Manual de instalação do PRESTO: PulsaR Exploration and Search TOolkit

Ana Rafaely Medeiros de Oliveira

Universidade Federal da Paraíba

Colaboração BINGO

2024

Sumário

1 Introdução ao PRESTO

O **PRESTO**[1]é uma suíte de software poderosa, desenvolvida por Scott Ransom, projetada para a pesquisa e análise de pulsares. Criado do zero e disponível sob a licença GPL (v2), o PRESTO é uma ferramenta essencial para astrônomos e pesquisadores interessados no estudo de pulsares, especialmente pulsares binários de milissegundos.

2 Objetivo Principal

O principal objetivo do PRESTO é facilitar a detecção e análise de pulsares binários de milissegundos através de observações prolongadas de aglomerados globulares. No entanto, suas funcionalidades são versáteis e vão além desse escopo. O software também é amplamente utilizado em diversas pesquisas, incluindo a análise de integrações curtas e o processamento de grandes volumes de dados de raios-X.

3 Características Técnicas

- **Linguagem de Programação**: O código-fonte do PRESTO é majoritariamente escrito em ANSI C, com várias rotinas mais recentes implementadas em Python.

- **Licença**: GPL (v2), o que garante que o software é livre para uso e modificação, respeitando os termos da licença.

4 Aplicações do PRESTO

- **Detecção de Pulsar Binário de Milissegundos**: Ideal para observações prolongadas de aglomerados globulares, permitindo uma pesquisa mais eficiente desses pulsares.

- **Integrações Curtas**: Utilizado para a análise de dados em observações de integração curta.

- **Processamento de Dados de Raios-X**: Adequado para lidar com grandes volumes de dados provenientes de observações de raios-X.

[1] PulsaR Exploration and Search TOolkit, em português, Kit de exploração e pesquisa Pulsar

5 Processamento de Dados

Segundo Scott, o *PRESTO* foi desenvolvido com foco em portabilidade, facilidade de uso e eficiência de memória. Atualmente, ele é capaz de processar dados brutos de várias máquinas e formatos de pulsar, incluindo:

- Dados no formato de pesquisa *PSRFITS* (como os gerados pelo *GUPPI* no *GBT*, *PUPPI* e pelos Mock Spectrometers em Arecibo, além de muitos dados novos e arquivados do Parkes).
- Formatos de banco de filtros de 1, 2, 4, 8 e 32 bits (em ponto flutuante) da *SIGPROC*.
- Séries temporais compostas por dados em ponto flutuante de precisão única (ou seja, 4 bytes), acompanhados por um arquivo de texto ".inf" que os descreve.
- Tempos de chegada de fótons (ou eventos) em formatos ASCII ou binários de precisão dupla [1].

No entanto, alguns formatos anteriormente suportados não estão mais incluídos:

- Processador Pulsar de Banda Larga Arecibo (WAPP) em Arecibo.
- Formatos de banco de filtros de 1 bit do Parkes e Jodrell Bank.
- SPIGOT no GBT.
- Máquina Pulsar Berkeley-Caltech (BCPM) no GBT.

Para processar esses formatos descontinuados, é recomendado verificar o ramo "clássico" do PRESTO, que não está mais em desenvolvimento ativo. Alternativamente, o *DSPSR* pode ser usado para converter esses formatos para o formato de banco de filtros `SIGPROC` ou, ainda melhor, para o formato de pesquisa *PSRFITS*. O *DSPSR* pode ser obtido em [2]. Se você necessita de suporte para algum desses formatos no PRESTO moderno, entre em contato para que possamos avaliar a possibilidade de implementação [1].

