# FRBGui

**Mohammed A. Chamma**

**Feb 21, 2024**

# INTRODUCTION

Welcome to FRBGui's documentation. FRBGui is a graphical user interface for measuring spectro-temporal properties of Fast Radio Bursts (FRBs) from their waterfalls using 2D autocorrelation functions (ACF). It can be used to split bursts with multiple components, change the dispersion measure (DM), add noise filters, and other preparation tasks before measurement. FRBGui is built with the Dear PyGui interface library.

Click here for a video demo of FRBGui.

Please visit *Overview and Installation* to get started.

CHAPTER

ONE

# FRBGUI

FRBGui is a graphical user interface for measuring spectro-temporal properties of Fast Radio Bursts (FRBs) from their waterfalls using 2D autocorrelation functions (ACF). It can be used to split bursts with multiple components, change the dispersion measure (DM), add noise filters, and other preparation tasks before measurement. FRBGui is built with the Dear PyGui interface library.

After measurement, FRBGui can be used to review and fix incorrect measurements by supplying initial guessues, producing an output PDF of all measurements and spreadsheets of measured values. Spreadsheets produced in this way can be loaded back into the FRBGui for further work or different sessions of measurements.

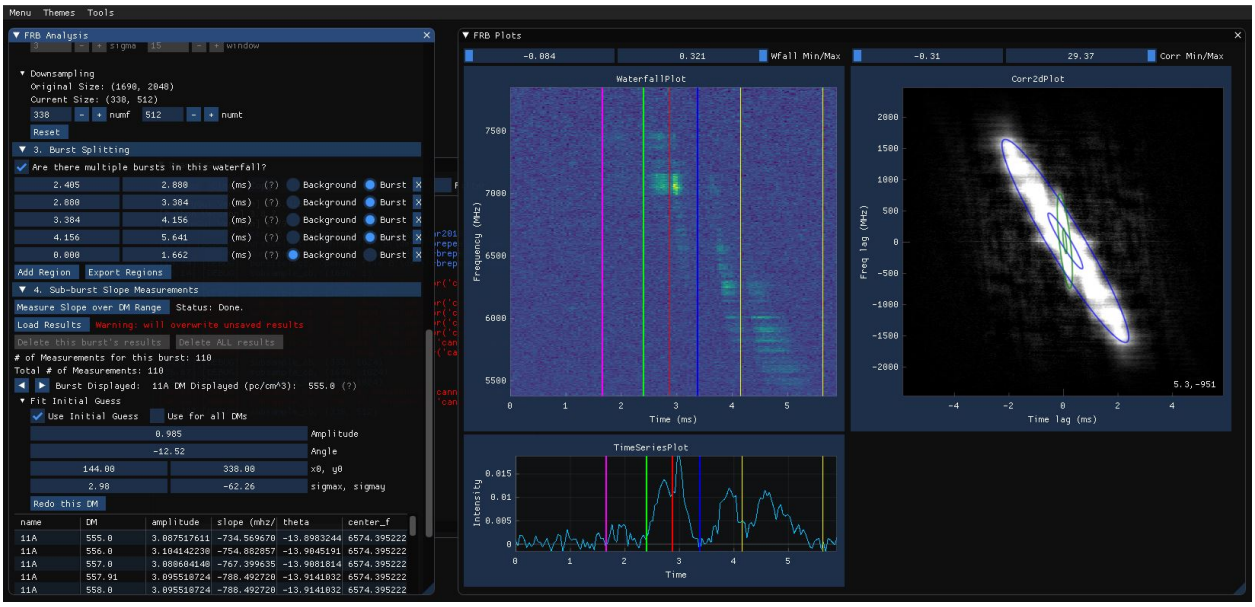
Measurements can be performed over a range of DMs and include the burst frequency, the sub-burst slope and/or drift rate, burst duration and burst bandwidth.

In this documentation you will find instructions on installing and getting started with FRBGui, an overview of its features, a guide on using FRBGui's scripting capabilities along with an API reference, tutorials for preparing data and performing burst measurements, and a reference table of FRBGui's measurement outputs. For advanced users, a section will introduce how custom tools, windows, and other interface elements can be added using Dear PyGui.

These pages are still being added, and if there is a specific topic you would like to see that is not yet available, please open an issue on GitHub or email me directly at chammam at mcmaster . ca

## 1.1 Features

- Change the DM of a burst
- Crop waterfalls
- Mitigate noise and RFI via background subtraction, SK-SG filter, manual channel zapping, and mask ranges
- Import and export of noise masks
- Measure over user-defined ranges of DMs
- Downsample waterfalls in frequency and time
- Split bursts with arbitrary numbers of sub-burst components
- Define inital fit guesses
- Review measurements via the output table
- Correct individual fits by DM
- Output measurements as a CSV spreadsheet and/or PDF with plots of each waterfall with its measurements.
- Provides `driftrate` and `driftlaw` python modules for scripting and automation.
- Automatic backups of measurements as they are made




## 1.2 Installation

Install FRBGui with:

```
pip install --user frbgui
```

For a local, editable installation with bleeding edge updates you may clone the repo and install locally:

```
git clone https://github.com/mef51/frbgui.git
cd frbgui
pip install --user --editable .
```

## 1.3 Usage

Run from the command-line with the following command to start in your current working directory:

```
frbgui
```

In a python script, you can invoke the gui in the following way:

```
from frbgui import frbgui
frbgui() # starts the GUI
```

## 1.4 Burst Format

FRBGui works with burst waterfalls that are prepared as python `.npz` archives.

The following snippet shows the format of the archive and an example of how a burst can be saved in the right format:

```
import numpy

wfall = # 2D numpy array with shape (num freq channels, num time channels)
burstmetadata = {
   ### required fields:
   'dfs'       : # 1D array of frequency channels in MHz, lowest first
   'DM'        : # dispersion measure (DM) in pc/cm^3, float
   'bandwidth' : # bandwidth of `wfall` in MHz, float (negative is ignored)
   'duration'  : # duration of `wfall` in seconds, float
   ### optional fields:
   'center_f'  : # burst frequency in MHz, optional,
   'freq_unit' : # string of freqeuncy unit, e.g. 'MHz', optional,
   'time_unit' : # string of time unit, e.g. 'ms', optional,
   'int_unit'  : # string of intensity unit, e.g. 'Jy', optional,
   'telescope' : # string of observing telescope, e.g. 'Arecibo', optional,
   'burstSN'   : # float of signal to noise ratio, optional,
   'tbin'      : # float of time resolution, optional
}

np.savez('burst.npz', wfall=wfall, **burstmetadata)
```

Optional fields are used for display purposes and do not otherwise affect measurements from within Frbgui.

## 1.5 Acknowledgements

If used in an academic study please cite:

- *A broad survey of spectro-temporal properties from FRB 20121102A*, Chamma, Mohammed A.; Rajabi, Fereshteh; Kumar, Aishwarya; Houde, Martin. MNRAS, 522, 2, 3036-3048, June 2023. arXiv:2210.00106

## 1.6 Publications

In addition to the above paper FRBGui has been used in the following studies:

- *Validating the Sub-Burst Slope Law: A Comprehensive Multi-Source Spectro-Temporal Analysis of Repeating Fast Radio Bursts*, Brown, Katie; Chamma, Mohammed A.; Rajabi, Fereshteh; Kumar, Aishwarya; Rajabi, Hosein; Houde, Martin. Submitted. arXiv:2308.11729

## 1.7 Attributions

Meteor icon created by Freepik - Flaticon

CHAPTER

TWO

# PREPARING DATA

Fast Radio Burst observations are taken with different telescopes from different countries around the world. Because of this, different studies will make their radio observations available in all kinds of formats and knowing how to adapt to and deal with different formats is a skill that one invariably develops when working with FRBs (and astronomical data in general).

This tutorial is intended as an example of how FRB data from different sources can be prepared for use in FRBGui and is intended for researchers new to FRBs with some technical experience programming in Python. Parts of the tutorial will require FRBGui to be installed (see *Installation*).

In this tutorial we will load FRB data from two different studies and prepare them in FRBGui's *Burst Format* and save them as numpy zipped `.npz` archive files.

These `.npz` files will be used in the tutorial "*Measuring FRBs*" to obtain spectro-temporal measurements of the FRBs. These tutorials can be done out of order if preferred, as the `.npz` files will be provided in the next tutorial for download.

The two studies we will process data from today will feature two different data formats and are

- Gajjar et al. (2018):
- Aggarwal et al. (2021)

Gajjar et al. (2018) features observations of the repeating source FRB 20121102A using the Green Bank Telescope. As stated in their paper, the data are available in the PSRFITS format at http://seti.berkeley.edu/frb121102/technical.html. The bursts in these data are already conveniently cut-out into individual files.

Aggarwal et al. (2021) also features observations of the repeating source FRB 20121102A, taken with the Arecibo telescope. These data are available in the filterbank format, and their results are available at https://github.com/thepetabyteproject/FRB121102. For these data, we will download the entire observation and need to locate the bursts and cut them out ourselves.

Following each paper's respective documentation, we will use Python packages that can read the data and prepare scripts to read the PSRFITS and filterbank data and save the bursts into `.npz` files that we can use with FRBGui.

Our goal is to prepare the following two bursts so that we may obtain their spectro-temporal properties in the next tutorial.

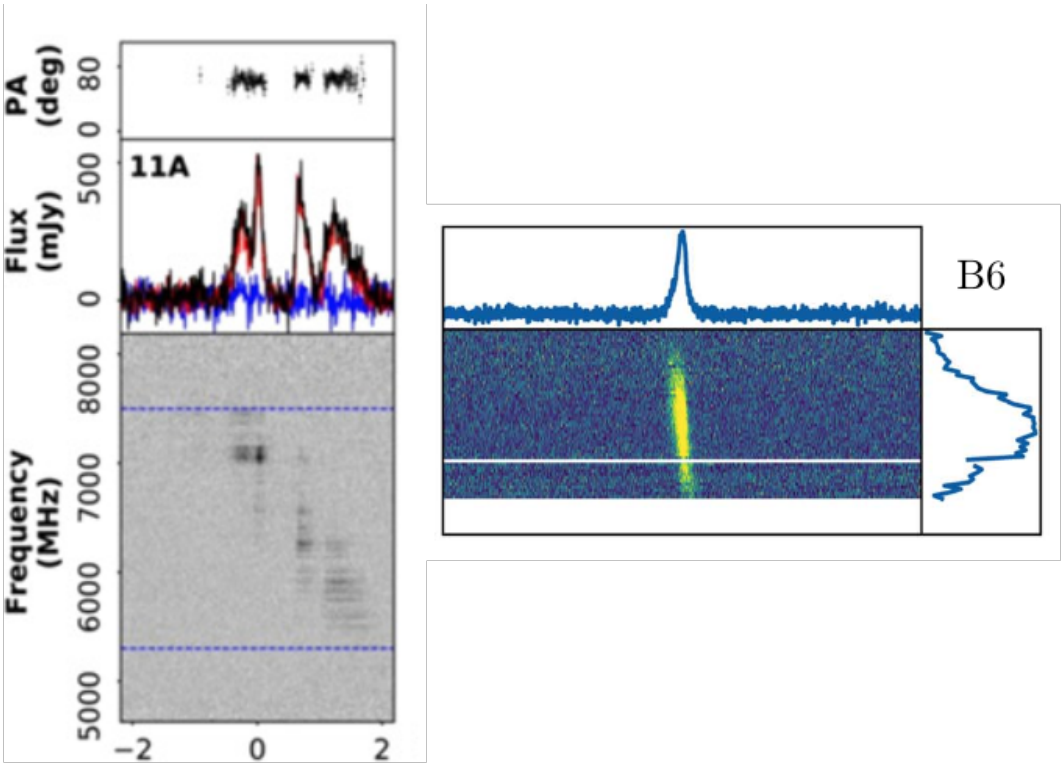



Fig. 1: On the left, burst 11A from Fig. 2 of Gajjar et al. (2018). On the right, Burst B6 from Fig. 1 of Aggarwal et al. (2021).

## 2.1 Reading PSRFITS (`.fits`)

In this section we will develop a script that reads a PSRFITS file and prepares the FRB contained within it for measurement.

>> Open the documentation for Gajjar et al. (2018) and download burst 11A which has the filename `11A_16sec.calib.4p` and is 308 MB. Save this file into your folder of choice.

As stated in their documentation, the package PyPulse can be used to read their PSRFITS format.

>> Install PyPulse from the command line

```
pip install --user pypulse
```

**Note:** Note that version 0.1.1 of PyPulse will not work with recent versions of numpy (>=1.20).

If a newer release of PyPulse is not yet available on pip, you will need to install PyPulse from a local copy of its repository which has resolved its issue with newer versions of numpy. You can do so in the following way:

```
git clone https://github.com/mtlam/PyPulse.git
cd PyPulse
pip install --user --editable .
```

With PyPulse installed you can open the data and start preparing it according to FRBGui's *Burst Format* with the following code:

```
import pypulse
import numpy as np
import matplotlib.pyplot as plt

ar = pypulse.Archive('11A_16sec.calib.4p', prepare=False) # load without dedispersing
ar.pscrunch() # average polarizations if any
ar.center() # center pulse in array

wfall = ar.getData()
burstmetadata = {
        'dt'        : ar.getTimes(),
        'dfs'       : ar.getFreqs(),
        'DM'        : ar.getDM(),
        'bandwidth' : ar.getBandwidth(),
        'duration'  : ar.getDuration(), # usually in seconds
        'center_f'  : ar.getCenterFrequency(),
        'freq_unit' : ar.getFrequencyUnit(),
        'time_unit' : ar.getTimeUnit(),
        'int_unit'  : ar.getIntensityUnit(),
        'telescope' : ar.getTelescope(),
        'burstSN'   : ar.getSN(),
        'tbin'      : ar.getTbin(),
}
```

In the above we have used the methods in `pypulse.Archive` to extract the information we will need from the fits file. For more information on these methods visit PyPulse's documentation.

We can now perform a few checks to ensure the values from the fits file will work with FRBGui and adjust them if not.

```
for item in burstmetadata.items():
        print(*item)
```

Which outputs each key and value in `burstmetadata`:

```
dt [0.01048303]
dfs [8188.87304688 8188.68994141 8188.50683594 ... 4626.92236328 4626.73925781
 4626.55615234]
DM 557.91
bandwidth -3562.5
duration 0.020966058666666648
center_f 6407.7148
freq_unit MHz
time_unit SEC
int_unit Jy
telescope GBT
burstSN 7.9028664
tbin 1.0239731655700501e-05
```

We can also see that `wfall.shape = (19456, 2048)`, indicating the data have 19456 frequency channels and 2048 time channels.

These values align well with FRBGui's *Burst Format* since the `freq_unit` and `time_unit` are already in MHz and seconds, respectively. However the `dfs` array and the `bandwidth` field are in slightly different formats. The `dfs` array should be sorted from smallest to largest and the negative sign in `bandwidth` is ignored by FRBGui, so it is a good idea to remove it. We can correct these by doing

```
burstmetadata['bandwidth'] = abs(burstmetadata['bandwidth'])
burstmetadata['dfs'].sort()
```

At this stage we could save the wfall and burstmetadata into an .npz file and the file will load in FRBGui.

However, due to the large size of the data array (with 19456 frequency channels) we may want to reduce the amount of data that is loaded into FRBGui for the sake of signal to noise as well as performance. We can also plot the data and check that we have correctly read the file.

Let's start by plotting the data using matplotlib to see what we are working with.

```
plt.imshow(wfall, aspect='auto', origin='lower')
```

We use `aspect='auto'` to get a more square figure and `origin='lower'` to specify we want the lowest frequency at the bottom. Feel free to experiment with these options.

The resulting plot is shown below.

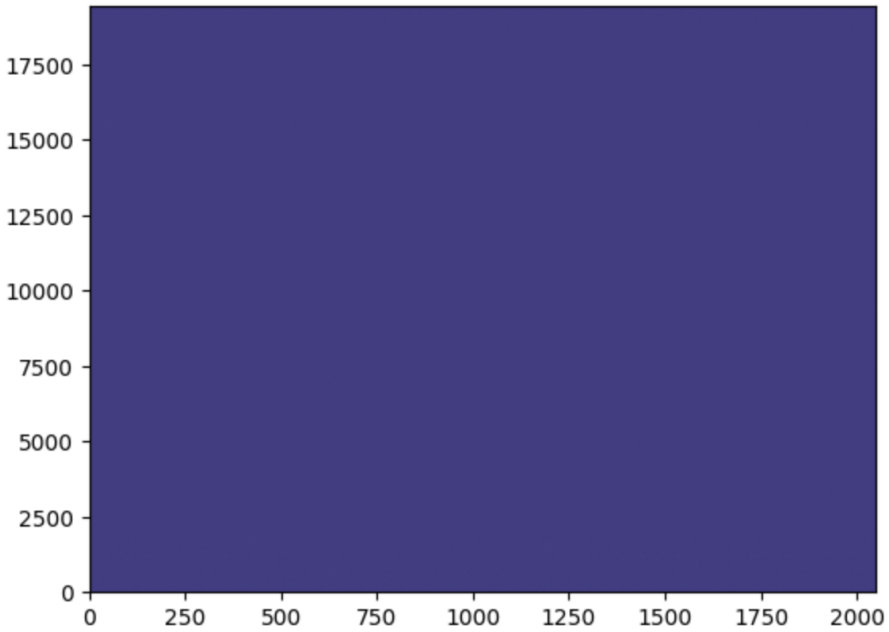



Fig. 2: Burst waterfall (frequency on y-axis, time on x-axis)

Oops! We don't see anything. The large number of channels may be washing out our signal, so let's temporarily downsample the burst using the *driftrate* module (which ships with FRBGui) to see if the signal to noise will increase.

```
import driftrate

wfall = driftrate.subsample(wfall, 304, 512)
plt.imshow(wfall, aspect='auto', origin='lower')
```

Here we've downsampled the original waterfall size of 19456 freq. channels and 2048 time channels to 19456/64 = 304 frequency channels and 2048/4 = 512 time channels.

While still faint we can begin to see the burst (though it appears flipped) and more importantly what appears to be two bands of radio frequency interference (RFI) near the bottom and top of the waterfall at around y = 20 and 245.

