Pulsar Observations with the ATA

Gurmehar Singh

gsingh@seti.org

May 2023

1 Introduction

There are a few key pieces of software that make pulsar observations with the ATA easier to work with. This guide covers these scripts so that anyone should be able to run pulsar observations with minimal or no introduction to the relevant software.

All of the scripts referenced in this guide without an absolute path are located in:
`/home/sonata/rfsoc_obs_scripts/atapulsarobs`.
Code is maintained at:
`https://github.com/gsingh-0-0-1/atapulsarobs`

2 Calibration

Calibration with the ATA usually involves setting the backend and then the postprocessor keys, after which we check if a source is visible, write an observing script for it, and then begin our calibration observation.

2.1 Setup

To set the backend for calibration, run:
`ansible-playbook /home/sonata/src/ansible_playbooks/hashpipe/xgpu_record.yml`
And the postprocessor:
`bash /home/sonata/src/observing_campaign/backend_setup_scripts/set_keys_uvh5_mv.bash`

2.2 Observation

The script `calib_obs.py` takes care of the latter part of all of this work by checking through a list of calibrator sources and determining which (if any) are observable.

Running `python calib_obs.py -h` returns the below information:

```
usage: ATA Calibration Observer [-h] [-s SOURCE] [-fb FREQB] [-fc FREQC] [-t TIME]

A script to observe calibrator sources

optional arguments:
    -h, --help            show this help message and exit
    -s SOURCE, --source SOURCE
                          Source name, leave empty to choose automatically
    -fb FREQB, --freqB FREQB
                          Frequency for Lo B in MHz, default 1236
    -fc FREQC, --freqC FREQC
```

```
                        Frequency for Lo C in MHz, default 1236 + 672 = 1908
  -t TIME, --time TIME  Observation time in seconds, default 300
```

For most pulsar observations, there isn't much of a need to observe at frequencies other than `1236` and `1908 MHz` – but you can set these differently if you like. Really, for a standard pulsar observation, all you need to do (after setting the backend and the postprocessor) is `python calib_obs.py`. In an edge-case scenario (which should really never be happening) where the script says that it could not find a source, refer to the ATA Beamformer Tutorial, which has a more extensive list of sources that you can check through.

3 Observation

3.1 Setup

To set the backend for calibration, run:

`ansible-playbook /home/sonata/src/ansible_playbooks/hashpipe/beamformer_mode_a_record.yml`

You will be prompted for some input. For a standard observation, you'll want to form just 1 beam, set textttfalse for the incoherent beam, produce 1 polarization, set 1 (disabled) for the channelizer, and use an integration size of 8 – which corresponds to 16 microseconds. And for the postprocessor: `bash /home/sonata/src/observing_campaign/backend_setup_scripts/set_keys_dspsr_rm.sh`

3.2 Observation

Like calibration, observation here is mostly automated. The script `auto_observe_pulsars.py` takes care of most of the observation work. In the file `PULSARS.csv`, you'll see a list of pulsars and their priorities. If you wish for a pulsar to be observed, simply add it to the file after checking that it's not already present. For now, simply list all pulsars with priority 1 – if you are interested in ranking pulsars priority-wise for observation, please read about the mechanics of the scheduler in the Scheduler Software section.

Let's go through the help menu for this script as well:

```
usage: ATA Pulsar Observer [-h] [-fb FREQB] [-fc FREQC] [-t TIME]

A script to observe pulsars

optional arguments:
    -h, --help            show this help message and exit
    -fb FREQB, --freqB FREQB
                          Frequency for Lo B in MHz, default 1236
    -fc FREQC, --freqC FREQC
                          Frequency for Lo C in MHz, default 1236 + 672 = 1908
    -t TIME, --time TIME  Observation time in minutes, default 150
    -tune, --tune         If changing frequencies, use this flag to autotune and change
```

This is pretty similar to the previous script – run the script without any options for the standard two-and-a-half-hour observation at the default frequencies. (Two and a half is chosen due to the standard pulsar observation windows being three hours, and leaving about 30 minutes for calibration and for possibilities of error.)

A quick note – the `--tune` flag should only be used in cases where you are manually swapping out solutions and switching to a different frequency, in which case this script will tune the array to the new frequencies that you've inputted using `--freqB` and `--freqC`. Please see the guide and code on how solution-swapping works if you're interested in that – you can see it at `https://github.com/gsingh-0-0-1/ata_soln_swapper`.

In addition to running the observation script, you'll want to make sure the data is processed. To do so, run `python auto_proc_pulsars.py` right after observing. So, after you've set the pulsar backend and appropriate postprocessor keys, you should run:

`python auto_observe_pulsars [options]; python auto_proc_pulsars.py`

That's all you need to know if you just want to observe. For a more in-depth explanation on how this all actually works, keep reading through this section – otherwise you can skip to the Data Visualization section.

