Searching for Pulsars with `PRESTO`

By Scott Ransom
NRAO / UVa

Getting `PRESTO`

- Homepage: http://www.cv.nrao.edu/~sransom/presto/
- `PRESTO` is freely available from github
https://github.com/scottransom/presto
- Note the new FAQ!
- You are highly encouraged to fork your own copy, study / modify the code, and make bug-fixes, improvements, etc....

For this tutorial...

- You will need a fully working version of `PRESTO` (including the python extensions)
- If you have questions about a command, just try it out! Typing the command name alone usually gives usage info.
- You need at least 1GB of free disk space
  - Linux users: if you have more than that amount of RAM, I encourage you to do everything in a subdirectory under `/dev/shm`
- Commands will be `> typewriter script`
- The sample dataset that I'll use is here (25MB) http://www.cv.nrao.edu/~sransom/GBT_Lband_PSR.fil

Outline of a PRESTO Search

1) Examine data format (readfile)
2) Search for RFI (rfifind)
3) Make a topocentric, DM=0 time series (prepdata and exploredat)
4) FFT the time series (realfft)
5) Identify "birdies" to zap in searches (explorefft and accelsearch)
6) Make zaplist (makezaplist.py Note: see simple_zapbirds.py)
7) Make De-dispersion plan (DDplan.py)
8) De-disperse (prepsubband)
9) Search the data for periodic signals (accelsearch)
10) Search the data for single pulses (single_pulse_search.py)
11) Sift through the candidates (ACCEL_sift.py)
12) Fold the best candidates (prepfold)
13) Start timing the new pulsar (prepfold and get_TOAs.py)

Examine the raw data

> readfile GBT_Lband_PSR.fil

```
> readfile GBT_Lband_PSR.fil
Assuming the data is a SIGPROC filterbank file.


1: From the SIGPROC filterbank file 'GBT_Lband_PSR.fil':
                  Telescope = GBT
                Source Name = Mystery_PSR
            Obs Date String = 2004-01-06T11:38:09
             MJD start time = 53010.48482638889254
                   RA J2000 = 16:43:38.1000
             RA J2000 (deg) = 250.90875
                  Dec J2000 = -12:24:58.7000
            Dec J2000 (deg) = -12.4163055555556
                  Tracking? = True
              Azimuth (deg) = 0
             Zenith Ang (deg) = 0
            Number of polns = 2 (summed)
           Sample time (us) = 72
         Central freq (MHz) = 1400
          Low channel (MHz) = 1352.5
         High channel (MHz) = 1447.5
        Channel width (MHz) = 1
         Number of channels = 96
      Total Bandwidth (MHz) = 96
                       Beam = 1 of 1
            Beam FWHM (deg) = 0.147
         Spectra per subint = 2400
           Spectra per file = 531000
      Time per subint (sec) = 0.1728
        Time per file (sec) = 38.232
            bits per sample = 4
          bytes per spectra = 48
        samples per spectra = 96
           bytes per subint = 115200
         samples per subint = 230400
                zero offset = 0
           Invert the band? = False
       bytes in file header = 365

> 
```

- `readfile` can automatically identify most of the datatypes that `PRESTO` can handle (in `PRESTO` v2, though, this is only SIGPROC filterbank and PSRFITs)
- It prints the meta-data about the observation

Search for prominent RFI: 1

> rfifind -time 2.0 -o Lband GBT_Lband_PSR.fil

```
> rfifind -time 2.0 -o Lband GBT_Lband_PSR.fil


               Pulsar Data RFI Finder
                 by Scott M. Ransom

Assuming the data are SIGPROC filterbank format...
Reading SIGPROC filterbank data from 1 file:
  'GBT_Lband_PSR.fil'

    Number of files = 1
       Num of polns = 2 (summed)
  Center freq (MHz) = 1400
    Num of channels = 96
    Sample time (s) = 7.2e-05
     Spectra/subint = 2400
   Total points (N) = 531000
     Total time (s) = 38.232
     Clipping sigma = 6.000
   Invert the band? = False
          Byteswap? = False
     Remove zeroDM? = False

File  Start Spec   Samples     Padding       Start MJD
----  ----------  ----------  ----------  --------------------
1              0      531000           0  53010.48482638889254

Analyzing data sections of length 28800 points (2.0736 sec).
  Prime factors are:  2 2 2 2 2 2 2 3 3 5 5

Writing mask data  to 'Lband_rfifind.mask'.
Writing  RFI data  to 'Lband_rfifind.rfi'.
Writing statistics to 'Lband_rfifind.stats'.

Massaging the data ...

Amount Complete =  37%^C
> 
```

- `rfifind` identifies strong narrow-band and/or short duration broadband RFI
- Creates a "mask" (basename determined by "`-o`") where RFI is replaced by median values
- `PRESTO` programs automatically clip strong, transient, DM=0 signals (turn off using `-noclip`) Usually a good thing!
- Typical integration times (`-time`) should be a few seconds
- Modify the resulting mask using "`-nocompute -mask ...`" and the other `rfifind` options

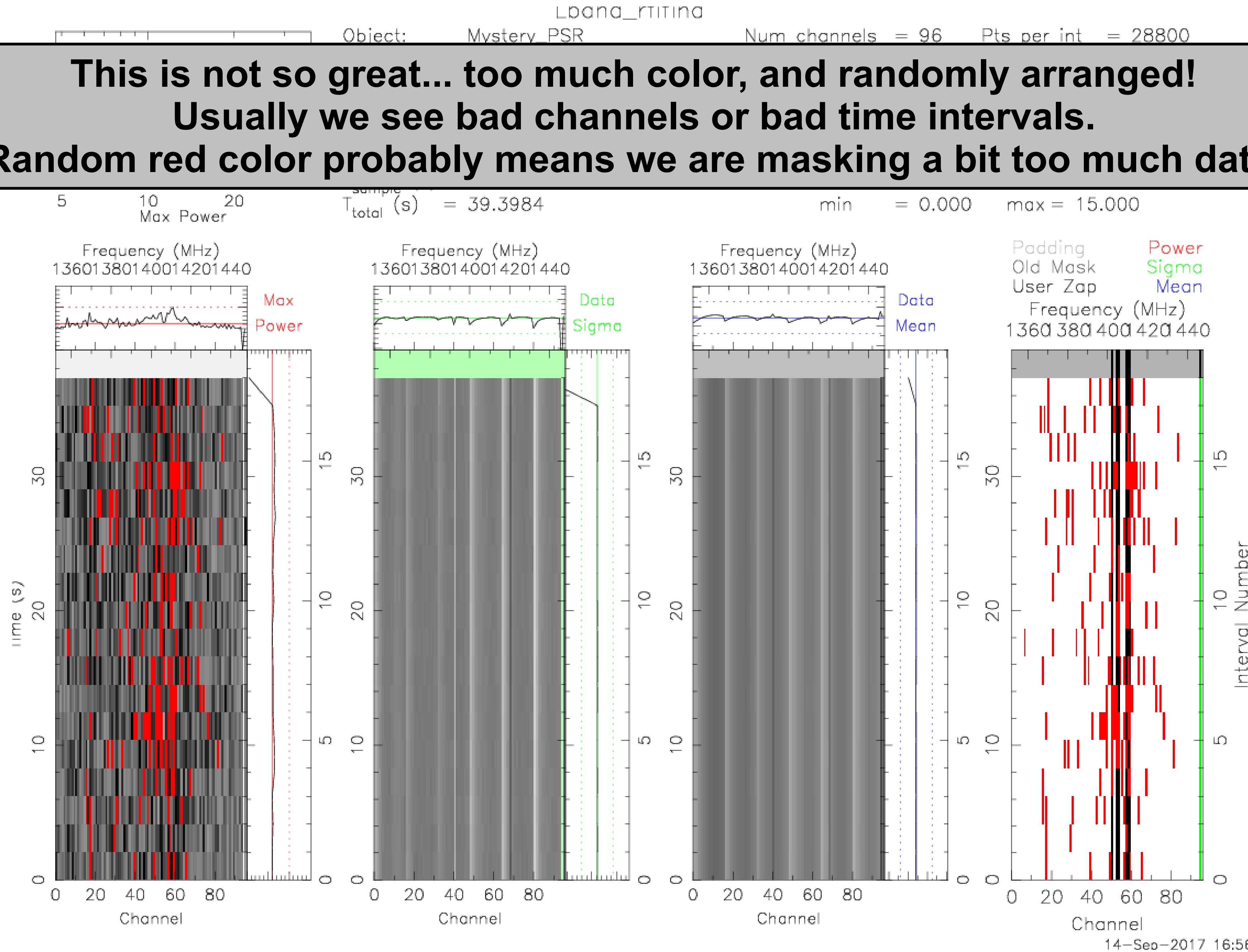
Search for prominent RFI: 2

```
Writing mask data  to 'Lband_rfifind.mask'.
Writing  RFI data  to 'Lband_rfifind.rfi'.
Writing statistics to 'Lband_rfifind.stats'.

Massaging the data ...

Amount Complete = 100%
There are 31 RFI instances.


Total number of intervals in the data:  1824

  Number of padded intervals:      96  ( 5.263%)
  Number of  good  intervals:    1487  (81.524%)
  Number of  bad   intervals:     241  (13.213%)

  Ten most significant birdies:
#  Sigma     Period(ms)       Freq(Hz)       Number
----------------------------------------------------
1  6.83      11.5521          86.5644        147
2  6.71      11.6494          85.841         170
3  6.68      11.6168          86.0822        146
4  6.57      8.76787          114.053        1
5  6.53      11.5844          86.3233        145
6  6.10      11.52            86.8055        135
7  5.96      11.4881          87.0467        107
8  5.89      11.7153          85.3588        21
9  5.88      11.6823          85.5999        23
10 5.65      11.7484          85.1177        24

  Ten most numerous birdies:
#  Number    Period(ms)       Freq(Hz)       Sigma
----------------------------------------------------
1  493       34.56            28.9352        4.82
2  351       34.8504          28.6941        4.75
3  280       17.28            57.8704        4.85
4  271       17.3523          57.6292        4.80
5  180       17.4252          57.3881        4.68
6  179       17.4987          57.147         4.67
7  170       11.6494          85.841         6.71
8  147       11.5521          86.5644        6.83
9  146       11.6168          86.0822        6.68
10 145       11.5844          86.3233        6.53

Done.
```