6 Funcionalidades do Software

O PRESTO é composto por inúmeras rotinas projetadas para lidar com três áreas principais da análise de pulsares:

- **Preparação de Dados**: Detecção (`rfifind`) e remoção (`zapbirds`) de interferências, de-dispersão (`prepdata`, `prepsubband` e `mpiprepsubband`), barycentrização (via TEMPO).
- **Busca**: Busca de aceleração e jerk no domínio de Fourier (`accelsearch`), busca de pulsos únicos (`single_pulse_search.py`) e busca de modulação de fase ou bandas laterais (`search_bin`).
- **Dobragem**: Otimização de candidatos (`prepfold`) e geração de Tempo de Chegada (TOA) (`get_TOAs.py`).
- **Miscelânea**: Exploração de dados (`readfile`, `exploredat`, `explorefft`), planejamento de de-dispersão (`DDplan.py`), conversão de datas (`mjd2cal`, `cal2mjd`), bibliotecas Python para pulsares/astrofísica, criação de pulso médio, estimativa de densidade de fluxo, entre outros.
- **Ferramentas Pós-Busca de Pulsos Únicos**: Algoritmo de agrupamento (`rrattrap.py`), produção de gráficos diagnósticos de pulsos únicos (`make_spd.py`, `plot_spd.py` e `waterfaller.py`).

Além disso, são fornecidos muitos utilitários adicionais para várias tarefas frequentemente necessárias ao trabalhar com dados de pulsares, como conversões de tempo, transformadas de Fourier, exploração de séries temporais e FFT, inversão de bytes, entre outros [1][3][4][5].

7 Instalação do PRESTO no Linux

Este tutorial é direcionado para a instalação do PRESTO no sistema operacional Linux. A instalação será realizada seguindo o processo padrão, conforme descrito no arquivo de referência do PRESTO.

O PRESTO tem alguns pré-requisitos e deve funcionar em um sistema semelhante ao Debian/Ubuntu. Portanto, certifique-se de instalar os seguintes pacotes:

- `FFTW3`
- `PGPLOT`
- `TEMPO`
- `GLIBv2`

- CFITSIO

Iniciaremos com a instalação principal do PRESTO e, posteriormente, abordaremos a instalação dos pacotes adicionais. Para qualquer duvida de erros pode verificar no meu github PULSAR_presto e o github original PRESTO .

7.1 Pré instalação

Instale os pacotes indicados a abaixo no seu computador, se você for linux, cole no terminal, ou ainda, se prefere pode instalar em algum ambiente conda.

Coloque no terminal (bash), o comando abaixo

```
Bash
sudo apt-get install git build-essential libfftw3-bin libfftw3-dev
```

```
Bash
sudo apt-get install pgplot5 libglib2.0-dev libcfitsio-bin libcfitsio-dev
    libpng-dev
```

```
Bash
sudo apt-get install latex2html gfortran tcsh autoconf libx11-dev python3-dev
    python3-numpy python3-pip
```

Certifique-se de que sua variável de ambiente PRESTO aponte para o PRESTO git checkout de nível superior. E certifique-se de que `$PRESTO/lib` e `$PRESTO/bin` não estejam em suas variáveis de ambiente `PATH` ou `LD_LIBRARY_PATH` ou `PYTHONPATH` pois pode ocasiona problemas para executa-los.

Que nesse caso, você precisará utilizar o comando `nano /.bashrc` para verificação se está de acordo com o indicado acima. Portanto, digite no terminal.

```
Bash
nano ~/.bashrc
```

Posteriormente as alterações que for necessário, utilize `Ctrl + 0` depois `Ctrl + X` e por fim `Y`, posteriormente você voltará para o terminal.

Para atualizar, você precisará digitar no terminal

```Bash
source ~/.bashrc
```

Provavelmente também é uma boa ideia limpar suas compilações anteriores. Basta entrar no diretório`src` e fazer um `make clean`, depois você volta o caminha fazendo o comando `cd .. .`

Veja o passo a passo:

```Bash
cd /presto/src
```

```Bash
make clean
```

```Bash
cd ..
```

No documento original ele da duas opções de para compilação, o Python virtual ou Conda ativado, no meu caso utilizei o um ambiente conda, por já ter mais domínio.

7.2 Ambiente conda

Para você manter seu código clean, é interessante você utilizar um ambiente virtual para a instalação dos próximos pacotes, com isso, instale o conda no se computador, após a sua instalação, crie o ambiente e configure, como:

```Bash
conda create --name presto_env python=3.10
conda activate presto_env
conda install -c matplotlib numpy pandas scipy ipykernel
python -m ipykernel install --user --name presto_env --display-name "presto_env
    "
```

É importante verificar no arquivo se você está instalando os pacotes no ambiente virtual criado, para não ter conflitos.