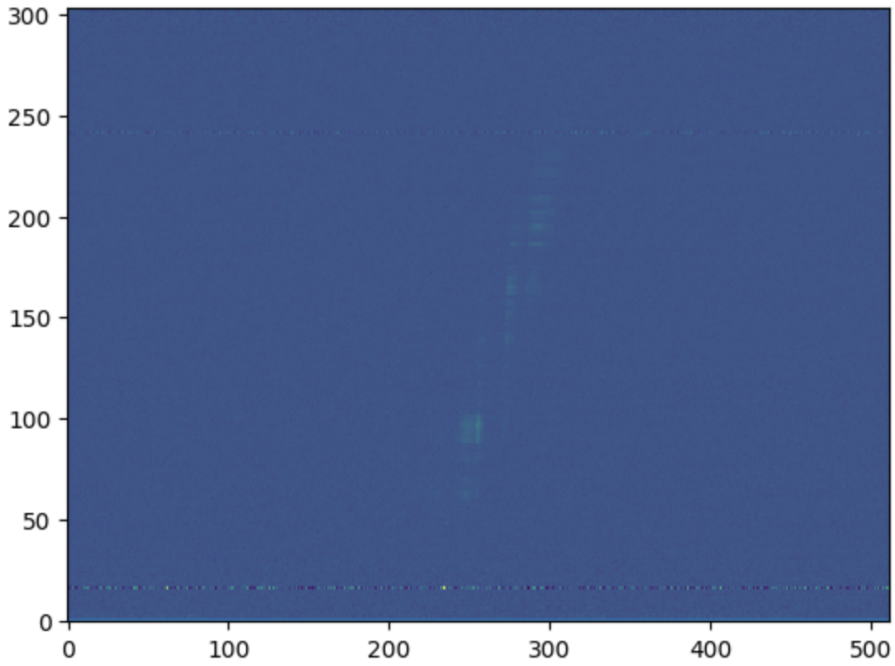



Fig. 3: Downsampled burst waterfall (frequency on y-axis, time on x-axis)

These RFI bands may be responsible for washing out the burst. If we can locate them precisely we can remove them and then plot the waterfall without them.

To precisely locate them we will plot the waterfall's spectrum, averaged over all time samples, to obtain a plot of intensity vs. frequency.

```
plt.plot(np.nanmean(wfall, axis=1))
```

The two spikes at around x = 1000 and 15300 are the RFI we saw in the earlier waterfall. By inspecting the figure more closely using matplotlib's graphical interface (you can use `plt.show()` in a script or `%matplotlib qt` in a jupyter notebook) we can more precisely determine the offending channel numbers and remove them from the waterfall with the following code. We will also flip the waterfall so that the burst is the right way up.

```
32 wfall[1053:1110] = 0
33 wfall[15360:15471] = 0
34 wfall = np.flipud(wfall)
35 wfall = driftrate.subsample(wfall, 304, 512)
36
37 plt.imshow(wfall, aspect='auto', origin='lower', interpolation='none')
```

Great! We can now clearly see the burst.

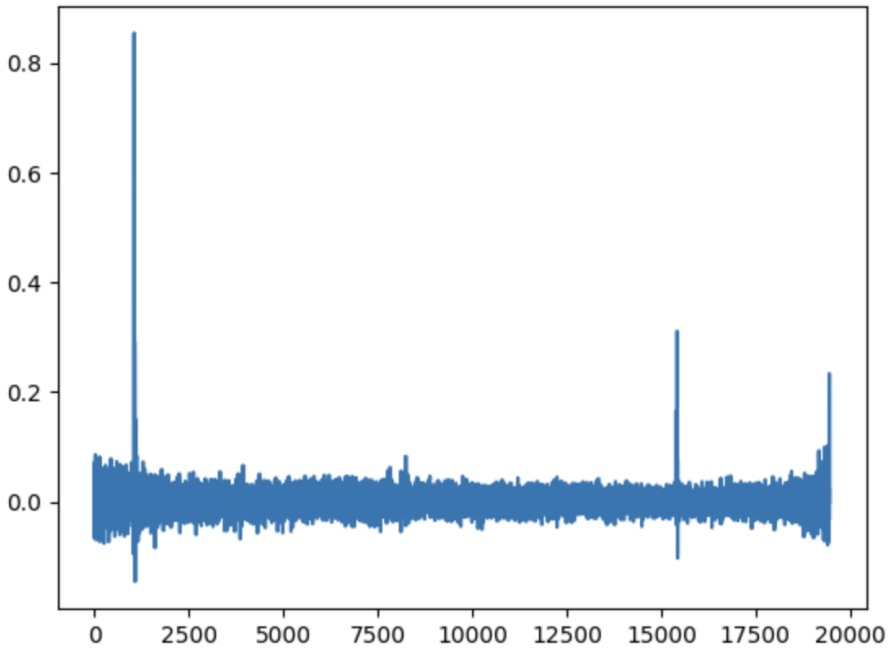



Fig. 4: Burst spectrum (intensity on y-axis, frequency on x-axis)

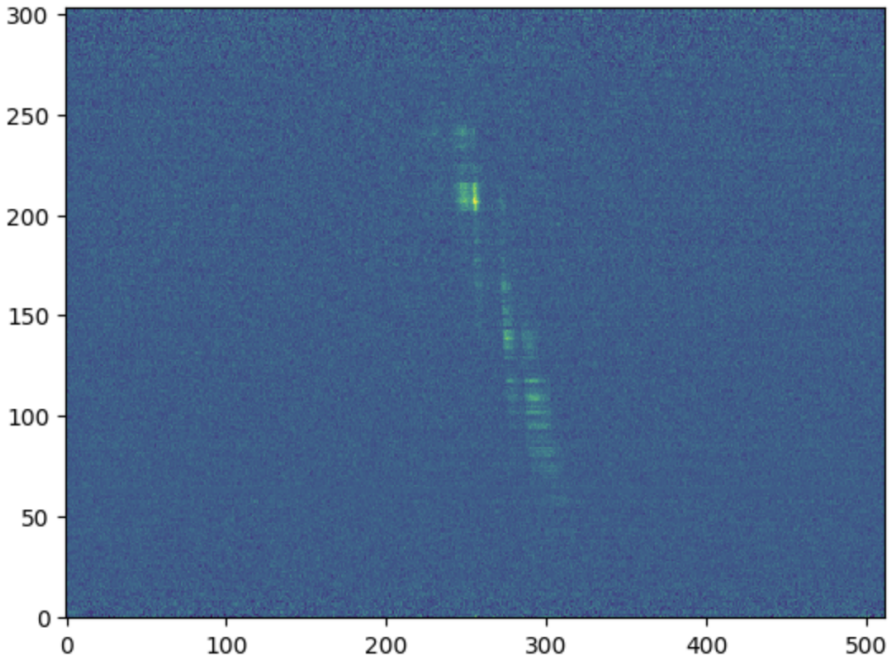



Fig. 5: Burst waterfall with high signal-to-noise (frequency on y-axis, time on x-axis)

### 2.1.1 Computing Axes

One last check we can perform is to add the units of the frequency (MHz) and time (ms) axes so that we can be sure we have loaded the data as it is presented in the paper (See the figure *above*).

The PSRFITS file contains information about the bandwidth, duration, frequency axis, and duration in the `burstmetadata['bandwidth']`, `burstmetadata['duration']`, `burstmetadata['dfs']`, `burstmetadata['dt']`, and `burstmetadata['tbin']` fields (which we printed above). The `dt` and `tbin` fields seem to slightly differ, and if we compute the resolution with the duration and original waterfall shape (before subsampling)

```
>>> burstmetadata['duration']/wfall.shape[0]*1000
0.010237333333333324
```

we get yet another slightly different value. In this case it may be best to just try all three and see which best matches up with the publication or contact the paper author for clarification.

For now we will simply compute the frequency and time resolutions using the `'bandwidth'` and `'duration'` fields from the file and the shape of the original waterfall. Downsampling the burst will change the resolutions since we are decreasing the number of channels, so we will store the original shape of the waterfall and update the resolutions based on the factor we downsampled by.

```
wfall[1053:1110] = 0
wfall[15360:15471] = 0
wfall = np.flipud(wfall)

df = burstmetadata['bandwidth']/wfall.shape[0] # MHz
dt = burstmetadata['duration']/wfall.shape[1]*1000 # ms
origshape = wfall.shape
wfall = driftrate.subsample(wfall, 2432, 2048)
df *= origshape[0]/wfall.shape[0]
dt *= origshape[1]/wfall.shape[1] # no change

plt.imshow(wfall, aspect='auto', origin='lower', interpolation='none')
```

Note here that we have changed our downsampling to 2432 by 2048 channels (more than before but less than the original) to preserve some of the data resolution. This will decrease the signal-to-noise but we will be able to modify this more dynamically later from inside FRBGui. A higher data resolution also helps with measurements stability and accuracy.

Using the frequency and time resolutions, we can now add axis labels and display the waterfall axes using `imshow`'s `extent` keyword, which takes a list of `[left, right, bottom, top]` limits for the axes.

```
lowest_freq = min(burstmetadata['dfs'])
extent = [
        0,
        dt*wfall.shape[1],
        lowest_freq,
        lowest_freq + df*wfall.shape[0]
]
# convenience function from driftrate module, same as above:
# extent, _ = driftrate.getExtents(wfall, df=df, dt=dt, lowest_freq=lowest_freq)

plt.imshow(wfall, aspect='auto', origin='lower', interpolation='none', extent=extent)
plt.xlabel("Time (ms)")
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("Frequency (MHz)")
plt.title(f"Burst 11A ({wfall.shape = })")
```

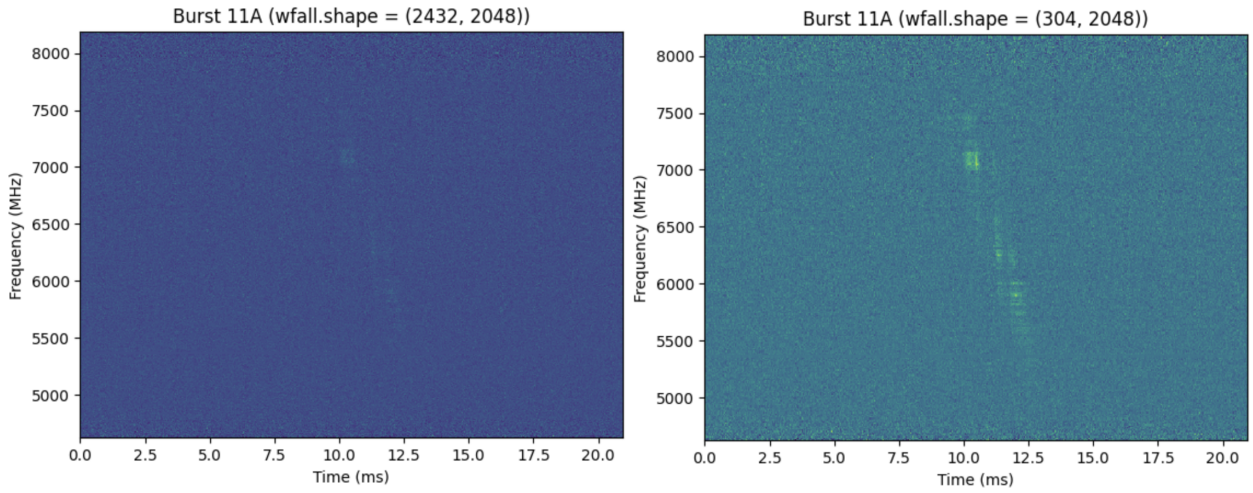



Fig. 6: The prepared waterfall, ready for saving. On the left the burst is faintly seen compared to on the right due to the difference in waterfall downsampling.

From here we can see that the frequency axes match with the figure in the paper and that the burst lasts just over 2 ms, also consistent with what is shown in the paper figure.

There are other tasks we could perform if desired, we could crop the waterfall to include less of the data before and after the burst or we could decrease the bandwidth. This however is sufficient for our purposes and we can now save the burst for measurement in FRBGui with the following command.

```
np.savez('burst11A.npz', wfall=wfall, **burstmetadata)
```

This creates a file `burst11A.npz` that can be loaded into FRBGui (see *Measuring FRBs*) in the directory of your script. Congratulations!

This script can now be used as the basis for preparing PSRFITS files, especially the other bursts made available from Gajjar et al. (2018). For example, it can be adapted to prepare all the bursts automatically in a loop over the downloaded data files.

### 2.1.2 Complete Code

For your reference, below is the complete script we developed for reading burst 11A in its PRSFITS format and preparing it as a numpy zipped `.npz` file, ready for further analysis in FRBGui or other Python scripts. Included are some optional lines for removing the bandwidth above and below the burst.

```
import pypulse
import numpy as np
import matplotlib.pyplot as plt
import driftrate

ar = pypulse.Archive('11A_16sec.calib.4p', prepare=False) # load without dedispersing
ar.pscrunch() # average polarizations if any
ar.center() # center pulse in array
```

(continues on next page)

(continued from previous page)

```
9
10  wfall = ar.getData()
11  burstmetadata = {
12          'dt'        : ar.getTimes(),
13          'dfs'       : ar.getFreqs(),
14          'DM'        : ar.getDM(),
15          'bandwidth' : ar.getBandwidth(),
16          'duration'  : ar.getDuration(), # usually in seconds
17          'center_f'  : ar.getCenterFrequency(),
18          'freq_unit' : ar.getFrequencyUnit(),
19          'time_unit' : ar.getTimeUnit(),
20          'int_unit'  : ar.getIntensityUnit(),
21          'telescope' : ar.getTelescope(),
22          'burstSN'   : ar.getSN(),
23          'tbin'      : ar.getTbin(),
24  }
25
26  burstmetadata['bandwidth'] = abs(burstmetadata['bandwidth'])
27  burstmetadata['dfs'].sort()
28
29  wfall[1053:1110] = 0
30  wfall[15360:15471] = 0
31  wfall = np.flipud(wfall)
32
33  df = burstmetadata['bandwidth']/wfall.shape[0] # MHz
34  dt = burstmetadata['duration']/wfall.shape[1]*1000 # ms
35  origshape = wfall.shape
36  wfall = driftrate.subsample(wfall, 2432, 2048)
37  df *= origshape[0]/wfall.shape[0]
38  dt *= origshape[1]/wfall.shape[1] # no change
39
40  # Optional: crop frequency band to match Fig 1 in Gajjar+2018
41  # burstmetadata['bandwidth'] = burstmetadata['bandwidth']*((wfall.shape[0] - ((wfall.
    ↪shape[0]-2200)+510))/wfall.shape[0])
42  # lowest_freq = burstmetadata['center_f'] - wfall.shape[0]/2*(df*(19456/2432)) +␣
    ↪(df*(19456/2432))*510
43  # wfall = wfall[510:2200, ...]
44  # df = burstmetadata['bandwidth']/wfall.shape[0]
45  # burstmetadata['dfs'] = np.linspace(lowest_freq, lowest_freq+burstmetadata['bandwidth'],␣
    ↪num=wfall.shape[0])
46
47  lowest_freq = min(burstmetadata['dfs'])
48  extent = [
49          0,
50          dt*wfall.shape[1],
51          lowest_freq,
52          lowest_freq + df*wfall.shape[0]
53  ]
54  # convenience function from driftrate module, same as above:
55  # extent, _ = driftrate.getExtents(wfall, df=df, dt=dt, lowest_freq=lowest_freq)
56
57  plt.imshow(wfall, aspect='auto', origin='lower', interpolation='none', extent=extent)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel("Time (ms)")
plt.ylabel("Frequency (MHz)")
plt.title(f"Burst 11A ({wfall.shape = })")

np.savez('burst11A.npz', wfall=wfall, **burstmetadata)
```

## 2.2 Reading Filterbank (`.fil`)

Our goal in this section is to read the bursts from Aggarwal et al. (2021), specifically burst B6 and prepare if for measurement in FRBGui.

The data are available in a large filterbank file that includes the entire observational scan. It's size is multiple gigabytes. Accompanying the paper is a spreadsheet with the timestamps of when the bursts were observed in the filterbank file. The paper also explains that it used a custom package called BurstFit for its analysis. Our strategy then will be to download the files, extract the timestamps from the paper's provided spreadsheet, and use BurstFit to cutout the data from the filterbank file. In order to extract basic information about the data file such as the resolutions and bandwidth, we will use another package called Your for convenience.

>> Download the filterbank data from Aggarwal et al. (2021) and extract it (6.2 GB). In it will be two filterbank files in deeply nested folders. Copy them to the directory of your choice and start a script.

```
files = [
        'puppi_57644_C0531+33_0021_subs_0001.fil',
        'puppi_57645_C0531+33_0029_subs_0001.fil'
]
```

Note that the '57644' and '57645' in the filenames refer to the day of the observation (the Modified Julian Date).

>> Download all_bursts_bary.csv which will contain metadata about the bursts as well as the burst timestamps.

>> Install the BurstFit and Your packages on the command-line. Read the "BurstData" class section of the BurstFit documentation.

```
pip install --user burstfit your
```

>> Load the spreadsheet using pandas (`pip install --user pandas` if you do not already have it)

```
import pandas as pd

canddf = pd.read_csv('all_bursts_bary.csv')
```

The spreadsheet does not have an explicit column for the "candidate times" but it does contain all the information for locating each burst in the "cand_id" column. For example,

```
>>> canddf['cand_id'][0]
'cand_tstart_57644.407719907409_tcand_61.2631000_dm_565.30000_snr_8.12529'
```

In this field we can find the day of the observation, the candidate time (the time from the start of the filterbank file where the candidate burst can be found), the DM, and the signal-to-noise ratio (SNR). Notice that the values are separated by a "_" character.

We will use the `BurstData` class from the BurstFit package to load the bursts. This class requires filename, DM, candidate time, width (number of samples to load), and the SNR of the burst.

>> Extract the properties needed for the `BurstData` class from the `'cand_id'` column and add these columns to the spreadsheet

```
canddf['tstart'] = [float(candid.split('_')[2]) for candid in canddf['cand_id']]
canddf['tcand']  = [float(candid.split('_')[4]) for candid in canddf['cand_id']]
canddf['dm']     = [float(candid.split('_')[6]) for candid in canddf['cand_id']]
canddf['snr']    = [float(candid.split('_')[8]) for candid in canddf['cand_id']]
canddf['label']  = canddf['bidx']
canddf['width']  = 256
canddf['file']   = [files[0] if '57644' in str(tstart) else files[1] for tstart in␣
↪canddf['tstart']]
canddf = canddf.set_index('label')
```

Here we have used the `.split()` function and the fact that `'cand_id'` contains the "_" character between values to extract the information we want. We have also used the `'bidx'` column to serve as a label for each burst, and chosen to extract 256 time samples around each candidate burst.

