s) of data (def=0.0)
-r readtime - read readtime (s) of data (def=all)
-i IFstream - write IFstream (IFstream=1,2,3,4)
-n nbits    - write n-bit numbers (def=input format)
-dt tsec    - add tsec seconds to tstart value in header (def=0.0)
-swapout    - perform byte swapping on output data (def=native)
-floats     - write floating-point numbers (equal to -n 32)
-sumifs     - sum IFs 1+2 to form total-power data
-headerfile - write header parameters to an ASCII file (head)
```

```
options for correlator data:
```

```
-hamming    - apply Hamming window before FFT (def=nowindow)
-hanning    - apply Hanning window before FFT (def=nowindow)
-novanvleck - don't do van Vleck correction before FFT (def=doit)
-zerolag    - write just the zero-lag value for each IF
-rawcfs     - write raw correlation functions (novanvleck)
-corcfs     - write corrected correlation functions (vanvleck)
```

Given just the name of the raw data file as the argument, `filterbank` will determine the origin of the data and, if it can read the file, unpack the samples before writing the header parameters and data as described in §3. The header and data go to the standard output by default but can be redirected to a file using the `-o filename` option, or in the standard way:

```
% filterbank rawdatafile > filterbankfile
```

With no further options, `filterbank` will read and unscramble all the data in the original file. A specific portion of the data can be specified using the `-r` and `-s` command-line options. For example:

```
% filterbank rawdatafile -r 10.0 > filterbankfile
```

reads just the first 10 seconds of data. These options are useful for a quick look at the data.

Selecting and/or summing IF streams: By default, all the IF streams (if there are more than one) in the file are read and processed. To select one or more of these, ignoring the others, use the `-i` option:

```
% filterbank rawdatafile -i 1 -i 2 > filterbankfile
```

will process just the first two IF channels of the raw data file. `filterbank` provides the option to sum *just the first two* IF channels (to form total-power data) via the `-sumifs` option:

```
% filterbank rawdatafile -sumifs > filterbankfile
```

This is a useful if, for example, to get just total power from polarimetry data for off-line searching.

ASCII headers: As mentioned in §3, `filterbank` will broadcast a header stream before writing the data. This header is used by other downstream SIGPROC programs to process the data. To make use of it in analysis with other programs, call the function `read_header` and link with the other routines contained in the file `read_header.c`. For those who prefer not to be bothered with these routines, use the `-headerfile` option when calling `filterbank`. For example:

```
% filterbank B0823+26.pspm -headerfile > B0823+26.fil
```

will create the file `B0823+26.fil` containing just the `filterbank` channels along with the relevant header parameters in an ASCII file head. In this case:

```
Original PSPM file: B0823+26.pspm
Sample time (us): 80.000002
Time stamp (MJD): 51740.882986111108
Number of samples/record: 512
Center freq (MHz): 430.000000
Channel band (kHz): 62.000000
Number of channels/record: 128
```

the user is then left to parse this file as he/she feels fit. An alternative means of getting header information would be to use the `header` program in the following example:

```
% filterbank B0823+26.pspm | header -tstart
```

which will return `51740.882986111108` to the standard output. Any of the header variable names listed in §3 can be given as a command-line option to the `header` program. Further details are given in §7.

Changing the number of bits per sample: By default, `filterbank` will write the outgoing data with the same number of bits per sample as the native format (e.g. 4 bits per sample for PSPM). For machines which write out larger numbers of bits (e.g. the WAPP) it is useful to be able to pack the data more efficiently using the `-n` option. For example, the sequence:

```
% filterbank wappdatafile -n 8 > filterbankfile
```

will process a WAPP data file (usually 16 bits per sample) and pack the outgoing samples as single-byte integers. For search purposes, where no loss in sensitivity is seen and data products are reduced significantly, use of this option is highly recommended.

Floating point output: Currently, no descaling parameters are given in the header when packing down data. This means that for applications where the absolute value of the data is necessary (e.g. polarization work) it is necessary to store the data as floating point numbers. The option `-floats` is provided for this purpose (just an alias for `-n 32`).

Byte swapping the output: Multi-byte precision data are written in different orders depending on your machine's operating system. The original WAPP data, for example, was written on a PC (little endian format). The `filterbank` program knows about this and *automatically* does any byte swapping required while reading. When it comes to writing the data out, however, the program will always write data in the native order of the processing machine. To swap the bytes around before writing for use on other machines, use the `-swapout` option.

Correlator-specific options: Presently, the WAPP is the only correlator machine recognized by SIGPROC which records auto- and, in polarization mode, cross-correlation functions for given numbers of lags. The autocorrelation function $R(\tau)$, as a function of lag τ is defined by:

$$R(\tau) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T V(t)V^*(t + \tau)dt,$$

where $V(t)$ is the complex signal voltage as a function of time t . From the Weiner-Khinchin theorem, the power spectral density function $P(f)$ is the Fourier transform of $R(\tau)$:

$$P(f) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} R(\tau)e^{-2\pi if\tau} d\tau.$$

In practice to obtain the equivalent of frequency channels of a filterbank, the lags from each IF channel need to be corrected for finite-level quantization — the so-called van Vleck correction (see for example Hagen & Farley 1973, *Radio Science*, **8**, 775–784) before the Fast Fourier Transform (FFT) to obtain the spectra. For reference, the three-level van Vleck formula used within `filterbank` to correct measured auto-correlation values (r) to unbiased ones (ρ) can be written as

$$r = \frac{1}{\pi} \int_0^\rho \left(\exp\left(\frac{-(\alpha/\sigma)^2}{1+x}\right) + \exp\left(\frac{-(\alpha/\sigma)^2}{1-x}\right) \right) \frac{dx}{\sqrt{(1-x^2)}},$$

where α is the digitizer threshold and σ the rms voltage. This correction is what `filterbank` does by default before FFTing the correlation functions to produce spectra.

A number of options exist to modify the default processing. To reduce FFT leakage, either a Hanning or Hamming window can be applied to the correlation functions via the `-hamming` and `-hanning` switches. Select `-rawcfs` to output the raw correlation functions quantized to the precision specified by `nbits`. To get at the raw correlation functions, include the floating-point option:

```
% filterbank wappdatafile -rawcfs -floats > rawcfile
```

The `-corcfs` option will write out the correlation functions *applying* the van Vleck correction.

Obscure correlator options: For completeness, we mention two other correlator specific options: `-novanvleck` and `-zerolag`. The `-novanvleck` option will not apply the quantization correction before the FFT. This feature is really for instructional purposes since, to FFT the data to get frequency channels,

signal-to-noise will be lost if not applying the van Vleck correction. Another option that is primarily used for testing is `-zerolag`. If selected, this outputs just the first correlation function for each IF (the so-called zero lag) as a floating point number. Inserting $\tau = 0$ into the above expression for $P(f)$, we note that the zero lag is just the sum over all the frequency channels — equivalent to a time series with no dispersion measure correction.

5 Creating mock data sets using fake

The `fake` program was written to create test data sets containing pulses hidden in Gaussian noise:

```
% fake help

fake - produce fake filterbank format data for testing downstream code

usage: fake -{options}

options:

-period    p - period of fake pulsar in ms (def=random)
-width     w - pulse width in percent (def=4)
-snrpeak   s - signal-to-noise ratio of single pulse (def=1.0)
-dm        d - dispersion measure of fake pulsar (def=random)
-nbits     b - number of bits per sample (def=4)
-nchans    n - number of filterbank channels (def=128)
-tsamp     t - sampling time in us (def=80)
-tobs      t - observation time in s (def=10)
-tstart    t - MJD time stamp of first sample (def=50000.0)
-nifs      n - number of IFs (def=1)
-fch1      f - frequency of channel 1 in MHz (def=433.968)
-foff      f - channel bandwidth in MHz (def=0.062)
-seed      s - seed for Numerical Recipes ran1 (def=seconds since midnight)
-nosmear   - do not add in dispersion/sampling smearing (def=add)
-swapout   - perform byte swapping on output data (def=native)
```

Default parameters are a filterbank similar to the PSPM. As an example, consider some fake PSPM data for a 42-s observation of a pulsar with a period of $\sim \pi$ ms, a duty cycle of 10% and a DM of 30:

```
% fake -period 3.1415927 -width 10 -dm 30 -tobs 42 -nbits 4 > pspm.fil
```

Each channel of fake data has a zero mean and unit rms. The signal-to-noise ratio refers to the height of a single pulse in each channel. In the above example, the default signal-to-noise was used. Weaker pulsars can be easily made to challenge limits of off-line search algorithms etc. By default, the fake pulse width w is smeared by an amount dependent on the filterbank setup using the quadrature sum:

$$\sqrt{w^2 + \text{tsamp}^2 + t_{\text{DM}}^2},$$

where t_{DM} is the dispersion smearing of the pulse over a single filterbank channel given by:

$$t_{\text{DM}} = 8.3 \times 10^6 \text{ms DM } \Delta\nu/\nu^3,$$

assuming the centre frequency ν is much larger than the channel bandwidth $\Delta\nu$ (both measured in MHz). Smearing can be disabled using the `-nosmear` option. Bit-format and byte-swapping options are identical to those described for the `filterbank` program in the previous section. The starting seed of the random number generator defaults to a number obtained by starting with the number of seconds since midnight and calling the random number generator that many times. This can be overridden by specifying a seed using the `-seed` option.

6 Looking at headers using header

The `header` program allows humans easy access to the binary header string in the filterbank data. As an example of the full output, here is the header of our PSPM test data:

```
% header B0823+26.fil

Data file           : B0823+26.fil
Header size (bytes) : 191
Data size (bytes)   : 2359296
Data type           : filterbank
Telescope           : Arecibo
Datataking Machine  : PSPM
Frequency of channel 1 (MHz) : 433.968000
Channel bandwidth   (MHz) : -0.062000
Number of channels  : 128
Time stamp of first sample (MJD) : 51740.882986111108
Gregorian date (YYYY/MM/DD) : 2000/07/15
Sample time (us)    : 80.00000
Number of samples   : 36864
Observation length (seconds) : 2.949120
Number of bits per sample : 4
Number of IFs       : 1
```

alternatively, `header` can be used with one or more of the above command-line options to return just the value of the parameter of interest (this is particularly useful when getting values from within scripts without having to parse the standard output). Currently available options are:

```
-telescope - return telescope name
-machine   - return datataking machine name
-fch1      - return frequency of channel 1 in MHz
-foff      - return channel bandwidth in MHz
-nchans    - return number of channels
-tstart    - return time stamp of first sample (MJD)
-tsamp     - return sample time (us)
-nbits     - return number of bits per sample
-nifs      - return number of IF channels
-headersize - return header size in bytes
-datasize  - return data size in bytes if known
-nsamples  - return number of samples if known
-tobs      - return length of observation if known (s)
```

It should be noted that `headersize`, `datasize`, `nsamples` and `tobs` are not header variables *per se*; they are derived by the program, based upon the file size and the real header variables.

7 Looking at data using bandpass and reader

The `bandpass` program is a simple utility to read incoming data and output a time-averaged bandpass:

```
% bandpass help

bandpass - outputs the pass band from a filterbank file

usage: bandpass {filename} -{options}

options:

    filename - filterbank data file (def=stdin)
-d numdumps - number of dumps to average over (def=all)
-t dumptime - number of seconds to average over (def=all)
```

In its simplest form, `bandpass` averages over the entire data file. The data for Fig. 1 were obtained using:

```
% filterbank B0823+26.pspm | bandpass > bandpass.ascii
```

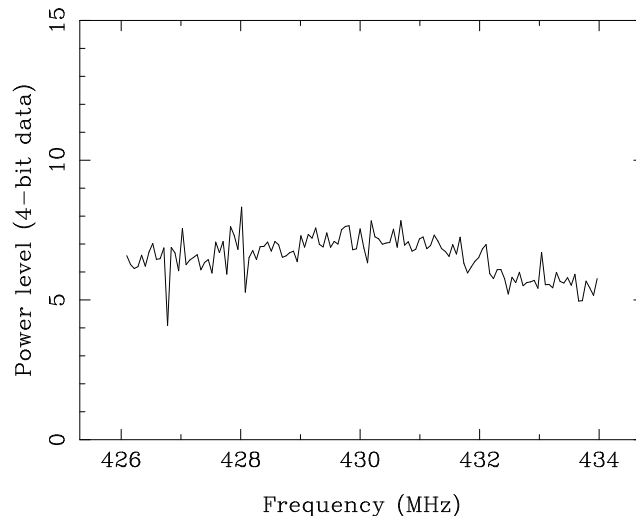


Figure 1: Output data from `bandpass` for the test observation of PSR B0823+26 using the PSPM.

The ASCII data is written in a simple format with one line for each frequency channel: `frequency if1 if2...` for up to `nifs` separate IFs. The `-d` and `-t` options allow averaging and output of the bandpass for a given number of dumps, or seconds. Each dump is encapsulated within `#START` and `#STOP` separators:

```
#START
freq(1)      if(1) .... if(nifs)
...         .     ..   ...
freq(nchans) if(1) .... if(nifs)
#STOP
```

where the `freq(1)` is the sky frequency of channel 1 in MHz and so on for all `nchans` channels.

A more general-purpose program is `reader` which will print out filterbank-format data as an ASCII stream to the standard output. This has been very useful in developing and testing the SIGPROC programs, and for getting ASCII data to make quick plots of filterbank and time series data.

```
% reader help
```

```
reader - look at filterbank data in ASCII format
```

```
usage: reader {filename} --{options}
```

```
filename is the filterbank data file (def=stdin)
```

```
options:
```

```
-c          c - output only frequency channel c (1..nchans) (def=all)
-i          i - output only IF channel i (1..nifs) (def=all)
-numerate  - precede each dump with sample number (def=time)
-noindex   - do not precede each dump with number/time
-stream    - produce a stream of numbers with #START/#STOP boundaries
```

In the general case, a filterbank file with `nchans` channels and `nifs` IFs, output is of the form:

```
% reader filterbankfile
```

```
time(1) if(1)c(1) if(1)c(2) .... if(1)c(nchans) ..... if(nifs)c(nchans)
time(2) if(1)c(1) if(1)c(2) .... if(1)c(nchans) ..... if(nifs)c(nchans)
time(3) if(1)c(1) if(1)c(2) .... if(1)c(nchans) ..... if(nifs)c(nchans)
```

the default case is to print out all IF and frequency channels. The output can be tailored by the `-i` and `-c` options to get just specific channels of interest. For example:

```
% filterbank B0823+26.pspm | reader -c 1 -c 2 -c 3 -c 4 | head
```

```
0.000000 5.000000 5.000000 6.000000 5.000000
0.000080 7.000000 5.000000 7.000000 6.000000
0.000160 7.000000 5.000000 6.000000 6.000000
0.000240 5.000000 5.000000 5.000000 7.000000
0.000320 5.000000 4.000000 5.000000 6.000000
0.000400 5.000000 4.000000 5.000000 6.000000
0.000480 5.000000 5.000000 5.000000 6.000000
0.000560 5.000000 5.000000 6.000000 5.000000
0.000640 6.000000 4.000000 5.000000 7.000000
0.000720 6.000000 6.000000 6.000000 5.000000
```

shows just the first four frequency channels of the PSPM data as a function of time. The `-numerate` switch will change this time stamp to an integer counter. Time or integer counters can be turned off completely via the `-noindex` option. The `-stream` option will, as in the case of the continuous `bandpass` output above, output a stream of numbers encapsulated by `#START` and `#STOP` separators.

8 Data reduction using decimate and dedisperse

Adding of adjacent time and/or frequency channels together to reduce the original resolution and size of the original data file is possible using the `decimate` program:

```
% decimate help
```

```
decimate - reduce time and/or frequency resolution of filterbank data
```

```
usage: decimate {filename} -{options}
```

```
options:
```

```
    filename - filterbank data file (def=stdin)
-c numchans - number of channels to add (def=all)
-t numsamps - number of time samples to add (def=none)
-T numsamps - (alternative to -t) specify number of output time samples
-n numbits  - specify output number of bits (def=input)
```

Output data from `decimate` is in standard filterbank format so that it can be easily read in by other SIGPROC programs. To get ASCII data, use the `reader` program (see §7). The following example adds all the frequency channels together, and every 32 time samples, to create the time series shown in Fig. 2.

```
% filterbank B0823+26.pspm | decimate -t 32 -n 32 | reader > timeseries.ascii
```

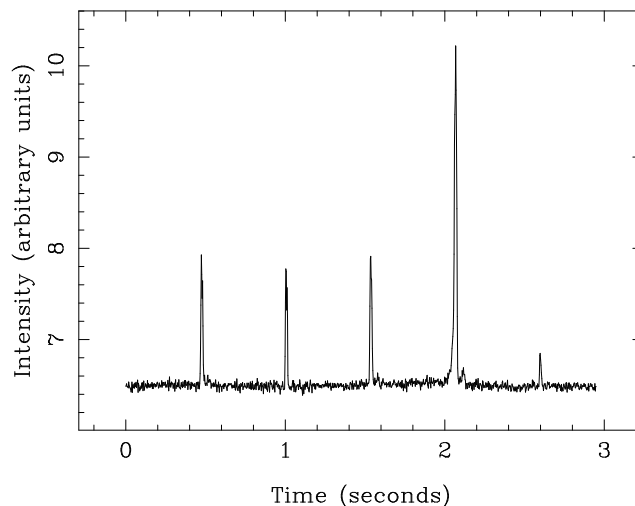


Figure 2: *Output time series from decimate for the test observation of PSR B0823+26 using the PSPM.*

Note that we have used the `-n` option to force the output number of bits per sample to be 32. By default `decimate` outputs data with the same number of bits as the incoming filterbank data. In this case, where there are strong single pulses, adding all the channels together would result in a signal-to-noise loss when trying to write the output time series with 4-bit precision.

While `decimate` is a good means for getting time series of weakly dispersed pulsars, it does not take into account the effects of dispersion by the interstellar medium where pulses emitted at higher radio

frequencies travel faster through the interstellar medium, arriving earlier than those emitted at lower frequencies. The time delay Δt between a high frequency ν_{hi} relative to a lower one ν_{lo} is

$$\Delta t = 4.15 \times 10^6 \text{ ms} \times (\nu_{\text{lo}}^{-2} - \nu_{\text{hi}}^{-2}) \times \text{DM},$$

where the frequencies are in MHz and the dispersion measure $\text{DM} = \int_0^d n_e dl$ ($\text{cm}^{-3} \text{ pc}$) is the integrated column density of free electrons along the line of sight. Here, d is the distance to the pulsar (pc) and n_e is the free electron density (cm^{-3}). For distant high-DM pulsars, especially those with short periods, dispersion needs to be accounted for to retain full time resolution. The `dedisperse` program does this by adding frequency channels with the appropriate time delays given a DM value:

```
% dedisperse help
```

```
dedisperse - form time series from filterbank data
```

```
usage: dedisperse {filename} -{options}
```

```
options:
```

```
    filename - full name of the raw data file to be read (def=stdin)
-d dm2ddisp - set DM value to dedisperse at (def=0.0)
-b numbands - set output number of sub-bands (def=1)
-o filename - output file name (def=stdout)
-c minvalue - clip samples that deviate more than minvalue*rms (def=noclipping)
-f reffreq   - dedisperse relative to refrf MHz (def=topofsubband)
-n num_bins - set number of bins if input is profile (def=256)
-swapout    - perform byte swapping on output data (def=native)
-nobaseline - don't subtract baseline from the data (def=subtract)
-headerless - write out data without any header info
```

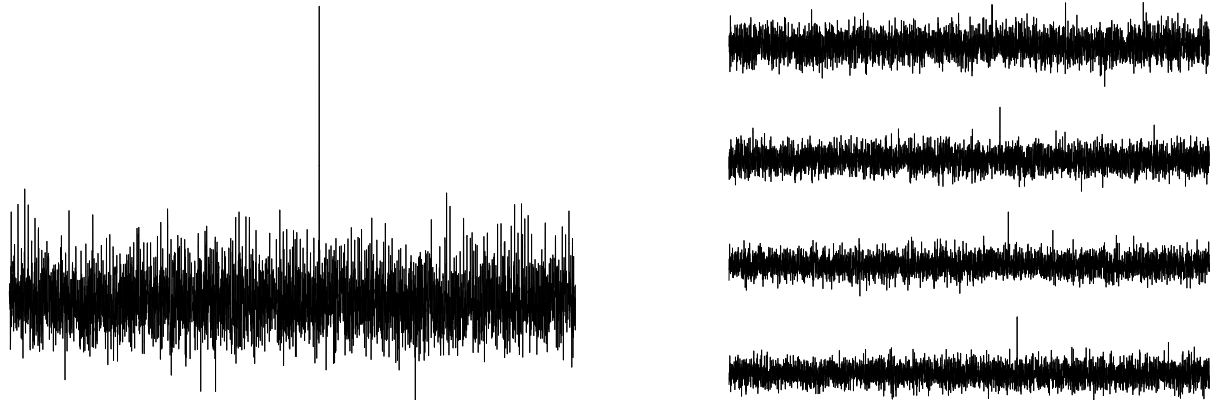


Figure 3: A WAPP observation of the millisecond pulsar B1937+21 showing a single “giant” pulse. Left: the dedispersed time series over the entire 100-MHz band. Right: the pulse seen in four dedispersed 25-MHz subbands. The length of the time series segment is ~ 0.26 s. The sampling time is $63.32 \mu\text{s}$.

In the example data for the 1.5578-ms pulsar B1937+21 shown in Fig. 3, the left panel was produced via:

```
% filterbank B1937+21.539 | dedisperse -d 71.04 | reader > timeseries.ascii
```

This single pulse is shown in four dedispersed frequency subbands in the right-hand panel of Fig. 3. These were obtained by adding a `-b 4` option into the dedisperse command-line in the above pipeline. In this case, dedispersion is carried out relative to the frequency of the first summed channel in each of the bands.

9 Getting pulse profiles using fold

The final analysis program in the SIGPROC package, `fold`, allows you to fold filterbank data modulo a pulse period to produce pulse profiles. `fold` accepts any number of IF and/or frequency channels, producing `nifs` \times `nchans` sets of profiles. The folding algorithm used is a simple one: for each time sample, compute the phase based on a, possibly time-dependent, value of the pulse period and add that sample to the nearest phase bin of the appropriate profile. The synopsis of `fold` is summarized below:

```
% fold help

fold - fold filterbank channels/time series data

usage: fold {filename} -{options}

options:

    filename - full name of the raw data file to be read (def=stdin)
-o out_file - output file for pulse profile data (def=stdout)
-p fold_prd - period to fold (ms) or polyco file (def=polyco.dat)
-f p_factor - multiply the period by p_factor (def=1.0)
-n num_bins - number of bins in folded profile(s) (def>window/tsamp)
-d time/num - dump profiles every time s or num pulses (def=nodumps)
-t samptime - hard-wire the sampling time (us) (def=header)
-l phaseval - phase value (turns) of left edge of pulse (def=0.0)
-r phaseval - phase value (turns) of right edge of pulse (def=1.0)
-ascii      - write profiles in ASCII format (def=epn)
-stream     - write profiles as ASCII streams with #START/#STOP boundaries
-nobaseline - don't subtract baseline from profiles (def=subtract)
```

Folding data at a fixed period: Consider folding a series containing our fake $\sim \pi$ -ms pulsar:

```
% fake -period 3.14159 -nchans 1 -nbits 32 | fold -p 3.14159 -ascii > profile.ascii
```

Note that the default profile output is in EPN format. This may be substituted by plain ASCII using the `-ascii` option as in the above example. The format of this output is a line for each bin:

```
bin_number if(1)c(1) if(1)c(2) .... if(1)c(nchans) ..... if(nifs)c(nchans)
```

In order to avoid overflows during folding, `fold` will by default subtract an offset from each folded sample calculated as the median value of a given data block. To turn off this feature, use the `-nobaseline` option. The default number of bins is given by the next largest integer value to the ratio of the folding period divided by the sampling time. This is, however, completely flexible. A lower number of bins would be

desirable, for example, when folding data for a faint pulsar or candidate. `fold` will permit oversampling which can pay dividends for high signal-to-noise observations of short-period pulsars.

Folding data using polynomial coefficients: For practical applications, the apparent pulse period is time-variable during the integration due to Doppler shifts resulting from the Earth’s motion and (for binary pulsars) from Doppler shifts induced by orbiting companions. To account for these the folding period needs to be updated during the integration. The TEMPO timing package can be used to create a set of polynomial coefficients to predict the change in period with time and `fold` can read these “polyco” files from TEMPO for these purposes. A script to run TEMPO to produce these files is described in §B. To tell `fold` to read a polyco file, supply the name of the filename with the `-p` option.

```
% filterbank B0823+26.pspm | fold -p polyco.dat -n 128 > B0823+26.epn
```

will fold each channel of the sample PSPM data for PSR B0823+26 to produce 128-bin profiles written to the file `B0823+26.epn` in EPN format. If no `-p` option is given to `fold` the program will look for the file `polyco.dat` as a matter of course so that, in the above case, it was not strictly necessary to specify the name of the polyco file. This is assumed in the following pipeline where the data are first dedispersed at the reference DM value of $19.4 \text{ cm}^{-3} \text{ pc}$ before being passed to `fold`:

```
% filterbank B0823+26.pspm | dedisperse -d 19.4 | fold > B0823+26.epn
```

Getting sub-integrations: In the above examples, `fold` produces one profile for each of `nchans` × `nifs` incoming data streams which corresponds to folding over the entire data set. It is often desirable to look at sub-profiles dumped at regular intervals during the observation — the `-d` (dump) option allows you to do this. Specifying a floating-point number, say f seconds, in this mode will output profiles to the nearest integer value of f/P pulses, where P is the pulse period in seconds. The following example on our fake millisecond pulsar daata would dump a subintegration approximately every 15 seconds:

```
% fold fakepulsar.fil -d 15.0 -p 3.1415927 > fakeprofiles.epn
```

Supplying an integer argument with the `-d` option, say n , the profiles are dumped every n pulses. So `-d 15` in the above example results in a profile being dumped every 15 periods (about 47 ms).

Single pulses and windowing profiles: Individual pulses can be obtained by specifying `-d 1` to the `fold` command line. The following example demonstrates this for the PSR B0823+26 PSPM data:

```
% filterbank B0823+26.pspm | dedisperse -d 19.4 | fold -d 1 > B0823+26.epn
```

The resulting EPN file contains a record for each single pulse. For this short data set, this amounts to just five single pulses shown in Fig. 4.

For single pulse applications, where the off-pulse region of the profile is usually not interesting, it is desirable to be able to set a window around the pulse. The `fold` program allows setting of windows via the `-l` and/or `-r` command-line options which specify the left and right-hand phase values of the windows. Phase values should be specified in turns ranging between 0.0 and 1.0. For example, the pulses in the lower panel of Fig. 4 were obtained using `fold -l 0.825 -r 0.925` for the PSR B0823+26 dataset. As before for the full profile, unless specified otherwise, `fold` will choose the number of bins based on the size of the window divided by the sampling interval.

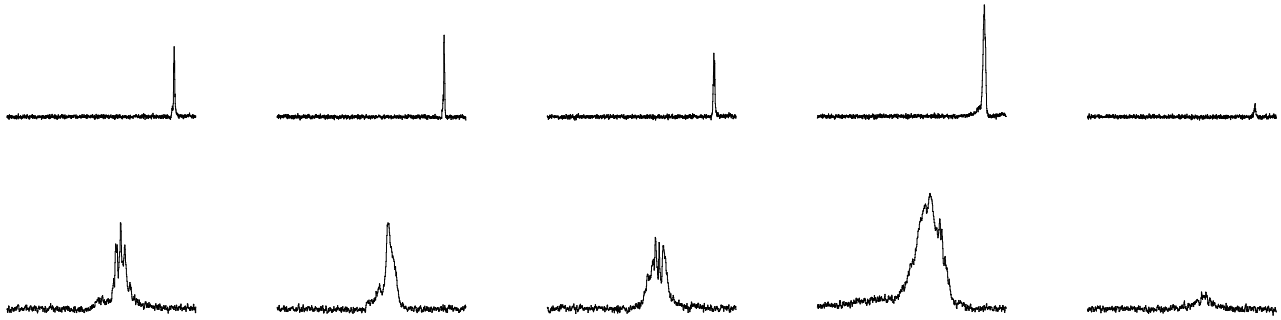


Figure 4: *Top: dedispersed single pulses for the PSPM test observation of PSR B0823+26. Bottom: the same data set after applying a phase window of 0.825 to 0.925 (see text)*

10 Putting it all together: the quicklook data reduction script

As an application of most of the programs discussed in the preceding sections, we conclude with the `quicklook` data analysis script which is designed to dedisperse and fold raw pulsar machine data taken on a known pulsar and produce a diagnostic output plot summarizing various aspects of the data:

```
% quicklook help

usage: quicklook <filename> -{options}

options:

-read time    - read and process only time (s) of data
-skip time    - skip the first time (s) of data
-addc nchans  - add nchans chans together before dedispersion (def=none)
-addt nsamps  - add nsamps samples together before dedispersion (def=none)
-nsints n     - specify number of time sub-integrations (def=8)
-nbands n     - specify number of frequency sub-bands (def=8)
-period p     - fold data at constant period (ms) (def=polyco)
-dm dmvalue   - dedisperse using dmvalue (pc cm-3) (def=polyco)
-clip value   - clip samples that deviate more than value*rms (def=noclipping)
-nbins n      - fold data using n bins (def=128)
-left phase   - specify left-hand phase window (def=0.0)
-right phase  - specify right-hand phase window (def=0.0)
-singlepulse  - set number of subints to be the number of pulses
-psr name     - fix source name for running TEMPO (def=filestem)
-mypolyco     - use an existing polyco file (def=make one)
-clean        - remove large files before and after (def=keep them)
-wipe         - remove large files beforehand (def=keep them)
```

As an example, the command:

```
% quicklook J1713+0747.744 -nbands 64 -nsints 32 -read 5
```

reduces the first 5 seconds of a WAPP observation of the millisecond pulsar J1713+0747 producing the plot shown in Fig. 10. N.B. Use of this script assumes that you have the `quickplot` program compiled (see §2 for further details).

QUICKLOOK: Arecibo WAPP datafile: J1713+0747.744
 Source: 1713+07 MJD: 51853.743935185186 Date: 2000/11/05
 Obs Freq: 1.420 GHz Bandwidth: 100.0 MHz Channels: 128
 t(samp): 127.32 us t(bin): 35.70 us N(bins): 128 SNR: 378.7
 Folding period: 4.5702374260 ms DM: 15.99 pc/cc

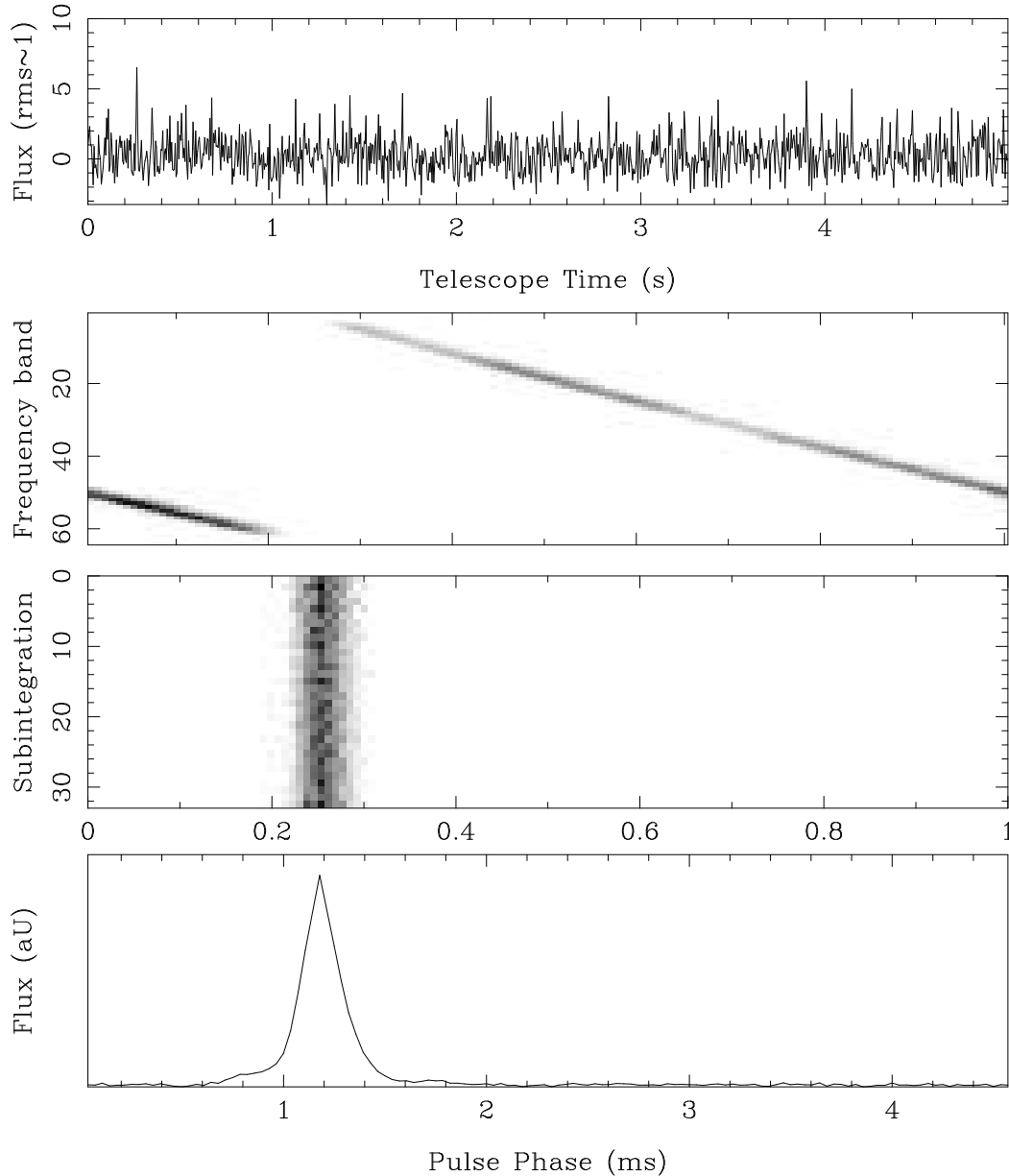


Figure 5: Sample output from the `quicklook` script for the millisecond pulsar *J1713+0747* observed by the WAPP. Top panel shows a 1024-point average of the dedispersed time series which has been normalised (as far as possible) so that it has a zero mean and unit rms. Below this are two panels showing the dedispersed frequency sub-bands (which clearly show the dispersion of the pulsar) and time sub-integrations as a function of pulse phase. The bottom plot is the integrated pulse profile. The signal-to-noise ratio of this profile is reported at the top of the plot along with essential header information.

11 Plans for Future Work

Current plans for future improvements to SIGPROC, once the dust has settled, include:

- Van Vleck corrections to cross-correlation functions for WAPP polarimetry
- Capability to read ABPP pulse profile data into `dedisperse`.
- A program to dedisperse the data using the tree algorithm.
- Improved C programs to look at EPN data
- Improved fake data - profiles and binary pulsars

All suggestions on these and other improvements, including offers to contribute write and/or improved routines for future releases of the package are most welcome. Please contact dunc@naic.edu

Acknowledgements

In putting together the SIGPROC package, I had the good fortune to work with a number of people who kindly donated existing routines or offered to write new ones. Andy Dowd wrote the original version of what became the `wapp2fb` routine for converting raw WAPP correlation values into spectra. Jeff Hagen also contributed to this effort and wrote the routines used for reading WAPP headers and byte swapping. Ingrid Stairs provided the `pspm_decode` routine — a C-version of an original Fortran-77 subroutine written by Alex Wolsczcan. Use was also made of some Numerical Recipes routines for generating random numbers for the `fake` program discussed in §5. Finally, many thanks to Jim Cordes and Maura Mclaughlin for their help in putting together and debugging some of the routines, and their suggestions for functionality.

A Monitoring programs using the Tk monitor widget

The SIGPROC programs run without any messages to the standard output. Instead, for each program, a `programname.monitor` file is created and tailing this file is a useful way of keeping track of what is going on. For example:

```
% tail fold.monitor
input J1713+0747.744.tim status time:0.0s:P(fold):0.0045702374s output stdout
input J1713+0747.744.tim status time:4.2s:P(fold):0.0045702374s output stdout
input J1713+0747.744.tim status finished output stdout
```

Another means of monitoring the programs is to use `monitor`, a `wish` script which keeps track of `programname.monitor` files as they appear in a given directory. Once the program is finished what it is doing, the monitor will go to sleep until the next time a `programname.monitor` file appears. To start the `monitor` script running, go to the directory where you are processing your files and type `monitor`. For example, starting a `filterbank` command:

```
% filterbank J1713+0747.744 -sumifs -r 5 > J1713+0747.744.fil
```

```
filterbank input: J1713+0747.744 status: time:30.5s output: stdout elapsed time: 37 s
```

will cause the following status bar to appear in the upper left-hand corner of the screen:

This counter will tick away updating as the file gets updated until the program is finished. If you have several jobs running, in a pipeline for example, several status bars will appear until their respective job is completed. To stop the monitor script at any time, type:

```
% monitor off
```

Note that you will confuse the script if you have two programs running from the same directory (for example two `filterbank` processes running on different raw datafiles) since the `programname.monitor` file will get updated by both programs. For such applications, run the programs from separate directories.

B Running TEMPO to generate polynomial coefficients

If you have `expect` in your path, you will also be able to take advantage of `polyco` a simple script designed to take the pain out of running TEMPO to generate files containing polynomial coefficients for use by `fold`. The synopsis of `polyco` is as follows:

```
% polyco help
```

```
polyco - a script to run TEMPO to generate a polyco.dat file
```

```
usage: polyco psrname -{options}
```

```
psrname    - name of the pulsar as it appears in tztot.dat
```

```
-freq  f - frequency in MHz (def=1410 MHz)  
-nspan n - span of each polyco set in minutes (def=15 min)  
-ncoeff n - number of coefficients in each polyco set (def=9)  
-maxha h - maximum hour angle (def=2 hours)  
-mjd    m - mjd to calculate for (def=today)  
-mjds   s - starting mjd (def=today)  
-mjdf   f - finishing mjd (def=today)  
-site   s - specify site code or alias (def=Arecibo)
```

It is assumed that you have TEMPO installed on your computer so that the `tempo` executable file is in your path, and the TEMPO environment variable set. At Arecibo, a solaris version of TEMPO can be found in `/home/pulsar/bin/tempo` and the TEMPO environment variable should be set to `/home/pulsar/tempo11`.

Running `polyco` is then a matter of giving a pulsar name from the list of ephemerides contained in `$TEMPO/tztot.dat` and the start and stop MJD ranges over which you wish the coefficients to apply. The default is to generate coefficients for use at the time you run TEMPO.

C The EPN Data format

The **E**uropean **P**ulsar **N**etwork (“**EPN**”) is an association of European astrophysical research institutes that co-operate in the subject of pulsar research. The EPN format was developed for the exchange of pulse profiles between different groups of individuals to permit a free interchange of data. The following text was taken from a paper which originally appeared in *Astronomy & Astrophysics Supplement Series* (1998) **128** 541–544 and is included here for quick reference.

Each EPN file consists of one or more blocks. The basic structure of an EPN block is shown in Fig. 6. Each file has a common fixed length *header* followed by a number of individual *data streams* of equal length. The header describes the data, containing information on the pulsar itself, the observing system used to make the observation as well as some free-form information about the processing history of the data. The onus is on the site-specific conversion process to ensure correct conversion to the standardized entries and reference to common catalogues (e.g. the Taylor et al. 1993 catalogue of pulsar parameters). The full list of header variables is given in Tables 1 and 2.

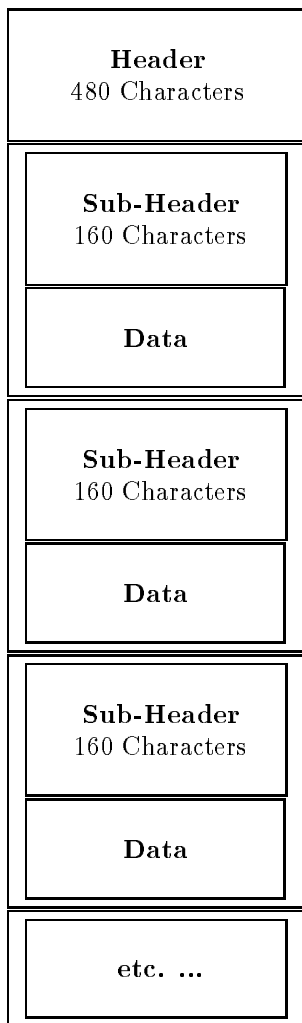


Figure 6: *Schematic representation of an EPN data block.*

The data streams themselves may be outputs of different polarization channels, or individual channels

| Position | Name | Format | Unit | Comment |
|----------|-------------------|------------|---------------------|--|
| 1 | version | A8 | | EPN + version of format (presently EPN05.00) |
| 9 | counter | I4 | | No. of records contained in this data block |
| 13 | history | A68 | | comments and history of the data |
| 81 | jname | A12 | | pulsar jname |
| 93 | name | A12 | | common name |
| 105 | P_{bar} | F16.12 | s | current barycentric period |
| 121 | DM | F8.3 | pc cm ⁻³ | dispersion measure |
| 129 | RM | F10.3 | rad m ⁻² | rotation measure |
| 139 | CATREF | A6 | | pulsar parameter catalogue in use |
| 145 | BIBREF | A8 | | bibliographical reference key (or observer's name) |
| 153 | | 8X | | blank space free for future expansion |
| 161 | α_{2000} | I2,I2,F6.3 | hhmmss | right ascension of source |
| 171 | δ_{2000} | I3,I2,F6.3 | ddmmss | declination of source |
| 182 | telname | A8 | | name of the observing telescope (site) |
| 190 | EPOCH | F10.3 | day | modified Julian date of observation |
| 200 | OPOS | F8.3 | degrees | position angle of telescope |
| 208 | PAFLAG | A1 | | A = absolute polarization position angle, else undefined |
| 209 | TIMFLAG | A1 | | A = absolute time stamps (UTC), else undefined |
| 210 | | 31X | | blank space free for future expansion |
| 241 | x_{tel} | F17.5 | m | topocentric X rectangular position of telescope |
| 258 | y_{tel} | F17.5 | m | topocentric Y rectangular position of telescope |
| 275 | z_{tel} | F17.5 | m | topocentric Z rectangular position of telescope |
| 292 | | 29X | | blank space free for future expansion |
| 321 | CDATE | I2,I2,I4 | d m y | creation/modification date of the dataset |
| 329 | SCANNO | I4 | | sequence number of the observation |
| 333 | SUBSCAN | I4 | | sub-sequence number of the observation |
| 337 | N_{pol} | I2 | | number of polarizations observed |
| 339 | N_{freq} | I4 | | number of frequency bands per polarisation |
| 343 | N_{bin} | I4 | | number of phase bins per frequency (1-9999) |
| 347 | t_{bin} | F12.6 | μs | duration (sampling interval) of a phase bin |
| 359 | t_{res} | F12.6 | μs | temporal resolution of the data |
| 371 | N_{int} | I6 | | number of integrated pulses per block of data |
| 377 | n_{cal} | I4 | t_{bin} | bin number for start of calibration signal |
| 381 | l_{cal} | I4 | t_{bin} | length of calibration signal |
| 385 | FLUXFLAG | A1 | | F = data are flux calibrated in mJy, else undefined |
| 386 | | 15X | | blank space free for future expansion |
| 401 | | 80X | | blank space free for future expansion |

Table 1: A description of the EPN format variables.

(bands) of a filterbank or a combination thereof. In total, there may be N_{freq} data streams of i.e. different frequencies for each polarization. Each data stream starts with a small, fixed length sub-header in front of the actual data values. The number of data streams and their length may vary between different EPN files, but is constant within each file. A character field and an ordinal number is provided for each stream for its identification.

Format Compatible Software

To incorporate the capability to read and write data in this format within existing analysis software, a simple routine exists which can read and write data in this format. In addition, we have written some sample programs which can plot the data and display the header parameters. The software are written

| Position | Name | Format | Unit | Comment |
|---------------------------------|--------------------------|--------|---------------|---|
| 481 | IDfield | A8 | | type of data stream (I,Q,U,V etc.) |
| 489 | n_{band} | I4 | | ordinal number of current stream |
| 493 | n_{avg} | I4 | | number of streams averaged into the current one |
| 497 | f_0 | F12.8 | | effective centre sky frequency of this stream |
| 509 | U_f | A8 | | unit of f_0 |
| 517 | Δf | F12.6 | | effective band width |
| 529 | U_Δ | A8 | | unit of Δf |
| 537 | t_{start} | F17.5 | μs | time of first phase bin w.r.t. EPOCH |
| 554 | | 7X | | blank space free for future expansion |
| 561 | SCALE | E12.6 | | scale factor for the data |
| 573 | OFFSET | E12.6 | | offset to be added to the data |
| 585 | RMS | E12.6 | | rms for this data stream |
| 597 | P_{app} | F16.12 | s | apparent period at time of first phase bin |
| 613 | | 28X | | blank space free for future expansion |
| 641 | Data(1) | I4 | | scaled data for first bin |
| $4(N_{\text{bin}} - 1) + 641$ | Data(N_{bin}) | I4 | | data for last bin of stream |
| $640 + N_{\text{records}} * 80$ | | | | end of first stream |

Table 2: *The sub-header variables within an EPN file*

in *Fortran—77*² and have been packaged into a single UNIX tar file which is freely available via the *Internet*. To down-load the package, log into the anonymous ftp area: **ftp.mpifr-bonn.mpg.de**, with the username **anonymous** using your complete E-mail address as the password. Once logged in, issue the following commands:

```
> cd pub/pulsar
> binary
> get epnsoft.tar.gz
```

Alternatively, the file can be down-loaded from the EPN *Internet* home-page:

<http://www.mpifr-bonn.mpg.de/div/pulsar/epn>

To uncompress and extract the contents of the tar file on a UNIX operating system, issue the commands:

```
% gunzip epnsoft.tar.gz
% tar xvf epnsoft.tar
```

The present package contains some sample data and two example programs — `plotepn` and `viewepn` which plot and view EPN files respectively. The ASCII file **00README** in this packages gives further details of the software and how to use it.

²some simple C utilities are planned for a future version of SIGPROC