Agora edite o arquivo **`/.bashrc`** com o comando **`nano /.bashrc`** e coloque na última linha o conteúdo abaixo:

```Bash
alias jupyter-notebook="~/.local/bin/jupyter-notebook --no-browser"
```

Verifique no se você fez o processo corretamente, portanto, digite no terminal **jupyter** com o comando **jupyter-notebook**

7.3 Instalação

Faça a clonagem do **github**, digite no terminal

```Bash
git clone https://github.com/scottransom/presto.git
```

Agora você precisa instalar os 3 pacotes, assim, digite no terminal

```Bash
conda install meson meson-python ninja
```

Agora configure as compilações de código **C/Fortran**, digite no terminal

```Bash
cd presto
meson setup build --prefix=$CONDA_PREFIX
```

Para evitar possíveis problemas de vinculação e execução, digite no terminal

```Bash
cd presto/python
```

Abra o python no se terminal, assim execute o teste, indicado na linha abaixo.

```Python
>>> python check_meson_build.py
```

Caso contrário, recomendo tentar corrigir os problemas detectados e começar novamente executando os seguintes comandos no terminal.

```
Bash
meson compile -C build
meson install -C build
```

Finalmente, instale o **Python** codes via **pip**:

```
Bash
cd $PRESTO/python
pip install --config-settings=builddir=build .
```

Faça os testes rapidamente para ver se a maioria das coisas está funcionando fazendo:

```
Bash
cd $PRESTO
python tests/test_presto_python.py
python examplescripts/ffdot_example.py
python python/fftfit_src/test_fftfit.py
```

7.4 Pacotes Adicionais

7.4.1 FFTW

Para o sistema linux, use os pacotes `libfftw3-bin` e `libfftw3-dev`

Você precisará compilar, coloque no seu terminal

```
Bash
./configure --enable-shared --enable-single
```

Se você estiver usando um processador Intel moderno e tiver uma versão recente do GCC, poderá obter um desempenho muito melhor adicionando [6]:

```
Bash
--enable-sse --enable-sse2 --enable-avx --enable-avx2 --enable-fma
```

7.4.2 PGPlot

Para instalar de maneira simples basta

```
Bash
sudo apt-get install pgplot5
```

Caso, não consiga instalar, seguia a alternativa. Você precisa realizar as seguintes etapas. Essas etapas são descritas mais detalhadamente abaixo para cada sistema operacional.

Copie o arquivo de distribuição por **FTP** anônimo da Caltech. Este é um arquivo **tar** compactado: download **PGPLOT**.

```
Bash
sudo apt-get update
sudo apt-get install build-essential gfortran libpng-dev
```

Baixar o PGPLOT Baixe o PGPLOT do site oficial. Você pode fazer isso usando `wget`, digite no terminal:

```
Bash
wget ftp://ftp.astro.caltech.edu/pub/pgplot/pgplot522.tar.gz
```

Agora, extraia o arquivo, abaixo:

```
Bash
tar zxvf pgplot522.tar.gz
cd pgplot
```

Para Configurar o PGPLOT

Edite o arquivo **drivers.list** para especificar os drivers de dispositivos que deseja incluir. A configuração padrão geralmente é suficiente, mas você pode personalizá-la conforme necessário.

```
Bash
nano drivers.list
```

Compilar o PGPLOT, digite no terminal

```
Bash
cp makefile.std makefile
make
```

Caso aconteça algum erro, verifique o manual do PGPlot.[7]

7.4.3 TEMPO

Para instalar o tempo, você precisará digitar no terminal

```
Bash
cd presto
git clone git://git.code.sf.net/p/tempo/tempo
```

Você notará que aparecerá uma nova pasta chamada tempo no repositório do presto, para verificar use o comando

```
Bash
cd presto
./prepare
```

Instalar `C shell (csh)` Primeiro, instale o csh no seu sistema:

```
Bash
sudo apt-get update
sudo apt-get install csh
```

Instale as ferramentas de Desenvolvimento Primeiro, instale as ferramentas de desenvolvimento, incluindo **`autoconf`**, **`automake`** e **`libtool`**.

```
Bash
cd ~/presto/tempo
./prepare
```

Configure e compile:

```
Bash
export PATH=/usr/local/bin:$PATH
echo 'export PATH=/usr/local/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Agora basta verificar a instalação, digitando [8].