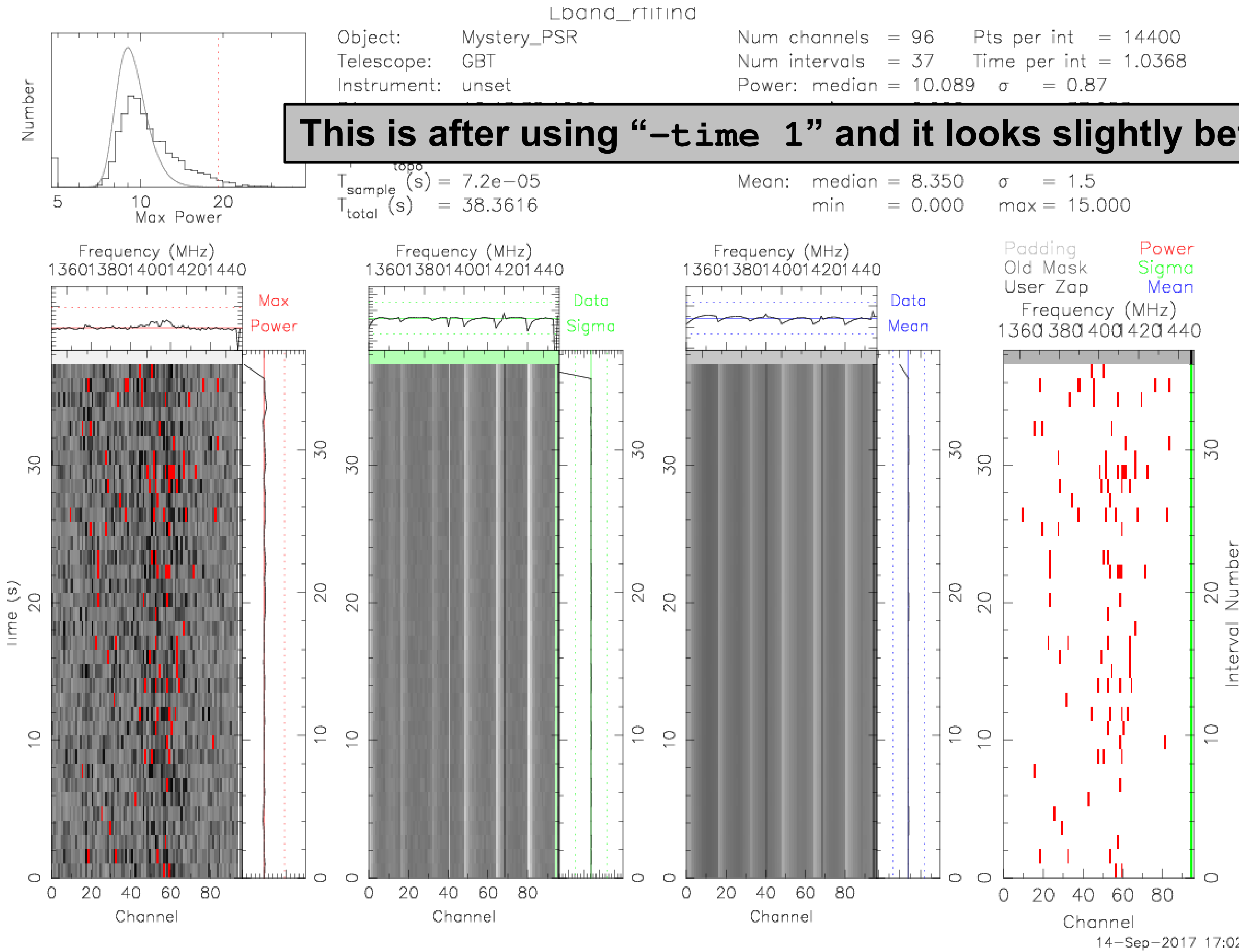
- Check the number of bad intervals. Usually should be less than ~20%
- Most significant and most numbers birdies are listed (to see all, use `-rfixwin`)
- Makes a bunch of output files including "...rfifind.ps" where colors are bad (red is periodic RFI, blue/green are time-domain statistical issues)
- Re-run with "`-time 1`" or re-compute with "`-nocompute`" in this case

Search for prominent RFI: 3

This is not so great... too much color, and randomly arranged!
Usually we see bad channels or bad time intervals.
Random red color probably means we are masking a bit too much data.



Search for prominent RFI: 4



This is after using "`-time 1`" and it looks slightly better.

