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

- 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`)
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_4bit.fil
Assuming the data is a SIGPROC filterbank file.


1: From the SIGPROC filterbank file 'GBT_Lband_PSR_4bit.fil':
                  Telescope = GBT
                Source Name = Mystery_PSR
                    Backend = BPP
             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 = 0 of 1
          Beam FWHM (deg) = 0.147
       Spectra per subint = 480
          Starting subint = 0
         Subints per file = 0
         Spectra per file = 531000
      Time per subint (sec) = 0.03456
        Time per file (sec) = 38.232
```

- `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_4bit.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_4bit.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 = 480
   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 27840 points (2.00448 sec).
  Prime factors are:  2 2 2 2 2 2 3 5 29
  WARNING:  The largest prime factor is pretty big!  This will
            cause the FFTs to take a long time to compute.  I
            recommend choosing a different -time value.

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

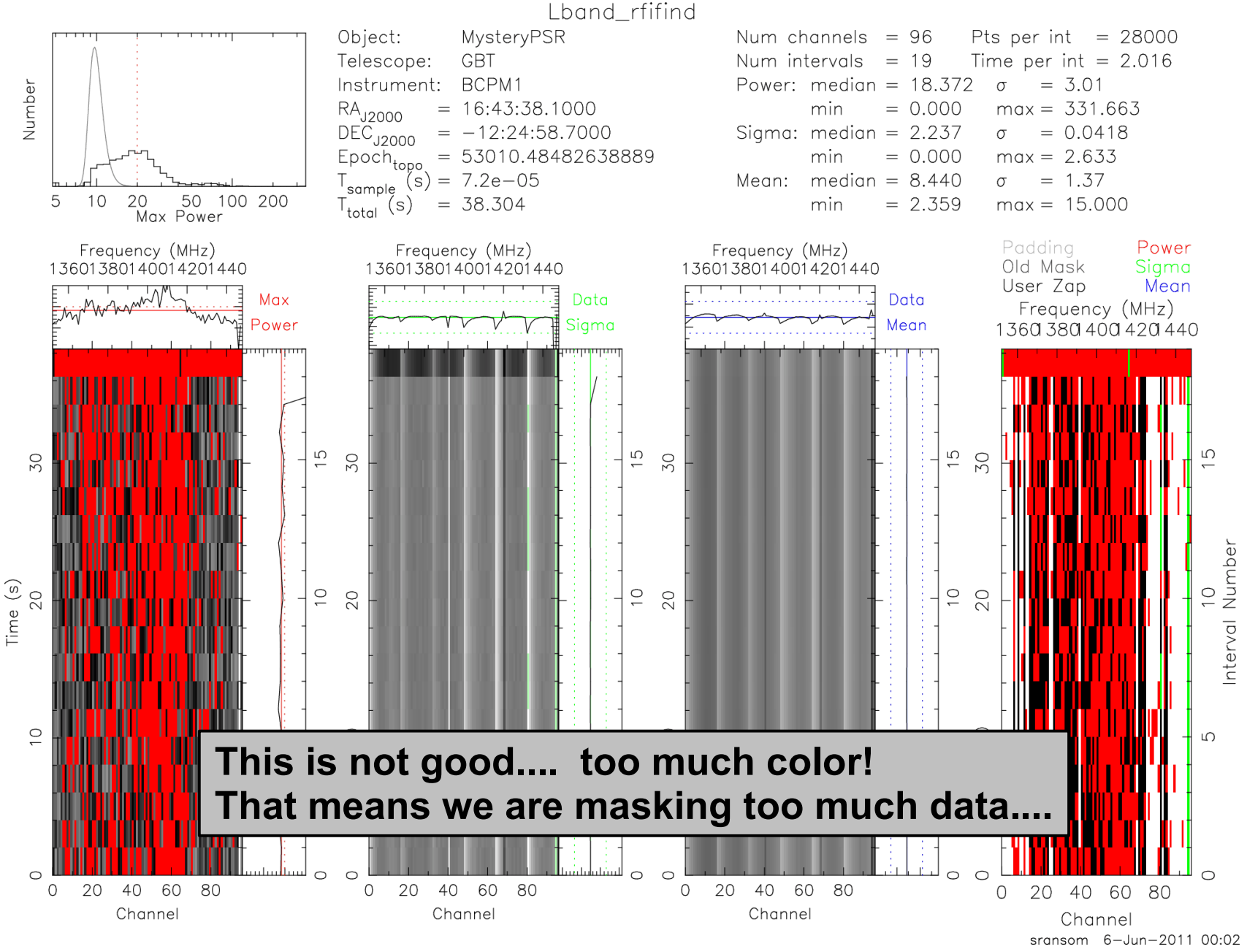
- `rfifind` identifies strong narrow-band and/or short duration broadband RFI
- Creates a “mask” (basename determined by “`-o`”) where RFI is replaced by median values
- All `PRESTO` programs automatically clip strong, transient, DM=0 signals (turn off using `-noclip`)
- Typical integration times (`-time`)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

Search for prominent RFI: 2

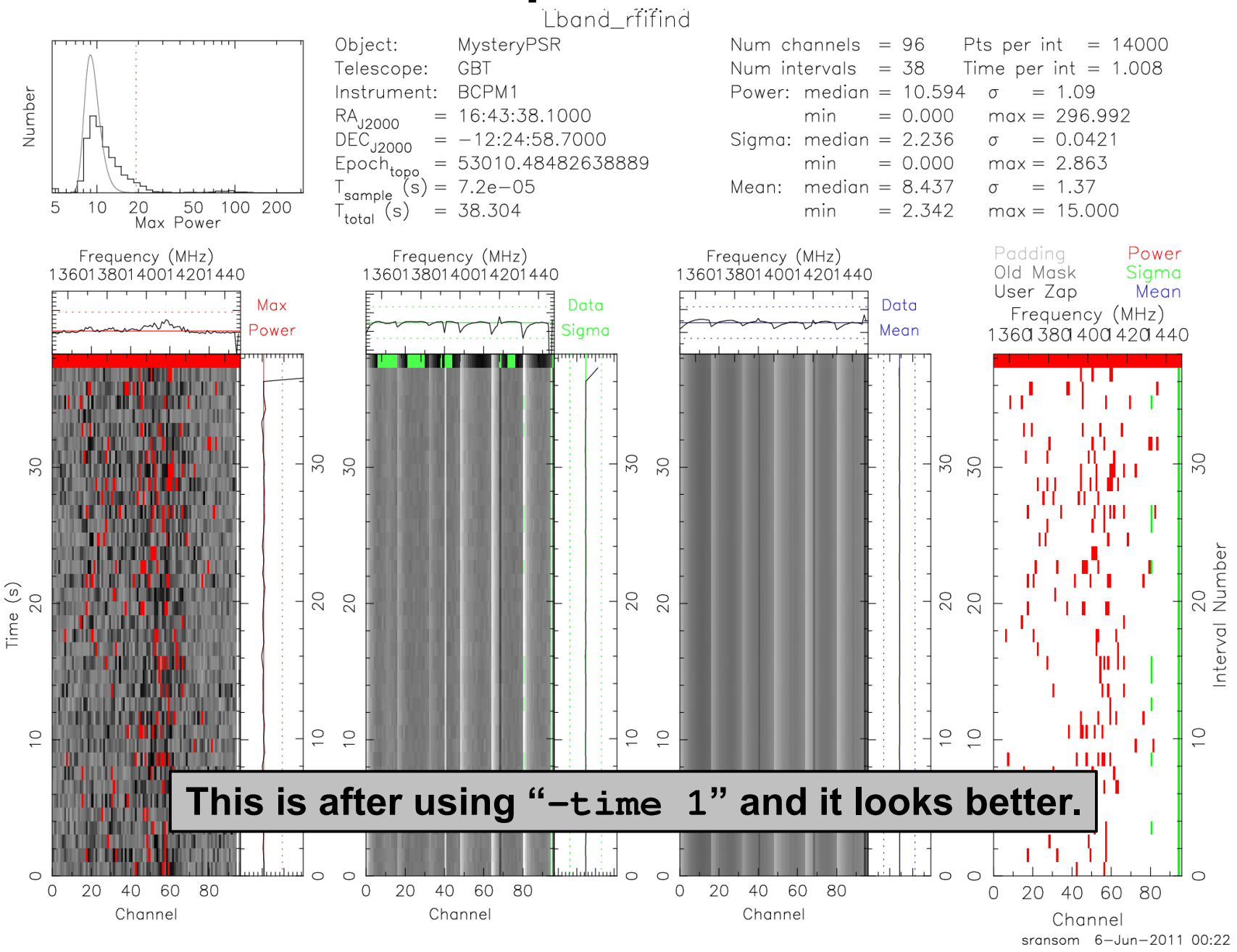
```
Amount Complete = 100%
There are 30 RFI instances.


