PSRSALSA

A Suite of ALgorithms for Statistical Analysis of pulsars

P. Weltevrede

April 20, 2020

Contents

1 The PSRSALSA package **2**
- 1.1 Programs part of the PSRSALSA package . . . 2
- 1.2 Help, pre-processing, header parameters and memory usage . . . 3
- 1.3 Data formats . . . 3
  - 1.3.1 PSRFITS . . . 3
  - 1.3.2 ASCII . . . 4
  - 1.3.3 PSRSALSA specific formats . . . 4
  - 1.3.4 PuMa . . . 4
  - 1.3.5 EPN . . . 4
  - 1.3.6 Sigproc . . . 4

2 Common first steps in data-analysis **5**
- 2.1 The test data-set . . . 5
- 2.2 Explore the header parameters of data (using pheader) . . . 5
- 2.3 Plotting data (using pplot) . . . 6
- 2.4 Manually removing RFI from data . . . 7
- 2.5 Removing the "baseline" (using pmod) . . . 9

3 Pulse energy distribution and null analysis **11**
- 3.1 Determining pulse energies . . . 11
- 3.2 Pulse energy distribution . . . 12
- 3.3 Fitting of energy distributions . . . 12

4 Measuring, plotting and fitting polarization properties **14**
- 4.1 Make a polarized profile . . . 14
- 4.2 Fitting a PA-swing . . . 15
  - 4.2.1 Basic RVM fit . . . 15
  - 4.2.2 Adding a ρ contour . . . 16
  - 4.2.3 Adding a ρ contour collection . . . 16

5 Subpulse modulation and fluctuation analysis **18**
- 5.1 Fluctuation spectra . . . 18
- 5.2 Modulation index . . . 19
- 5.3 Making nice plots . . . 19
- 5.4 Analysis of features in fluctuation spectra . . . 20
- 5.5 Subpulse phase track . . . 21
- 5.6 P_3 folding . . . 22

Chapter 1

The PSRSALSA package

1.1 Programs part of the PSRSALSA package

The following programs are part of the PSRSALSA package and some of the functionality is described in this documentation and examples are provided.

- avrg_bin_files — Program to average binary data, which can be useful in imaging processing as used in section 4.2.3.
- fakeDist — Program to generate a random list of values drawn from various (combinations of) distribution functions (see chapter 3).
- padd — Program to add pulsar data together.
- pconv — Program to convert pulsar data into other types of pulsar data.
- pdist — Program to generate or plot a histogram by binning data. Also a cumulative distribution can be generated (see chapter 3).
- pdistFit — Program to fit a measured distribution with a model distribution using various possible test statistics (see chapter 3).
- penergy — Program for calculating pulse energy statistics (see chapter 3).
- pfold — Program to fold (folded) single pulse data thereby visualising the subpulse modulation cycle (see section 5.6).
- pheader — Program to retrieve header information from files.
- pmod — Program to modify pulsar data in various ways (see chapter 2 and examples throughout this document).
- pplot — Program to plot pulsar data in various ways (see chapter 2 and examples throughout this document).
- ppol — Program to convert Stokes parameters in position angle and linear intensity. The result can be plotted and/or written out (see chapter 4).
- ppolFit — Program to fit the rotating vector model to a position angle swing as generated by ppol (see chapter 4).
- pspec — Program to analyse (folded) single pulse data using mostly Fourier techniques (see chapter 5).
- pspecDetect — Interactive program designed to analyse features in the 2dfs to obtain centroid P_2 and P_3 values and corresponding error-bars (see chapter 5).
- pspecFig — Program to plot some of the pspec output (see chapter 5).
- pstat — Program to perform various statistical tests on input data. This program is used by pdistFit (see chapter 3), but can be useful as a stand-alone program.
- rmsynth — Program to measure rotation measures.

1.2 Help, pre-processing, header parameters and memory usage

If you want a short description of a particular piece of software, including the possible command line options, run the program without any command line options. Example:

```
prompt> pmod
```

The command line options are described in different categories. First come the general options which are the same in different programs part of this package. This include things like "General Input/Output options". In this context general means that these are options that are not just specific to pmod, but different programs use identical options and show an identical description. An example is the "-header" option, which allows you to override header parameters as being interpreted by the software while reading in an input data-set. For example, this allows you to set values of not specified parameters which might be useful when for instance reading in an ascii format with limited header information. Using this option does not change the input file itself, just how the software interprets the data. The option "-headerlist" gives you a list of parameters that can be changed.

Most programs will read in the whole data-set in memory, so you have to be careful using this software on enormous files. Some programs allow you to change the default behavior with the an option like "-memsave", although not all operations might be compatible with this mode. The "General pre-process options" are options which are applied after reading the whole file in memory and allow you to, for instance, rebin the data. This is done before any processing specific to the program is being executed. Again, different programs share the same pre-process command-line options.

Most programs expect an input file and command line options to be specified. The expected command line is in general expected to be the name of the program, followed by the command-line options. The input files are in general expected at the very end of the command. Not all programs allow multiple input files to be specified. An example of a command line is:

```
prompt>  pmod -rebin 256 -ext ar.rebin *.ar
```

This particular command reads in all .ar files, rebins the data to 256 pulse longitude bins and writes out the data as files with the extension ".ar" replaced with ".ar.rebin".

1.3 Data formats

Different file types are supported. Most file types are automatically recognized by the software. Use the -formatlist to get a list of supported formats. If the file format cannot be automatically determined, the -iformat option can be used to let the software know what format the data is in. However, chances are that reading the input file will fail, or the data might be corrupted, because the input format is somewhat different from what is expected. The -oformat option allows you to specify what format to use for the output, which can be different compared to the input. For example the following command will write out data in an ascii format.

```
prompt>  pmod -rebin 256 -oformat ascii -ext ar.rebin *.ar
```

1.3.1 PSRFITS

PSRFITS is supported by the software. Note that the data files written out do not 100% conform with the PSRFITS definition, so proper behavior in other software cannot be guaranteed. Especially timing experiments are not recommended, as this software package is not designed with that aim in mind. For example, the ephemeris/polyco/tempo2 predictor used to fold the data is ignored by this software and will not appear in output data written in this format. Note that usually by default PSRCHIVE does not output PSRFITS files. To generate PSRFITS files you might want to try the following PSRCHIVE command

```
prompt> pam -a PSRFITS -e psrfits inputfile
```

Reading in a PSRFITS file in a PSRSALSA application might result in a system call to vap from PSRCHIVE.

1.3.2 ASCII

This format is identical to the format used by PSRCHIVE when writing out data with the pdv -t command. The header information that can be stored in this format is limited.

1.3.3 PSRSALSA specific formats

Data written out by ppol (an ascii format) are automatically recognized by the software. There are two flavours, one containing more information than the other.

1.3.4 PuMa

This is a binary format used by the no longer existing PuMa backend at the WSRT. Some PSRSALSA specific header information is written out which does not conform with the PuMa format, but it should be backwards compatible.

1.3.5 EPN

The EPN format comes in different versions. Data written out is probably not strictly compatible with a certain fixed version, but it appears to work well for most input files in EPN formats between versions 6.0 and 6.3. Files written out in this format can be read in with PSRCHIVE.

1.3.6 Sigproc

Either folded or non-folded sigproc data in a binary format is supported in the PSRSALSA package, but the data should be written as 8-bit unsigned bytes or 32-bit floating points. This can either be raw filterbank data, data dedispersed with sigproc's dedisperse or folded data using sigproc's "fold -bin -d 1".

There is also a sigproc ascii format which is handled as well. This is for folded data containing multiple subints (sigproc's "fold -d 1"). The ascii format is currently not automatically recognized, so it relies on the user specifying the -iformat option.