Shortcuts for big observations

Sometimes for long observations, or those with many channels, fast sampling, or lots of RFI, `rfifind` can take a ***long*** time to run. You can often mask most of the RFI doing a few shortcuts and using `-ignorechan`:

- Run `rfifind` on a subset of the data (one or more of the individual files)
- Tweak the results, primarily using `-nocompute` and different values of `-freqsig` and `-timesig`, so the worst channels are marked for masking
- Run `rfifind_stats.py` on one of the resulting rfifind files. That will average the stats over the rfifind file and make a ".weights" file that shows which channels should be zero weighted (also an average ".bandpass" file)
- You can then convert that weights file into a list of channels to ignore using the `weights_to_ignorechan.py` routine, which also gives you a `paz` command (from PSRCHIVE) to zap folded archives made from the data
- "ignorechan" syntax lists channels (starting from 0), or start:end ranges of channels, separated by commas which can be used with `prepfold`, `prepdata`, `prepsubband`, or `mpiprepsubband`, for example:

```
> prepdata … -ignorechan 0:10,15,20:25,67 … myfiles*.fil
```

Look for persistent low-level RFI

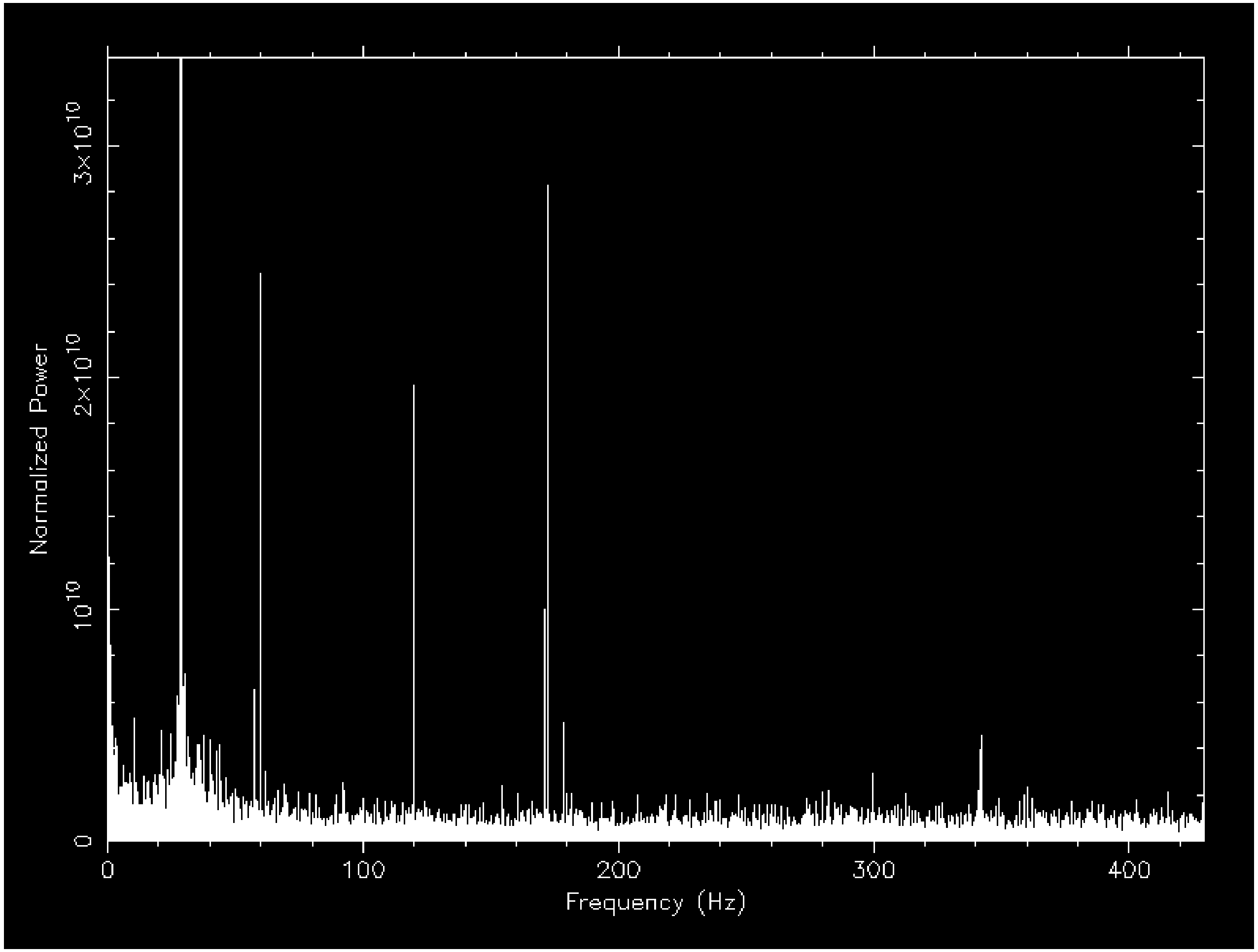
```
> prepdata -nobary -o Lband_topo_DM0.00 \
    -dm 0.0 -mask Lband_rfifind.mask \
    GBT_Lband_PSR.fil
```

- `prepdata` de-disperses a single time-series. The “`-nobary`” flag tells `PRESTO` not to barycenter the time series.
- If you need to de-disperse multiple time-series, use `prepsubband`
- We used to need to set the number of points (`-numout`) to make it a nice round number for FFTing, but PRESTO does that automatically now

```
           Pulsar Data Preparation Routine
    Type conversion, de-dispersion, barycentering.
                 by Scott M. Ransom

Assuming the data are SIGPROC filterbank format...
Reading SIGPROC filterbank data from 1 file:
  'GBT_Lband_PSR.fil'

    Number of files = 1
       Num of polns = 2 (summed)
  Center freq (MHz) = 1400
    Num of channels = 96
    Sample time (s) = 7.2e-05
     Spectra/subint = 2400
   Total points (N) = 531000
     Total time (s) = 38.232
     Clipping sigma = 6.000
   Invert the band? = False
          Byteswap? = False
     Remove zeroDM? = False

File  Start Spec   Samples     Padding       Start MJD
----  ----------  ----------  ----------  --------------------
1              0      531000           0  53010.48482638889254

Read mask information from 'Lband_rfifind.mask'

Attempting to read the data statistics from 'Lband_rfifind.stats'...
...succeded.  Set the padding values equal to the mid-80% channel averages.
Writing output data to 'Lband_topo_DM0.00.dat'.
Writing information to 'Lband_topo_DM0.00.inf'.

Massaging the data ...

Amount Complete = 100%

Done.

Simple statistics of the output data:
             Data points written:  530000
           Maximum value of data:  909.05
           Minimum value of data:  674.91
              Data average value:  785.54
         Data standard deviation:  23.12

>
```

Explore and FFT the time-series

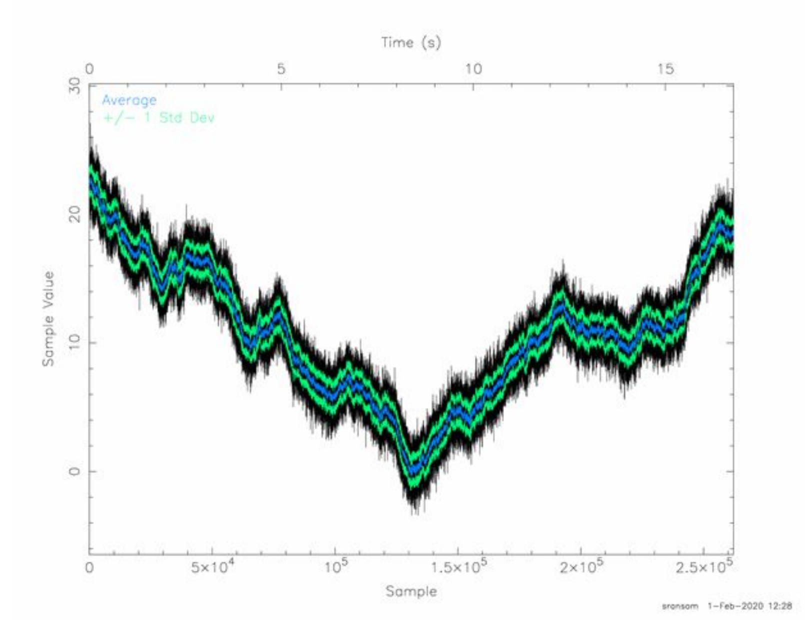
```
> exploredat Lband_topo_DM0.00.dat
> realfft Lband_topo_DM0.00.dat
> explorefft Lband_topo_DM0.00.fft
```