Total number of intervals in the data:  1920

  Number of padded intervals:       96  ( 5.000%)
  Number of  good  intervals:     1561  (81.302%)
  Number of  bad   intervals:      263  (13.698%)

  Ten most significant birdies:
#  Sigma     Period(ms)      Freq(Hz)       Number
-------------------------------------------------
1  7.06      11.5532         86.5561        198
2  6.83      17.5832         56.8726        10
3  6.82      11.654          85.8078        195
4  6.81      11.6202         86.0572        189
5  6.80      17.6606         56.6232        10
6  6.68      11.5866         86.3067        166
7  6.31      11.52           86.8056        181
8  6.11      11.487          87.055         140
9  5.37      8.79158         113.745        4
10 5.34      11.7565         85.0595        12

  Ten most numerous birdies:
#  Number    Period(ms)      Freq(Hz)       Sigma
-------------------------------------------------
1  556       34.56           28.9352        4.81
2  333       34.8605         28.6857        4.73
3  325       17.28           57.8704        4.78
4  311       17.3548         57.6209        4.81
5  198       11.5532         86.5561        7.06
6  195       11.654          85.8078        6.82
7  189       11.6202         86.0572        6.81
8  181       11.52           86.8056        6.31
```

- 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



Search for prominent RFI: 4



Look for persistent low-level RFI

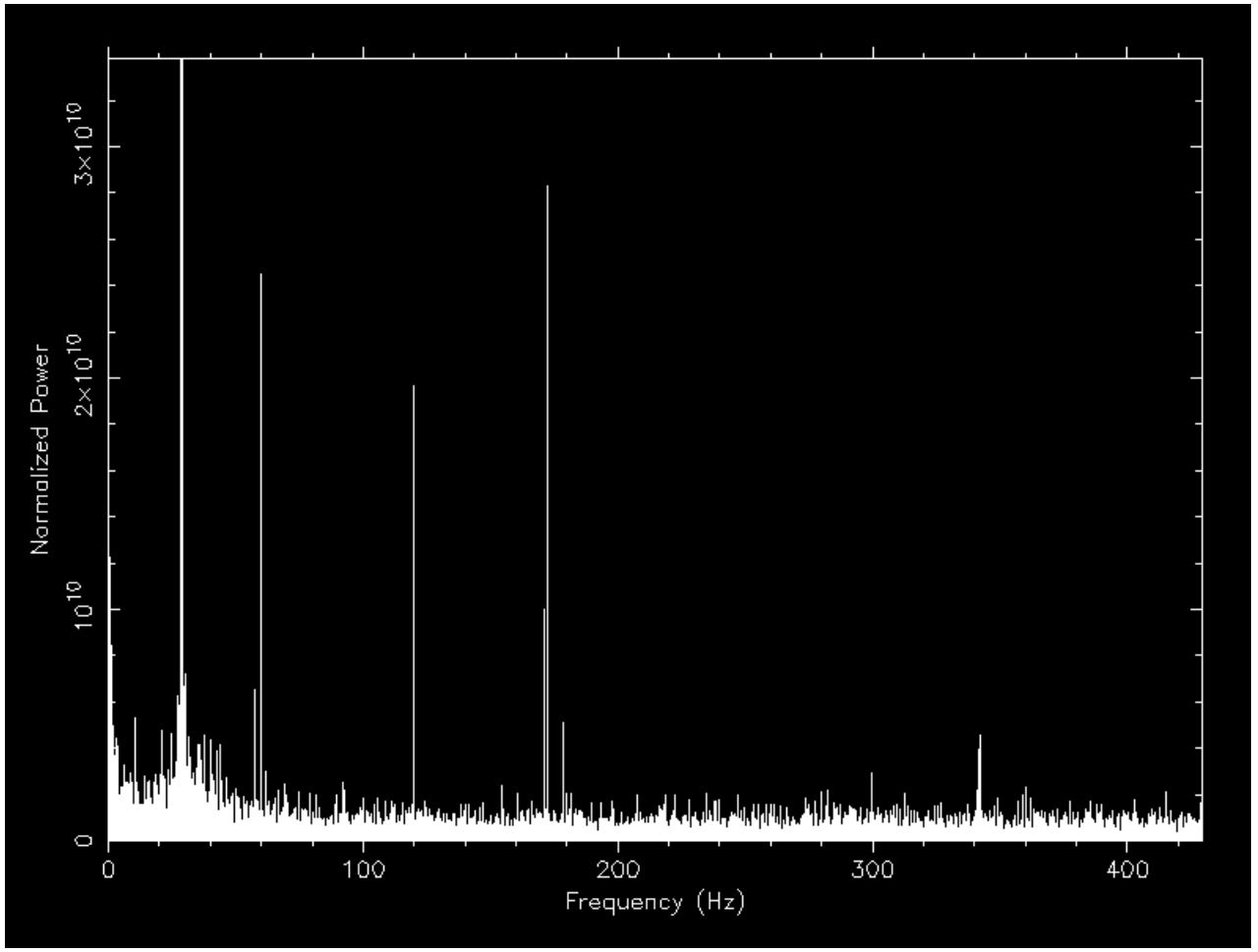
```
> prepdata -nobary -o Lband_topo_DM0.00 \
    -dm 0.0 -mask Lband_rfifind.mask \
    -numout 530000 GBT_Lband_PSR.fil
```

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

Assuming the data is from a GBT BCPM...
Reading Green Bank BCPM data from 1 file:
  'GBT_Lband_PSR.bcpm2'

BCPM input file information:
  Number of files = 1
     Points/block = 1000
  Num of channels = 96
 Total points (N) = 530000
 Sample time (dt) = 7.2e-05
   Total time (s) = 38.16

File  Start Block    Last Block     Points      Elapsed (s)      Time (s)           MJD           Padding
----  -----------  -----------  ----------  --------------  --------------  ------------------  ----------
 1              1          530      530000               0           38.16  53010.484826388885           0

Writing output data to 'Lband_topo_DM0.00.dat'.
Writing information to 'Lband_topo_DM0.00.inf'.
```

- `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`
- Since we will search these data (and FFT them), make sure that the resulting time-series has a "good" number of points (`-numout`)

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 fully interactively view a time-series or its power spectrum
- These commands are very useful for checking for data problems
- `realfft` requires that the time-series is easily factorable (and at least has 1 factor of '2'). Use the "`factor`" program to check.