```
Bash
tempo
```

7.4.4 GLIBv2

Em máquinas Linux é quase certo que isso já esteja em seu sistema (verifique em `/usr/lib e /usr/include/glib*)`. Embora você possa precisar instalar um pacote de desenvolvimento simplista para ter os arquivos de inclusão necessários. No Ubuntu, o pacote que você precisa é: **`libglib2.0-dev`**. [9]

7.4.5 CFITSIO

Eu recomendo fortemente o uso de pacotes pré-compilados, mais uma vez (no Ubuntu eles são **`libcfitsio-bin e libcfitsio-dev)`**, porém, esta é uma instalação muito fácil via fonte. [10]

8 Manuseio inicial do PRESTO

Essa parte é basicamente uma tradução modificada do autor scott, nela alguns pontos importantes são abordados, assim como suas recomendações.

Para testar se o Presto está funcionando corretamente, você precisará de 1 GB livre no seu disco para excussão do código a seguir.

Para isso também precisará executar alguns comandos como **`> typewriter script`**

8.1 Dados

Baixe o exemplo abaixo do arquivo, seu tamanho é de 25MB (GBT_Lband_PSR.fil).

8.2 Examinando o formato de dados `readfile)`

Coloque no seu terminal **`readfile GBT_Lband_PSR.fil`**

Digite no terminal

```
Bash
readfile GBT_Lband_PSR.fil
```

Ao realizar esse comendo espera-se que tenha essa imagem de retorno **??**.

```
> 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
>
```

Figura 1 – Retorno esperado. Fonte: [11].

- **`readfile`** pode identificar automaticamente a maioria dos tipos de dados que PRESTO pode alça (em PRESTO `v2`, porém, isso é apenas `SIGPROC` banco de filtros e `PSRFITs`)
- Ele imprime os metadados sobre a observação

Digite no terminal

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

Ao realizar esse comendo espera-se que tenha essa imagem de retorno 2.

```
> 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
>
```

Figura 2 – Retorno esperado. Fonte: [11].

- **`rfifind`** identifica fort banda estreita e/ou curta duração da banda larga RFI
- Ele imprime os metadados sobre a observação
- Programas `PRESTO` automaticamente clipe forte, transitório, $\texttt{DM} = 0$

Digite no terminal

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

```
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.
```

Figura 3 – Retorno esperado. Fonte: [11].

- Verifique o número de coisas ruins intervalos. Normalmente deveria ser menos de 20%
- Mais significativo e mais números de passarinhos estão listados (para veja tudo, use `-rfixwin`).
- Cria vários arquivos de saída incluindo `..rfifind.ps` onde as cores são ruins (o vermelho é periódico RFI, azul/verde são questões estatísticas no domínio do tempo)

8.3 Atalhos para grandes observações

Às vezes, para observações longas ou com muitos canais, amostragem rápida ou muita **RFI**, o **rfifind** pode levar muito tempo para ser executado. Muitas vezes você pode mascarar a maior parte da RFI fazendo alguns atalhos e usando **-ignorechan**:

- Execute rfifind em um subconjunto de dados (um ou mais arquivos individuais).
- Ajuste os resultados, principalmente usando **-nocompute** e valores diferentes de **-freqsig** e **-imesig**, então os piores canais são marcados para mascaramento.
- Você pode então converter esse arquivo de pesos em uma lista de canais a serem ignorados usando a rotina **weights_to_ignorechan.py**, que também fornece um comando paz (de **PSRCHIVE**) para zapear arquivos dobrados feitos a partir dos dados.

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

8.3.1 Procure RFI persistente de baixo nível

Digite no terminal o seguinte comando

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

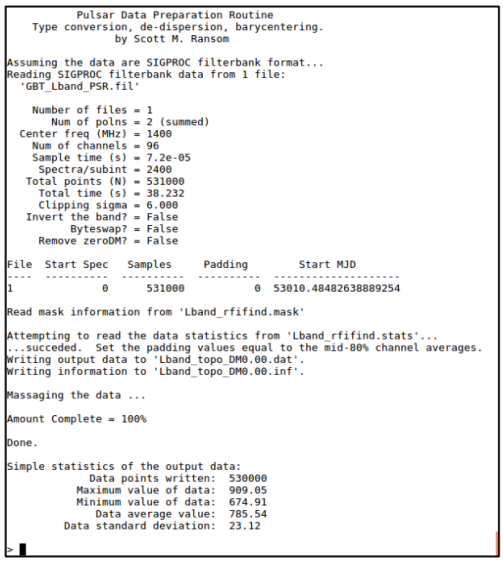

Figura 4 – Retorno esperado. Fonte: [11].

- **prepdata** dispersa um único série temporal. A bandeira **"-nobary"** diz ao PRESTO para não fazer baricentro a série temporal.
- Se você precisar de-dispersar várias séries temporais, use **prepsubband**.
- Costumávamos precisar definir o número de pontos (**-numout**) para torne-o um bom número redondo para **FFTing**, mas o PRESTO faz isso automaticamente agora.

8.4 Explore e FFT a série temporal

Digite no terminal o seguinte comando