Chapter 2

Common first steps in data-analysis

2.1 The test data-set

You can try out the commands in this and other chapters by running the commands using the file tutorial1.data. This file should have been provided together with this document. This file contains an artificial pulsar signal, containing drifting subpulses and nulls. In addition there is some RFI and a gradient in the background noise which should be removed prior to any analysis. How to do this is explained in this chapter.

In addition, the result of the cleaning process described in Section 2.4 can be obtained straight away with the command

```
prompt> pmod -ext clean.zero -zap "105 106" -fzap "0 0" -fzap "2 2" \
             -fzap "6 6" -fzap "8 8" -fzap "14 14" tutorial1.data
prompt> pmod -ext clean.removed -zap "105 106" -fzap "0 0" -fzap "2 2" \
             -fzap "6 6" -fzap "8 8" -fzap "14 14" -remove tutorial1.data
```

This should have made a file called tutorial1.clean.zero and tutorial1.clean.removed. The result of the baseline removal (tutorial1.clean.debase.gg) can be reproduced by

```
prompt> pmod -debase -onpulse "121 178" -debase -device /null tutorial1.clean.removed
```

2.2 Explore the header parameters of data (using pheader)

Let us first explore what type of data we have generated. Header parameters can be explored with the program pheader. The following command gives a summary of the header parameters:

```
prompt> pheader tutorial1.data
```

The information shown is very similar to what you get if you run other tools, like pmod, with the vebose option (-v). You can also get a list of parameters you're interested in by running for example:

```
prompt> pheader -c "nbin nsub nfreq npol" tutorial1.data
```

This should tells us that there are 300 bins in pulse phase, 1024 sub-integrations, 16 frequency channels and 4 polarization channels. To find out how the different parameters are called you can include in the -c option you can run

```
prompt> pheader -H
```

To find out what command line option there are, run the program without any command line options:

```
prompt> pheader
```

2.3 Plotting data (using pplot)

To make a plot of the data you can use the program pplot. So for example if you run:

```
prompt> pplot tutorial1.data
```

This shows all data in the file. In this case there are 16 frequency channels plotted (on the vertical axis) and there are 1024 subintegrations (on the horizontal axis). By default it shows the first polarization channel, which for this data-set means Stokes I. There are a number of things to see in the data, which become more obvious when plotting the data differently. The following command shows a similar plot, now with all the subints summed together.

```
prompt> pplot -TSCR tutorial1.data
```

The pulsar signal now can be weakly seen in the middle of the plot and note that 5 frequency channels look somewhat worse than the others. We will look into how to remove RFI in section 2.4. To look to the evolution of the signal over time you can do

```
prompt> pplot -FSCR tutorial1.data
```

Now all the frequency channels are added together first. You should see that subintegration ~ 100 looks particularly bad. The pulsar signal itself is not really visible in this plot. However, the pulsar signal is present, which can be seen by forming the pulse profile.

```
prompt> pplot -FSCR -TSCR tutorial1.data
```

Notice that the noise level is not zero. This means there is a "baseline" present, which will need to be removed before any analysis. In section 2.5 it is discussed how you can try to get rid of it. For now, this effect can be suppressed by running

```
prompt> pplot -debase -FSCR -TSCR tutorial1.data
```

This subtracts the average for each channel/subint independently. The effects of the baseline should be largely disappeared. Notice that the noise level has become slightly negative, which is because the average of noise plus pulsar signal has been removed from the data. To explore the data in more detail you might want to run pplot in interactive mode:

```
prompt> pplot -ia -v tutorial1.data
```

The option -v means "verbose", implying that more information is sent to the terminal. In general this is a good idea, since it gives you a better idea of what steps are done by the software, hence it allows you to so if (and why) things go wrong. For a start it will show you the header information, allowing you to assess in the data is interpreted correctly. Once you start this command the terminal will ask you to provide key presses to change what is plotted. Press ? (in the terminal) for a list of commands. For example you can type in the terminal:

```
l 100 110
```

From the help you can see that this specifies the left/right bounds, i.e. the plotted x-range. You can should see there are two subintegrations which look particularly bad and it will be explained in section 2.4 how to remove these subintegrations. Lets now zoom in a bit more.

```
l 300 301
```

So this the command you entered zooms in on a single subintegration somewhere in the middle of the data-set. The pulsar signal is not obvious because of the 5 corrupted frequency channels. To see the pulsar clearer, lets zoom in the vertical range to avoid the bad channels.

```
v 9 13
```

This shows part of the vertical range (without much RFI) the pulse should have appeared in the middle of the subint. You can experiment with the b and y to change what units are used for the x and y axis. With the M option you can specify the type of plot. So for example M followed by l results in a line plot. If you do this, you see horizontal flat lines, which in this case is because the different channels are separated by 1 unit on the vertical axis (if the y-axis is in bin nr mode rather than in MHz), while the amplitude of the signal is much less than 1 unit, making the curve appear to be flat. What you could do is press a, which auto-scales the data and should allow you to show the data as non-flat lines. Another option available is the p option. If you press p it cycles back to previous settings. You can press p multiple times to go back to previous plots. To cycle through polarizations you can use P. If you look carefully you can see that the polarization channels (counting from zero) 1 and 2 (Stokes Q and U) show evidence for Faraday rotation. To quit press q.

There are more options to play with in interactive mode, and many more command-line options to change the layout of the plot (type pplot without command-line arguments to get a list of options). For example you could try out the following command:

```
prompt> pplot -showwedge -showtop tutorial1.data
```

The top plot shows clearly the RFI in the two subints as identified earlier. In section 2.5 it is discussed how you can try to get rid of it.

2.4 Manually removing RFI from data

To identify RFI, you probably want to show a plot of frequency vs subint using pplot in interactive mode. If the file is very large, consider making a file with a reduced time resolution first, which is still good enough to identify the frequencies you would like to remove. If your data contains polarization channels written out as coherency parameters rather than Stokes parameters, you might want to use the -stokes option. The -debase option will remove the baseline, i.e. the non-zero noise level we noticed in section 2.3. This is especially useful if the level of the background noise is varying significantly during the observation. Since we didn't define an onpulse region (i.e. a pulse longitude range at which the pulsed emission occurs), the -debase option will make the average of the noise + pulsar signal zero, making the baseline negative. This doesn't matter here, since we just want to take out the largest variations (especially those which are caused by strong RFI), which otherwise might it hard to see the data. See section 2.5 how to properly remove the baseline, although this is not important for what we're trying to achieve here. To improve the quality of the fake data-set we start by running:

```
prompt> pplot -v -debase -ia tutorial1.data
```

The strategy is to first identify badly affected subints, before identifying bad frequency channels which will be zapped throughout the whole data-set. In general it is some sort of iterative process to identify which combination of zapped subints/channels results in the most optimum result. Some commands that might be useful:

- z - Toggle between interactive channel/subint zapping
- Z - Start zapping, see terminal for help about how this is done.
- v - 'v' is followed by two numbers, this allows you to zoom in the vertical range. To zoom out, you can use the range 0 to a very large number.
- n - Go to next block in the vertical range. This allows you to step through the data.
- W - Write out a zap file containing the zapped channel numbers/subints (counting from zero)
- l 0 1 - Zoom in on first subint
- ,/. - Move to left, right when zoomed in
- q - quit

Lets start with removing the bad subintegrations we identified earlier. Start with zooming in to clearly see which of the subints are effected.

```
l 103 109
```

Now make sure you're in subint zapping mode by pressing the 'z' key an appropriate number of times in the terminal and by watching what is reported on the terminal. The terminal should tell you if you're in subint or frequency channel zap mode. Now press 'Z' to remove a few subints and experiment with taking out a range as well. Note that the result is only visible after you quit the 'Z' option by pressing in the plot window a different key. Note that when you select a range, the 'Z' option is quit as well. The 'Z' option does not remove any data from the input data-file. It just removes the data from the plot. When you're done, write out a list of subints with the 'W' option, which we can use to apply these zapping choices to the data. This should have created a file called tutorial1.subint.zap.