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)
1	714.5	255319	5722.62	4	4.02(5)x10^3	0.249(3)	9.50(13)	0.0000(3)	0.00(50)	0.0(4.1)x10^5
2	172.0	14824.	3338.97	1	34.77(2)	28.76(1)	1097.50(50)	0.000(1)	0.0(2.0)	0.0(1.4)x10^4
3	20.17	224.56	547.63	2	16.669(2)	59.991(7)	2289.25(25)	0.0000(7)	0.0(1.0)	0.0(3.4)x10^3
4	10.55	81.48	105.43	4	5.8484(1)	170.987(3)	6524.88(13)	0.0000(3)	0.00(50)	0.0(6.0)x10^2
5	7.62	53.27	47.55	4	5.6024(1)	178.495(3)	6811.38(13)	0.0000(3)	0.00(50)	0.0(5.8)x10^2
6	6.27	34.91	11.13	1	1.46190(3)	684.04(1)	26103.00(50)	0.000(1)	0.0(2.0)	0.0(6.0)x10^2
7	2.33	17.09	3648.31	1	779(8)	1.28(1)	49.00(50)	0.000(1)	0.0(2.0)	0.0(3.2)x10^5

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)
1	1	24.98	316(25)	1.83e+16	4.001(35)	9.50	-13.80(42)	0.00	5.037(40)	0.790(11)
	2	5.78	19.4(6.2)	1.33e+16	6.00(15)	19.00	18.9(1.9)	0.00	1.29(16)	0.182(46)
	3	1.31	2.4(2.2)	3.51e+15	28.00(56)	28.50	69.8(9.6)	0.00	1.42(46)	0.10(13)
	4	125.1	7.83(13)x10^3	1.28e+13	37.4968(26)	38.00	1.132(12)	0.00	4.7122(80)	0.4912(23
2	1	81.65	3339(82)	7.58e+12	1097.4212(68)	1097.50	-0.038(53)	0.00	2.473(12)	0.5037(35
3	1	12.07	76(12)	3.3e+10	2289.285(44)	2289.25	0.52(33)	0.00	5.456(81)	0.462(23)
	2	21.82	242(22)	4.39e+10	4578.519(25)	4578.50	0.20(19)	0.00	5.195(45)	0.510(13)
4	1	7.46	30.8(7.8)	1.08e+10	6524.859(70)	6524.88	-0.89(53)	0.00	1.52(13)	0.491(37)
	2	4.67	13.4(5.2)	5.29e+09	13050.12(11)	13049.75	1.52(87)	0.00	4.51(19)	0.457(56)

- We “trick” `accelsearch` into find all of the main peaks (7 in this case)
- Will parse the output files to make our “birds” file
- “.inf” file is ASCII and is therefore human readable. Also in the ACCEL file.

Make a “birds” file

- Use `exploreffft` 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

```
sransom@neve:~/presto/src$ cat /dev/shm/tutorial/Lband.birds
#Freq      Width  #harm  grow?  bary?
1.2        0.01   2      0      0
5.55555    0.05   12     0      0
11.11111   0.12   7      0      0
50.0       0.04   2      0      0
60.0       0.1    2      1      0
13.88985   0.05   30     0      1
sransom@neve:~/presto/src$
```

- Need to use these same columns
- “Freq” and “Width” in Hz and the number of harmonics to zap
- If the zapping regions should get bigger with harmonic, set “grow?” to 1
- If the frequency is barycentric (i.e. a known pulsar), set “bary?” to 1

Convert the “birds” file to a zaplist

- 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).