```
Bash
exploredat Lband_topo_DM0.00.dat
```

```
Bash
realfft Lband_topo_DM0.00.dat
```

```
Bash
explorefft Lband_topo_DM0.00.fft
```

Você deve encontrar um resultado próximo a

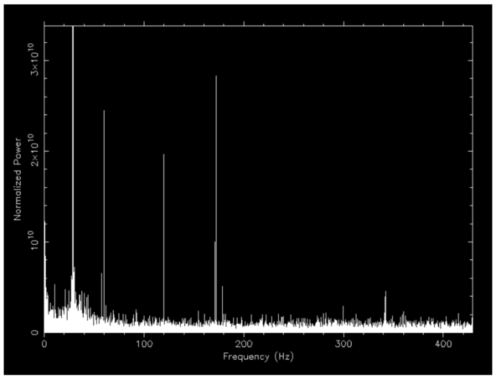

Figura 5 – Gráfico de Potencia junto a frequência Retorno esperado. Fonte: [11].

- `exploredat` e `explorefft` permite que você para visualizar interativamente um série temporal ou seu poder espectro (para encontrar RF.
- mudando o poder normalização (chave 'n') em `explorefft` é frequentemente muito útil.
- `realfft` exige que a série temporal é facilmente fatorável (e pelo menos tem 1 fator de '2'). Verifique usando `"fator"`.

8.5 Encontre a interferência periódica

Digite no terminal

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

Você deve encontra algo do tipo

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)	

Figura 6 – Resultado esperado. Fonte: [11].

- Nós "trick" o Accelsearch para encontrar interferências periódicas (ele encontrou 6 candidatos, com vários harmônicos em cada um).
- Essas informações serão usadas para criar um arquivo "birds".

- O arquivo ".inf" é ASCII legível por humanos (também é encontrado no arquivo ACCEL)

8.6 Determine o Plano da De-Dispersão

Digite no terminal

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

- DDplan.py determina maneiras quase ideais de dispersar seus dados para mantenha a sensibilidade a pulsares rápidos e ainda economize tempo de CPU e E/S
- Pressupõe o uso de pré-subband para fazer múltiplas passagens pelos dados usando desdispersão de "subbanda"
- Especifique as informações da linha de comando do arquivo readfile ou (Novo!) forneça o filename e DDplan.py determinarão os detalhes da observação
- A nova opção "-w" gravará um arquivo dedisp*py que você pode executar desdisperse seus dados (e edite conforme necessário, ou seja, para adicionar máscaras rfifind)

Será retornado a figura abaixo 7.

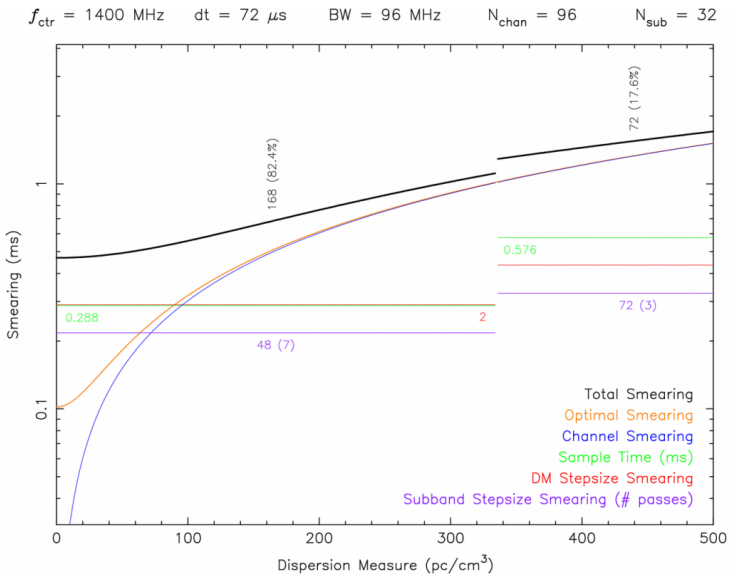

Figura 7 – Resultado esperado. Fonte: [11].

8.7 Subband De-Dispersion 2

Digite no terminal

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

Esse comando vem da primeira chamada da primeira linha do plano:

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

Figura 8 – Caption

- Execute o `prepsubband` tantas vezes quantas forem as "chamadas" no plano.
- Os formatos de arquivo aceitos para rodar o `prepsubband` são SIGPROC filterbank (".fil") e PSRFITS (".sf" ou ".fits").
- Se você tiver um computador paralelo (e observações longas), pode usar o `mpiprepsubband` totalmente paralelo para que uma CPU leia os dados, transmita para outras CPUs, que cada uma efetivamente faz uma "chamada".
- O script `dedisp.py` em `$PRESTO/examplescripts` pode ajudá-lo a automatizar esse processo (e gerar sub-bandas também, o que pode ser usado para dobrar candidatos mais rapidamente do que dobrar dados brutos). Quando o arquivo tiver sido editado, faça: `python dedisp.py`.

Referências

1 PRESTO. <https://github.com/scottransom/presto>. Accessed: 25 de julho de 2024. Citado 2 vezes nas páginas 4 e 5.

2 DSPSR. <https://dspsr.sourceforge.net/>. Accessed: 25 de julho de 2024. Citado na página 4.

3 FOURIER Techniques for Very Long Astrophysical Time-Series Analysis. <https://ui.adsabs.harvard.edu/abs/2002AJ....124.1788R/abstract>. Accessed: 25 de julho de 2024. Citado na página 5.

4 ANDERSEN BRIDGET C. SEARCH BY ORCID ; RANSOM, S. M. s. b. o. A fourier domain "jerk" search for binary pulsars. arXiv:1807.07900, 2018. Citado na página 5.

5 RANSOM SCOTT M. SEARCH BY ORCID ; CORDES, J. M. . E. S. S. A new search technique for short orbital period binary pulsars. arXiv:astro-ph/0210010, 2018. Citado na página 5.

6 FFTW. <https://www.fftw.org/>. Accessed: 25 de julho de 2024. Citado na página 9.

7 PGPLOT Graphics Subroutine Library. <https://sites.astro.caltech.edu/~tjp/pgplot/>. Accessed: 25 de julho de 2024. Citado na página 11.

8 TEMPO. <https://tempo.sourceforge.net/>. Accessed: 25 de julho de 2024. Citado na página 11.

9 GLIB – 2.0. <https://docs.gtk.org/glib/>. Accessed: 25 de julho de 2024. Citado na página 12.

10 CFITSIO - A FITS File Subroutine Library. <https://heasarc.gsfc.nasa.gov/fitsio/>. Accessed: 25 de julho de 2024. Citado na página 12.

11 RANSOM, S. M. . U. Searching for pulsars with presto. 2018. Citado 5 vezes nas páginas 13, 14, 15, 16 e 17.