3.3 Scheduler Software

At the moment, the script `dev_timeslots.py` takes care of auto-generating pulsar schedules. The name comes from the fact that until recently, it was `timeslots.py` that did this job – but that's quite a bit messier, and so I made the decision to rewrite the bulk of the scheduler into something easier to use, with added flexibility. The `auto_observe_pulsars.py` script imports this and uses the `Schedule` class to create a schedule which is then looped through during observation.

The basic principle is that, given a time window and a list of pulsars, the script will check which pulsars are capable of being observed during the time window based on both their locations in the sky and how long they'd need to be observed for.

The scheduler is initialized with three necessary parameters: (1), the start time, relative to now, in minutes, (2), the end time, relative to now, in minutes, and (3), a list of observing targets. A fourth optional parameter is the observation buffer time, in minutes – but this is set by default to two minutes, and that seems to be a nice sweet-spot from trial-and-error.

In addition to the `Schedule` class, there is the `Window` class – which really just holds a start and end time. But the purpose of this class is to provide the scheduler an easy way to actually schedule targets. When it's given a list of pulsars, it simple checks if the longest free window it has is greater than the shortest observation time. If so, it continues through a loop where it takes the first pulsar and observation time in its list, finds when it's available to be observed, and attempts to schedule it. Then, if it is indeed available, it is appended to the end of the target list, and if not, it's simply removed. This process continues until either there are no targets left or there is no observing time left.

You'll remember that the `PULSARS.csv` file requires you to input a priority as well as the pulsar name. This is because the older system that was used to schedule pulsar observations explicitly looped through higher priority pulsars separately. However, you might have noticed that given the mechanics of the current scheduler, simply placing a pulsar at the front of the queue ends up "prioritizing" it in the sense that it will have first preference on the entirety of the observing window. So, if you really want to make sure a pulsar gets observed in any given observation, just make sure it's either the first one in the `PULSARS.csv` file, or that you pre-pend it to the `dev_timeslots.ATA_PULSARS` list.

3.4 Observation Time Calculation

Currently, the formula being used to calculate the observation time for any pulsar is:

$$\min(\max(T_{min}, \frac{T_{min}}{(\frac{\phi}{\phi_{max}})^2}), T_{max})$$

where T_{min} is the minimum observing time, T_{max} is the maximum observing time, ϕ is the source flux density, and ϕ_{max} is a chosen parameter corresponding to the flux density at which we should observe for a time of T_{max}.

The reasoning behind this formula stems from the need to obtain a quantity for observation time that decreases with increasing flux density – we want to observe less on brighter sources, and more on dimmer ones. Thus, we take our minimum observing time and divide it by the flux density of our source. Of course, if we did just that, we would assume that a flux density of 1 – in this case mJy – would yield our minimum observing time. However, realistically, we might want to set our minimum observing time to correspond to some other flux density – say, 30 mJy – and so when a source is that bright, we'll observe for our minimum time. Thus we can divide the flux density by some chosen ϕ_{max} in order to accomplish this.

Now, we know that signal-to-noise ratio is correlated with the square root of time, and of luminosity. Thus, we'd have to square the fraction $\frac{\phi}{\phi_{max}}$ to stay in accordance with this correlation. So, if the flux density of our source decreases by a factor of two, we have to observe for four times as long.

However, this alone as a function of flux density for observing time would present a problem due. The behavior of this function as flux density approaches infinity or zero would result in either observing times nearing zero or infinity, respectively – and so we employ the `max` function to ensure that the observing time does not dip below T_{min} for very bright sources. Then, we employ `min` to deal with the second part of the problem, to ensure that observing time does not exceed T_{max} for very dim sources.

4 Data Visualization

Now that you've observed, there's the question of being able to see the results of your observations. The processing script that you queued up just after the observation creates images and dumps them into a directory that's readable by a webserver hosted on `obs-node1`. At the time of writing, you can see all your data at `obs-node1:8080`[1]. You'll see a list of date-time stamps, and if you click on any of those, you'll see a list of pulsars drop down – those are the ones observed in that observing session. Clicking on any of those should take you to a page where you can browse the processed images from the observation.

And if you want to see results from a specific pulsar, just type something in the search bar over to the right – you'll get a list of result hyperlinks containing observation date-time stamps as well as the name of the pulsar that the result links to.

5 Sample Plots

Lastly, here are a few plots from past observations.