Determining a De-Dispersion Plan

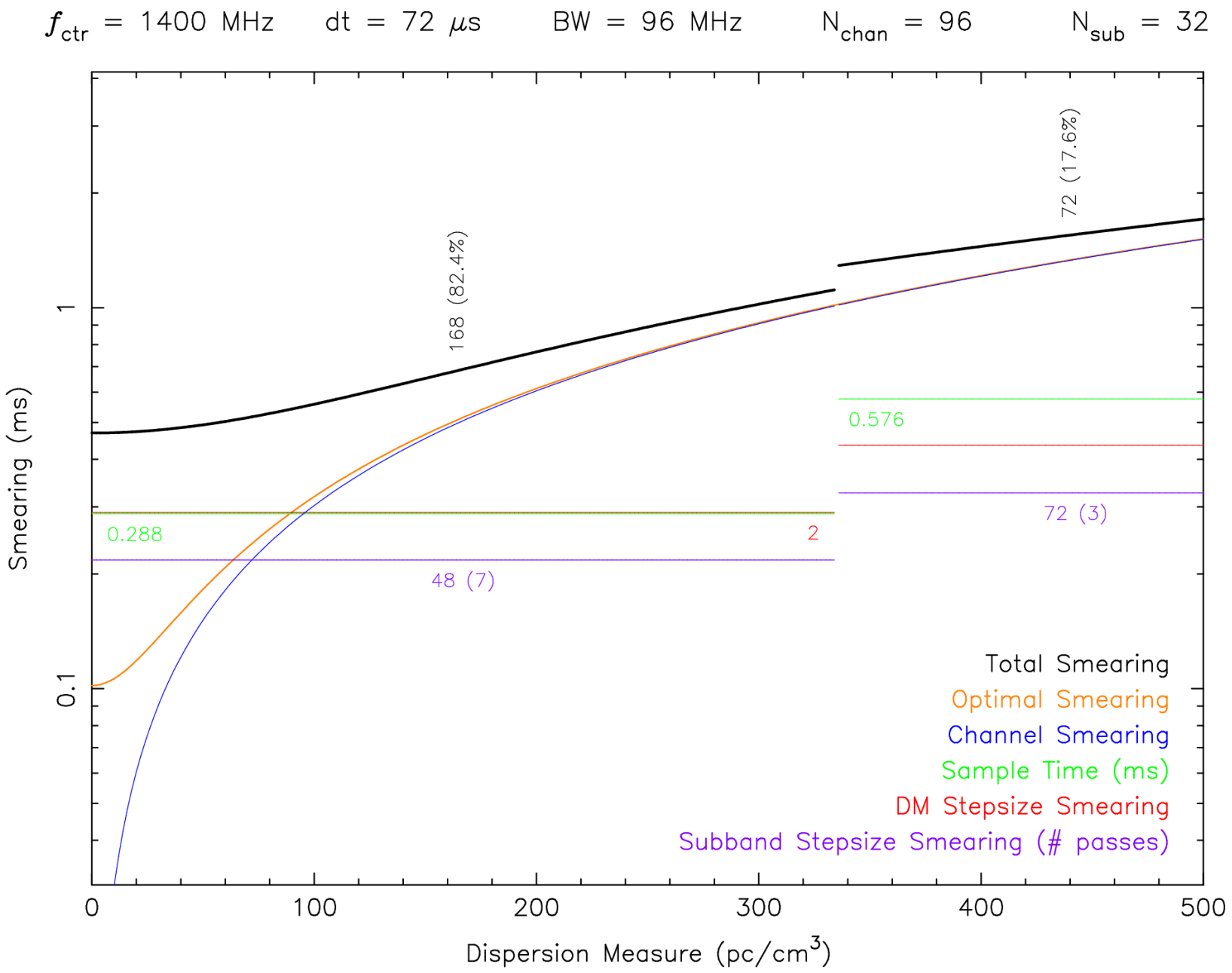
```
> DDplan.py -d 500.0 -n 96 -b 96 -t 0.000072 \
-f 1400.0 -s 32 -r 0.5
```

```
>
>
> 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

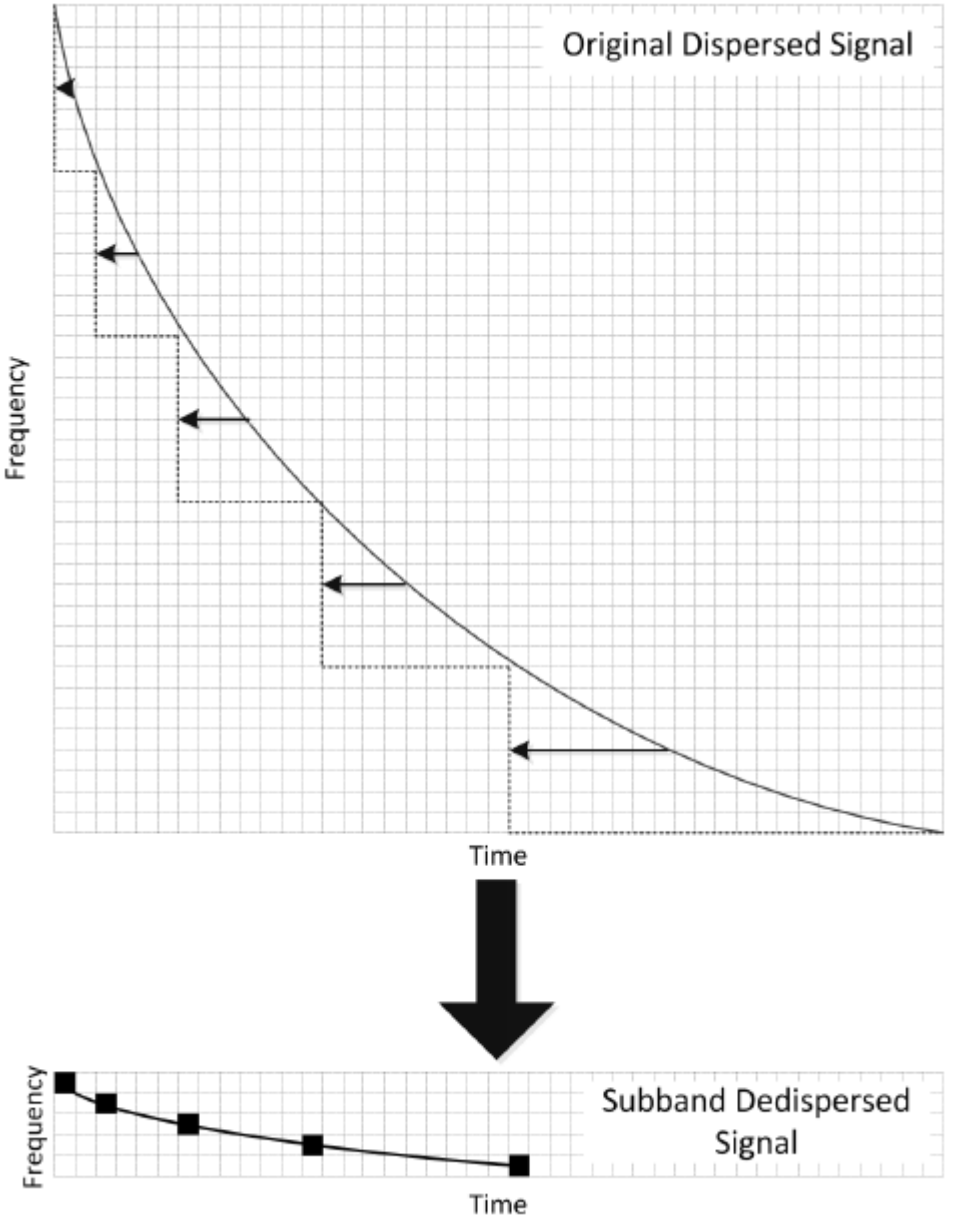
- `DDplan.py` determines near-optimal ways to de-disperse your data to maintain sensitivity to fast pulsars yet save CPU and I/O time
- Assumes using `prepsubband` to do multiple-passes through the data using "subband" de-dispersion
- Specify command line information from `readfile`

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 -numout 132500 -downsamp 4 -mask
Lband_rfifind.mask -o Lband GBT_Lband_PSR.fil
```