At this point we could just load the single burst we want, but we have done all this work and it is perfectly setup to just extract every burst from the study in a nice for-loop. So let's do that.

```
import numpy as np
import matplotlib.pyplot as plt
from burstfit.data import BurstData

for bid, row in canddf.iterrows():
        bd = BurstData(
                fp=row['file'],
                dm=row['dm'],
                width=row['width'],
                snr=row['snr'],
                tcand=row['tcand'],
        )
        bd.prepare_data(time_window=0.1) # 0.1 seconds
        wfall = np.flipud(bd.sgram.astype(np.float64).copy())
        if bid == 'B6.1':
                plt.imshow(wfall, aspect='auto', interpolation='none')
```

Here we have added an if-condition to plot the waterfall if it is burst B6 so we can see some output.

To save these bursts we need the bandwidth and resolution information of the file, which is not readily available from the BurstFit package. The Your package is a general library for reading filterbank files (amongst others) and will provide this information.

Then, we will create the `burstmetadata` object and save the `.npz` for each burst, as in the previous section.

```
import your
yourdata = your.Your(files[1]) # assuming both files have same res and band
datameta = yourdata.your_header

for bid, row in canddf.iterrows():
        bd = BurstData(
                fp=row['file'],
                dm=row['dm'],
                width=row['width'],
                snr=row['snr'],
```

(continues on next page)

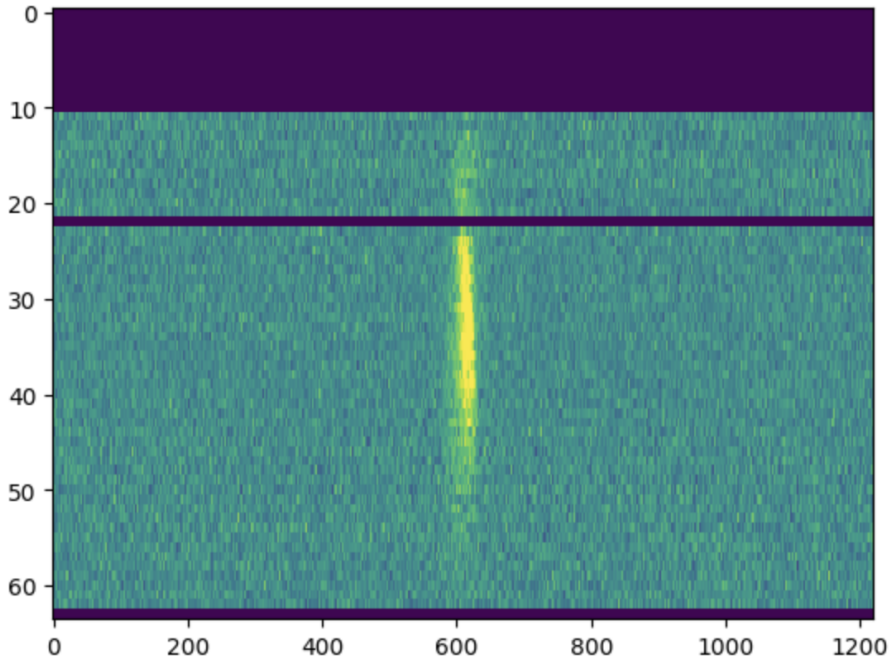



Fig. 7: Burst B6 of Aggarwal et al. (2021) loaded.

(continued from previous page)

```
                tcand=row['tcand'],
        )
        bd.prepare_data(time_window=0.1) # 0.1 seconds
        wfall = np.flipud(bd.sgram.astype(np.float64).copy())
        if bid == 'B6.1':
                plt.imshow(wfall, aspect='auto', interpolation='none')

        burstmetadata = {
                'dt'        : datameta.tsamp,
                'dfs'       : np.linspace(datameta.fch1-abs(datameta.bw), datameta.fch1,␣
↪num=datameta.nchans),
                'DM'        : row['dm'],
                'bandwidth' : abs(datameta.bw),
                'duration'  : 0.1,
                'center_f'  : datameta.center_freq,
                'freq_unit' : 'MHz',
                'time_unit' : 's',
                'int_unit'  : 'arb',
                'telescope' : 'Arecibo',
                'burstSN'   : row['snr'],
                'raw_shape' : wfall.shape
        }

        wfallout = f'{bid}.npz'
        np.savez(wfallout, wfall=wfall, **burstmetadata)
```

**Note:** You may, as in the previous section, compute the axes and plot the figures to ensure that they match with the

publication. Another option however is to load the burst into FRBGui and check the axes from there, as FRBGui will compute them for you from the information provided in the .npz file.

---

You have now completed the "Preparing Data" tutorials. These have hopefully given you an impression of the kind of tasks that may be involved in loading data from different research groups using different telescopes and different formats. The techniques shown here are certainly not exhaustive, but will hopefully have given you some experience in dealing with the data formats that FRBs can be found in.

### 2.2.1 Complete Code

For your reference, below is the complete script we developed for reading the bursts from Aggarwal et al. (2021) in filterbank format and preparing it as a numpy zipped `.npz` file, ready for further analysis in FRBGui or other Python scripts.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from burstfit.data import BurstData
from burstfit.utils.plotter import plot_me
import your

canddf = pd.read_csv('all_bursts_bary.csv')
files = [
        'puppi_57644_C0531+33_0021_subs_0001.fil',
        'puppi_57645_C0531+33_0029_subs_0001.fil'
]

canddf['tstart'] = [float(candid.split('_')[2]) for candid in canddf['cand_id']]
canddf['tcand']  = [float(candid.split('_')[4]) for candid in canddf['cand_id']]
canddf['dm']     = [float(candid.split('_')[6]) for candid in canddf['cand_id']]
canddf['snr']    = [float(candid.split('_')[8]) for candid in canddf['cand_id']]
canddf['label']  = canddf['bidx']
canddf['width']  = 256
canddf['file']   = [files[0] if '57644' in str(tstart) else files[1] for tstart in␣
↪canddf['tstart']]
canddf = canddf.set_index('label')

yourdata = your.Your(files[1]) # assuming both files have same res and band
datameta = yourdata.your_header
for bid, row in canddf.iterrows():
        bd = BurstData(
                fp=row['file'],
                dm=row['dm'],
                width=row['width'],
                snr=row['snr'],
                tcand=row['tcand'],
        )
        bd.prepare_data(time_window=0.1)
        wfall = np.flipud(bd.sgram.astype(np.float64).copy())
        if bid == 'B6.1':
                plt.imshow(wfall, aspect='auto', interpolation='none')

```

(continues on next page)

(continued from previous page)

```
        burstmetadata = {
                'dt'         : datameta.tsamp,
                'dfs'        : np.linspace(datameta.fch1-abs(datameta.bw), datameta.fch1,␣
↪num=datameta.nchans),
                'DM'         : row['dm'],
                'bandwidth'  : abs(datameta.bw),
                'duration'   : 0.1,
                'center_f'   : datameta.center_freq,
                'freq_unit'  : 'MHz',
                'time_unit'  : 's',
                'int_unit'   : 'arb',
                'telescope'  : 'Arecibo',
                'burstSN'    : row['snr'],
                'raw_shape'  : wfall.shape
        }

        wfallout = f'{bid}.npz'
        np.savez(wfallout, wfall=wfall, **burstmetadata)
```

CHAPTER

# THREE

# MEASURING FRBS

This tutorial will describe how to perform measurements of FRB waterfalls using two examples. At the end of the tutorial we will have produced a CSV spreadsheet with all of our measurements that can be used for further analysis and a PDF that displays plots of each burst with each of its measurements overlaid on top for review. This tutorial is intended for those with some knowledge on the observational characteristics of FRBs and with basic experience programming and using a terminal. For an introduction to FRBs, see (e.g.) Petroff et al. (2022)

The first FRB is a simple bright pulse with some noise in its waterfall, and the second FRB consists of multiple sub-bursts at a high data resolution that we would like to separate and measure independently.

These examples will broadly demonstrate the capabilities of FRBGui and what a typical workflow looks like.

To follow along, you may download the two burst files which have been prepared in `.npz` format for FRBGui here (see *Preparing Data*):

- aggarwal2021_B006.npz
- gajjar2018_11A.npz

The first burst is burst B006 published in Aggarwal et al. (2021) and discovered in data taken by the Arecibo telescope while observing the repeating source FRB 20121102A. With FRBGui installed (see *Installation*), navigate to the folder where your bursts have been downloaded and saved and run the following command in the terminal:

```
frbgui
```

This will start the graphical user interface and you should see your first burst already loaded and displayed:

The main interface consists of the "FRB Analysis" window which is where we will manipulate the waterfall and the "FRB Plots" window, which displays the FRB waterfall (left), the two-dimensional autocorrelation function (ACF) on the right, and the frequency integrated time-series of the waterfall at the bottom.

At this stage it is important to verify that the frequency and time axes are correctly displayed, usually by cross-referencing with the corresponding figure in the publication. If the axes of the waterfall are incorrect, this can indicate an issue with the way the `.npz` file was prepared, either with the frequencies, bandwidth, or duration supplied and should be double-checked against FRBGui's *Burst Format*.

If the axes appear correct we can proceed with measurement.

For this burst we want to measure its sub-burst slope, or the rate of change of frequency with time, a quantity related to the drift rate and a characteristic feature of FRBs. Because the choice of Dispersion Measure (DM) affects the value of the sub-burst slope by "rotating" the burst as it appears in its waterfall we will repeat our measurements over a range of DMs. These measurements can then be used at a later stage of analysis to estimate and characterize the uncertainty on our measurements.

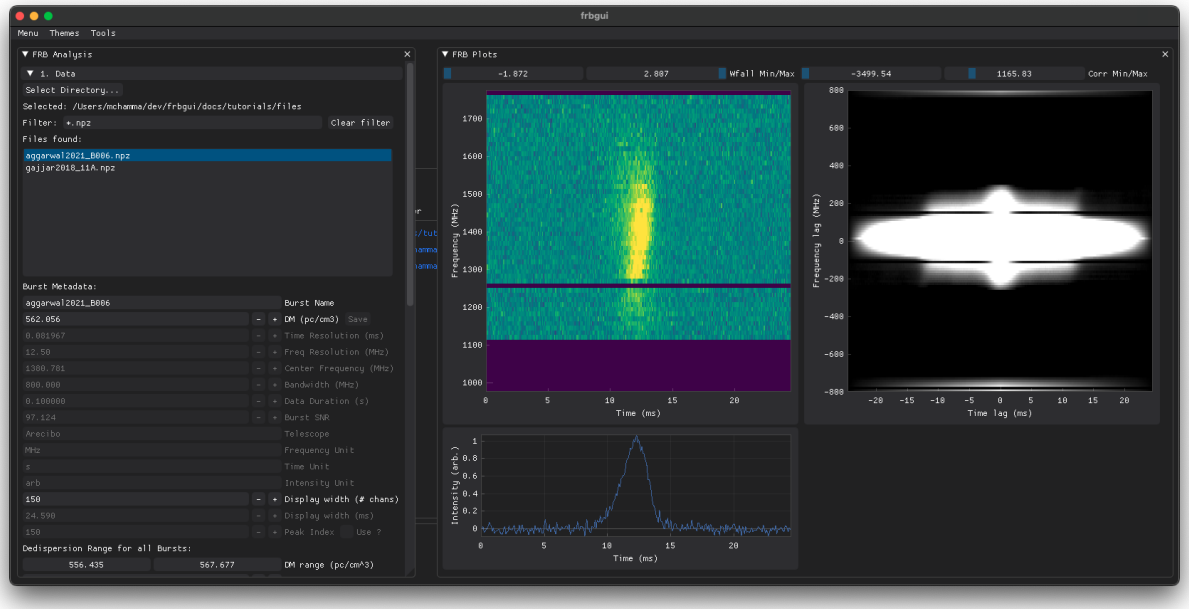



Fig. 1: The main FRBGui window. Click to enlarge.

## 3.1 Measurement Properties

Under the fold-out section "1. Data" in the FRB Analysis window you will find the Burst Metadata subsection. In here we see that the DM of the burst is 562.056 pc/cm $^3$. If needed we could change this DM and save it back into the burst's `.npz` file using the +/- keys next to the DM and the grayed out "Save" button. We will leave this value as is for this burst.

Turning our attention to the "Display Width" field, we see the value 150, which is the number of time channels that are currently being displayed in the waterfall plot. This default value is not the full size of the waterfall saved in the `.npz`, and we can increase this value to display more channels.

Since this burst will rotate as it is measured at different DMs, we want to ensure that there is enough room for it to do so:

**>>** Increase the display width for this burst to 200 channels by typing in the value manually or by clicking the +/- keys, which increments the value by 10 channels each time.

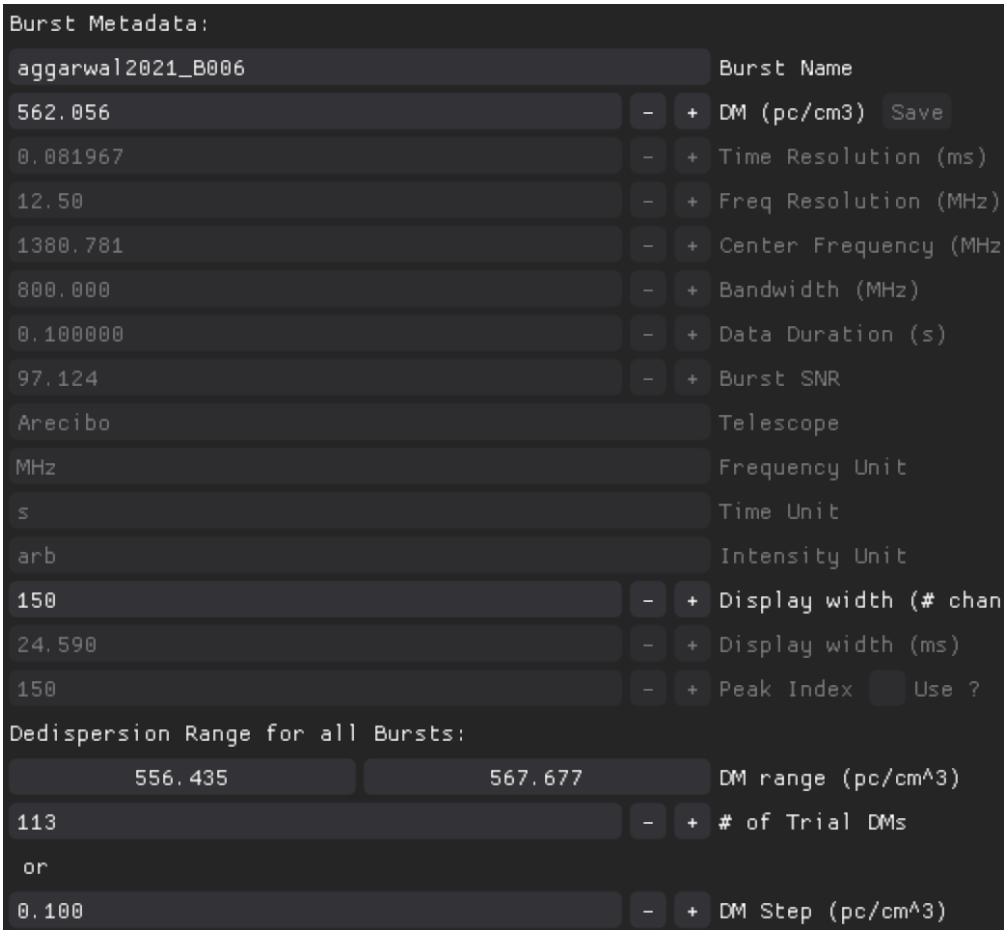
Below the Display Width we can set the Dedispersion Range and the # of Trial DMs which will determine the range of DMs that measurements are repeated over and the number of measurements obtained. The pair of values are displaying a default range of 556.435 to 567.677 pc/cm $^3$. In the interface, the value on the left is the start of the DM range and the value on the right is the end.

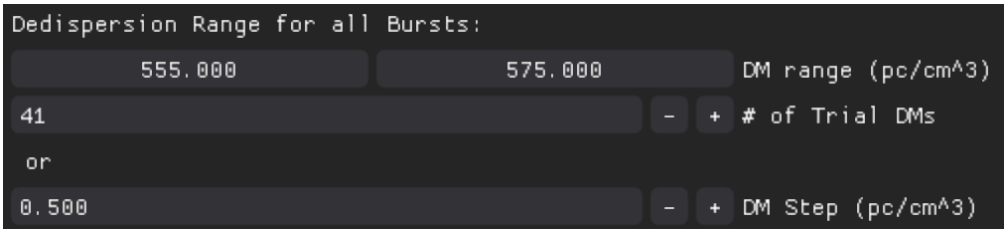
**>>** Change the DM range to 555 to 575 pc/cm by **double-clicking** the values in the range input and set the DM step to 0.5 pc/cm $^3$. This will tell FRBGui to perform measurements over the DMs 555.0, 555.5, 556.0, … 574.5, 575.0 pc/cm $^3$, as well as the burst DM from the loaded file, for a total of 41+1 measurements.

---

**Note:** FRBGui will not allow you to specify the start of the range to be greater than the end of the range. It will overwrite the value for the start of the range if it detects this error. Therefore it is best to input the end of the DM range before inputting the beginning.

In addition, if the DM range inputted does not include the burst's DM as read from the loaded file, a warning will be

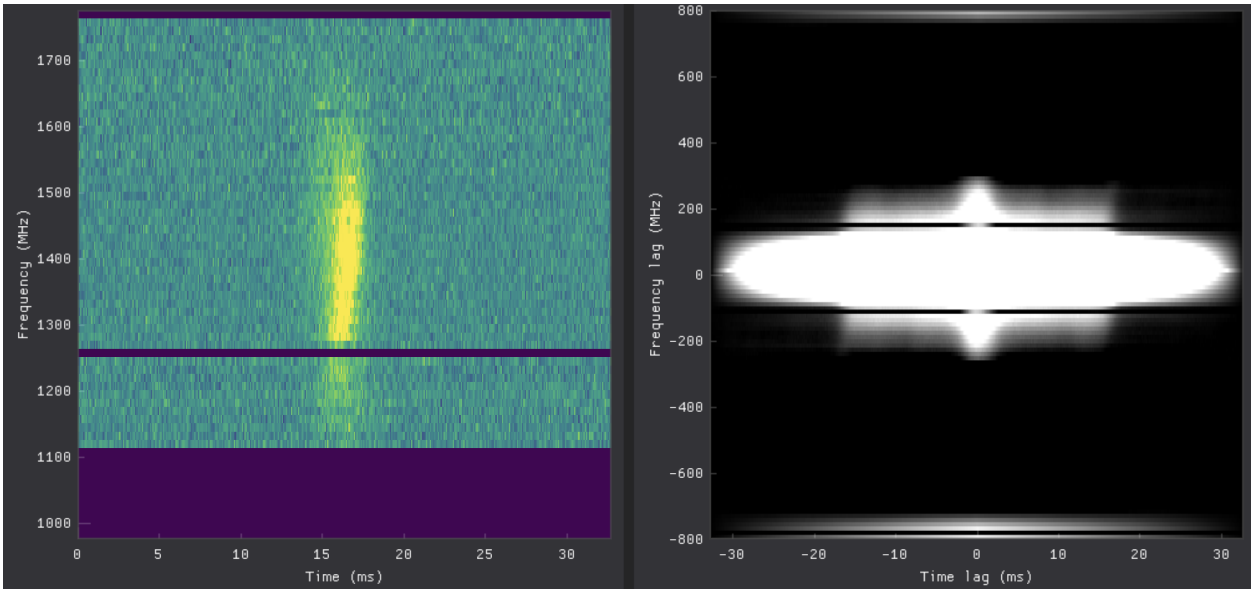

Burst Metadata:
aggarwal2021_B006
Burst Name
562.056
DM (pc/cm3)
Save
0.081967
Time Resolution (ms)
12.50
Freq Resolution (MHz)
1380.781
Center Frequency (MHz
800.000
Bandwidth (MHz)
0.100000
Data Duration (s)
97.124
Burst SNR
Arecibo
Telescope
MHz
Frequency Unit
s
Time Unit
arb
Intensity Unit
150
Display width (# chan
24.590
Display width (ms)
150
Peak Index
Use ?
Dedispersion Range for all Bursts:
556.435
567.677
DM range (pc/cm^3)
113
# of Trial DMs
or
0.100
DM Step (pc/cm^3)


Dedispersion Range for all Bursts:
555.000
575.000
DM range (pc/cm^3)
41
# of Trial DMs
or
0.500
DM Step (pc/cm^3)

displayed. Ensure your chosen DM range includes the DM of the burst.

## 3.2 Cleanup (Background Subtraction)

At this stage we will now look at the waterfall and its 2D autocorrelation function (ACF) in the plot windows in order to assess what kind of cleanup will be necessary before measurement. The burst now appears as in the following:



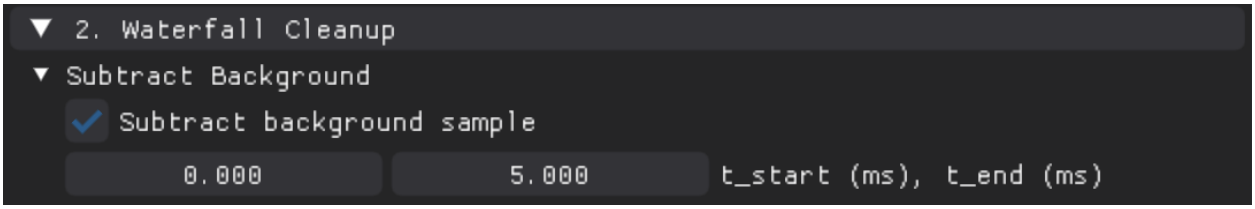

Notice in the waterfall plot, some channels have already been masked, and there is little radio frequency interference (RFI) remaining. RFI typically appear as bright horizontal bars across the waterfall. The data resolution is good and the SNR of the burst is high, as can be seen by the contrast between the colors of the bright burst signal and the background noise.

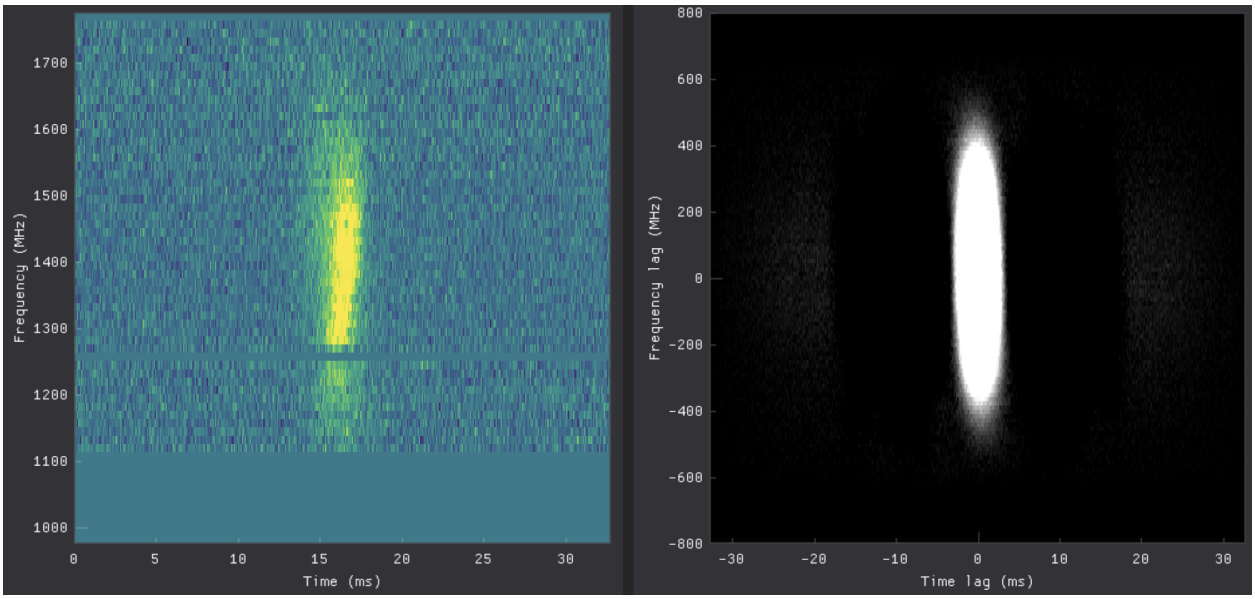
While the burst appears well defined, its ACF however appears blocky and filled with artifacts. Our goal is for the ACF to resemble an ellipse as much as possible. The artifacts in the ACF will be caused by irregularities in the burst waterfall. In this case, the masked out noise channels may not be properly zeroed out, resulting in the blocky structures we see.

To fix this we will use the "Subtract background" feature under the second fold-out section "2. Waterfall Cleanup". Subtract background will take a time averaged sample of the noise as a function of frequency and subtract this sample from each column of data, and is a good and simple way of removing frequency dependent noise that does not change in time from a waterfall. FRBGui allows you to pick the start time (t_start (ms)) and end time (t_end (ms)) of this background sample. Typically you will choose the first few milliseconds of a burst waterall before the burst starts as the background.

**>>** Check the "Subtract background sample" checkbox and set the second value of the number pair (which corresponds to t_end (ms)) to 5.0 ms. Feel free to scrub this value with your mouse to dynamically see the effect this has on the waterfall.

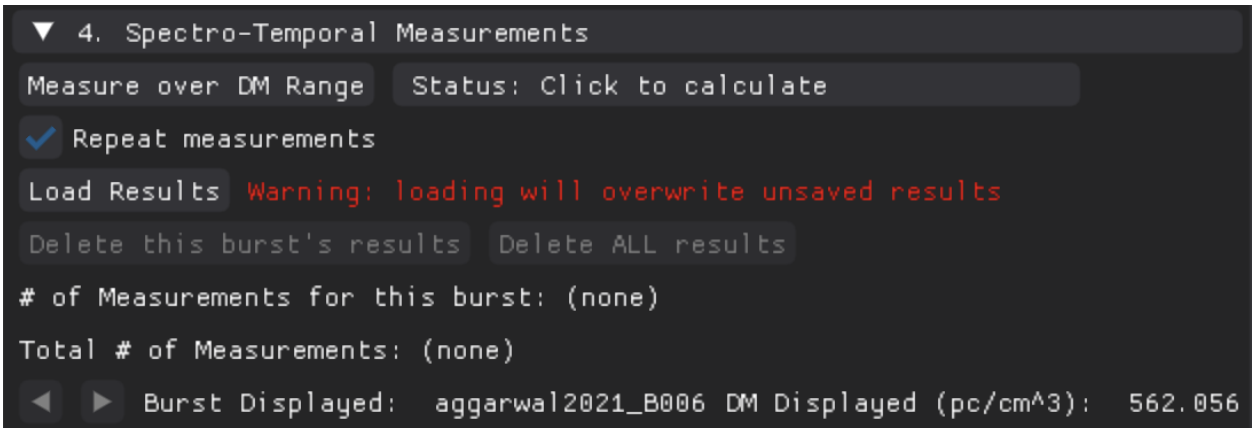
Immediately we see the colors of the burst waterfall update and the artifacts disappear from the ACF leaving behind a well-defined ellipse with a high contrast with its background, indicating a high signal-to-noise. Since there are no components to seperate, we can skip the third fold-out section "3. Burst Splitting". This burst is now ready for measurements.




## 3.3 Performing Measurements

>> In the last fold-out section "4. Spectro-Temporal Measurements", click "Measure over DM Range".

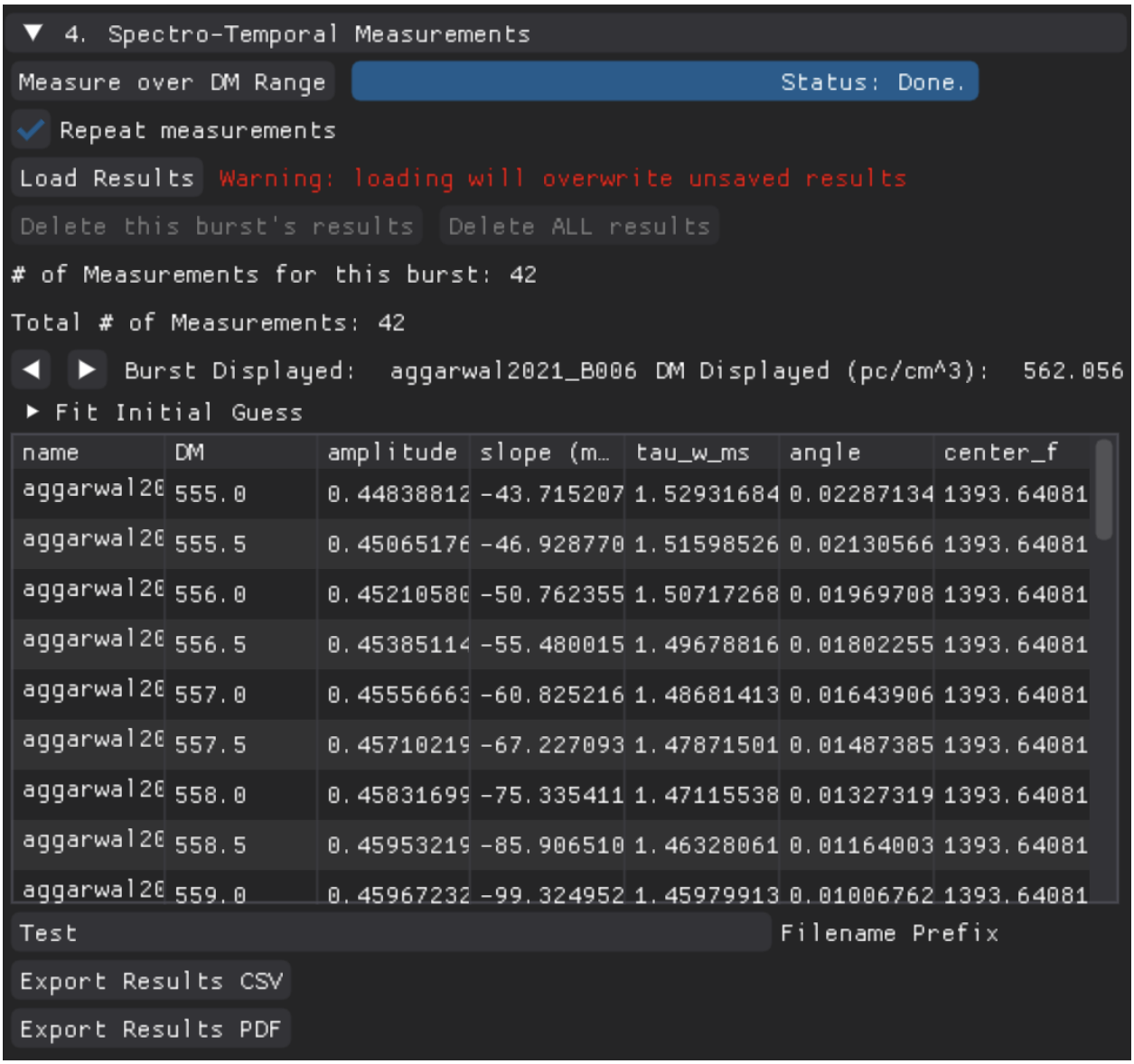



FRBGui will now begin the process of incoherently dedispersing the waterfall to each DM we specified in the DM range and fitting a 2D gaussian the ACF of the burst at that DM. From the parameters of the 2D gaussian, spectro-temporal properties such as the sub-burst slope, duration, bandwidth, and center frequency are computed (See *2D Autocorrelation Method*). Leaving the "Repeat Measurements" box checked (recommended) will tell FRBGui to perform two passes: the first finds a rough fit and then the rough fit is used to refine the fit further in the second pass to improve accuracy.

As FRBGui performs measurements the Status bar next to the "Measure…" button will fill up until measurements are complete. When done, a table with the results (the "results table") will be displayed in the fold-out section along with

information on

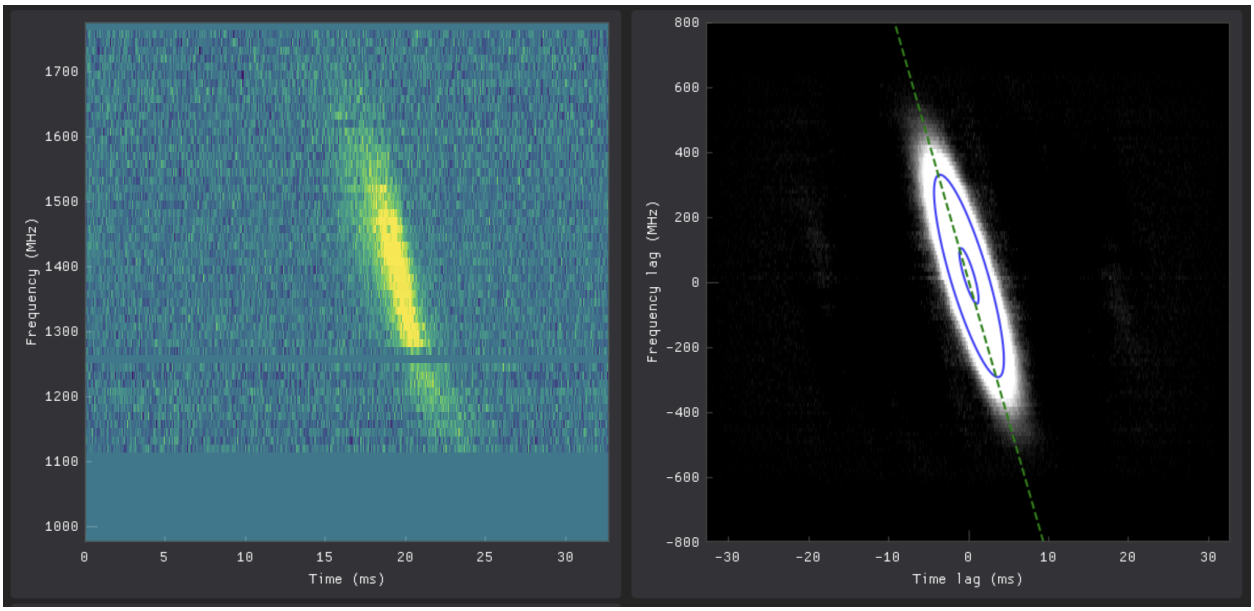
- the total number of measurements
- the burst displayed and the DM it is displayed at




**Note:** If FRBGui crashes for whatever reason, any measurements you have made will have automatically been saved in the `backups` folder FRBGui creates in the same directory you started FRBGui in. You can load the spreadsheet back into FRBGui and continue where you left off from using the "Load Results" button in the "4. Spectro-Temporal Measurements" section or continue afresh knowing that your measurements have already been saved.

## 3.4 Reviewing Measurements

With our measurements complete, we will now review our measurements to ensure that the fits found by FRBGui are accurate to the ACF of the burst. With the measurements complete, the plot figures now show a blue elliptical contours overlaid onto the ACF and a dashed line that corresponds to the sub-burst slope found, as shown in the image below. The blue contours correspond to 90% and 25% of the peak of the 2D Gaussian model fit to the ACF.




By clicking on rows of the results table you can choose which measurement is displayed on the plots. In addition, just above the results table are the left and right triangle scroll buttons, which can be clicked to go through each measurement one by one.

**>>** Click on the first row of the results table to display the first measurement. Check that the blue contours and dashed line line up well with the plot of the burst ACF as shown in the image above.

**>>** Click the button to load the each measurement one-by-one and note whether each contour aligns with its autocorrelation. The currently displayed DM will be listed in the "DM Displayed:" status text just above the results table.

Once you have reached the measurements for the DM of 575 pc/cm $^3$ you may notice the burst tilted so much that it has begun to roll around to the other side of its plot and blobby artifacts appearing on its ACF:

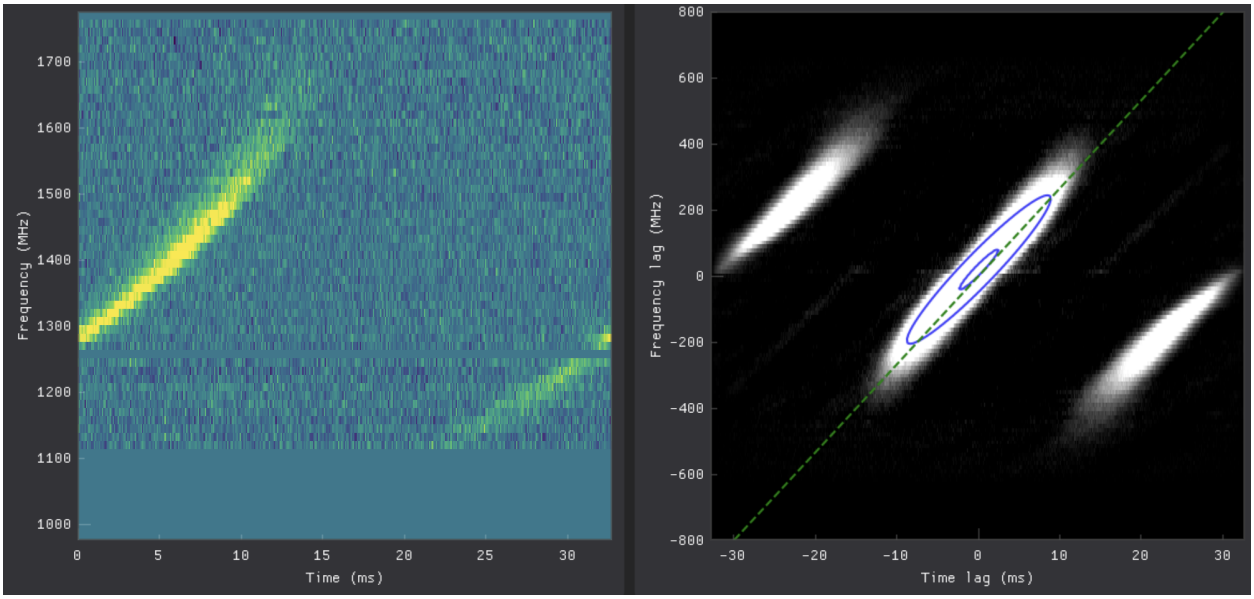
In typical analyses this burst measurement would be excluded on the basis of it being greatly over dedispersed (as evidenced by the characteristic $1/\nu^2$ curve) and even excluded on the basis of its positively sloping sub-burst slope, which is usually assumed to be unphysical.

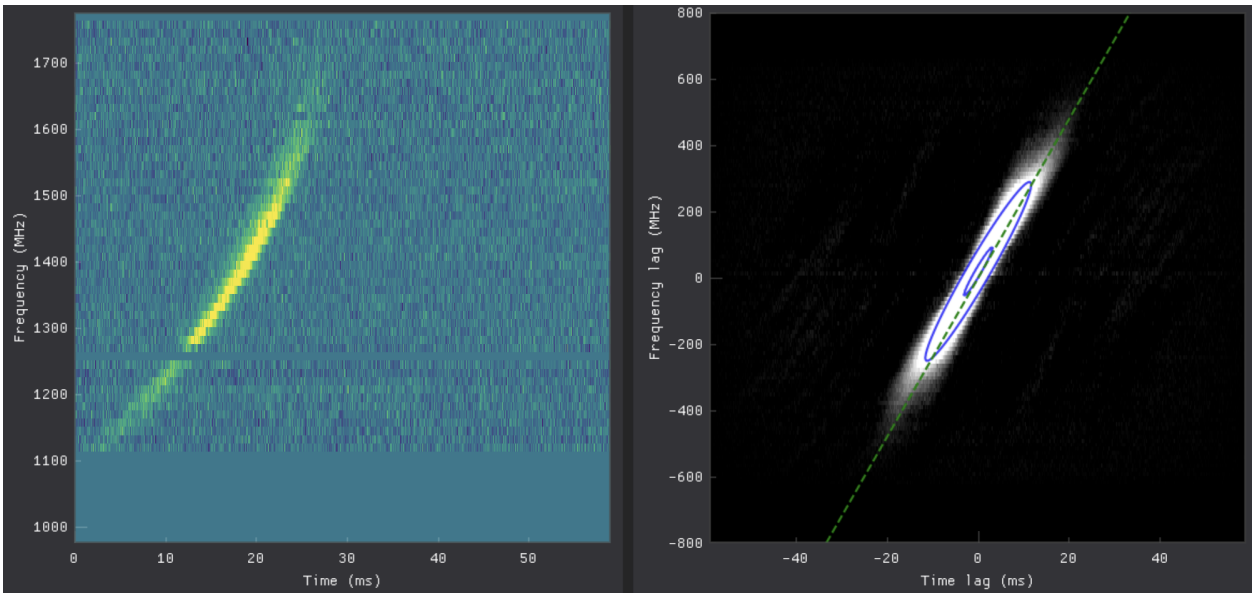
However, we may still want to ensure our set of measurements is accurate so we will adjust our measurement properties and re-measure to correct this issue.

**>>** Scrolling back up to the first fold-out section, increase the "Display Width" further under "Burst Metadata" to 360 channels. Notice the burst appears narrower as FRBGui loads more data into the displayed plot

**>>** Return to "4. Spectro-Temporal Measurements" and click "Measure over DM range" to repeat the measurements with the new measurement properties.

When reviewing the measurements now at a DM of 575 pc/cm $^3$ we should see a clean ACF without artefacts and a more accurate measurement:

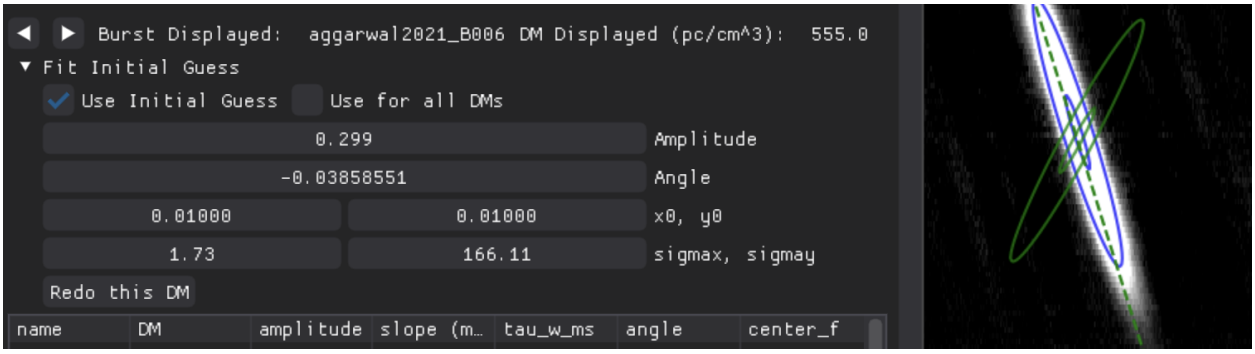

Frequency (MHz)
1700
1600
1500
1400
1300
1200
1100
1000
0
5
10
15
20
25
30
Time (ms)
Frequency lag (MHz)
800
600
400
200
0
-200
-400
-600
-800
-30
-20
-10
0
10
20
30
Time lag (ms)


Frequency (MHz)
1700
1600
1500
1400
1300
1200
1100
1000
0
10
20
30
40
50
Time (ms)
Frequency lag (MHz)
800
600
400
200
0
-200
-400
-600
-800
-40
-20
0
20
40
Time lag (ms)

## 3.5 Manual Initial Fit Guess

In this case the fitting algorithm found accurate measurements automatically without the need for us to input an initial approximate guess manually.

However, if needed, we could input an initial guess by expanding the "Fit initial Guess" section just above the results table, and checking "Use Initial Guess". This adds a green contour set based on the best fit so far to the ACF plot that represents our initial guess. We can modify its parameters using the Amplitude, Angle, x0, y0, sigmax, and sigmay inputs in the interface.




To perform a measurement with our initial guess, we click "Redo this DM", which will redo the fit for the currently displayed measurement. Checking "Use for all DMs" will automatically repeat the measurement using the initial guess one-by-one and display the result as it works. This allows you to monitor the measurements as they are found.

For this burst, the fits found automatically are accurate without the need for a manual initial guess. The measurement of this burst is now complete and we can proceed to our second burst.

## 3.6 Downsampling an FRB

The next FRB we will measure is burst 11A published in Gajjar et al. (2018) which was observed by the Green Bank Telescope from the same source as before, FRB 20121102A, using a 4–8 GHz receiver. This burst is at a much higher data resolution (more frequency channels and more time channels) and is made up of four sub-bursts. That is, four distinct pulses are seen spanning a duration of just 3 milliseconds.

>> Scroll to the top of the FRB analysis window and select the burst "gajjar2018_11A.npz" to load and display the burst.