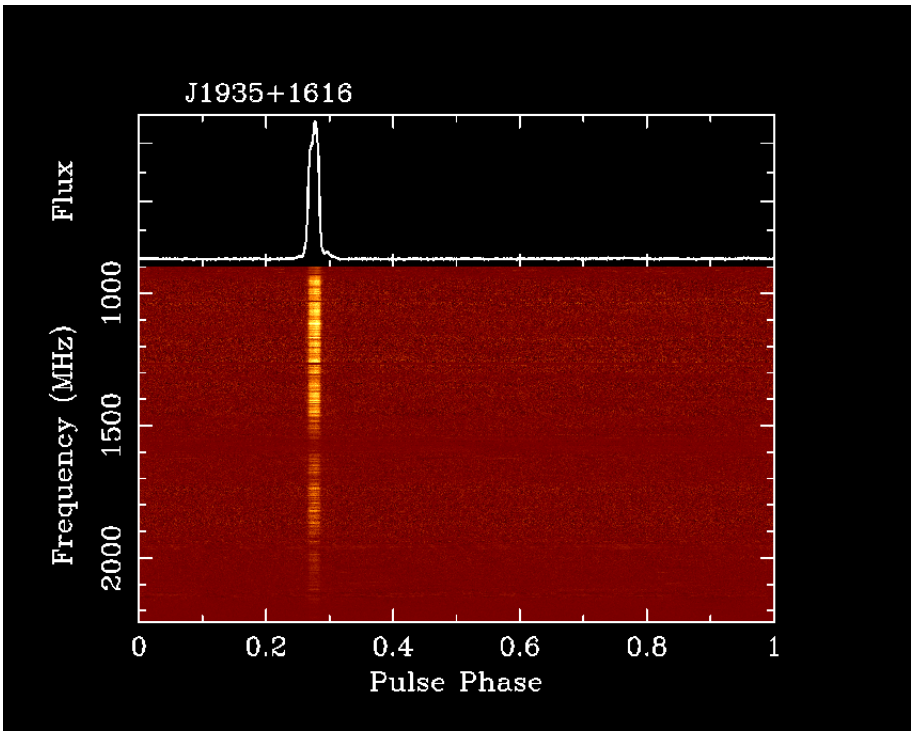

Figure 1: J1935+1616, 2023-05-15-08:32:50

[1]Check out the pulsar text animation! Definitely spent an appropriate amount of time getting that to look right... :)