- That command line comes from the first call of the first de-dispersion 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

- You need to call `prepsubband` as many times as there are “calls” in the de-dispersion plan
- If you have a parallel computer (and very long observations), you can use the fully parallel program `mpiprepsubband` to have one machine read the data, broadcast it to many other CPUs and then each CPU effectively makes a “call”
- The `dedisp.py` script in `$PRESTO/python` can help you automate this process (and generates subbands as well, which can be used to fold candidates and see the DM-curve much faster than by folding raw data). When the file has been edited, do: `python dedisp.py`

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*
```

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

```
> ls *.dat | xargs -n 1 realfft
> ls *.fft | xargs -n 1 zapbirds -zap \
-zapfile Lband.zaplist -baryv -5.69726e-05
```

- Remember that we can get the barycentric value by running a fake `prepdata` or `prepfold` command on the raw data. The value we use is “The average topocentric valocity”
- Now we are ready to run `accelsearch` on the `*.fft` files
- Note that if you are using short time series (like we are), you can use accelsearch to do its own FFTing and zapping. 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 and/or 16.
- "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 < 300 or so (beyond that non-linear acceleration start to creep in).
- You can use `xargs`: `ls *.fft | xargs -n 1 accelsearch` ...

Sifting the periodic candidates

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

- `ACCEL_sift.py` is in $PRESTO/python 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 2 -accelfile \
Lband_DM62.00_ACCEL_0.cand Lband_DM62.00.dat
```