In general you probably would like to start with removing some subints, if they are badly affected by RFI, so we end up with a cleaner data-set in which affected frequency channels can be easier identified. So let's apply the selection of subints to the data. This can be done using the following command:

```
prompt> pmod -ext zap1 -zapfile tutorial1.subint.zap tutorial1.data
```

This will write out a new file with the selected subints set to zero intensity (called tutorial1.zap1). Note that the number of subints in your data-set is therefore not changed. You probably want to see the result by using pplot. A second file written out is tutorial1.zapped.gg, which contains all the subints which were zapped.
Depending on what you want to achieve, it might be desirable to remove the subints from the data-set (reducing the number of subints in your file). This is achieved by adding the -remove flag in pmod.

```
prompt> pmod -ext zap2 -zapfile tutorial1.subint.zap -remove tutorial1.data
```

The new tutorial1.zap2 should not have any zero-weighted subints and therefore have less subints than the original data-set.

After removing the worst affected subintegrations you want to repeat the procedure, but now identifying the frequency channels you want to remove. Make a plot with pplot in interactive mode using the file you just created, but now adding the -TSCR option to add all subintegrations together first. This should enable you to see weaker RFI than without the -TSCR option. Since we sum the two subints together, it should make no difference if we use the tutorial.zap1 or tutorial.zap2 file

```
prompt> pplot -v -debase -TSCR -ia tutorial1.zap1
```

After making sure you're in vector zap mode, zap the 5 bad channels and write out the list of "vectors" with the 'W' option. This should have created a file called tutorial1.freq.zap. On the command line you could check its contents by typing

```
prompt> cat tutorial1.freq.zap
```

It should contain the numbers 0, 2, 6, 8 and 14. A similar pmod command line as used before can be used to remove these channels from the data, but use -fzapfile rather than -zapfile to zap frequency channels. We apply the same zapping (by zero-weighting the frequency channels) on the two files previously generated: one the zapped subints removed, the other with the subints set to zero.

```
prompt> pmod -fzapfile tutorial1.freq.zap -ext clean.zero tutorial1.zap1
prompt> pmod -fzapfile tutorial1.freq.zap -ext clean.removed tutorial1.zap2
```

Note that in both cases the zapped frequency channels are weighted to zero. If you would remove the channels, the frequency labeling will no longer be correct, hence things like dedispersion will go wrong. You can confirm with pplot that the resulting tutorial1.clean.gg indeed has the desired channels removed. If we make a plot showing the time variability of the signal after summing all frequency channels

```
prompt> pplot -FSCR tutorial1.clean.zero
```

The dominating effect to be seen is a linearly increasing signal from bottom to top. This is a slope in the background ("baseline") which is slowly increasing throughout the observation. In section 2.5 it is discussed how you can try to get rid of it. Since there is a positive baseline, the two subints which were set to zero have a lower intensity, and they appear as black lines. For now, this effect of the baseline can be suppressed by running

```
prompt> pplot -debase -FSCR tutorial1.clean.zero
```

The effect of the varying baseline should have been removed and the pulsar signal should be very clear. You should see the drifting subpulses of the pulsar and you should see two episodes where the pulsar is switched off ("nulls"). Notice that the noise during the nulls look different. This is because the noise level is on average zero during the nulls, but negative when there is the pulsar signal, since the average of noise plus signal is forced to be zero.

As stated above, you might go through this process multiple times, identifying the strongest RFI first, which after zapping allows you to see weaker RFI you want to remove as well. Remember that removing RFI implies removal of signal as well. So removing all data which contains RFI might not result in an optimal S/N. Depending on the quality of your data and what you would like to achieve, it might be possible to combine/skip some steps, especially for smaller data-sets.

2.5 Removing the "baseline" (using pmod)

Here it is assumed that you have the files tutorial1.clean.removed and tutorial1.clean.zero made following the instructions in section 2.4. If not, you can make it directly from tutorial1.data with the commands given in section 2.1.

In general there is a baseline in your data you will have to remove, i.e. setting the average noise level to zero. If you plot the data without doing anything, the noise is most likely not zero. You can check this by running pplot to produce a pulse profile

```
prompt> pplot -FSCR -TSCR tutorial1.clean.removed
```

Two pre-process options are selected, which in this case add all frequency channels and all subintegrations together. Pre-process means that these options are applied directly after reading in the data, but before the program gets access to the data. Many pre-process option are available via the same command line options in different programs. As you can see, the noise is not zero on average (the noise level is positive in this case) and there is also a very small slope in the baseline. The reason is that a linearly increasing slope has been added to this artificial data-set. This is more obvious if you run:

```
prompt> pplot -FSCR tutorial1.clean.removed
```

Now only frequency channels are summed, and the plot shows subsequent subintegrations on top of each other. In the artificial data "drifting subpulses" are generated, and there is a baseline which is increasing over time. To remove the baseline you can use pmod.

```
prompt> pmod -debase tutorial1.clean.removed
```

The program asks you to specify a pgplot device, so for instance /xs can be used. The program asks you to identify the onpulse region in order to estimate the noise level. So all bins which are *not selected* are used to determine the baseline. Therefore it is crucial that everything that is not selected, does not contain any signal. So it is better to make the selected regions too wide rather than too narrow. In this particular case, maybe select a region starting at bin ~ 120 up to bin ~ 180. You can select more than one region if desired: only not selected regions are used in the baseline calculation. After pressing S inside the pgplot window, the baseline is subtracted. A plot is shown which shows the baseline value (average value of the noise) found as function of subintegration number. The plot is for the first frequency channel and first polarization channel only, but the actual baseline values are determined for each frequency and polarization channel separately. The produced plot indeed shows that there is a linearly increasing baseline over time.

The accuracy with which the baseline can determined depends on the timescale with which the baseline is varying and the number of off-pulse bins given that the average of the off-pulse region is determined by the rms of the white noise times the square-root of the number of off-pulse bins. For some pulsars there might be only a very small nr of off-pulse bins if the duty cycle of the pulsar is very large. In that case, you might want to experiment with the -debase_length option, which allows you to use a "running mean" calculated using multiple subintegrations rather than determining the average on a single subint basis. See help (just type pmod on the terminal) for more available options.

In this particular case the default process is good enough for our purposes here. You can check the result by plotting a profile made from the output.

```
prompt> pplot -TSCR -FSCR tutorial1.clean.debase.gg
```

The average of the noise is indeed zero. There is still a slight slope present, since we just subtracted an average value of the baseline for each subintegration. By making a plot of the individual subintegrations you can see that most of the baseline is no longer there and the drifting subpulses stand out more clearly.

```
prompt> pplot -FSCR tutorial1.clean.debase.gg
```

The strange appearance of the noise during the "nulls" has also been resolved.

In principle, this process should be done on the two versions of the cleaned data-set (removed/zero-weighted subints), as different types of analysis prefer one over the other. In the other chapters we will only make use of tutorial1.clean.debase.gg.

Chapter 3

Pulse energy distribution and null analysis

The following commands in this chapter assume you have a tutorial1.clean.debase.gg, as made in chapter 2. *The baseline should have been removed before doing any analysis.* If you don't know what this means, read chapter 2 first. If you want to analyse pulse energy distribution and you had to get rid of bad subintegrations (single pulses), you probably want to remove them from the pulse stack rather than setting those pulses to zero. Otherwise you will, for instance, in a pulse energy distribution get an excess of pulse energies at exactly zero which is completely artificial without any physical meaning.