The burst is loaded and we see that the contrast in the figures is a little lower. This may indicate a low signal-to-noise ratio, but we see the burst pulses prominently in the time series plot below the waterfall so this is more likely due to the choice of color scale. Note that the color scale of the waterfall and ACF plots can be adjusted with the sliders just above each respective plot.

Nonetheless we can increase signal to noise and decrease computation time by downsampling the waterfall, i.e. by decreasing the number of frequency and time channels by averaging them together. While it is best to perform your measurements at full resolution when accuracy is a concern, many bursts may be adequately measured at a lower resolution and much more quickly by downsampling. For bursts with very low signal to noise, fits may be difficult to find, and downsampling is an essential strategy for boosting the signal to noise and securing a good fit.

The DM range settings persist from the first burst so we will move immediately to the waterfall cleanup stage, and downsample this waterfall. This will increase the signal-to-noise but in this case also make it a little easier to manipulate settings in the interface, as this large burst can slow down the response time of the interface.

>> Under "2. Waterfall Cleanup > Downsampling", set `numt` to 1024 and `numf` to 845. These correspond to the number of time and frequency channels in the downsampled waterfall.

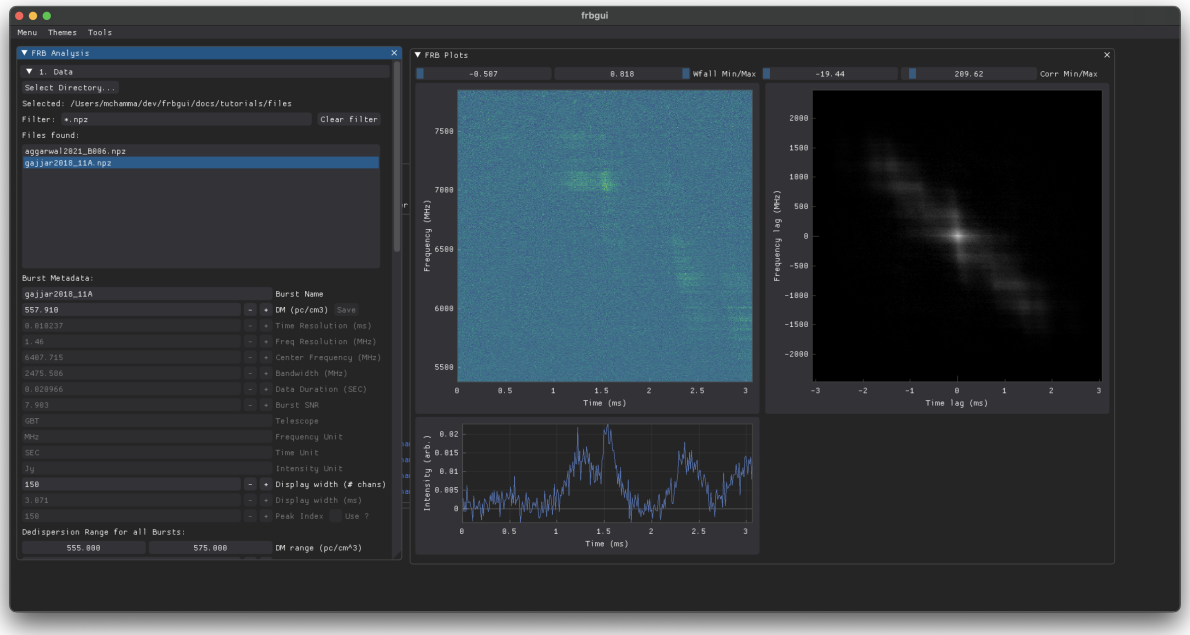
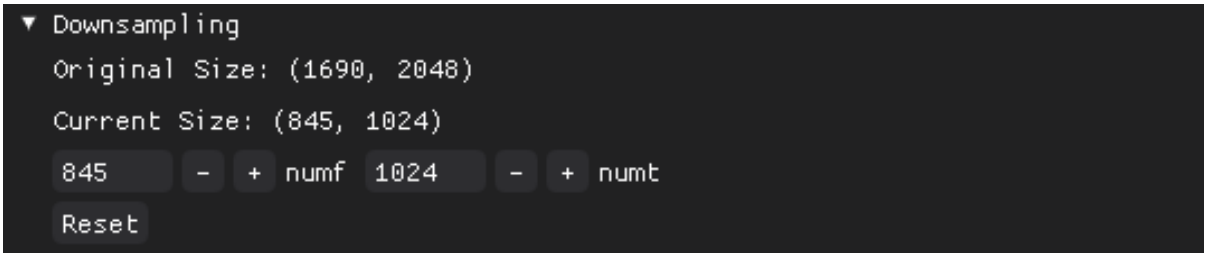



Fig. 2: FRBGui with burst 11A loaded

"Original Size" shows a tuple of the waterfall's shape (frequency channels, time channels). The number you set must evenly divide the original shape. For example, since the original waterfall has 2048 channels, only 1024, 512, 256, 128, 64, etc. are valid inputs. Likewise with 1690, valid inputs are 845 (1690/2), or 338 (1690/5) (for example).

When downsampling the width of the waterfall may change as FRBGui replots the waterfall, so typically there is no need to change the "Display Width" until you have downsampled. In this case, the last pulse is cutoff, so we need to adjust the width.

**>>** Adjust the "Display Width" to show 150 channels after downsampling to 845 by 1024 channels.




Your waterfall now appears as in the following and you can notice a higher contrast in the colors of the waterfall and ACF plot after downsampling.

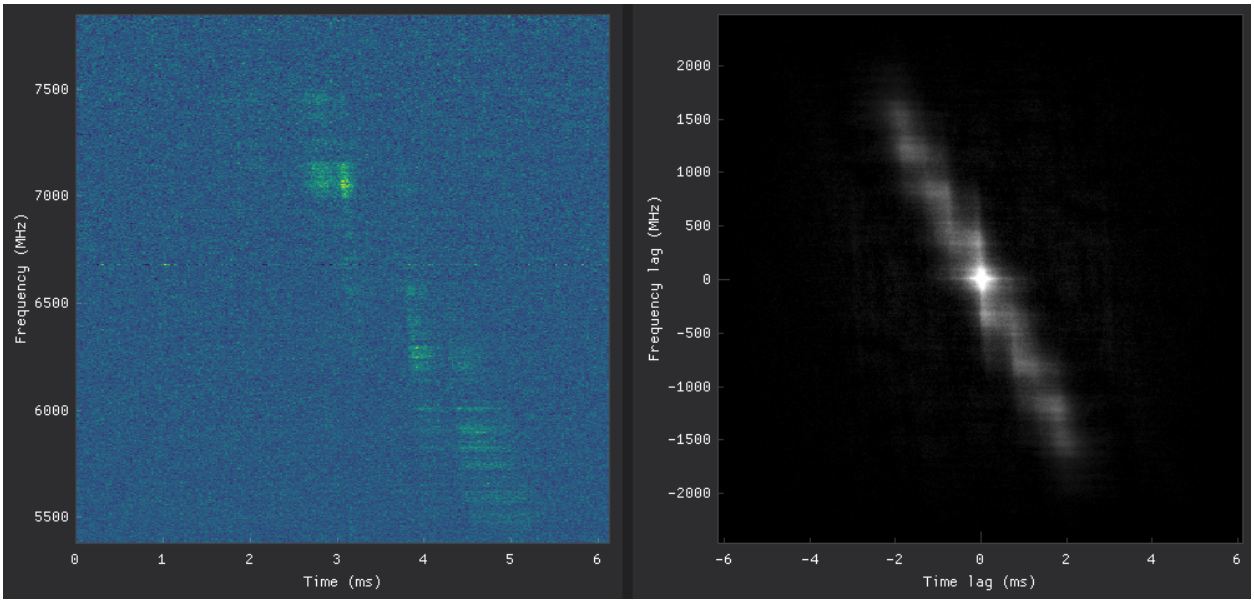
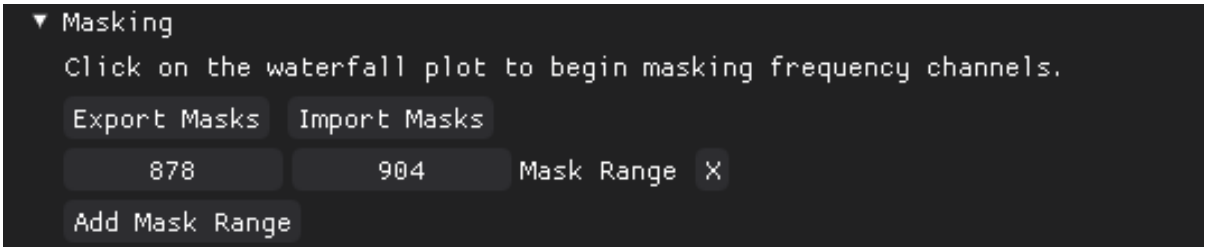



Fig. 3: Click to enlarge

## 3.7 Mask Ranges

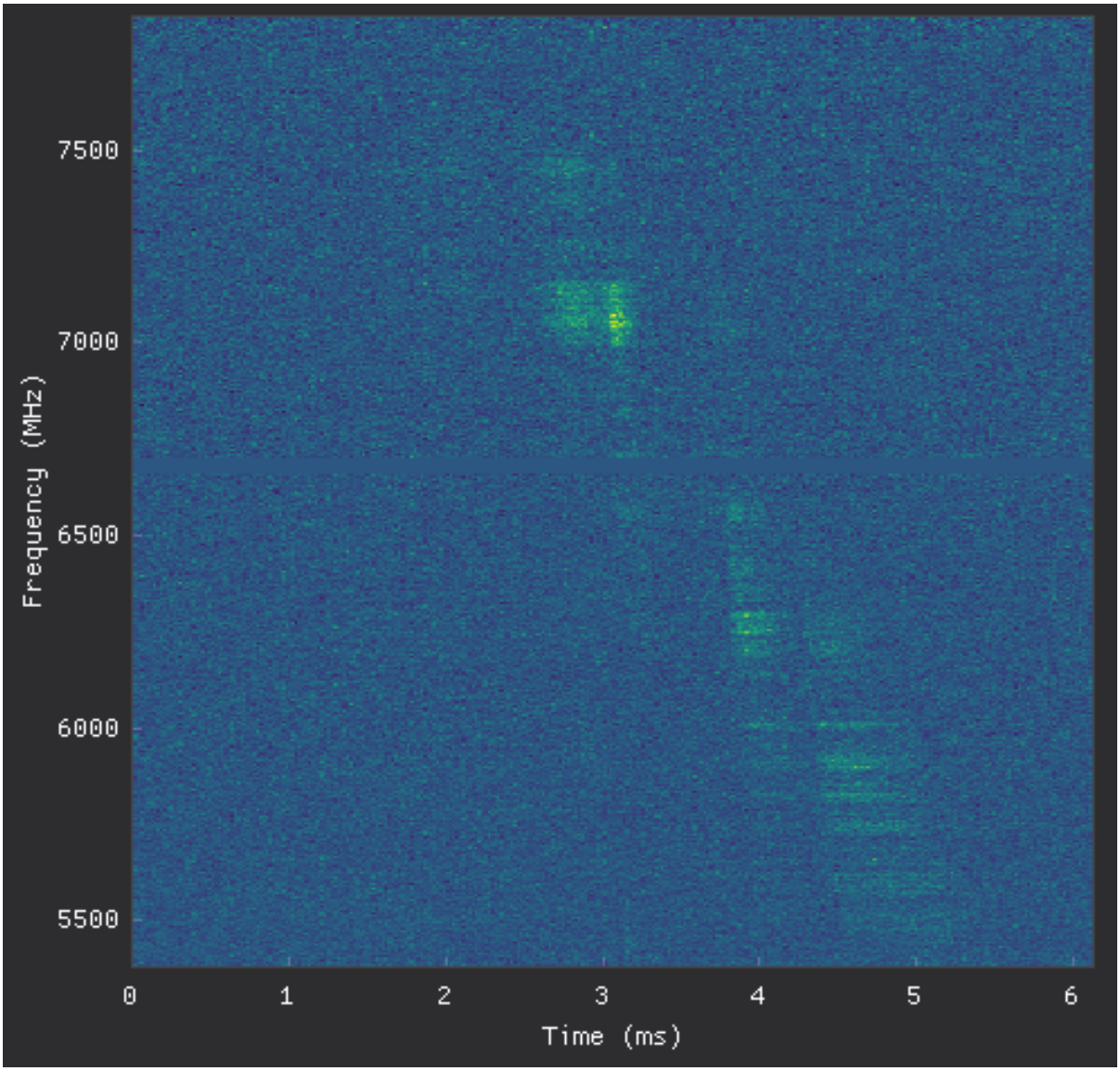
It will be difficult to see, but in the enlarged view of the above image you may notice a faint but distinct thin line streaking the waterfall at around 6600 MHz. This noise feature is minor and will likely not affect measurement, however we can zero it out using the "Mask Range" feature under "Masking" in the "2. Waterfall Cleanup" fold-out.

**>>** Under "Masking", click "Add Mask Range" to add a pair of number fields for specifying a mask range. The left and right values are the starting and ending frequency channels of the mask range. Drag the fields to change the values and notice the blank horizontal gap that appears in the waterfall. Drag the values until the mask range overlaps with the thin narrow feature, or input 880 and 905 into the fields.




After masking your waterfall will appear with a zeroed out band.

The mask range can be removed by selecting the "X" button next to it.


7500
7000
6500
6000
5500
Frequency (MHz)
0
1
2
3
4
5
6
Time (ms)

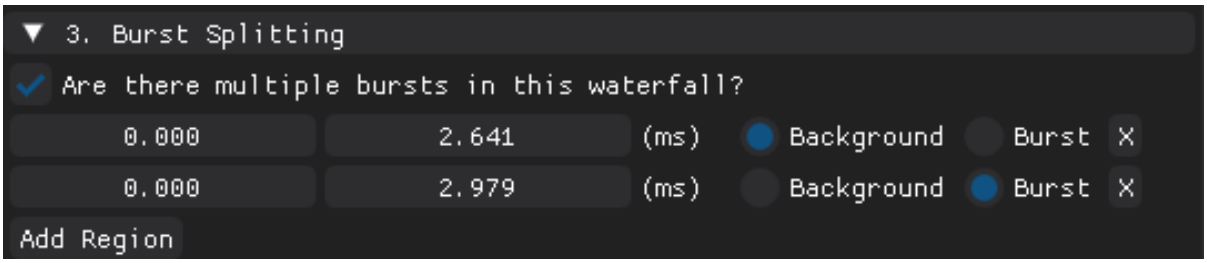
## 3.8 Burst Splitting

We may now choose to proceed with measurement, in which case the Gaussian fit will fit the ACF that is formed by the entire FRB, that is, the 4 sub-bursts that comprise it. This measurement is important and will include the so-called drift rate, or the change in frequency between consecutive resolved bursts. FRB drift rates obey characteristic relationships that are potentially a clue to the underlying emission mechanism that remains a mystery. However, we may also want to measure the change in frequency with time of the individual sub-bursts to obtain a sub-burst slope (also called intra-burst drift rate) as well as the individuated spectro-temporal properties (such as the durations of each pulse).

We can use FRBGui's burst splitting feature to encode the start and end times of each burst and FRBGui will automatically measure each individual sub-burst we specify after measuring the entire FRB. This is done by sectioning the waterfall into regions.

**>>** Under "3. Burst Splitting" check the "Are there multiple bursts in this waterfall" box. The grayed out fields will be enabled.

These fields consist of the beginning and end start times (the left and right values, respectively) as well the type of region you are specifying, either "Background" or "Burst". The first region will default to the "Background" type. Since the spaces between sub-bursts can be quite short, when FRBGui measures sub-bursts it will add a zero-padded region to the left and right of the sub-burst region before measuring. This "Background" region is the amount of zero-padding to add. Typically you can simply specify the region before the burst starts.




**>>** Slide the end of the "Background" region until the line that has appeared on the waterfall representing it is near the beginning of the burst, around 2.6 ms.

**>>** Click "Add Region" to add a new region. This second region will default to "Burst". Slide the end of this region to the very end of the first pulse. Use the time series plot beneath the waterfall to line up the region line that appears to the valley between the two pulses.

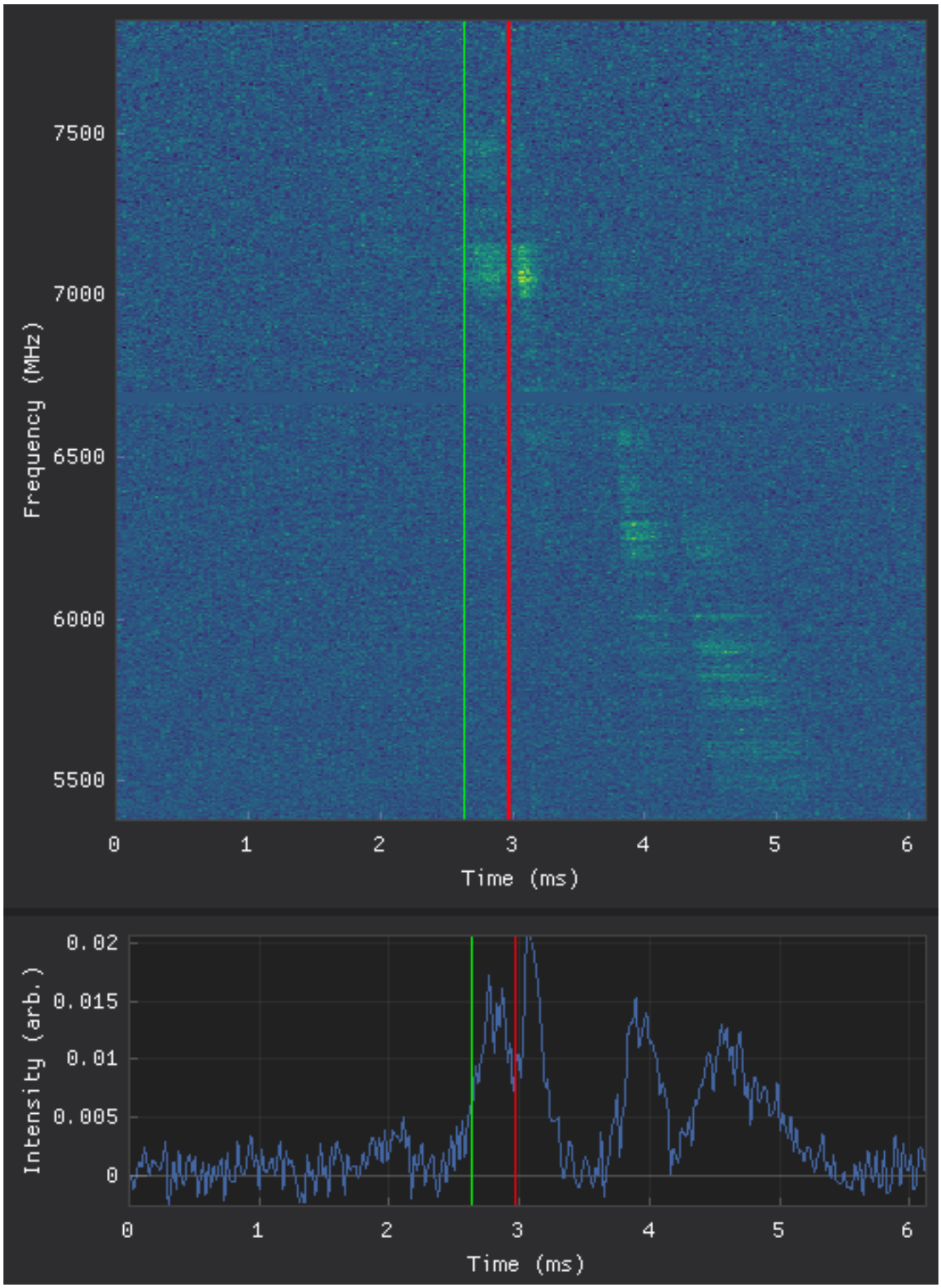
**>>** Continue adding regions until you have specified one background region and the four sub-bursts. Notice that as you add "Burst" regions, the end of the previous region is automatically set as the start of the new region.

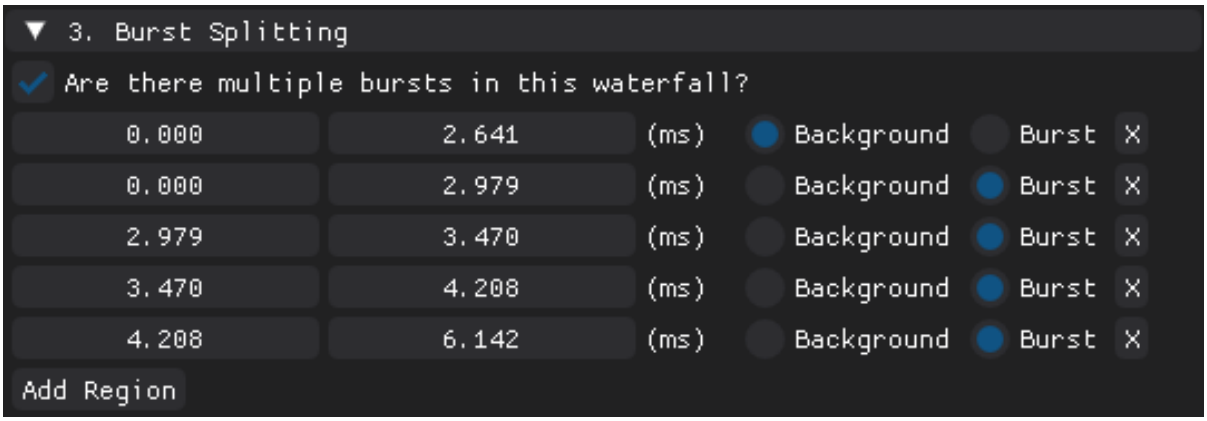
---

**Note:** When splitting sub-bursts it is important to specify the region around the sub-burst to be as large as possible while minimizing overlap. This improves measurement stability and reproducibility.

---

When finished your regions and waterfall may appear as in the following image:


7500
7000
6500
6000
5500
Frequency (MHz)
0
1
2
3
4
5
6
Time (ms)
0.02
0.015
0.01
0.005
0
Intensity (arb.)
0
1
2
3
4
5
6
Time (ms)




## 3.9 Saving and Exporting Results

>> With the regions specified you click "Measure over DM Range" in the "4. Spectro-Temporal Measurements" section and sit back while FRBGui first obtains measurements for the entire waterfall over the DM range and then each of the four individual sub-burst you have specified. The terminal you started may offer additional feedback on the progress as this process can take a few minutes.

When complete all the measurements will again be displayed in a results table. This time, the table includes the sub-bursts suffixed by "_a", "_b", "_c", "_d" corresponding to each of the sub-burst regions specified. Note the number of measurements is 210 which is equal to the 1+4 regions specified times 42 measurements per region.

When reviewing measurements, notice that clicking on the row of a sub-burst measurement, such as "gajjar2018_11A_a", will display the extracted waterfall of the first isolated sub-burst region along with its own ACF. In this way you may verify the fit accuracy of the sub-bursts in addition to the fits of the entire waterfall.

>> Adjust the column sizes of the tables until you can clearly see the burst name and DM. Scroll down and select one of the measurements for "gajjar2018_11A_a". Notice the zero padded vertical regions in the waterfall when viewing sub-bursts. Click rows or use the  and  buttons to review all the measurements in the table.

Since we have finished the list of bursts we wanted to measure, we may now export the measurements to a CSV spreadsheet as well as a PDF of the measurements.

>> At the very bottom of the "FRB Analysis" window below the results table, enter a filename prefix (such as "B006_11A_results") and click "Export Results PDF". This automatically exports a CSV and then plots each measurement and saves it in a corresponding PDF. Note that saving a PDF with many measurements can take a lot of time. If you don't want the PDF you can just click "Export Results CSV".

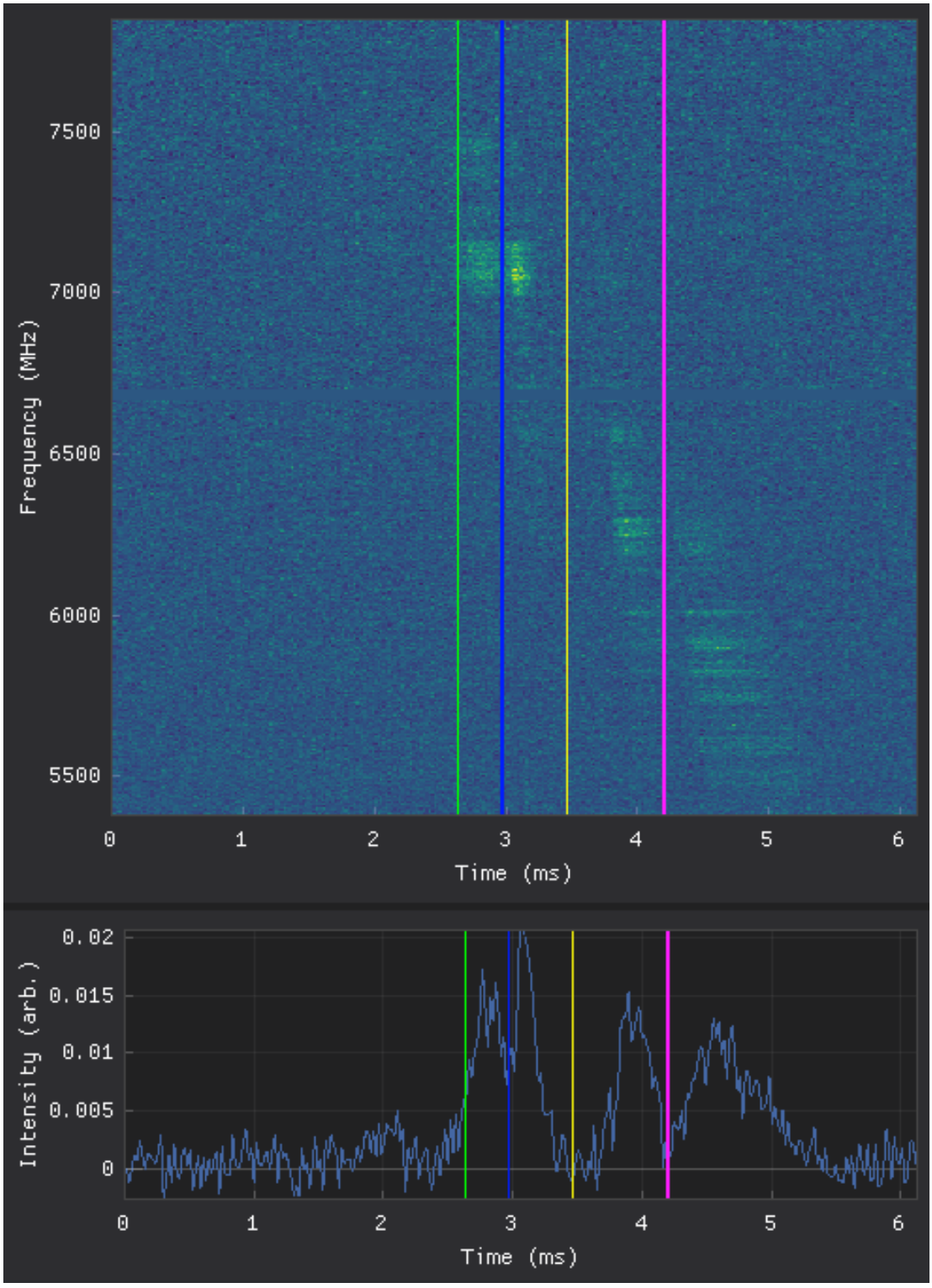
The results files will be saved in the same directory you started FRBGui in. The CSV file also doubles as a save file of your FRBGui session since all the information needed to reproduce your measurements will be saved in the CSV. You can reload your FRBGui session by starting FRBGui in the same directory, clicking "Load Results", and selecting the CSV you previously exported.

Example of the CSV Output. See *Output CSVs* for the meanings of the columns in detail.

Example of the PDF Output. Note that the waterfall plots here show the sub-burst slope or drift rate (dashed line), the duration (the correlation length) with the horizontal bar, and the frequency bandwidth with the vertical bar.

In this tutorial we have taken two FRBs and used FRBGui to prepare the bursts for measurement and measure their spectro-temporal properties, producing CSV and PDF outputs that are useful for later analysis outside of FRBGui and for reviewing measurements visually.

Thank you for following along this tutorial and best of luck on your FRB measuring days.


7500
7000
6500
6000
5500
Frequency (MHz)
0
1
2
3
4
5
6
Time (ms)
0.02
0.015
0.01
0.005
0
Intensity (arb.)
0
1
2
3
4
5
6
Time (ms)

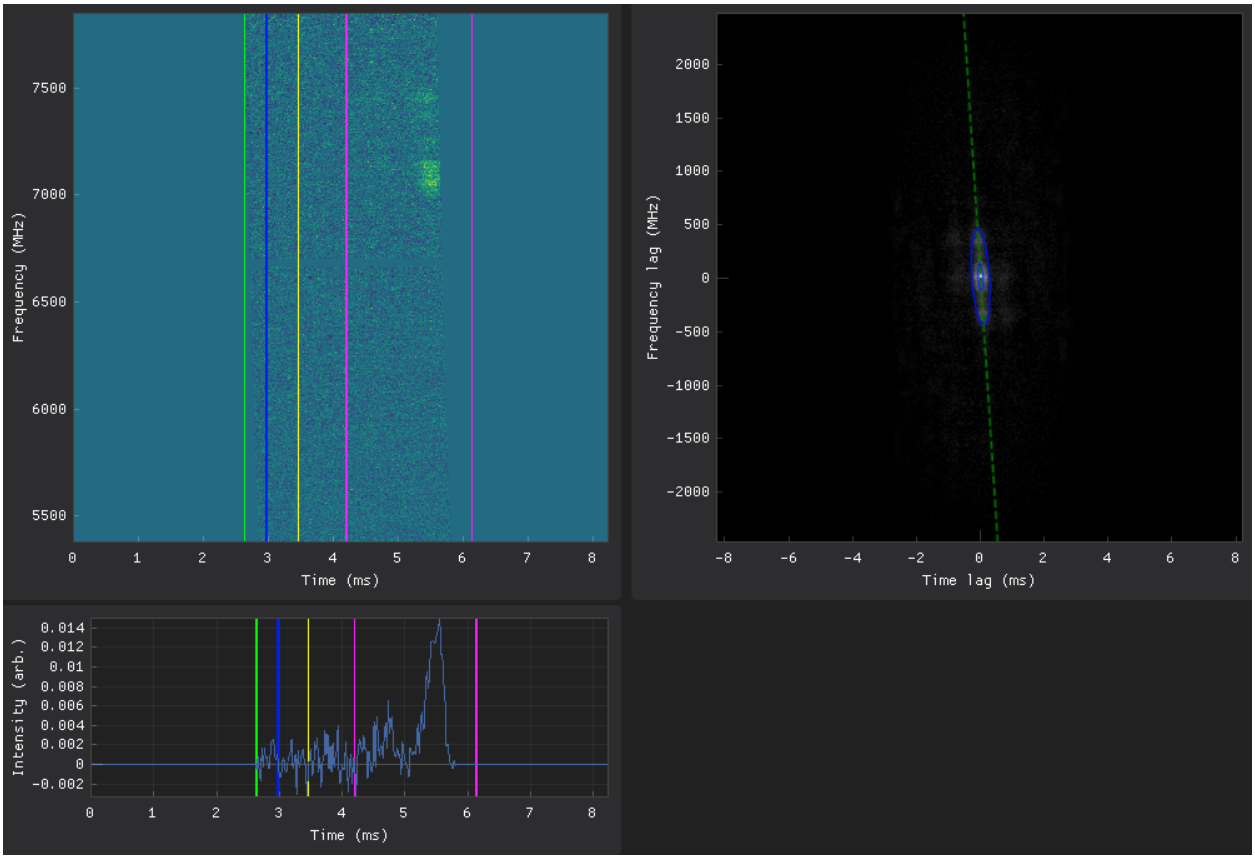
# of Measurements for this burst: 210

Total # of Measurements: 210

◀ ▶ Burst Displayed: gajjar2018_11A_a DM Displayed (pc/cm^3): 555.0

▶ Fit Initial Guess

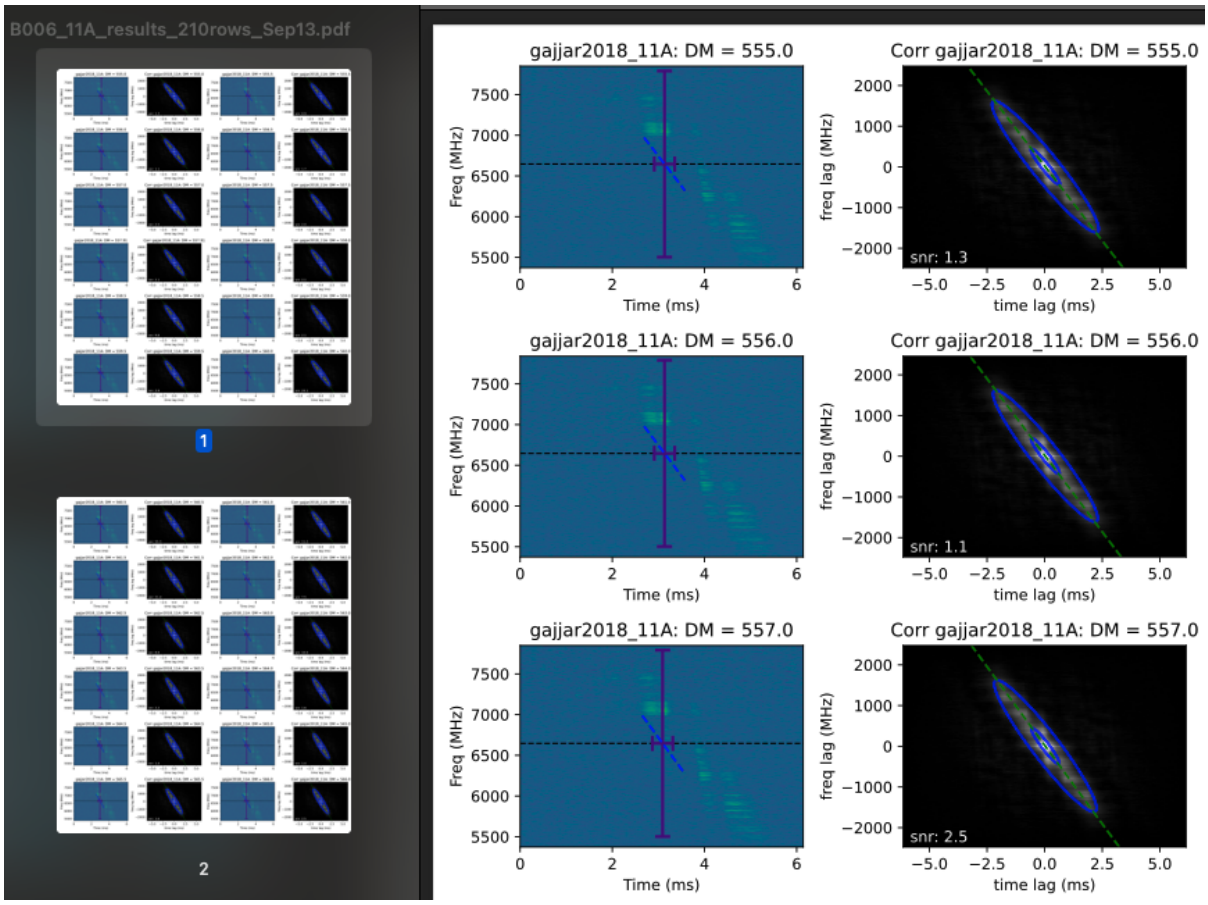
| | | | | | | |
|---|---|---|---|---|---|---|
| gajjar2018_11A | 573. | 0.028351 | -1281.7269 | 0.44671311 | 0.00078019 | 6646.59594 |
| gajjar2018_11A | 574. | 0.028350 | -1307.5269 | 0.44687672 | 0.00076480 | 6646.59594 |
| gajjar2018_11A | 574. | 0.028340 | -1335.8399 | 0.44717690 | 0.00074859 | 6646.59594 |
| gajjar2018_11A | 575. | 0.028321 | -1363.8782 | 0.44742176 | 0.00073320 | 6646.59594 |
| gajjar2018_11A_a | 555. | 0.027235 | -4639.9371 | 0.16621025 | 0.00021552 | 6933.04038 |
| gajjar2018_11A_a | 555. | 0.027169 | -4717.0975 | 0.16680637 | 0.00021199 | 6933.04038 |
| gajjar2018_11A_a | 556. | 0.027172 | -5117.8124 | 0.16711201 | 0.00019539 | 6933.04038 |
| gajjar2018_11A_a | 556. | 0.027188 | -5575.4274 | 0.16721416 | 0.00017935 | 6933.04038 |




B006_11A_results Filename Prefix

Export Results CSV Saved to B006_11A_results_210rows_Sep13.csv

Export Results PDF Saved to B006_11A_results_210rows_Sep13.pdf




| name | DM | center_f | center_f_err | slope (mhz/ms) | slope error (mhz/ms) | t |
|---|---|---|---|---|---|---|
| gajjar2018_11A | 555.0 | 6646.595941541430 | 1.970331862266630 | -734.9800112880090 | 22.46320873701810 | - |
| gajjar2018_11A | 555.5 | 6646.595941541430 | 1.9625629148358200 | -745.0019474609870 | 22.854356075044000 | - |
| gajjar2018_11A | 556.0 | 6646.595941541430 | 1.9583074072227400 | -752.6596409764680 | 23.420664610678300 | - |
| gajjar2018_11A | 556.5 | 6646.595941541430 | 1.9564213555744900 | -761.1687590617170 | 23.934537438813600 | - |
| gajjar2018_11A | 557.0 | 6646.595941541430 | 1.9555848612093500 | -772.4800917160730 | 24.483161155607800 | - |
| gajjar2018_11A | 557.5 | 6646.595941541430 | 1.949120918392660 | -778.919624681142 | 24.96766659896370 | - |
| gajjar2018_11A | 557.91 | 6646.595941541430 | 1.9364152698845200 | -788.1720448156490 | 25.468958755157000 | - |
| gajjar2018_11A | 558.0 | 6646.595941541430 | 1.9364152698845200 | -788.171881059219 | 25.468910848738300 | |
| gajjar2018_11A | 558.5 | 6646.595941541430 | 1.952582144245640 | -802.7789240308150 | 26.441987852650800 | - |
| gajjar2018_11A | 559.0 | 6646.595941541430 | 1.9543700543817500 | -809.0992031099100 | 26.946031531464000 | - |
| gajjar2018_11A | 559.5 | 6646.595941541430 | 1.9556015982009200 | -820.045821149214 | 27.472537946923700 | |
| gajjar2018_11A | 560.0 | 6646.595941541430 | 1.9562469557162300 | -829.4763543906720 | 28.351496456067100 | - |

CHAPTER

# FOUR

# SCRIPTING

FRBGui ships with the following modules that can be used for scripting outside of the graphical interface:

- `driftrate.py` : Utilities for manipulating FRB waterfalls and performing measurements on them.
- `driftlaw.py` : Utilities for processing FRBGui measurements for the purpose of studying spectro-temporal relationships.

These can be accessed with