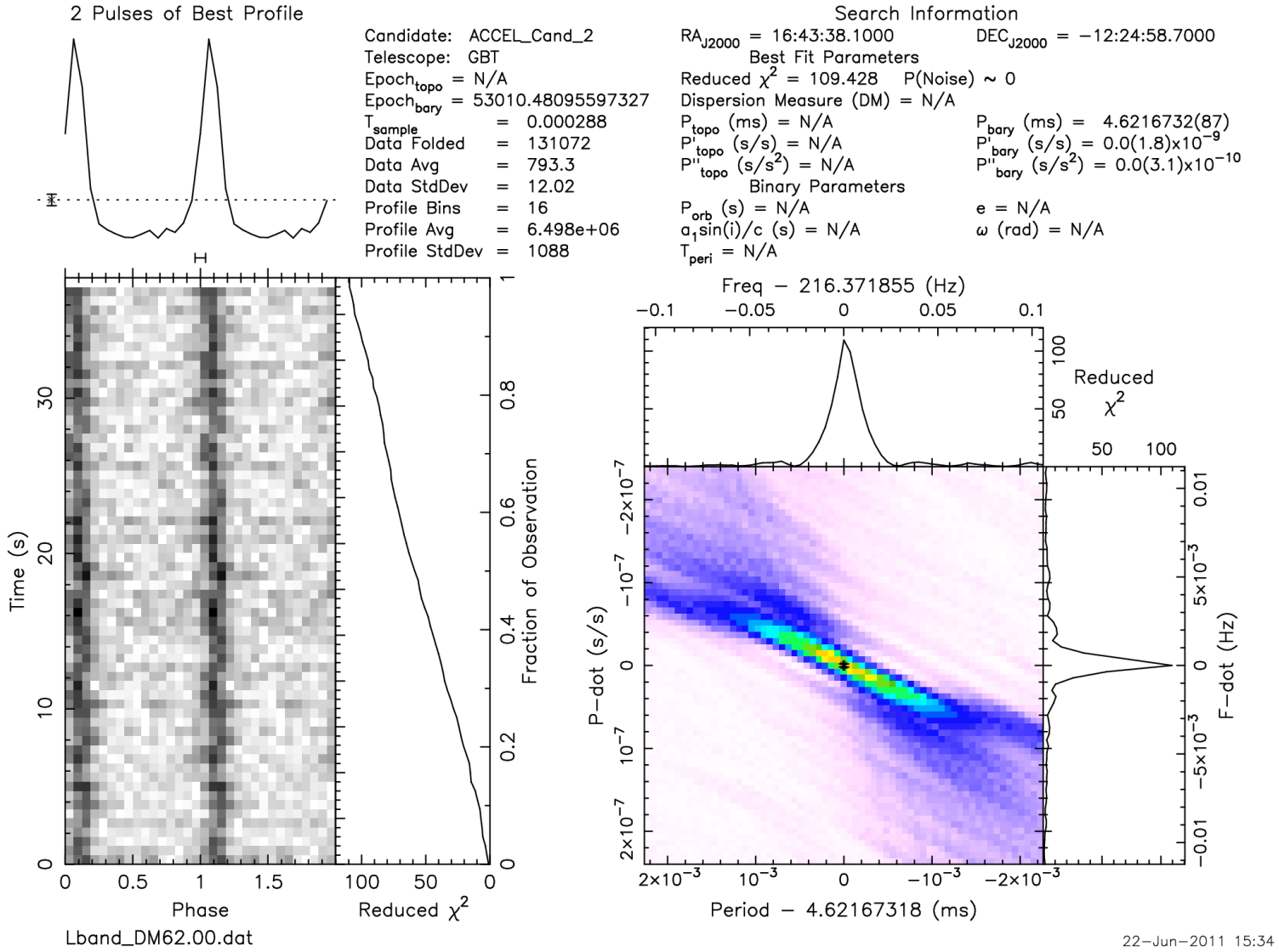
- `prepfold` can fold time-series (*.dat files), subbands (*.sub?? files), or rawdata files. Many ways to specify period (`-p`) / freq (`-f`) etc.
- Folding time-series is very fast and is useful to decide which candidates to fold the raw data
- When you fold subbands and/or the raw data, make sure that 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 2 -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
```

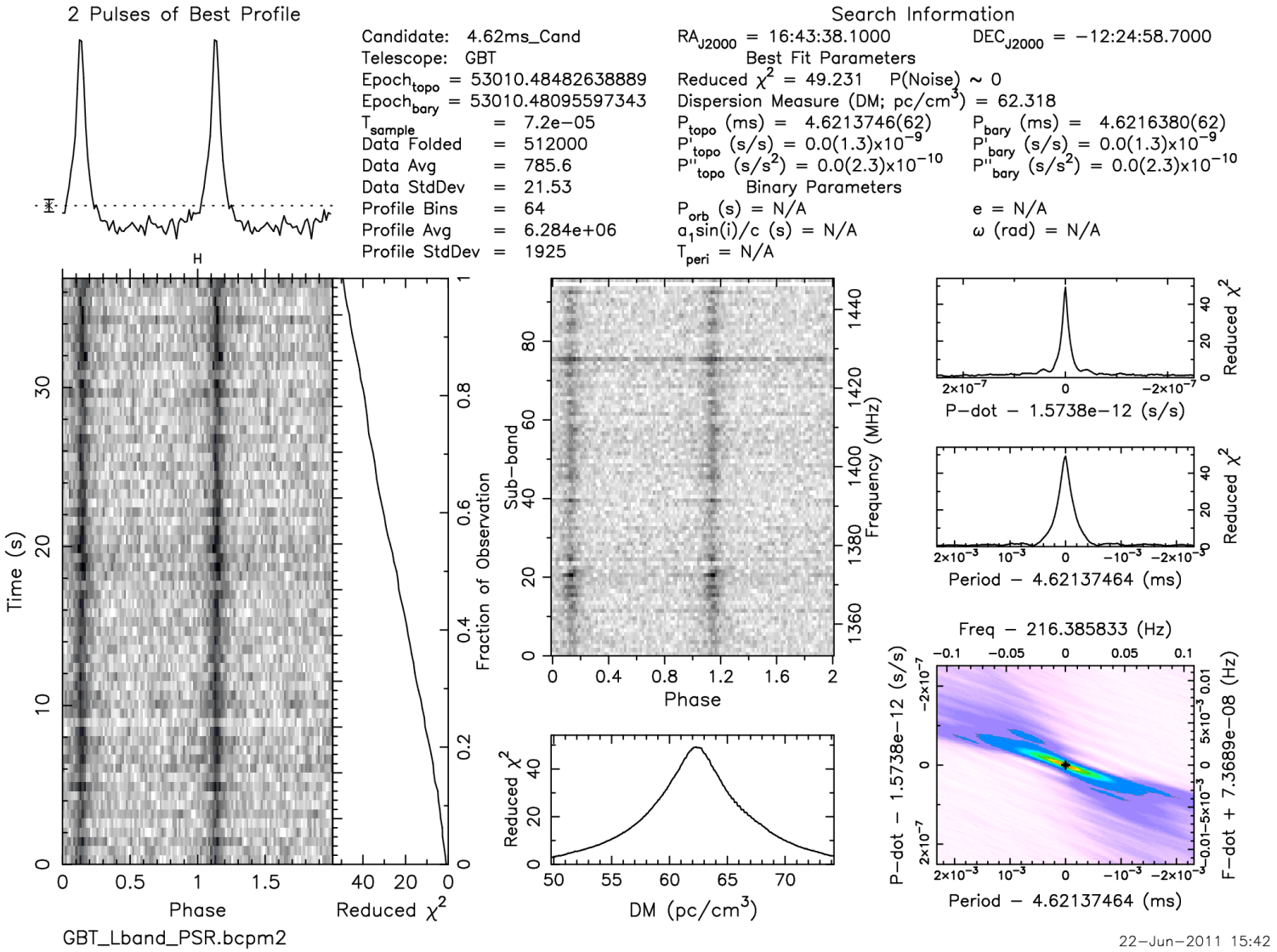
Pulsar! (timeseries)