The first step we do here is to add noise to the data, thereby more typical of what often can be expected in real data.

```
prompt> pmod -ext weak -addnoise 5e-6 tutorial1.clean.debase.gg
```

3.1 Determining pulse energies

The term "pulse energy" is used to indicate the integrated intensity over a certain pulse longitude range. If the baseline is not removed, all values are offset. If the baseline is variable the determined values will be smeared out. So removing the baseline is important. The program penergy can be used to get energy values for each individual subintegration.

```
prompt> penergy -v -FSCR tutorial1.clean.debase.weak
```

Here the -v option results in more information in the terminal, which helps you understand what is being done. The -FSCR pre-process option ensures that all frequency channels are summed before calculating the pulse energies. By default only the first polarization channel will be analysed (which is total intensity in this case). Since no onpulse regions were selection on the command line, it asks you to specify a pgplot device to manually make the selection. When selecting multiple onpulse regions, only the first is used to calculate the onpulse statistics. Non-selected regions are used for the off-pulse statistics. This allows you for instance to select a first onpulse region which is relatively narrow to ensure that all included bins have sufficient S/N, while you could select a second onpulse region which is a bit too broad to ensure that no signal is included in the noise statistics. It also allows you to consider the pulse energy statistics of part of the profile, for instance the main pulse while not including an interpulse signal in the noise statistics. The choice you made result in a suggestion of how to repeat the command with an identical selection of the on-pulse regions. For instance:

```
prompt> penergy -v -FSCR -onpulse "127 174" -onpulse "121 181" tutorial1.clean.debase.weak
```

The output is an ascii file called tutorial1.clean.debase.weak.en, which is in ascii format (see section 1.3.2). At the end of the file a short description of the different columns and relevant on-pulse selection that have been made can be found. The columns are:

1. Polarization channel number
2. Frequency channel number

3. Subintegration number
4. On-pulse peak intensity, which is the maximum intensities within the first selected onpulse region.
5. Off-pulse peak intensity, which is the maximum intensities within the not selected regions.
6. On-pulse energy, which is the sum of the intensities within the first selected onpulse region.
7. Off-pulse energy, which is the sum of the intensities of the not-selected regions.
8. on-pulse RMS, which is rms within the first selected onpulse region.
9. off-pulse RMS, which is rms within the not selected regions.
10. S/N, which is the summed on-pulse intensity, divided by the expected noise resulting from adding the given number of on-pulse bins together, each containing a rms equal to that of the offpulse region.

When measuring pulse intensities, be aware that interstellar scintillation can affect your measurements.

3.2 Pulse energy distribution

Another thing you might want to do is to make a distribution of the on-pulse energies. The program pdist can be used for this purpose, which is fact can make a histogram of any list of numbers. If you use the -plot option a plot is generated rather than an output file. Run pdist without command line options to get a list of possible parameters. When you run

```
prompt> pdist -v -plot -n 50 tutorial1.clean.debase.weak.en
```

pdist will ask you for a plotting device (/xs for example). A plot should be produced with 50 bins. Other things can be specified on the command-line including for instance the specification of a bin-width rather than the number of bins as is used in the next command we use. The histogram shows two peaks: one centered at zero, which are the nulls, and one at a positive energy, which are the active pulses. The -v option allows you to see more information, including for instance the bin-width that the software has chosen. Instead of a graph the histogram can also be outputted to an ascii file.

```
prompt> pdist -dx 5e-05 tutorial1.clean.debase.weak.en
```

The above command outputs the histogram to the file tutorial1.clean.debase.weak.en.hist. We can make a similar distribution for what can be expected from the noise. Notice that in the command below the first "on-pulse" region has an equal width to what was used above, but now located in the off-pulse region. Hence the integrated energies are those which can be expected from the noise within a region with a width equal to what we used for the on-pulse region.

```
prompt> penergy -v -ext off.en -FSCR -onpulse "27 74" -onpulse "121 181" \
                tutorial1.clean.debase.weak
prompt> pdist -dx 5e-05 tutorial1.clean.debase.weak.off.en
```

Note that the histograms are written out with the same bin widths. The two histograms can be plotted in gnuplot

```
gnuplot> plot "tutorial1.clean.debase.weak.en.hist" with histeps, \
              "tutorial1.clean.debase.weak.off.en.hist" with histeps
```

3.3 Fitting of energy distributions

The program pdistFit can be used to fit energy distributions (or any distribution function you might have) with various analytic distributions. Noise, either white noise or the observed noise distribution, can be taken into account, as well as a nulling fraction. The aim is to find a mathematical description of a distribution which, once convolved with the noise distribution, matches the observed distribution. Run pdistFit without any command-line options to see what possibilities there are.

What follows is a particular example of how you can fit the distribution. Here it is chosen to use the observed off-pulse distribution we generated earlier as an input in the fitting process. This distribution should be available as a list of numbers, which can be extracted from the earlier made file with the following Linux commands (If you copy/paste from this document, you might have to replace the single quotes with proper single quotes in the awk bit).

```
prompt> grep -v "File" tutorial1.clean.debase.weak.off.en | grep -v "#" \
            | awk '{print $6}' > noise.dist
```

The command quoted below which can be used to fit the pulse energy distribution depends on the average of the measured energy distribution. A way to find the average value of the observed distribution is to make a distribution with pdist in verbose mode, such as the command

```
prompt> pdist -v -n 50 -plot tutorial1.clean.debase.weak.en
```

The output should tell you that the average is 4.752696×10^{-4}.

The pulse energy distribution can be fitted with the following command.

```
prompt> pdistFit -v -plotcdf -lognorm "-7 1 0.03 0.01" -null "4.752696e-04 0 1" \
                 -noisefile noise.dist tutorial1.clean.debase.weak.en
```

Run pdistFit without command-line parameters to see what the options mean. In this particular command the -v option ensures that some useful information is given in the terminal, -plotcdf means that a plot with the cummulative distribution (fit and data) will be shown after the fitting is completed. The pulse-energy distribution is assumed to be a lognormal distribution and initial parameters and stepsizes are provided. In addition, nulls are added to ensure that the modelled distribution has the same average as the observed average. Although this is a parameter we can fit for, in this command the average is used as a fixed parameter. The modelled noise distribution is convolved with the observed noise distribution before being compared with the observed energy distribution. Since no particular statistical test has been specified on the command-line, the optimisation is based on a χ^2 test of the cummulative distribution.

The resulting plot should show that the fitted distribution (white) and the measured distribution (red) are relatively similar, indicating that the measured distribution can be described relatively well with a combination of a lognormal distribution, nulls and the noise distribution. If you run the same command again you can notice that the reported numbers are different. This is because the fitting process runs for each itteration the program fakeDist, which randomly picks values from the model distribution. This distribution is compared to the measured distribution. Since the model distribution is based on a finite number of samples (which can be specified on the command line), the paramters will be affected by this uncertainty. The comparison is done by calling pstat (type pstat to get a better idea what various statistical tests do, including the one used by pdistFit). To get a better idea of what is happening in the background, you can consider using the -debug option, which gives you more information than -v.

The terminal should (if -v was used) report how to generate a model distribution with fakeDist. As an exercise you can try to overplot a histogram of this distribution with the observed distribution (using pdist). In addition the fit parameters are reported and a KS-test (Kolmogorov-Smirnov test).

Chapter 4

Measuring, plotting and fitting polarization properties

You should have a cleaned dataset in the sense that the frequency channels with bad RFI are already removed. If there are subintegrations you want to remove, either removing them completely (pmod -remove) them to zero is fine if you want to make a profile with a PA-swing. If you want to study the statistics of individual subintegrations, you're probably want to remove them. The following commands in this chapter assume you have a tutorial1.clean.debase.gg, as made in chapter 2. *The baseline should have been removed before doing any analysis.* If you don't know what this means, read chapter 2 first.

Polarization products are derived as follows by PPOL.