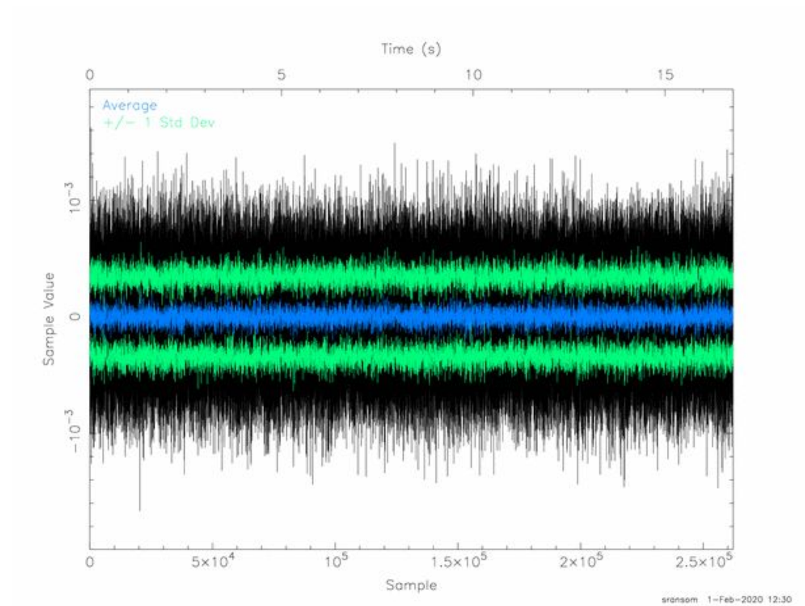


- `exploredat` and `explorefft` allow you to interactively view a time-series or its power spectrum (for finding RFI)
- changing the power normalization (key 'n') in `explorefft` is often very helpful
- `realfft` requires that the time-series is easily factorable (and at least has 1 factor of '2'). Check using "`factor`".

Note: Rednoise and its suppression

- If your time series looks like the one on the right, you have a rednoise problem
- Rednoise makes searches for, and folding of, slow pulsars (in particular), problematic
- You can suppress much of that rednoise in your .fft using the `rednoise` program (which is described in Lazarus et al. 2015)
- That program makes a new .fft file (and corresponding .inf file) that ends in *_red.fft, which you can search
- Or, you can use `realfft` on the *_red.fft file to create a de-reddened time series (*_red.dat), as seen to the right (which can then be folded with `prepfold`)
- Beware that rednoise will always decrease your S/N at the frequencies where it is present! Removing it with the `rednoise` program will not fix that!





Find the periodic interference

`> accelsearch -numharm 4 -zmax 0 \`
`Lband_topo_DM0.00.dat`

Cand	Sigma	Summed Power	Coherent Power	Num Harm	Period (ms)	Frequency (Hz)	FFT 'r' (bin)	Freq Deriv (Hz/s)	FFT 'z' (bins)	Accel (m/s^2)	Notes
1	60.87	1876.6	3637.40	2	34.777(8)	28.754(6)	1113.00(25)	0.0000(7)	0.0(1.0)	0.0(7.0)x10^3	
2	20.01	229.74	671.54	4	16.6698(9)	59.989(3)	2322.00(13)	0.0000(3)	0.00(50)	0.0(1.7)x10^3	
3	9.20	57.94	57.02	1	5.7945(4)	172.58(1)	6680.00(50)	0.000(1)	0.0(2.0)	0.0(2.3)x10^3	H 6 of Cand 1
4	7.93	55.92	53.33	4	5.8484(1)	170.986(3)	6618.38(13)	0.0000(3)	0.00(50)	0.0(5.9)x10^2	
5	4.26	31.23	59.09	4	5.6024(1)	178.494(3)	6909.00(13)	0.0000(3)	0.00(50)	0.0(5.6)x10^2	
6	3.90	25.02	5.39	2	2.92384(6)	342.016(6)	13238.50(25)	0.0000(7)	0.0(1.0)	0.0(5.9)x10^2	

Cand	Harm	Sigma	Power / Loc Pow	Raw Power	FFT 'r' (bin)	Pred 'r' (bin)	FFT 'z' (bins)	Pred 'z' (bins)	Phase (rad)	Centroid (0-1)	Purity <p> = 1	Notes
1	1	78.99	3125(79)	1.99e+03	1113.1595(70)	1113.00	-0.022(55)	0.00	2.477(13)	0.4943(37	0.9895(57	
	2	5.87	19.9(6.3)	16.9	2226.319(91)	2226.00	-0.04(73)	0.00	5.12(16)	0.481(46)	0.962(74)	
2	1	12.38	80(13)	90.9	2322.080(43)	2322.00	0.26(32)	0.00	5.424(79)	0.462(23)	1.021(35)	
	2	20.96	224(21)	143	4644.161(26)	4644.00	0.52(20)	0.00	5.411(47)	0.508(14)	0.997(21)	
	3	4.17	11.1(4.7)	12.9	6966.24(11)	6966.00	0.78(87)	0.00	3.75(21)	0.511(61)	1.024(93)	
	4	3.49	8.3(4.1)	7.02	9288.32(14)	9288.00	1.0(1.2)	0.00	2.05(25)	0.418(71)	0.94(12)	
3	1	10.37	57(11)	70.8	6680.255(56)	6680.00	0.32(47)	0.00	3.222(94)	0.469(27)	0.927(45)	
4	1	6.98	27.2(7.4)	24.8	6618.261(77)	6618.38	-1.01(62)	0.00	1.94(14)	0.483(39)	0.968(63)	
	2	3.62	8.8(4.2)	10.3	13236.52(15)	13236.75	-2.0(1.4)	0.00	4.05(24)	0.350(69)	0.85(13)	
	3	2.58	5.3(3.3)	6.47	19854.78(40)	19855.12	-3.0(7.5)	0.00	4.62(31)	0.290(88)	0.42(33)	
	4	2.95	6.4(3.6)	15.3	26473.04(20)	26473.50	-4.0(2.1)	0.00	5.09(28)	0.342(80)	0.76(16)	
5	1	6.12	21.5(6.6)	19.6	6909.061(89)	6909.00	-0.35(73)	0.00	4.98(15)	0.412(44)	0.942(72)	
	2	2.87	6.2(3.5)	4.55	13818.12(16)	13818.00	-0.7(1.2)	0.00	3.82(28)	0.416(82)	0.99(13)	
	3	2.43	4.9(3.1)	4.33	20727.18(26)	20727.00	-1.1(2.9)	0.00	4.43(32)	0.519(92)	0.69(21)	
	4	2.54	5.2(3.2)	6.43	27636.25(18)	27636.00	-1.4(1.4)	0.00	0.28(31)	0.391(90)	0.96(14)	
6	1	1.43	2.6(2.3)	3.81	13238.68(17)	13238.50	1.18(94)	0.00	3.36(44)	0.43(13)	1.41(14)	
	2	4.45	12.4(5.0)	25	26477.37(12)	26477.00	2.4(1.0)	0.00	4.14(20)	0.394(58)	0.929(97)	

- We “trick” `accelsearch` into finding periodic interference (it found 6 candidates, with several harmonics in each)
- That information will be used to create a “birds” file
- “.inf” file is human readable ASCII (it is also found in the ACCEL file).