```
import driftrate
import driftlaw
```

**This page is currently under construction and may be incomplete.**

## 4.1 `frbgui` module

The `frbgui` function itself has multiple options that can be used to configure the startup of FRBGui

As stated in the Overview, FRBGui can be called from a script like so:

```
from frbgui import frbgui
```

`frbgui.`**`frbgui`**(*filefilter='*.npz'*, *datadir=''*, *maskfile=None*, *dmrange=None*, *numtrials=10*, *dmstep=0.1*, *winwidth=1700*, *winheight=850*, *winpos=(0, 0)*)

Start FRBGui window. Every option has a default so all parameters are optional.

**Parameters**

- **`filefilter`** (*`str`*) – A glob pattern to filter for by default. e.g. '*.npz'
- **`datadir`** (*`str`*) – path to the directory to start frbgui in
- **`maskfile`** (*`str`*) – if masks were exported previously, they can be loaded by default by adding a path to the mask file here
- **`dmrange`** (*`tuple`*) – a list or tuple of the default DM range to use (e.g. [555, 575])
- **`numtrials`** (*`int`*) – number of measurements to perform over dmrange
- **`dmstep`** (*`float`*) – size of DM step to use. Will overwrite numtrials
- **`windiwdth`** (*`int`*) – width in pixels of FRBGui window
- **`winheight`** (*`int`*) – height in pixels of FRBGui window
- **`winpos`** (*`tuple`*) – (x,y) coordinates on the screen of where to start FRBGui

## 4.2 driftrate module

This module contains Utilities for manipulating FRB waterfalls (such as dedispersion and subsampling) and performing measurements on them.

driftrate.**autocorr2d**(*data*, *maskcorrpeak=True*)

Returns a 2D autocorrelation computed via an intermediate FFT

**Parameters**

- **data** (*np.ndarray*) – 2D array of size MxN
- **maskcorrpeak** (*bool, optional*) – Set to False to keep the zero lag peak

**Returns**
2D array of size 2M-1 x 2N-1. The autocorrelation of `data`.

**Return type**
np.ndarray

driftrate.**cropwfall**(*wfall*, *twidth=150*, *pkidx=None*)

Crops a waterfall and attempts to center the burst in the array.

**Parameters**

- **wfall** (*np.ndarray*) – the array to be cropped of size M x N
- **twidth** (*int*) – the number of channels on either side of the burst. Total width is therefore 2*twidth
- **pkidx** (*int, optional*) – the index of the waterfall to center. If None will use np.argmax of the timeseries as the index to center.

**Returns**
cropped waterfall of size M x (2*twidth)

**Return type**
np.ndarray

driftrate.**dedisperse**(*intensity*, *DM*, *nu_low*, *df_mhz*, *dt_ms*, *cshift=0*, *a_dm=4149377.59336*)

Incoherently dedisperse a dynamic spectrum to the specified DM.

Computes the time delay at each frequency and correspondingly rolls the data

$$\Delta t = -a\Big(\frac{1}{\nu_i^2} - \frac{1}{\nu_{\text{high}}^2}\Big)\text{DM}$$

By default uses the pulsar community dispersion constant of

$a = 4.14937759336e6 \quad \text{MHz}^2\text{cm}^3\text{pc}^-1\text{ms}.$

**Parameters**

- **intensity** (*np.ndarray*) – the dynamic spectrum (or waterfall) to dedisperse. `intensity[0]` should correspond to the lowest frequency.
- **DM** (*float*) – the dispersion measure in pc/cm^3
- **nu_low** (*float*) – the frequency at the bottom of the band
- **df_mhz** (*float*) – the frequency resolution of the channels in MHZ
- **dt_ms** (*float*) – the time resolution of channels in ms

- **cshift** (*int, optional*) – additional horizontal shift to add after dedispersion in number of channels
- **a_dm** (*float*) – the dispersion constant to use when dedispersing. See units above.

**Returns**
the dedispersed 2d intensity array

**Return type**
np.ndarray

driftrate.**exportresults**(*results*)

Creates a dataframe of measurement results.

See `driftrate.columns`.

**Parameters**
**results** (*list*) – the results list

driftrate.**findCenter**(*burstwindow*)

Find the center *index* with uncertainties of the peak of the waterfall. This is done by time averaging the waterfall to obtain a spectrum and then finding the average channel weighted by the square of the intensity.

This is calculated as

$$\bar{\nu_i} = \frac{\sum_{\nu_i} \nu_i I^2(\nu_i)}{\sum_{\nu_i} I^2(\nu_i)}$$

The full uncertainty (after taking into account units and standard error propagation) is given by

$$\delta\bar{\nu} = \sqrt{\nu_{\text{res}}^2/12 + 4\sigma_I^2\left(\nu_{\text{res}}\frac{\sum_\nu(\nu-\bar{\nu})I(\nu)}{\sum_\nu I^2(\nu)}\right)^2}$$

**Parameters**
**burstwindow** (*np.ndarray*) – 2d array of the waterfall

**Returns**

(meanfreqi, errorsum) - tuple of the mean peak frequency index and summation term of the uncertainty.

You may then calculate the mean frequency in MHz with