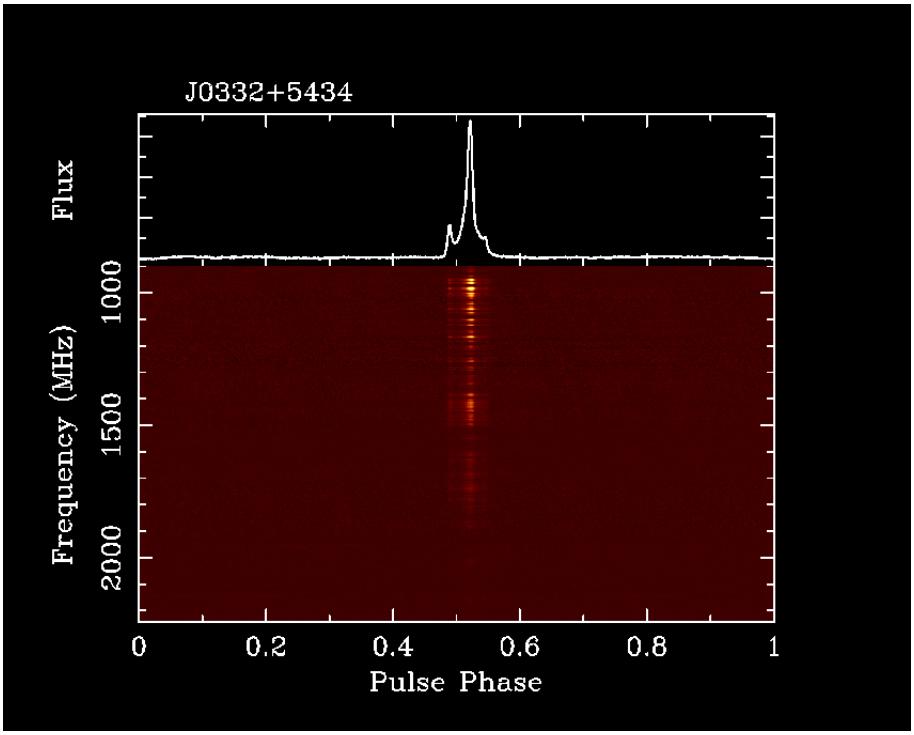

Figure 2: J0332+5434, 2023-04-13-01:31:50

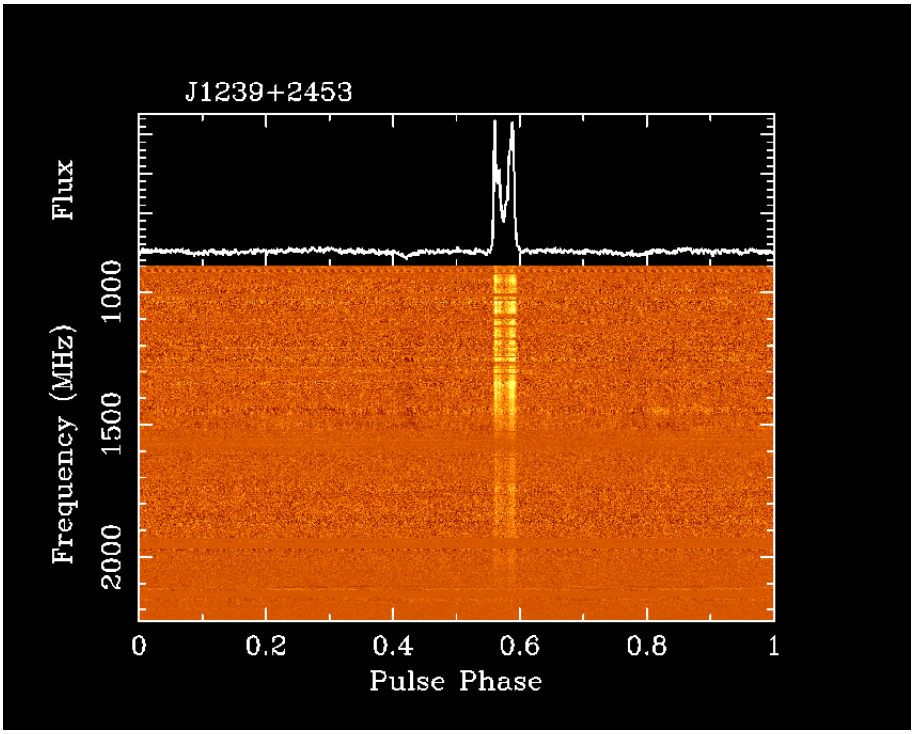

Figure 3: J1239+2453, 2023-05-02-03:50:54

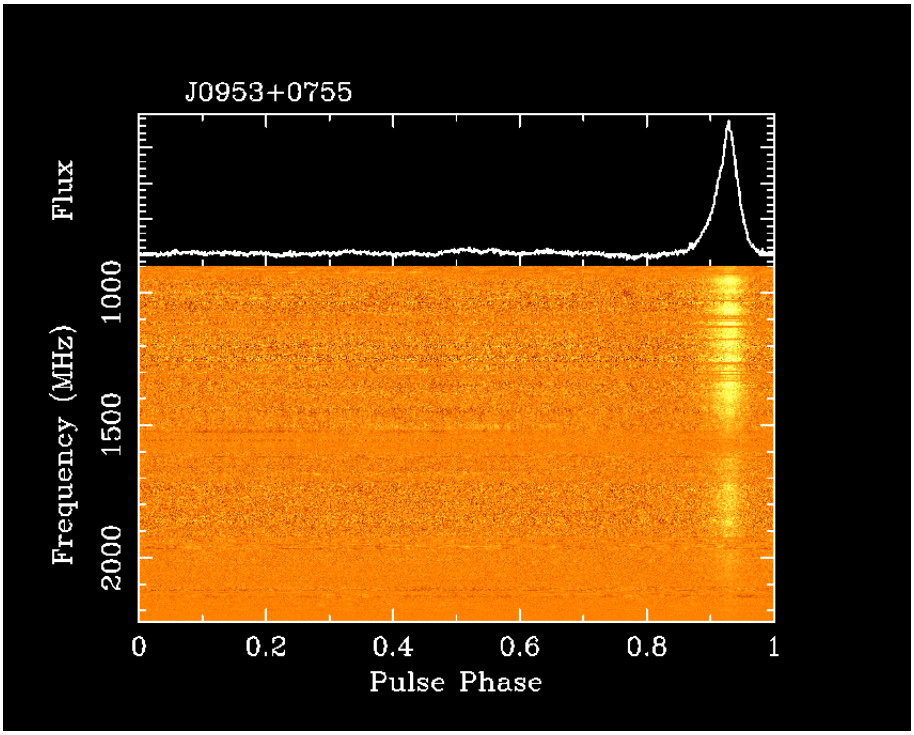

Figure 4: J0953+0755, 2023-04-04-04:30:25