Make a “birds” file

- What the heck is a “birds” file?
    - “birds” are pulsar astronomer jargon for periodic interference that shows up in our power spectra. We usually “zap” them by zeroing them out before we search the power spectrum.
- In PRESTO, a .birds file is a simple ASCII text file with 5 columns
    - The fundamental frequency of the periodic interference in Hz
    - The width of the interference in Hz (power lines RFI at 50 or 60 Hz is often quite wide, but some interference is only a single FFT bin wide)
    - The number of harmonics of the fundamental to zap, and then 0/1 (no/yes) for whether the width of the harmonics should grow with harmonic number and whether the freqs are barycentric or not (e.g. the ATNF database freq for a strong pulsar in the data is barycentric)
    - A row starting with a “#” is a comment
    - Here is an example .birds file:

```
> cat Lband.birds
#Freq     Width    #harm    grow?   bary?
1.2       0.02     5        0       0
25.0      0.01     20       0       0
60.0      0.1      5        1       0
100.0     0.02     24       0       0
>
```

Make a “birds” file

- Use `explorefft` and the *ACCEL_0 files to identify the main periodic signals. Since these are DM=0, they are *almost* certainly RFI.
- Edit the .birds file with a text editor
- Given the results of our earlier `accelsearch` run, here is an example (where I examined the signals with `explorefft` to check their widths):

```
> cat Lband.birds
#Freq    Width   #harm   grow?  bary?
28.760   0.1     2       0      0
60.0     0.05    2       1      0
>
```

- **Notes**:
  - Don’t stress out too much over getting a perfect .birds file (especially about high frequency not-too-strong signals – they will be smeared out at high DMs). You mainly want to get the really strong stuff, with Fourier powers more than 50 or so.
  - Usually I make a .birds file only for a certain type of data (like once for a whole project where the data are all the same) or for really important single pointings.

Convert the “birds” file to a zaplist

Note: The command `simple_zapbirds.py` **can do all the following now!**

- Make an associated “.inf” file for the “.birds” file

```
> cp Lband_rfifind.inf Lband.inf
```

- Now convert all of the “birds” and harmonics into individual freqs/widths

```
> makezaplist.py Lband.birds
```

- The resulting “Lband.zaplist” is ASCII and can be edited by hand
- It can also be loaded into `explorefft` so you can see if you are zapping everything you need (see the `explorefft` help screen)
- Apply the zaplist using “`zapbirds`”:

```
> zapbirds -zap -zapfile Lband.zaplist \
     Lband_topo_DM0.00.fft
```

- Zapping barycentric time-series requires “`-baryv`” to convert topocentric RFI freqs to barycentric. Get that by running `prepdata` or `prepfold` on raw data (you can ctrl-c to stop them). As an example:

```
> prepdata -o tmp GBT_Lband_PSR.fil | grep Average
   Average topocentric velocity (c) = -5.697334e-05
```

Determining a De-Dispersion Plan

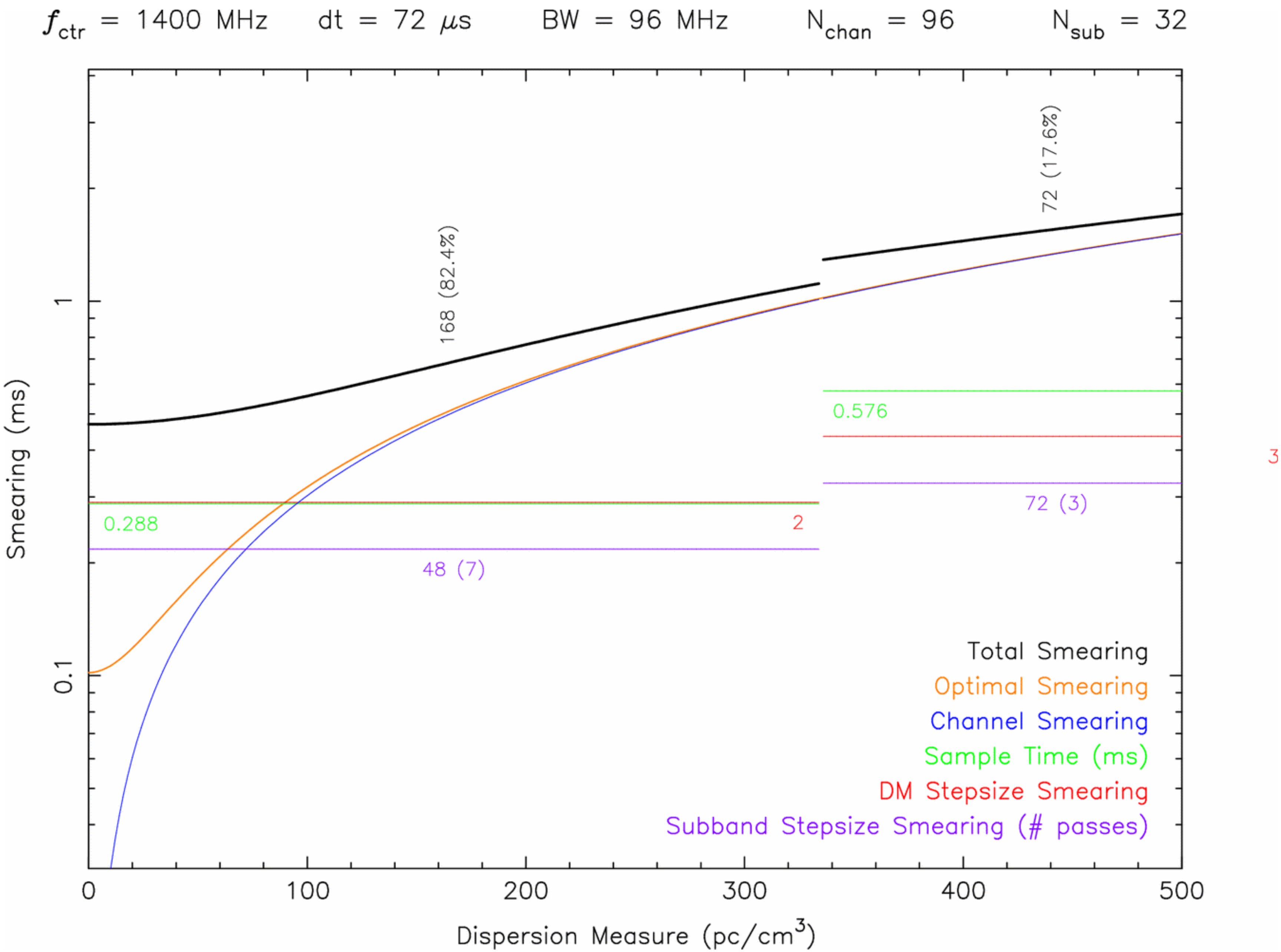
```
> DDplan.py -d 500.0 -n 96 -b 96 -t 0.000072 \
-f 1400.0 -s 32 -r 0.5
```

```
>
>
> DDplan.py -d 500.0 -n 96 -b 96 -t 0.000072 -f 1400.0 -s 32 -r 0.5

Minimum total smearing     : 0.102 ms
--------------------------------------------
Minimum channel smearing   : 1.51e-05 ms
Minimum smearing across BW : 0.00145 ms
Minimum sample time        : 0.072 ms

Setting the new 'best' resolution to : 0.5 ms
   Note: ok_smearing > dt (i.e. data is higher resolution than needed)
         New dt is 4 x 0.072 ms = 0.288 ms
Best guess for optimal initial dDM is 1.984

  Low DM    High DM     dDM  DownSamp   dsubDM   #DMs  DMs/call  calls  WorkFract
    0.000    336.000    2.00        4    48.00    168      24        7    0.8235
  336.000    552.000    3.00        8    72.00     72      24        3    0.1765
```