```
center_f = meanfreqi*fres_MHz + lowest_freq
```

The uncertainty on `center_f` is then

```
center_f_err = np.sqrt(fres_MHz**2/12 + 4*wfallsigma**2*(errorsum*fres_
↪MHz)**2)
```

`wfallsigma` ($\sigma_I$) is the standard deviation of the intensity noise. This can be sampled from the noise in the waterfall that doesn't include the burst. For example:

```
wfallsigma = np.std( burstwindow[:, 0:burstwindow.shape[1]//20] )
```

**Return type**
tuple

driftrate.**fitdatagaussiannlsq**(*data*, *extents*, *p0=[]*, *sigma=0*, *bounds=(-inf, inf)*)

Fit 2d gaussian to data with the non-linear least squares algorithm implemented by scipy.optimize.curve_fit.

Uses the physical units as the data's coordinates.

**Parameters**

- **data** (*np.ndarray*) – 2D array of the data to fit on
- **extents** (*tuple*) – Tuple of the initial time, final time, initial frequency, and final frequency (ti, tf, nu_i, nu_f). See *getExtents()*.
- **p0** (*list, optional*) – initial guess to used, input as a list of floats corresponding to [amplitude, x0, y0, sigmax, sigmay, theta]. See *twoD_Gaussian()*.
- **sigma** (*float, optional*) – uncertainty to use during fitting. See Scipy Documentation for more information
- **bounds** (*tuple, optional*) – lower and upper bounds on parameters. See Scipy Documentation for more information

**Returns**

tuple of (popt, pcov) where popt is the list of gaussian parameters found and pcov is the covariance matrix.

**Return type**

tuple

driftrate.**fitgaussiannlsq**(*data*, *p0=[]*, *sigma=0*, *bounds=(-inf, inf)*)

Deprecated function. See `fitdatagaussianlsq()`

driftrate.**getDataCoords**(*extents*, *shape*)

driftrate.**getExtents**(*wfall*, *df: float = 1.0*, *dt: float = 1.0*, *lowest_freq: float = 1.0*, *lowest_time: float = 0.0*)

Given a waterfall's time and frequency resolutions, return the extents of the axes as well as the axes of its autocorrelation.

Convenience function for plt.imshow's extent keyword

**Parameters**

- **wfall** (*np.ndarray*) – 2D array of the waterfall
- **df** (*float*) – frequency resolution
- **dt** (*float*) – time resolution
- **lowest_freq** (*float*) – lowest frequency in the band

**Returns**

(extents, corrextents) where extents is a list of the waterfall extents and corrextents is a list of the corresponding autocorrelation extents

**Return type**

tuple

driftrate.**getSubbursts**(*wfall_cr*, *df*, *dt*, *lowest_freq*, *regions*)

Split a waterfall into regions.

Specify regions with a dictionary. For example