- The error on Stokes I and V samples is taken to be the off-pulse rms, and is therefore the same for all samples in a given sub-integration/channel number.
- For the linear polarization intensity L, by default the Wardle and Kronberg 1974 (ApJ 194, 249) de-bias is applied, although it is not assumed that the rms on each Stokes parameter is equal. This means that

$$L = \sqrt{Q^2 + U^2}\sqrt{1 - \left(\frac{\sigma}{\sqrt{Q^2 + U^2}}\right)^2}. \tag{4.1}$$

  If $\sigma = \sqrt{(\sigma_Q^2 + \sigma_U^2)/2} > \sqrt{Q^2 + U^2}$, then L is taken to be zero. It is possible to disable this de-bias (use -noLdebias), or subtract the median off-pulse $\sqrt{Q^2 + U^2}$ instead (use -medianLdebias). The error on L is taken to be the rms of the off-pulse L, which is therefore slightly conservative.
- For the error in PA Gaussian error propagation on the Stokes parameters is applied, without assuming that the rms of all Sokes parameters are equal. By default a threshold in the significance of L is applied, where the ratio of L and the error (as defined above) need to exceed 3. The threshold can be changed with the -sigma option.
- $P = \sqrt{Q^2 + U^2 + V^2}$ can be computed by using the -extendedpol option. The median offpulse P is subtracted. This is far from ideal, but so far a better alternative (such as done for L) has not been implemented. The error is taken to be the offpulse rms (before the any bias has been subtracted).

4.1 Make a polarized profile

The position angle (PA) and linear intensity (L) etc. can be calculated using ppol. To get a polarized profile you can run:

```
prompt> ppol -TSCR -FSCR tutorial1.clean.debase.gg
```

If you run the above command ppol will ask you to specify a pgplot device, which allows you to select on-pulse regions. This information is only used to determine the *off-pulse* statistics. The non-selected regions are used for the off-pulse statistics. So like with pmod -debase, in general it is therefore better to

have the region a bit too broad to ensure that no signal is included in the noise statistics. More than one region can be selected, for instance to deal with an interpulse. The choice you made result in a suggestion of how to repeat the command with an identical selection of the on-pulse regions.

After pressing S, you can specify an additional pgplot device, which shows you the polarized profile and PA points. The white, red and green lines are Stokes I, total amount of linear polarization and Stokes V. The resulting plot shows that the signal is not 100% linearly polarized. In fact, this data is not properly de-Faraday rotated. Another thing that could potentially go wrong is confusion between Stokes parameters and coherency parametes. Here the problem is that the rotation measure (RM) is set to zero in the header, while there is Faraday rotation present in the data. To fix the problem we can overwrite header paramters with the -header option (the input file will not be changed). See the help of ppol for more information of these and other options. ppol should have reported a warning about the RM being zero, so make sure you understand all warnings you encounter. In general it is always a good idea to run all software with the -v option to get a better idea what is being done to the data, which allows you to spot where things might go wrong. To fix this particular issue run:

```
prompt> ppol -v -onpulse "125 177" -header "rm 100" -TSCR -FSCR tutorial1.clean.debase.gg
```

The profile now should be fully linearly polarized. Note that each PA-point is shown twice, as by default the full PA range is 360 degrees rather than 180. It is probably good to check if the linear polarization is correctly de-Faraday rotated. You could run something like the following command to check if Stokes Q is independent of frequency.

```
prompt> pplot -ia -stokes -defarad -TSCR tutorial1.clean.debase.gg
```

The above command runs pplot in interactive mode. By pressing P you can cycle through the Stokes parameters. You should see obvious Faraday rotation in Stoke Q and U, despite having requested for de-Faraday rotation. Inclusion of the above used -header option should get rid of the Faraday rotation.

ppol has a number of options to make nice looking figures, which you can output for instance as postscript files. By using the option -ofile or -ext you can generate an ascii file containing the polarized profile. Run the following command to write out a polarized profile with PA points.

```
prompt> ppol -v -onpulse "125 177" -header "rm 100" -TSCR -FSCR \
             -ext paswing tutorial1.clean.debase.gg
```

4.2 Fitting a PA-swing

4.2.1 Basic RVM fit

Here we will fit the Rotating Vector Model (RVM) to the polarized profile with PA points we produced in Section 4.1. Run ppolFit to see what the unexplained command-line options in the examples below do, and what other options are available.

A good first step in obtaining a RVM solution is to take a coarse α-β grid (the -g option) to get a feel for the parameter space. After this step we can refine things on the command line. The -l option can be used to make an initial guess for the location of the inflection point of the RVM. Since the pulse is centered at pulse longitude 180 this seems a reasonable first step. The initial step size in longitude will be 5 degrees with the following command.

```
prompt> ppolFit -g "50 50" -l "180 5" -showwedge -best tutorial1.clean.debase.paswing
```

This should result in a reduced-χ^2 map and a model PA-swing with the observed PA-points superimposed. The cross in the reduced-χ^2 map indicates where the best solution was found, which is the model PA-swing in the other plot. The reduced-χ^2 surface is in this case located on the positive β side, so the grid can be constructed more efficiently by specifying a β range with the -B option. Looking at the output on the terminal, the inflection point (l0) was indeed found close to 180 degrees. However, the pa0 value is something like 110 degrees, so adding the -pa option will result in better results than the default assumption made by the program.

```
prompt> ppolFit -g "50 50" -l "180 5" -pa0 "110 5" -B "-5 20" \
                -showwedge -best tutorial1.clean.debase.paswing
```

Clearly this very high signal-to-noise data-set which furthermore has a perfect RVM PA-swing results in a very small allowed range in the reduced-χ^2 map. Zooming in further, this time also in alpha (-A option) results in a resolved reduced-χ^2 map. A higher resolution map has been specified as well.

```
prompt> ppolFit -g "200 200" -l "180 5" -pa0 "110 5" -A "50 90" -B "8 11.5" \
                -showwedge -best tutorial1.clean.debase.paswing
```

Note that if -best is removed from the command line, the program will stay in interactive mode once being run. Press 'h' inside the pgplot window to get a list of available commands in interactive mode.

4.2.2 Adding a ρ contour

Often, the RVM fit is poorly constrained in α-β space and a typical banana-shape is seen. Here only a small part of the banana (which starts at $(\alpha = 0, \beta = 0)$ and extends to $(\alpha = 180deg, \beta = 0)$). Further constraints can often be obtained by making assumptions and measuring the pulse width.

Start by measuring the pulse width. This can be guessed by using pplot in interactive mode (toggle 'b' such that you see the horizontal axis in degrees and zoom in with 'l')

```
prompt> pplot -ia tutorial1.clean.debase.paswing
```

The pulse-width can be defined in different ways (width at different intensity levels, different ways to deal with asymmetries), but in this example the pulse width is taken to be 50 degrees.

Secondly a opening angle for the beam (assuming it is circular) has to be chosen. Looking at the output of ppolFit, the inflection point (l0) is at pulse longitude ~ 183, so ~ 3 degrees after the symmetry point (peak) of the profile. This can be interpreted in terms of the "BCW effect". Given the pulse period and assuming the emission is from the open field line region of a (static) dipole this can be used to obtain a predicted half opening angle of the beam. The combination of this prediction and the measured pulse width leads to an additional constraint on α and β.

```
prompt> ppolFit -g "200 200" -l "180 5" -pa0 "110 5" -A "50 90" -B "8 11.5" -wmp 50 \
                -showwedge tutorial1.clean.debase.paswing
```

Note that by taking out the -best option an interactive mode is entered. Press 'r' in the pgplot window to enable the drawing of rho contours. Nothing should appear, which is because by default the contours are drawn too far apart to see within the narrow range which we selected in α-β space. Press 'n' in the pgplot window and enter the number of requested contours on the terminal. Select 180 contours (those for the interpulse are not used, so set it to zero, or anything else). This should result in contours separated by 1 degree in ρ. We can see that if the predicted half opening angle ρ is in the range 23-27 degrees, it would be consistent with the RVM fit result.