> **“-r” reduces the effective time resolution to speed up search**

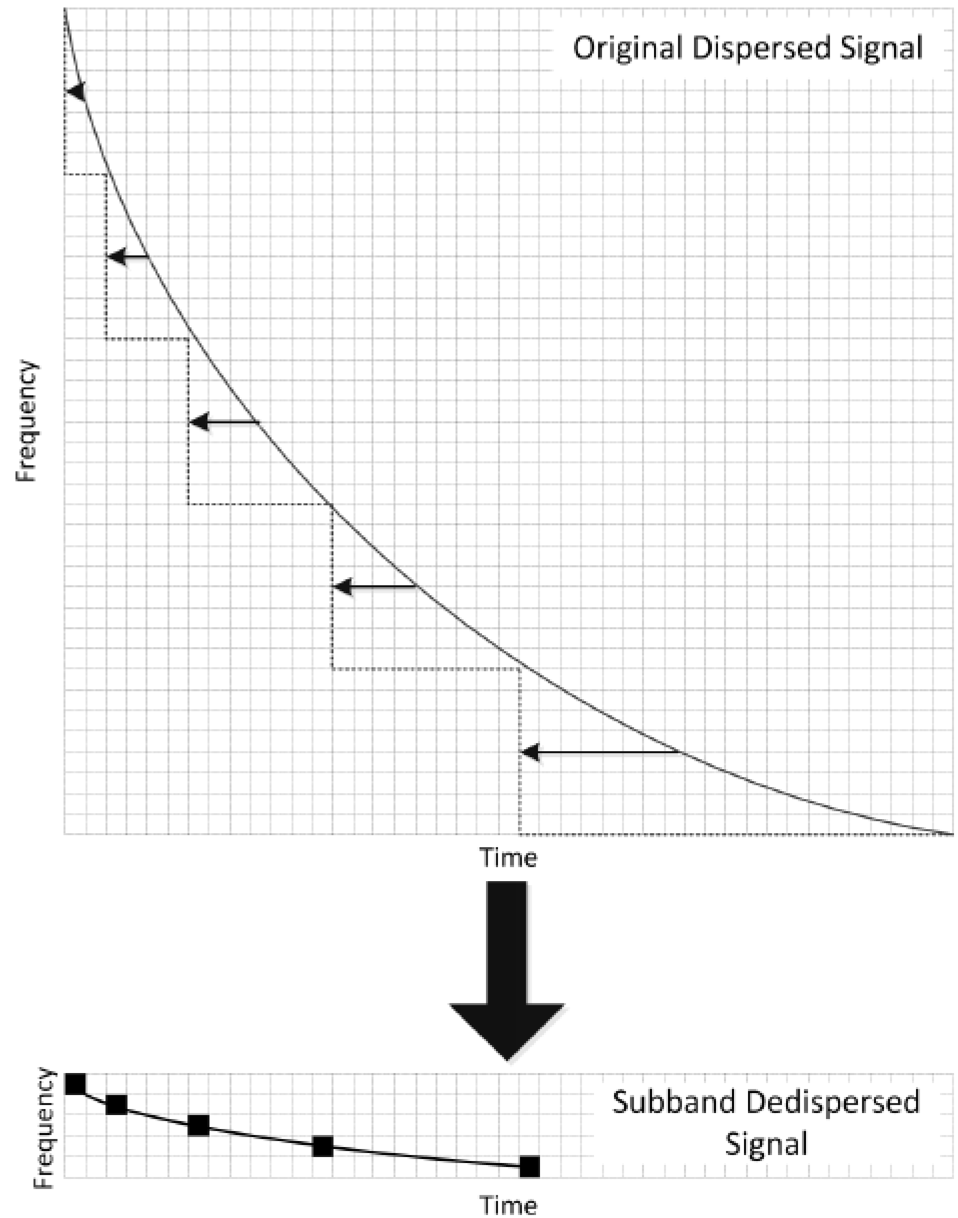
- `DDplan.py` determines near-optimal ways to de-disperse your data to maintain sensitivity to fast pulsars yet save CPU and I/O time
- Assumes using `prepsubband` to do multiple-passes through the data using “subband” de-dispersion
- Specify command line information from `readfile`, or **(New!)** give the filename and `DDplan.py` will determine the observation details
- The **new “-w” option** will write out a dedisp*py file that you can run to dedisperse your data (and edit as needed, i.e. to add rfifind masks)

Determining a De-Dispersion Plan



Subband De-Dispersion 1

- Incoherent de-dispersion requires you to shift the arrival times of each input channel for a particular DM
- This can be made much quicker by partially shifting groups of channels (subbands) to some nominal DM
- The resulting subband dataset can then be de-dispersed around neighboring DMs with many fewer calculations
- In PRESTO, we do this subband de-dispersion with `prepsubband` and `mpiprepsubband`



From Magro and Zarb Adami, MNRAS in press

Subband De-Dispersion 2

```
> prepsubband -nsub 32 -lodm 0.0 -dmstep 2.0 -numdms 24 -downsamp 4 -mask Lband_rfifind.mask -o Lband GBT_Lband_PSR.fil
```

- That command comes from the first call of the first plan line:

Low DM	High DM	dDM	DownSamp	dsubDM	#DMs	DMs/call	calls	WorkFract
0.000	336.000	2.00	4	48.00	168	24	7	0.8235
336.000	552.000	3.00	8	72.00	72	24	3	0.1765

- Run `prepsubband` as many times as there are "calls" in the plan
- Accepted file formats to run `prepsubband` on are SIGPROC filterbank (".fil") and PSRFITS (".sf" or ".fits")
- If you have a parallel computer (and long observations), you can use the fully parallel `mpiprepsubband` to have one CPU read the data, broadcast it to other CPUs, which each effectively makes a "call"
- The `dedisp.py` script in `$PRESTO/examplescripts` can help you automate this process (and generate subbands as well, which can be used to fold candidates faster than folding raw data). When the file has been edited, do: `python dedisp.py`
- `DDplan.py` can now generate `dedisp.py` scripts with the -w option

Prepare for Searching the Data

- First we'll clean up this directory but putting the subband files in their own directory and getting rid of the temporary topocentric files

```
> mkdir subbands
> mv *.sub* subbands/
> rm -f Lband*topo* tmp*
```

- Use `xargs` (awesome Unix command) to fft and zap the `*.dat` files

```
> realfft *.dat  # works with multiple files now
> ls *.fft | xargs -n 1 zapbirds -zap \
-zapfile Lband.zaplist -baryv -5.697334e-05
```

- New recommended zapping alternative:

```
> simple_zapbirds Lband.birds *.fft
```

- Remember that we can get the barycentric value (i.e. average topocentric velocity) by running a fake `prepdata` or `prepfold` command on the raw data
- Now we are ready to run `accelsearch` on the `*.fft` files
- If your time series are short (like these), you can use `accelsearch` to do its own FFTing and zapping by calling it on the ".dat" file. See the `-zaplist` and `-baryv` options for `accelsearch`.

Searching for Periodic Signals

```
> accelsearch -zmax 0 Lband_DM0.00.fft
```

- `Accelsearch` conducts Fourier-domain acceleration (or not, if `zmax=0`) searches for periodic signals using Fourier interpolation and harmonic summing of 1, 2, 4, 8, 16 and/or 32 harmonics (8 is default).
- “zmax” is the max number of Fourier bins the highest harmonic for a particular search (i.e. fundamental or 1st harm. for a 1 harm. search, 8th harm. for a 8 harm. search) can linearly drift in the power spectrum (i.e. due to orbital motion). Sub-harmonics drift proportionally less (i.e. if 2nd harmonic drifts 10 bins, the fundamental will drift 5).
- The time that the searches take doubles for each additional level of harmonic summing, and is linearly proportional to zmax.
- For MSPs, 8 harmonics is almost always enough. And zmax < 200 or so (beyond that non-linear acceleration start to creep in).
- You can use `xargs`: `ls *.fft | xargs -n 1 accelsearch` …
- For this tutorial data, which is very short, you might want to use “`-flo 15`” so that the rednoise at the very lowest freq bins aren’t detected