```
> prepfold -accelcand 2 -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

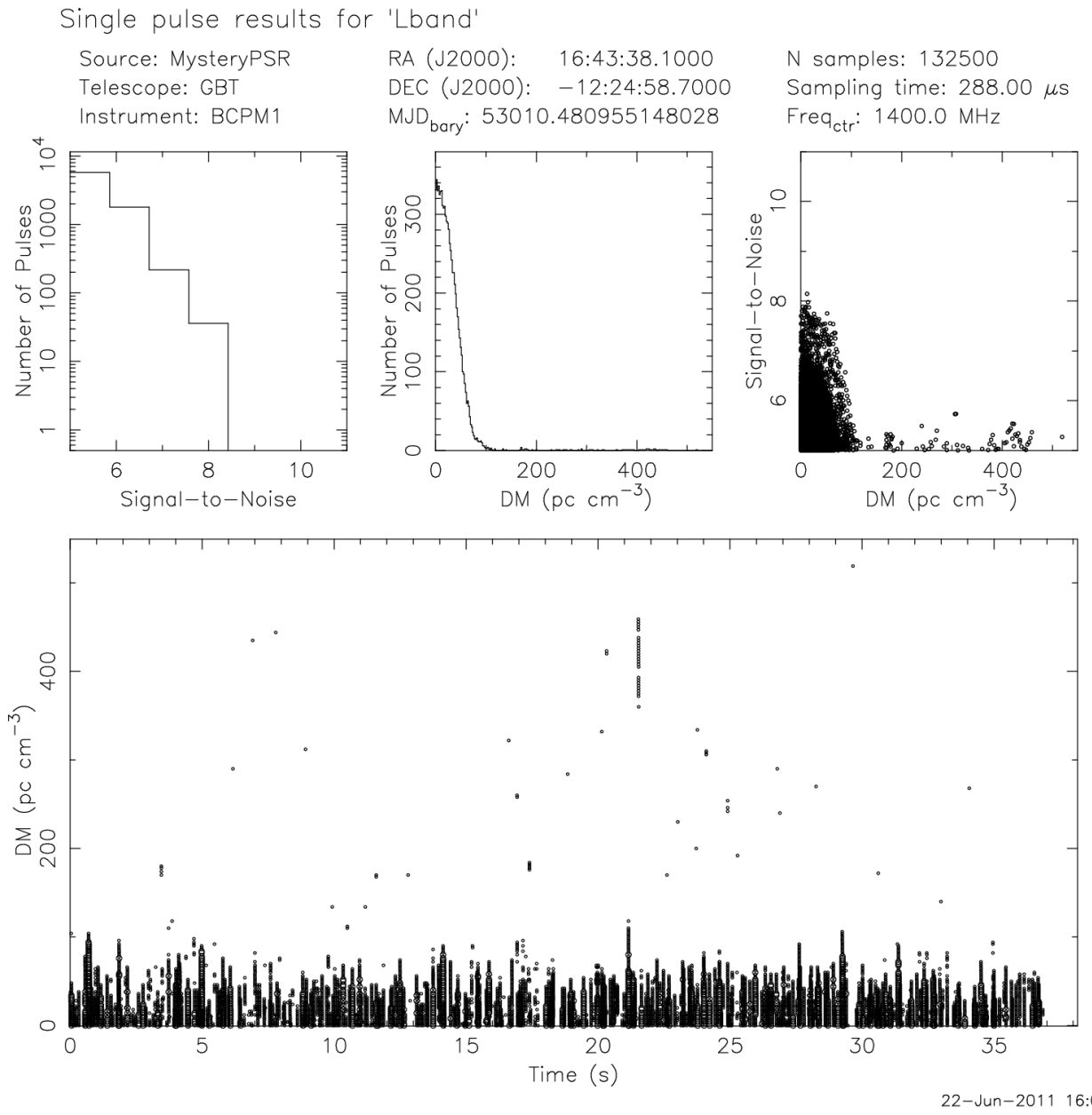
```
> single_pulse_search.py *.dat
```

- `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)
- 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 for this tutorial can be found here:

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

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

Scott Ransom <sransom@nrao.edu>