4.2.3 Adding a ρ contour collection

Continuing with the fit made in the previous subsection, lets now consider the case where we have a range of allowed ρ and pulse widths. This should correspond to a region in parameter space which is allowed. Working out what region is parameter space is allowed for a range of contours will be at best complicated, and potentially impossible to do analytically. Therefore to generate the area allowed in α-β space it is much easier to simply generate contours corresponding to a finely sampled range of acceptable parameters. If it is sampled finely enough, it will resemble a continuous region. So each combination of allowed values result in a contour, eventually resulting in a region covered by this collection of contours. Unfortunately pgplot does not allow transparency, so the contours will block the χ-square surface. Below it is explained how to resolve this, but lets start with defining which collection of contours to consider.

First of all, a file needs to be generated defining what contours need to be generated. So for instance by running the following command

```
prompt> awk '{for(x=24; x<=26;x+=0.01) printf("50 %f\n", x)}' > contours.list
```

followed by a press of 'return' and a 'ctrl-d' should result in a file contours.list. Each line will define a contour, all corresponding to a pulse width of 50 degrees in this case, and ρ being between 24 and 26 degrees. Start ppolFit once more and note that the -wmp option is no longer required since we're going to use the file as input.

```
prompt> ppolFit -g "200 200" -l "180 5" -pa0 "110 5" -A "50 90" -B "8 11.5" \
                -showwedge tutorial1.clean.debase.paswing
```

Press 'R' in the pgplot window to specify the input file with contours to be used. Type the file name (contours.list) in the terminal. We consider here the main pulse only, so type MP. Say n to disable plotting the found cross-points between the contours and the χ^2 surface. Make sure to toggle 'r' to show the contours. You might want to take out the labeling of the contours 't'.

To make the region appear as a transparent region you need to first write out two separate files: one being a plot of the χ^2 surface only and one of the region covered by the ρ contours. Continuing from the command above, we can first disable the plotting of the ρ contours by pressing 'r' once more. To avoid the cross indicating the best fit to appear in the generated plot, press 'C' twice. To write out the plot to a file, press 'w'. Say yes to the question if you want to specify the plot device yourself. For instance "chi2surface.ps/cps" will write out the plot as a colour postscript file named chi2surface.ps. You probably want to invert the gray-scale, so that white correspond to a high χ^2. Next the program asks you to provide an α and β value. These values are used to make a file called pa.ps, which contains the observed PA-swing with a RVM fit corresponding to the specified values. Since this graph is not used for what we're doing here, so what values of α and β you choose is irrelevant. After writing out the graphs to a postscript file, you might want to check in a different terminal if the generated postscript file looks like the χ^2 grid you generated.

Now we need another postscript file with the contours. The final result looks visually better by leaving the gray-scale surface in this plot[1] Press 'r' to enable the drawing of the ρ contours. Now proceed with writing out the a new postscript file with the 'w' option. You could, for instance, now use the device "allowedregion.ps/cps". We now have everything we need, so you can quit by pressing 'q'.

To combine the two images, you can use the following commands. Here avrg_bin_files is part of the PSRSALSA package, while convert and identify are part of the ImageMagick.

```
prompt> identify -density 200 allowedregion.ps chi2surface.ps
prompt> convert -depth 8 -density 200 allowedregion.ps allowedregion.rgb
prompt> convert -depth 8 -density 200 chi2surface.ps   chi2surface.rgb
prompt> avrg_bin_files allowedregion.rgb chi2surface.rgb combined.rgb
prompt> convert -depth 8 -size "1447x1464" combined.rgb combined.ps
```

Here the first command identifies the resolution of the two plots (here assumed to be 1447x1464). The second and third line converts the postscript files in a simple headerless binary format. The avrg_bin_files command combines the two images by taking the average, giving the impression of transparency. The final command converts the image back in a postscript file. The -density option ensures that the resulting postscript file has a high resolution. You can experiment in the last step with converting it into a postscript with a .ps2 or .ps3 extension, which might reduce the file size.

[1] In principle, by pressing 'g' the grayscale χ^2 surface disappears. To make the χ^2 contours disappear, press 'c' and type "0" in the terminal. However, the final result will look better by not doing this.

Chapter 5

Subpulse modulation and fluctuation analysis

You should have a cleaned dataset in the sense that the frequency channels with bad RFI are already removed. If there are subintegrations (presumably single pulses) you want to remove, there are two options: either you remove them completely (pmod -remove) or you set them to zero. Which one is best depends on what aspect of subpulse modulation you're investigating. For fluctuation spectra you probably want to set the subintegrations to zero. This ensures the periodicities (for instance P_3) are less distorted. However, extra modulation power will be introduced since by setting pulses to zero, intensity fluctuation has been artificially introduced. Any periodicities in the pulse numbers set to zero will end up in the fluctuation spectra. At the other hand the calculation of the modulation index is independent of periodicities in the data. In fact, the order of the pulses is irrelevant. This means that removing the affected subintegrations will be better, since this ensures the modulation index in not artifically increased by intruducing modulation power.

The following commands in this chapter assume you have a tutorial1.clean.debase.gg, as made in chapter 2. *The baseline should have been removed before doing any analysis.* If you don't know what this means, read chapter 2 first. In this data two pulses were removed. Since they happened during a null, this process did not impacted the periodicity of the drifting subpulses. Therefore for all processes discussed in this chapter, the data with removed subintegrations can be used.
You probably want to consider Stokes I and sum all frequency channels first to end up with a pulse-stack. This is something we can do first, which will shorten the commands used below.

```
prompt> pmod -stokes -polselect 0 -FSCR -output tutorial1.pulsestack tutorial1.clean.debase.gg
```

The inclusion of -stokes was not necessary, since the data already was written in Stokes parameters. The above command should have written out the file tutorial1.pulsestack. This dataset has very obvious drifting subpulses, as you can see with

```
prompt> pplot tutorial1.pulsestack
```

5.1 Fluctuation spectra

Various type of fluctuation spectra can be calculated using pspec, which can also do various other operations, see help (run pspec without command line options). Here we will generate a longitude resolved fluctuation spectrum (lrfs) and a two-dimensional fluctuation spectrum (2dfs).

```
prompt>  pspec -lrfs -2dfs tutorial1.pulsestack
```

The program ask you to identify a (or multiple) on-pulse region. These regions are used as follows:

- Everything what is *not selected* is used for noise calculations, so make sure that all signal is selected in one or more ranges you define. For a lrfs, this is the only way the selected regions are used.
- A 2dfs is generated *for each* of the selected on-pulse regions.

So a strategy could be to make a first selection which only contains bins with a clearly detected signal. This ensures that the first 2dfs to be produced is only based on clearly detected signal and less noise is included compared to when a wider selection would have been made. If you adopt this strategy you probably want to make a seconds selection which is wider, to ensure that the off-pulse region does not contain any signal. After pressing 'S' in the pgplot window it shows you what selection you have made. In this case you could have selected the regions '131 170' and '118 187' for example.

The next plot which is shown is the lrfs. In this case all power is concentrated in a very low frequency P_3 fluctuation, which corresponds to 0.01 cycles per period (cpp), which means P_3 is about 100 pulses. However part of the low-frequency modulation detected is because of the nulls, which also corresponds to intensity modulation at a low frequency. The P_3 resolution of the lrfs (and also the 2dfs) is determined by the length of the fft's which are performed. This length can be set with the -nfft option.

The next plot is the 2dfs of the first component, which shows power at the same P_3 frequency, which in this plot is clearly offset from the vertical axis. This indicates that the subpulse modulation is drifting in pulse phase. In the mathematical definition of the 2dfs used in this software, a negative P_2 corresponds to a drift towards later phases. A rough estimation of P_2 would be 22 cpp = 1/22 period = 0.045 period = 13.5 pulse longitude bins since the data has 300 bins covering a full period.

Since we made two onpulse selections, also a second 2dfs is shown. This essentially is the same plot as the previous 2dfs, but it contains slightly more noise. Note also that covered P_2 range depends on the with of the onpulse region, which might be different for the two plots.