Sifting the periodic candidates

```
>  python ACCEL_sift.py > cands.txt
```

- `ACCEL_sift.py` is in `$PRESTO/examplescripts` and can be edited and tweaked on an observation specific basis
- It uses several heuristics to reject bad candidates that are unlikely to be pulsars. And it combines multiple detections of the same candidate signals over various DMs (and harmonics as well).
- The output is a human-readable ranked list of the best candidates
- ASCII “plots” in the cands.txt file allow you to see rough signal-to-noise versus DM (if there is a peak at DM != 0, that is good)
- The format for the “candidate” is the candfile:candnum (as you would use them with prepfold)
- You can also look through the ACCEL files themselves. The ones ending in numbers are human readable (use `less -S`). Summaries of the candidates are at top and details of their harmonics at bottom.
- For large single ACCEL files, you can use `quick_prune_cands.py`

Folding Pulsar Candidates

```
> prepfold -accelcand 1 -accelfile \
Lband_DM62.00_ACCEL_0.cand Lband_DM62.00.dat
```

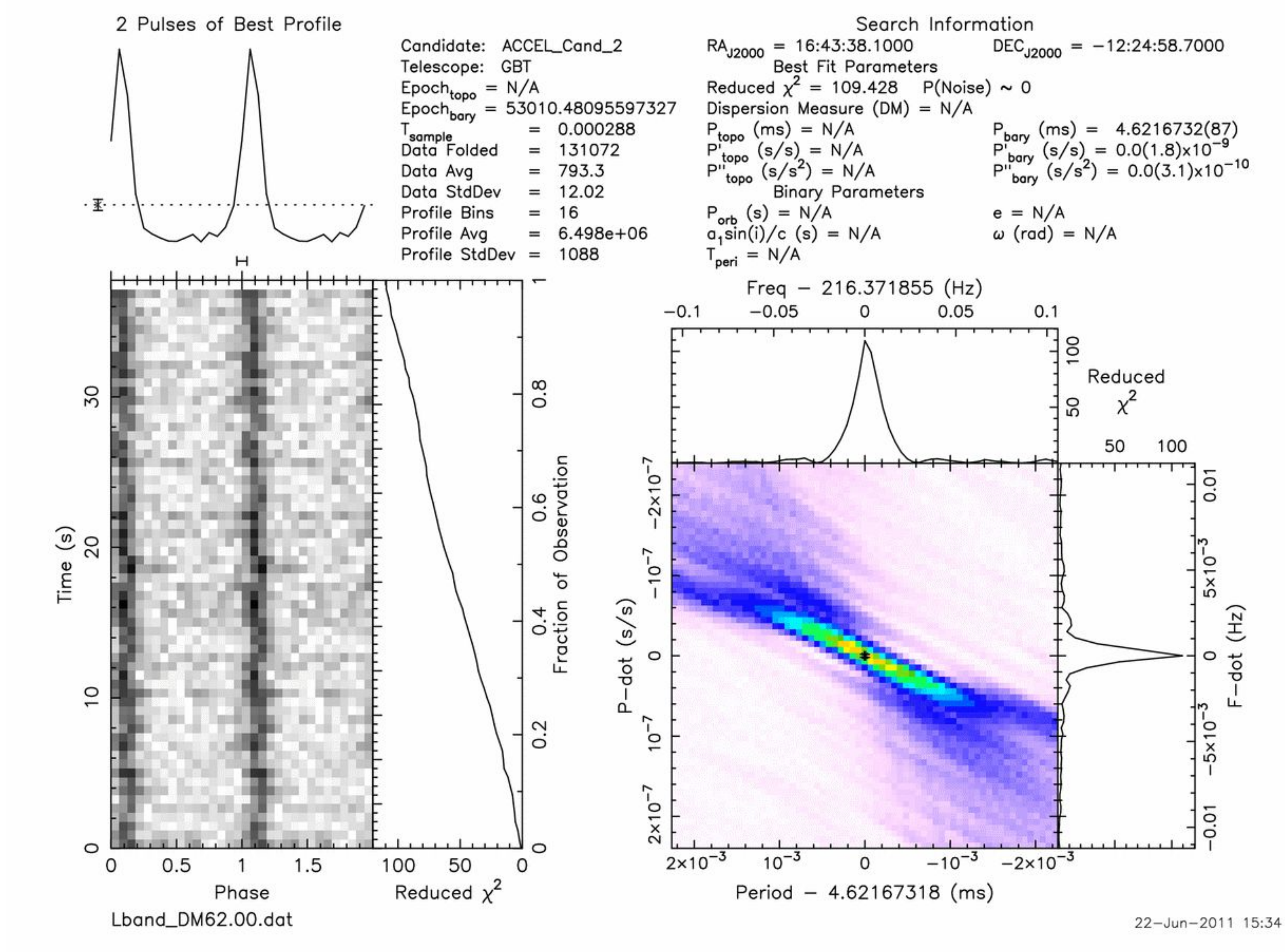
- `prepfold` can fold time-series (*.dat files), subbands (*.sub?? files), or rawdata files. Many ways to specify period (`-p`) / freq (`-f`) etc.
- Folding time-series is very fast and is useful to decide which candidates to fold the raw data
- When you fold subbands and/or the raw data, make sure that you specify the DM (and choose the set of subbands with closest DM).
- For modern raw data, using 64 or more subbands (`-nsub`) is a good idea for folding (to see narrow band RFI and scintillation better)
- If RFI is bad, can zap it using `show_pfd` or re-fold using `-mask`

```
> prepfold -dm 62.0 -accelcand 1 -accelfile \
Lband_DM62.00_ACCEL_0.cand \
subbands/Lband_DM72.00.sub??
```

```
> prepfold -n 64 -nsub 96 -p 0.004621638 -dm 62.0 \
GBT_Lband_PSR.fil
```

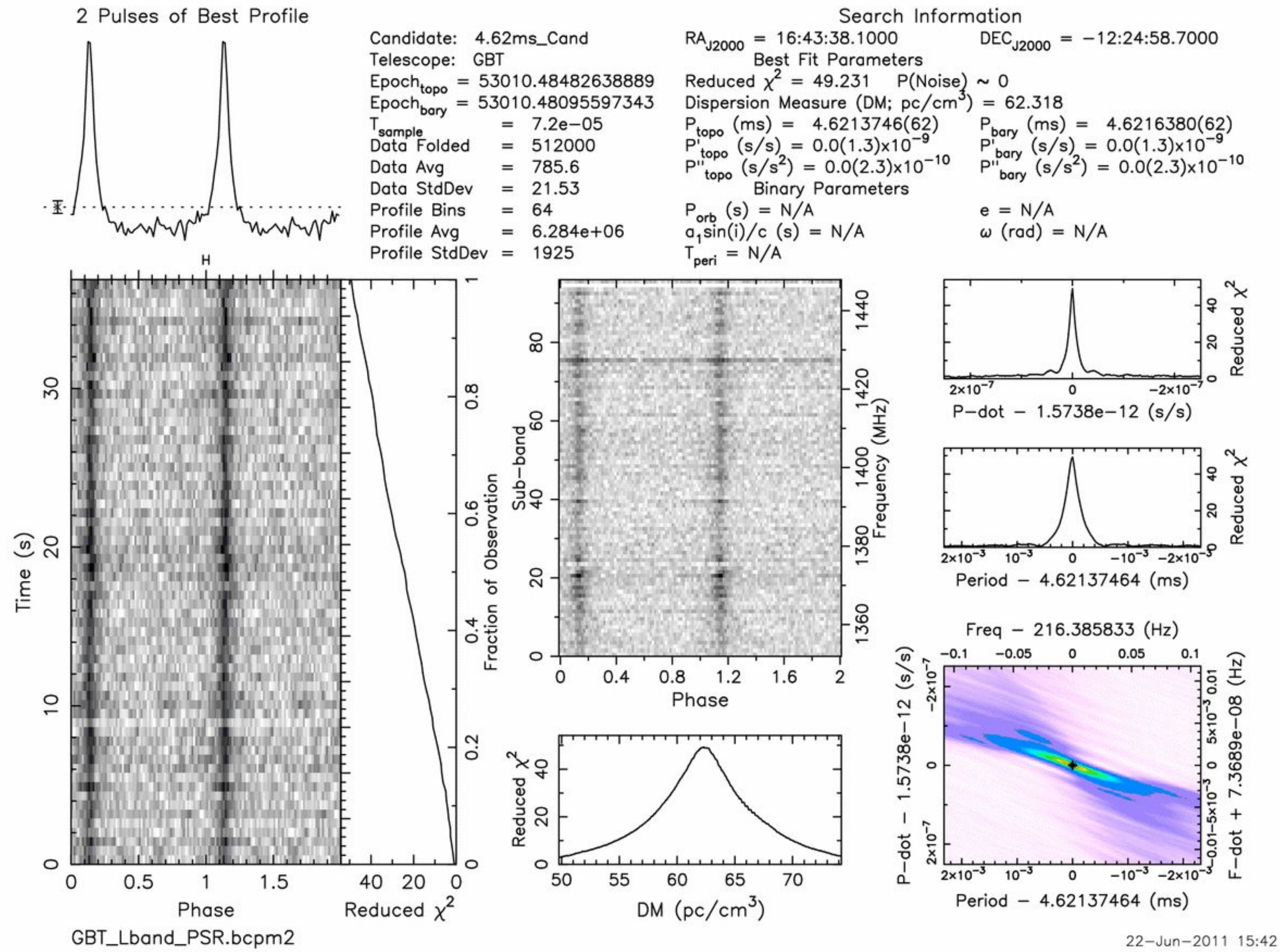
Pulsar! (timeseries)

```
> prepfold -accelcand 1 -accelfile \
Lband_DM62.00_ACCEL_0.cand Lband_DM62.00.dat
```



Pulsar! (raw data)

```
> prepfold -n 64 -nsub 96 -p 0.004621638 -dm 62.0 \
GBT_Lband_PSR.fil
```



Searching for Transient Bursts

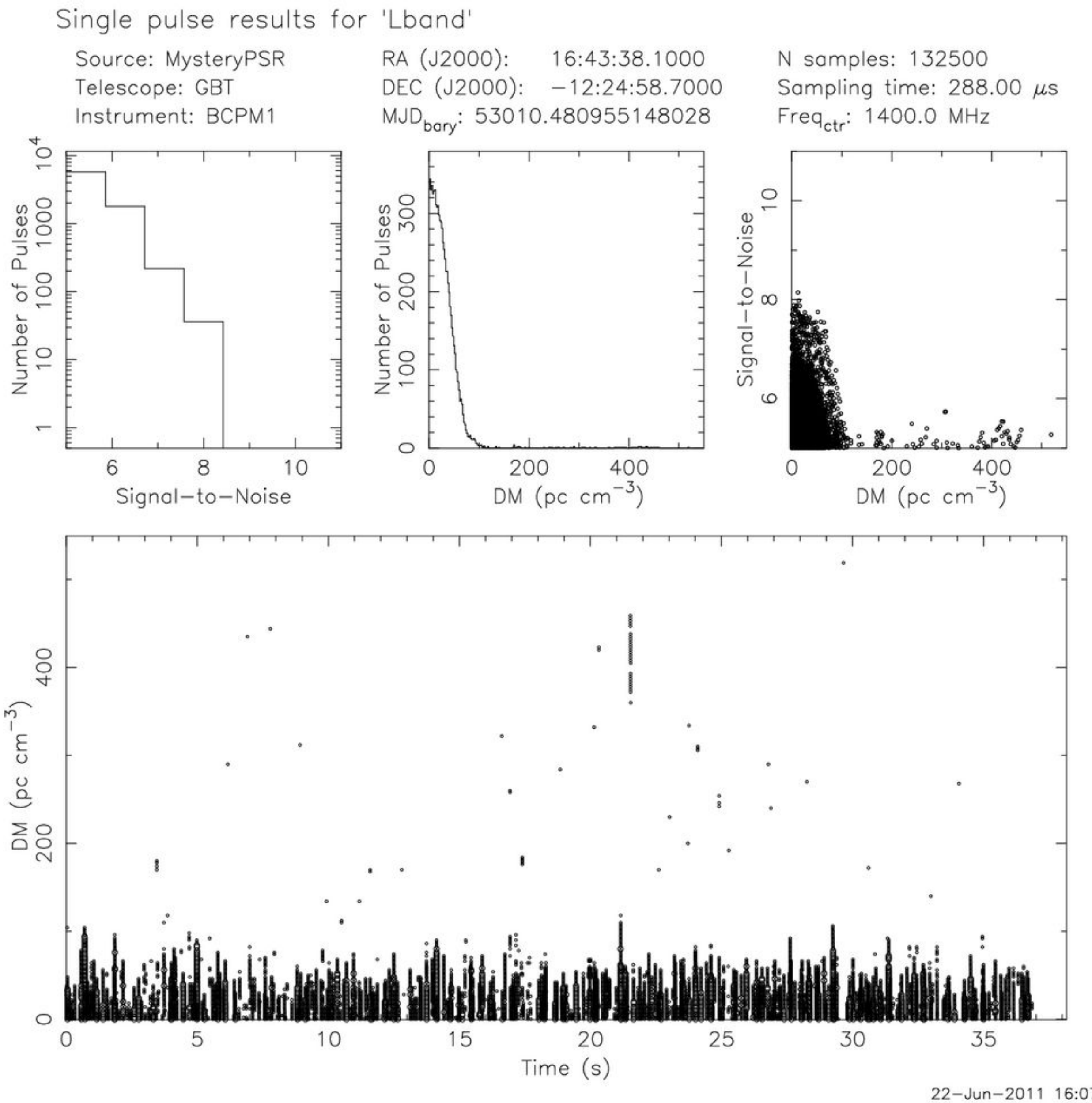
```
> single_pulse_search.py *.dat
```

- `single_pulse_search.py` conducts matched-filtering single-pulse searches using “boxcar” templates.
- `--fast` can make things about a factor of 2 faster, but only use it if the data are well-behaved (relatively constant power levels)
- If there are very strong pulses in your data, they can look like RFI. For those cases, turn off bad-block finding (`--nobadblocks`)
- Generates *.singlepulse files that are ASCII and a single-pulse plot
- Can regenerate a plot using (for instance)

```
> single_pulse_search.py *DM1??.??*.singlepulse
```

- Can choose start and end times as well (`--start` and `--end`)

Searching for Transient Bursts



Making TOAs from the discovery obs

- `get_TOAs.py` needs to be run on a prepfold file of either a topocentric time series or a fold of raw data. The fold must have been made either using a parfile (use `-timing`) or with the (`-nosearch`) option.
- The must be either a single gaussian (`-g` FWHM), an ASCII profile (i.e. a bestprof file from `prepfold`) or a multi-gaussian-template (derived using `pygaussfit.py`: "`-g template.gaussian`")
- `-n` is the number of TOAs (and must factor the number of parts (`-npart`) from the `prepfold` file
- `-s` is the number of subband TOAs to generate (1 is default)

```
> get_TOAs.py -g 0.1 -n 20 newpulsar.pfd
```

Now try it from scratch...

- There is another sample data set (with mystery pulsar) here:

  http://www.cv.nrao.edu/~sransom/Parkes_70cm_PSR.fits

- Command history (and properly formatted `dedisp.py` file) for this tutorial can be found here:

  http://www.cv.nrao.edu/~sransom/GBT_Lband_PSR_cmd_history.txt

  http://www.cv.nrao.edu/~sransom/dedisp.py

- Note the new PRESTO FAQ! Check it out!

- Let me know if you have any problems or suggestions!

Scott Ransom <sransom@nrao.edu>