To confirm that the P_3 and P_2 values make sense, you might want to check the original data itself by plotting it in interactive mode:

```
prompt> pplot -ia tutorial1.pulsestack
```

You might want to zoom in in the vertical range (using the v key) and the pulse longitude range (l option).

5.2 Modulation index

You can calculate and show a modulation index with

```
prompt> pspec -mod -prof tutorial1.pulsestack
```

It asks for an onpulse region, in this case it is only used to identify the noise, so make it slightly wider than the pulse to be at the save side. The option -prof calculates the average profile. Without this option the result is not plotted. After making the selection the profile and modulation index are shown.

As noted in the help on the command line, the analytic errorbars are in general not very accurate. Instead you can use a process called bootstrapping to get more reliable errorbars, so you could run the following command to determine the errorbars with 100 iterations. As explained in the help the resulting errorbars are reliable, but slightly conservative.

```
prompt>  pspec -mod -prof -bootstrap 100 tutorial1.pulsestack
```

5.3 Making nice plots

The above results can be combined in a nice looking plot. In order to do this the results should be written to files first, so one can do something like:

```
prompt> pspec -lrfs -2dfs -mod -prof -bootstrap 100 -w  -nfft 256 \
             -onpulse "131 170" -onpulse "118 187" tutorial1.pulsestack
```

These are all the previously used options plus the -w option which allows you to write out the results. Here we also changed the default FFT-size to 256. Since the input data-set only consists of 1022 pulses, this means that the first 768 pulses will be analysed rather than only the first 512. Reducing the FFT-size will result in a lower P_3 resolution.

The results are written to:

- tutorial1.profile - ASCII file containing the profile + standard deviation + modulation index. The columns are: (1) bin number (2) intensity of profile (3) std. dev. (4) 1 sigma error on std. dev. (5) modulation index (6) error on modulation index.
- tutorial1.lrfs - The lrfs which you can plot with for example pplot.
- tutorial1.1.2dfs - The 2dfs based on the first selection you made.
- tutorial1.2.2dfs - The 2dfs based on the second selection you made.

These results can be combined in a single figure using pspecFig. To make the figure with the default options you can run:

```
prompt>  pspecFig tutorial1.pulsestack
```

You probably want to output the results to a postscript file, either by typing in the appropriate device name, or running something like

```
prompt> pspecFig -device "tutorial1.ps/ps" tutorial1.pulsestack
```

By default only the first 2dfs is shown. Note also that the 2dfs is flipped, as the feature in the 2dfs now appears at the right-hand-side in the figure. So by default a positive P_2 value in the pspecFig figures mean positive drifting, which is drifting towards later phases.

There are various options to make the plot look nicer. Run pspecFig without command line option to see what options are available. For example you could try out:

```
prompt> pspecFig -device "tutorial1.ps/ps" -l "100 260" -scalel 10 -scale2 30 -nostddev \
                 -title "pspecFig test" -xlabel3 -ylabel3 -ytop tutorial1.pulsestack
```

5.4 Analysis of features in fluctuation spectra

The obtained fluctuation spectra can be interactively explored with the program pspecDetect. This program is designed to analyse features in the 2dfs to obtain centroid P_2 and P_3 values and corresponding error-bars. To start the program run

```
prompt> pspecDetect -v tutorial1.pulsestack
```

which assumes the lrfs and 2dfs are written out with the default extensions. Press 'h' inside the pgplot window to get a list of supported key presses.

The first plot shown is the 2dfs. By pressing the space-bar you can cycle through the different plots. The next plot shows the lrfs, whith the pulse profile superimposed to more easily indentifying what part of the profile corresponds to what modulation power. The next plot shows the 2dfs again, from which you can zap power by clicking with the mouse and making boxes. The remainder of the power is assumed to be power associated with white noise, so get rid of anything that looks like signal. In the terminal a sigma value is reported, which should be something like 2.8×10^{-7} after removal of all signals. After pressing space another time, you should end up with the first plot.

In the plot showing the 2dfs (first plot) you can zoom in on features by defining boxes with the mouse. 'f' resets the zoom range only, while 'r' also resets the earlier flagged points to identify the noise. By default zooming in always selects a symmetric P_2 range, to avoid any bias in the centroid P_2 value derived. By zooming in on the low-frequency power and pressing return you should be able to find out that the centroid values (indicated by the cross) correspond to something like $P_3 = 140$ periods and $P_2 = -44$ cpp.

In this case it is clear there are drifting subpulses detected, while the reported centroid values are heavely biassed because of the power centred at the origin, which is related to the nulling. By pressing 'c' the centring of the P_2 range can be switched off, allowing you to zoom in to the drift feature itself, without considering the power close to the origin. This should allow you to determine that P_2 is more like -14.5 cpp in reality. Note that the reported error-bar is tiny. However, a major source of error on centroid values is the what region in the spectra are included in the centroid calculation, since this is at some level an arbitrary decision. Therefore it makes sense to use a few different selected ranges to see what spread in the centroid values you get.

5.5 Subpulse phase track

Analysing the subpulse phase as function of pulse longitude gives you some feeling for the average shape of the drift bands in your data. In particular, it shows for instance if there is any curvature, or if the drift bands are straight.

The method is based on the same type of analysis done to compute the LRFS. The pulse stack is divided into equal sized blocks (in pulse number) with a length given by the chosen length of the fft's. FFT's are computed in columns of data, i.e. for a given pulse longitude, the FFT is done over the recorded intensities for the subsequent pulses. The FFT results in complex Fourier components. The amplitude is related to the strength of the modulation, while the phase tells you something about the required delay of a sinusoid with a given frequency to match the data. For linear drift bands this phase (subpulse phase) should change linearly with pulse longitude. As a side note: the subpulse phase is actually the negative of the phase obtained from the FFT's. This choice has been made in PSRSalsa to ensure that a positive gradient of the subpulse phase as function of pulse longitude corresponds to positive drifting.

To do the analysis, first the P_3 spectral bin of interest from the LRFS needs to be identified. In section 5.4 we identified that $P_3 = 140$ periods, corresponding to 0.0071 cpp. To compute the phase track you could do

```
prompt> pspec -v -freq "0.0070 0.0072" -track tutorial1.pulsestack
```

This fails, because there is no complete frequency bin within the specified range. So the frequency range should be specified to be wide enough. The following will work

```
prompt> pspec -v -freq "0.005 0.009" -track tutorial1.pulsestack
```

Note that a warning is generated stating that the frequency range is now so wide that two spectral bins are being used. Although the software will try to do something sensible, it is highly recommended to only analyse a single spectral bin. So if you want to use this frequency range, it would be better to use shorter FFT lengths, resulting in wider spectral channels, to ensure that a single frequency bin falls within the selected frequency range. A balance needs to be struck between making your spectral channels wide enough so that enough signal is present within the analysed channel, without it being so wide that it includes more than just the signal you're interested in. So to ensure that a single frequency bin falls within the selected range we could do

```
prompt> pspec -nfft 256 -v -freq "0.005 0.009" -track tutorial1.pulsestack
```

We can also compute the amplitude, as well as the phase, using

```
prompt> pspec -amplitude -w -nfft 256 -v -freq "0.005 0.009" -track tutorial1.pulsestack
```

Here we also specified the -w option, which writes out the found results to a files (tutorial1.track and tutorial1.amplitude). The first column is the pulse longitude bin number, followed by the subpulse phase in one case, and subpulse amplitude in the other. The file with the subpulse phase information has -1 in the last column. This corresponds to the error on the subpulse phase, which is currently undefined. Errors can be computed using the -bootstrap option. So we could do the following to obtain errors

```
prompt> pspec -bootstrap 100 -amplitude -w -nfft 256 -v -freq "0.005 0.009" \
              -track tutorial1.pulsestack
```

You can use pspecFig to plot the results, if desired.

```
prompt> pspecFig -phaseplot -device "phaseplot.ps/cps" -l "140 220" tutorial1.pulsestack
```

Note that the subpulse phase is increasing with pulse longitude (and is wrapping a few times). This is consistent with the positive drift in the input data-set.

The subpulse amplitude shows structure, which in this case is most likely because the very small number of complete cycles in the data-set. The subpulse phase is not quite linear. You can play with the

-phaseslope option of pspecFig to subtract out a linear gradient, thereby more clearly show the deviations. A similar result can be obtained by using the -slope and the -track_dphase option in pspec.

A final note about the computation of the subpulse phase track: A subpulse phase track is determined for each separate block of data with a length given by the FFT length. These phase tracks are averaged coherently by aligning the subpulse phase tracks of the individual blocks. Although the shape of the subpulse phase tracks are expected to be similar for the different blocks, they will have essentially arbitrary offsets in subpulse phase. The offsets are determined by a cross-correlation technique. By default the fully specified on pulse region is used. However, the total on pulse region selected, also with the aim of specifying which pulse longitudes to exclude from the determination of the off pulse rms, is in general wider than the longitude range where the subpulses are strong. This implies that the -track_firstregion option is useful. This allows you to first specify a relatively narrow on pulse region (the first selected), which will be used to align the subpulse phase tracks of the individual blocks (the cross-correlation is only using the first specified on pulse region). Then additional wider on pulse regions can be selected, which reduces the pulse longitude range to be used for the noise calculation.

5.6 P_3 folding

P_3 folding is the process of averiging the modulation cycle with the periodicity P_3, which means we get a plot of how the average drift band looks like. A complication is that P_3 is in general not constant throughout the data, hence we have to resolve the fluctuations in P_3. This is something pspec can do. We start with a fake data-set called tutorial2.pulsestack, which includes clear drifting subpulses, but now with a somewhat variable P_3 value. If you want, you can plot the data with pplot, which should show that the driftbands are closer and further apart in different stretches of the data. You can also see that the slope of the driftbands is varying at the same time (you probably need to zoom in while using the interactive mode), so different parts of the data should be folded with a different P_3 value.

Before we're going to fold the data, lets start with calculating the LRFS in order to find the typical value of P_3 in the data.

```
prompt> pspec -lrfs -v tutorial2.pulsestack
```

You should see that the feature in the lrfs is smeared out in P_3, confirming it is indeed varying. Maybe a good input value for P_3 to use would be a value corresponding to 0.14 cpp (so $P_3 = 7.15$ periods). Hopefully the software will be able to find the correct P_3 value throughout the whole data set. Note that it probably takes a bit of experimenting what input parameters gives you the best results. In the end, P_3 folding should be considered to be a nice way to graphically represent the data, and 100% accuracy should not be expected since in reality the driftbands might not be repetitive enough, for instance because of nulling, mode-changing etc. So lets try this P_3 value as the initial guess. In addition it needs to be specified how in how many (equally spaced) bins the P_3 cycle will be divided. For now, lets generate 7 pulse shapes throughout the modulation cycle. The intrinsic resolution of the P_3 fold will be one pulse period, so making the number of bins much larger than the value of P_3 (expressed in periods) has very little use. In the command below the P_3 cycle will be separated in 7 bins.

```
prompt> pfold -p3fold "7.15 7" -v tutorial2.pulsestack
```

The selected on-pulse region is used to do the correlations (as explained below in more detail), but the full pulse longitude range will be folded. When you select the onpulse region, a relatively small on-pulse region might give the best result (include no noise, and maybe even leave out the outer bits of the profile). The reason is that the correlations are done on whatever you selected, hence the correlation can be done most accurately by considering those parts of the profile which are strong. The output (the average drift band) is shown twice on top of each other for continuity. It should show very clearly what the average drift band looks like. This command tries to resolve the variations in P_3. If you want, you can find out what would be obtained if you would use a fixed P_3 value by adding the -p3fold_norefine option, which results in complete failure to detect the drifting subpulses.

If you want to see in more detail how the driftbands deviate from a diagonal line it could be useful to subtract a constant slope from the observations. This might be especially useful if the driftbands are steep. You can see the result using the following command:

```
prompt> pfold -p3fold "7.15 7" -v -slope 25 tutorial2.pulsestack
```

Here a slope of 25 deg/deg is subtracted from the drift bands. The drifting subpulses now appear in the plot as an on/off modulation, i.e. the P_3 modulation without the drift in pulse longitude. You can introduce a vertical shift to the result using the -p3fold_dphase option, which might make your plot nicer. For instance the following comment should result result in a plot which starts at the minimum of the modulation cycle.

```
prompt> pfold -p3fold "7.15 7" -v -slope 25 -p3fold_dphase 90 tutorial2.pulsestack
```

Resolving variations in P_3 is a iterative process. There are two important parameters which specify the way this is done. There is an option -p3fold_nritt, which sets the number of iterations which are performed. In order to find the variations in P_3, knowledge about the shape of the driftbands is required. This is because essentially you take a block of P_3 pulses, i.e. a full cycle, then take the next block and you fit for the offset between the patterns observed in the two blocks. So if P_3 is a bit larger than the typical value provided on the command line, the driftband appears a bit late in the second block compared to the first. Hence when folding the data, the extra shift, or offset, has to be subtracted first. However, after folding the full data set, the result, i.e. the "average drift band" can be used as a template to do the comparisons to find the offsets in each block. This has the advantage that the obtained average drift band has a high S/N, hence the offsets can be determined more accurately than just using the previous block, i.e. the offsets obtained by making comparisons between individual drift bands. By default this loop is only done once, but you can increase this number by using the -p3fold_nritt option. You can experiment with increasing this number to see if the result is improving and to see after which number no changes in the end result are apparent. Another parameter to try out to refine the results is the -p3fold_cpb option. This specifies the number of P_3 cycles used per block of data analysed. So for instance if this number is set to 2 (default is 1), it means the software takes the first $2 \times P_3$ pulses to generate the average of two driftbands (using a fixed P_3 folding with the number specified on the command line), it then takes the next block of $2 \times P_3$ pulses and the offset between the driftbands is obtained by a cross-correlation of the two blocks. In other words, increasing this number increases the S/N of the drifting subpulses in each block, which increases the precision of the determined offsets, however at the same time the drifting subpulses are smeared out. The latter is because the fixed period folding done before doing the cross-correlation. Again, experimenting is required to find the optimum compromise. In this example you cannot do much better than using the default values since the simulated pattern is quite regular.

You might want to "over-sample" the data, which means specifying more P_3 bins than the number of available pulses per P_3 cycle. This might result in nicer looking plots, but remember that each row will no longer be independent on the other rows. Try out the following command which will generate 20 P_3 bins:

```
prompt> pfold -p3fold "7.15 20" -slope 25 -v tutorial2.pulsestack
```

Depending on the number of bins used, you might see strange patterns arising which are caused by unequal sampling of different parts of the output. In order to remove these artifacts you can use the -p3fold_smooth option, which allows you to replace the top-hat weight used to assign power of each individual pulse to the different P_3 bins with a Gaussian weight with this width in P_3 phase bins. This will smooth the output, hence could make oversampling look nicer. The following command smooths the output with 1 P_3 bin (i.e. the intrinsic resolution). In the example below also the -w option is included, which means that the result is written to an output file.

```
prompt> pfold -w -p3fold "7.15 20" -slope 25 -v -p3fold_smooth 1 tutorial2.pulsestack
```

Note that this makes the calculation considerably slower, hence you might want to include this option after you're happy with the result obtained using the other options. The output by default is in psrfits format and hence can be read in by PSRCHIVE and other software of this package, for instance you can make a plot using PPLOT

```
prompt> pplot -v -showtwice -showright -showtop -showwedge tutorial2.p3fold
```