```
regions = {"a": [0, 3], "b": [3, 5.5]}
```

where the arrays are the timestamps in milliseconds.

Regions should be named as "a", "b", "c", etc. See `driftrate.subburst_suffixes`.

Meant for use by frbgui.

**Parameters**

- **wfall_cr** (*np.ndarray*) – waterfall to split
- **df** (*float*) – frequency resolution
- **dt** (*float*) – time resolution
- **lowest_freq** (*float*) – lowest frequency
- **regions** (*dict*) – Dictionary of {"a": [start_ms, end_ms], etc..}

**Returns**
A dictionary of cropped waterfalls. Follows {"a": np.ndarray}

**Return type**
dict

driftrate.**makeDataFitmap**(*popt*, *corr*, *extents*)

Make a fitmap using physical coordinates

**Parameters**

- **popt** (*list*) – List of 2D gaussian model parameters. [amplitude, xo, yo, sigmax, sigmay, theta)]
- **corr** (*np.ndarray*) – the 2d autocorrelation. Can simply pass a 2d array of the same size
- **extents** (*list*) – Return value of *getExtents()*

**Returns**
a 2d array of the 2d gaussian specified by popt

**Return type**
np.ndarray

driftrate.**makeFitmap**(*popt*, *corr*)

Deprecated function. See *makeDataFitmap()*

driftrate.**moments**(*data*)

Returns (height, x, y, width_x, width_y) initial gaussian parameters of a 2D distribution by calculating its moments

**Parameters**
**data** (*np.ndarray*) – the data to compute moments of

**Returns**
(height, x, y, sigma_x, sigma_y)

**Return type**
list

driftrate.**plotResults**(*resultsfile*, *datafiles=[]*, *masks=None*, *figsize=(14, 16)*, *nrows=6*, *ncols=4*, *clip=1*, *snrLines=False*, *show=False*)

Given a results CSV produced by FRBGui will plot fit results by burst at each DM in a stampcard and save a PDF with the same name.

**Parameters**

- **resultsfile** (*str*) – Filename of CSV file
- **datafiles** (*list[str]*) – list of burst filenames in FRBGui's .npz *Burst Format*
- **masks** (*str*) – filename to FRBGui maskfile
- **figsize** (*tuple(float)*) – (width, height) of figure
- **nrows** (*int*) – number of rows in stampcard
- **ncols** (*int*) – number of cols in stampcard
- **clip** (*int*) – factor to clip the color scale of the autocorrelation figure for improving display SNR
- **snrLines** (*bool*) – If true plots 1 sigma lines around the autocorrelation
- **show** (*bool*) – Show the figure in a window if true. Displays a pdf otherwise.

**Returns**
True if completed. Saves a PDF file when completed.

**Return type**
bool

driftrate.**plotStampcard**(*loadfunc*, *fileglob='*.npy'*, *figsize=(14, 16)*, *nrows=6*, *ncols=4*, *twidth=150*)

Plot bursts and their autocorrelations in a stampcard

Optionally do custom processing to find the slope and plot the resulting fit as well.

**Parameters**

- **loadfunc** (*function*) – a function that accepts "filename" to a waterfall as an argument and loads the waterfall and returns (subfall, pkidx, wfall) where `wfall` is the loaded waterfall, `subfall` is the subbanded waterfall that you want to see, and `pkidx` is the index in the timeseries of the data with peak intensity. Implement this yourself to load your data files
- **fileglob** (*str*) – a glob that matches on your data files (e.g. "*.npy")
- **figsize** (*tuple[float]*) – Tuple of (width, height) for the resulting figure
- **nrows** (*int*) – number of rows in the stampcard
- **ncols** (*int*) – number of columns in the stampcard
- **twidth** (*int*) – number of channels on either side of the burst to plot

**Returns**
shows a figure. Useful in jupyterlab for example.

**Return type**
None

driftrate.**processBurst**(*burstwindow*, *fres_MHz*, *tres_ms*, *lowest_freq*, *burstkey=1*, *p0=[]*, *popt_custom=[]*, *bounds=(-inf, inf)*, *nclip=None*, *clip=None*, *plot=False*, *corrsigma=None*, *wfallsigma=None*, *maskcorrpeak=True*, *verbose=True*, *lowest_time=0*)

Given a waterfall of a burst, will use the 2d autocorrelation+gaussian fitting method to perform spectro-temporal measurements of the burst

Can optionally plot the measurement.

**Parameters**

- **burstwindow** (*np.ndarray*) – 2d array of the burst waterfall
- **fres_MHz** (*float*) – frequency resolution in MHz

- **tres_ms** (*float*) – time resolution in ms
- **lowest_freq** (*float*) – lowest frequency in MHz
- **burstkey** (*int, optional*) – Burst number. Used in plot title
- **p0** (*list, optional*) – Initial 2d gaussian guess to use. [amplitude, x0, y0, sigmax, sigmay, theta]
- **popt_custom** (*list, optional*) – Override the fit and plot your own 2D gaussian by placing your popt list here
- **bounds** (*tuple, optional*) – parameter bounds. See `fitdatagaussianlsq()`
- **nclip** (*float, optional*) – minimum clip value of autocorrelation. Applied before fitting.
- **clip** (*float, optional*) – maximum clip value of autocorrelation. Applied before fitting.
- **plot** (*bool, optional*) – If true will display a diagnostic plot of the fit
- **corrsigma** (*float, optional*) – Standard deviation of noise of autocorrelation. Used when fitting.
- **wfallsigma** (*float, optional*) – Standard deviation of noise of waterfall. Used when fitting.
- **verbose** (*bool, optional*) – Set to False to limit console output
- **maskcorrpeak** (*bool, optional*) – Set to False to keep the zero lag correlation peak
- **lowest_time** (*float, optional*) – starting time (ms) of waterfall. Default 0.

**Returns**
(slope, slope_error, popt, perr, theta, red_chisq, center_f, center_f_err, fitmap)

**Return type**
tuple

driftrate.**processDMRange**(*burstname*, *wfall*, *burstdm*, *dmrange*, *fres_MHz*, *tres_ms*, *lowest_freq*, *p0=[]*, *corrsigma=None*, *wfallsigma=None*, *tqdmout=None*, *progress_cb=None*, *lowest_time=0*)

Process burst measurements over a range of DMs.

Dedisperses to each DM in `dmrange` and calls *processBurst()* on the resulting waterfall. Returns a list of all measurements as well as a dataframe. Columns of the dataframe are given by `driftrate.columns`

**Parameters**

- **burstname** (*str*) – name of burst to use in results dataframe
- **wfall** (*np.ndarray*) – 2d array of burst waterfall
- **burstdm** (*float*) – the DM of the burst in `wfall`
- **dmrange** (*list[float]*) – a list of the DMs you would measurements at
- **fres_MHz** (*float*) – frequency resolution in MHz
- **tres_ms** (*float*) – time resolution in milliseconds
- **lowest_freq** (*float*) – lowest frequency of the waterfall
- **p0** (*list, optional*) – optionally provide an initial guess for the 2d gaussian fit with list of [amplitude, x0, y0, sigmax, sigmay, theta]
- **corrsigma** (*float, optional*) – standard deviation of the burst's autocorrelation

- **wfallsigma** (*float, optional*) – standard deviation of the burst waterfall
- **tqdmout** (*object, optional*) – output for TQDM's progress bar
- **progress_cb** (*function, optional*) – callback to run after each DM is processed. Called with ( #of DMs processed) / len(dmrange) and a string of "{#}/{len(dmrange)}"
- **lowest_time** (*float, optional*) – the starting time of the waterfall, passed to *processBurst()*.

**Returns**
(list of measurement results, dataframe of measurement results)

**Return type**
tuple

driftrate.**readRegions**(*resultsdf*)

Parse regions out of a results csv produced by FRBGui

**Parameters**
**resultsdf** (*pd.DataFrame*) – dataframe of results. Can load with `pd.read_csv(filename).set_index('name')`

driftrate.**scilabel**(*num*, *err*)

Utility for pretty scientific notation labels

e.g. (6.0 +/- 0.3) $times 10^2$ MHz/ms

**Parameters**

- **num** (*float*) – the number to label
- **err** (*float*) – the uncertainty of `num`

**Returns**
the formatted label string

**Return type**
str

driftrate.**structureParameter**(*wfall*, *dt*, *tstart*, *tend*)

wip. Compute the structure parameter of a burst.

See eq. 1 in gajjar et al. 2018

**Parameters**

- **wfall** (*np.ndarray*) – burst waterfall
- **dt** (*float*) – time resolution
- **tstart** (*int*) – time channel to start at
- **tend** (*int*) – time channel to end at

**Returns**
the structure parameter

**Return type**
float

driftrate.**subband**(*wfall*, *nsub*)

Downsample frequency channels of a waterfall.

See *subsample()* for more general method.

**Parameters**

- **wfall** (*np.ndarray*) – 2d array
- **nsub** (*int*) – number of channels. nsub should evenly divide `wfall.shape[0]`

**Returns**
2d subbanded array.

**Return type**
np.ndarray

driftrate.**subsample**(*m*, *nfreq*, *ntime*)

Subsample a waterfall in time and frequency

**Parameters**

- **m** (*np.ndarray*) – 2d array to subsample.
- **nfreq** (*int*) – the number of frequency channels desired. Should evenly divide `m.shape[0]`
- **ntime** (*int*) – the number of time channels desired. Should evenly divide `m.shape[1]`

**Returns**
the subsampled array

**Return type**
np.ndarray

driftrate.**subtractbg**(*wfall*, *tleft: int = 0*, *tright: int = 1*)

Subtract a background sample from a waterfall

This will compute the mean along the time axis to produce an array of noise as a function of frequency and subtract that array from the entire waterfall.

Avoid sampling the burst if possible to improve accuracy of measurements.

**Parameters**

- **wfall** (*np.ndarray*) – 2d array of burst waterfall
- **tleft** (*int*) – time channel to start sample from
- **tright** (*int*) – time channel to end sample from

**Returns**
the waterfall subtracted by the background sample

**Return type**
np.ndarray

driftrate.**twoD_Gaussian**(*point*, *amplitude*, *xo*, *yo*, *sigma_x*, *sigma_y*, *theta*)

2D rotatable Gaussian Model function. Used as the model function in scipy.optimize.curve_fit

**Parameters**

- **point** (*tuple*) – (y, x) point to evaluate the model at
- **amplitude** (*float*) – amplitude of the 2D gaussian
- **xo** (*float*) – central x position of the gaussian
- **yo** (*float*) – central y position of the gaussian
- **sigma_x** (*float*) – standard deviation in x-direction
- **sigma_y** (*float*) – standard deviation in y-direction

- **theta** (*float*) – angle of gaussian

**Returns**
1d array of raveled 2d gaussian data

**Return type**
list

driftrate.**updatenpz**(*npz*, *field*, *val*)

Update a field in a npz file and resave it.

**Parameters**

- **npz** (*str*) – filename of .npz file
- **field** (*str*) – the field you would like to update
- **val** (*Any*) – the value to update the field to

**Returns**
None

## 4.3 driftlaw module

driftlaw.**atanmodel**(*B*, *x*)

driftlaw.**bakeMeasurements**(*sources*, *names*, *exclusions*, *targetDMs*, *logging=True*, *tagColors=['r', 'r', 'b', 'g', 'y', 'c', 'tomato', 'c']*, *showDMtraces=False*, *exclude_set=None*)

Process completed measurements performed over a range of DMs for the purposes of analysing them in the context of the triggered relativistic dynamical model (TRDM).

Performs fits of the sub-burst slope law (i.e. slope = A/duration, A is a constant) by grouping measurements by DM. Provides detailed logging information.

The logging information of this function should be carefully reviewed to ensure the science and questions you are investigating are not being affected. Reviewing the information in this function should provide insight into how your bursts as a cohort are behaving as the DM changes.

The outputs (the "baked measurements") of this function are used to produce figures of spectro-temporal properties of the measurements and explore their relationships.

Specifically:

1. Filter measurements based on uncertainties and/or unphysical values. Measurements with uncertainty greater than 40% of the measurement value are discarded. Measurements with uncertainties greater than 10^8 are discarded. Measurements where the sub-burst slope is positive are discarded due to assumed over-dedispersion. Measurements with no valid fit (negative gaussian amplitude) are excluded.
2. Color data points based on spreadsheets.
3. Finds range of fits by source.
4. Tags data that is at the target DM and splits up measurements by DM. This can be then used to plot burst measurements at a "representative" DM.
5. Compute ranges of measurements and use these as conservative uncertainty bars on plots of bursts at their representative DM.

Example usage: .. code-block:: python

bakeddata, fitdata, sources, extradata = driftlaw.bakeMeasurements(sources, names, exclusions, targetDMs, logging=False, tagColors=tagColors)

See here for a (nontrivial) usage example.

**Parameters**

- **sources** (*list[pd.DataFrame]*) – list of DataFrames of frbgui results spreadsheets. Spreadsheets should be complete, that is, they are the output of *computeModelDetails()*.
- **names** (*list[str]*) – list of names for each dataset listed in sources. used in figures. One name per data source.
- **exclusions** (*list[list[str]]*) – list of bursts to exclude by name. One list per data source.
- **targetDMs** (*list[float]*) – Representative DM to display each burst at. One per data source. After running this function once it is a good idea to run *getOptimalDMs()* and use that list as your representative DMs. *getOptimalDMs()* requires the baked data so cannot be run first.
- **logging** (*bool, optional*) – Set to False to disable logging. Do this after you've reviewed the logging output.
- **tagColors** (*list[str], optional*) – list of matplotlib colors for plotting purposes. One per data source
- **showDMtraces** (*bool, optional*) – if True will display plots of Slope vs. duration for each burst over the DM range. Useful for looking at the change in measurements as DM varies.

**Returns**

(bakeddata, fitdata, sources, extradata).

- `bakkeddata` is an dictionary containing "frames", "labels", and "colors". Used in *plotSlopeVsDuration()* to produce figures of measurements at a representative DM with uncertainties estimated from the DM range used.
- `fitdata` is a DataFrame of the fit results (i.e. fitting slope = A/duration) at each DM and is used by *getOptimalDMs()*. It contains the following columns and information:

```
'name': # source name
'DM': # DM fit was performed at
'param': # The fit parameter to the fit A/duration
'err': # the error on A ,
'red_chisq': # the reduced chisquared,
'numbursts': # the number of bursts used, after measurement exclusions
```

- `sources` is the updated list of dataframes after measurement exclusions.
- `extradata` Some extra information regarding the sub-burst slope vs frequency relationship.

**Return type**

tuple

driftlaw.**cleanAngle**(*row*)

driftlaw.**computeModelDetails**(*frame*, *channelSpaceDuration=False*)

Takes a dataframe and computes columns related to the triggered relativistic dynamical model from the 2d gaussian parameters found in the measurement stage.

Importantly, computes the sub-burst slope, duration, bandwidth, and uncertainties associated with these.

Can be run even if the columns this function computes already exist and can serve to recalculate or update computed values.

**Parameters**
**frame** (*pd.DataFrame*) – the results dataframe from FRBGui or *driftrate.processDMRange()*

**Returns**
the dataframe with new computed columns

**Return type**
pd.DataFrame

driftlaw.**fitodr**(*frame, beta0=[1000], errorfunc=<function log_error>, log=True*)

driftlaw.**fitreciprocal**(*x*, *data*, *sigma=1*)

driftlaw.**fitreciprocal_log**(*x*, *data*, *sigma=1*, *loglog=False*)

driftlaw.**getOptimalDMs**(*fitdata*, *log=False*)

Using the fitdata found in *bakeMeasurements()*, identify which DM results in the lowest reduced chi-squared when fitting the sub-burst slope law while using all the bursts.

Thus this method hypothesizes that the sub-burst slope law must hold and uses that to define an "optimal" DM.

Because some measurements are excluded, some DMs (usually on the higher end of the range) may lead to entirely excluding a burst from analysis. Here we choose to interpret this as the DM being too high and unphysical.

That is, in addition to having the best fit to the sub-burst slope law, the "optimal" DM is also defined to be the DM where all bursts in our sample have physically valid (under our assumptions) measurements.

**Parameters**

- **fitdata** (*pd.Dataframe*) – Dataframe output from *bakeMeasurements()*
- **log** (*bool, optional*) – if True, ouptut tables of the fitdata with their reduced chi-squareds and number of remaining bursts by DM. Can be used to manually pick an "optimal" DM.

**Returns**
list of optimal DMs by source. Use this as the input to py:meth:*bakeMeasurements* to produce publication figures of your measurements at a representative DM that is the optimal DM as defined here.

**Return type**
list

driftlaw.**limitedDMrangeerror**(*frame*)

driftlaw.**limitedDMrangeerror_odr**(*frame*)

The range errors are asymmetric. Take the largest error

driftlaw.**limitedDMrangeerror_recip**(*frame*)

driftlaw.**limitedDMsloperanges**(*fitdf*, *source*, *threshold=0*)

Like *sloperanges* but only for DMs where all the bursts in a sample have valid measurements

driftlaw.**log_error**(*frame*)

see modelerror()

driftlaw.**log_log**(*x*, *k*, *b*)

driftlaw.**modelerror**(*frame*)

driftlaw.**modelerror_recip**(*frame*)

driftlaw.**offset_atanmodel**(*B*, *x*, *zero_ddm_fit=6.554*)

driftlaw.**plotSlopeVsDuration**(*frames=[]*, *labels=[]*, *title=None*, *logscale=True*, *annotatei=0*, *markers=['o', '^', 'v', 'd', 'p']*, *hidefit=[]*, *hidefitlabel=False*, *fitlines=['r-', 'b--', 'g-.']*, *fitextents=None*, *figsize=(17, 9)*, *errorfunc=<function modelerror>*, *fiterrorfunc=<function rangelog_error>*, *dmtrace=False*, *ax=None*)

Plot the normalized sub-burst slope vs. sub-burst duration for a list of baked measurements.

Finds a fit of the form

$$\frac{1}{\nu}\left|\frac{d\nu}{dt}\right| = \frac{A}{t}$$

Bake your spectro-temporal measurements with *bakeMeasurements()* first and use the outputs as the `frames` and `labels` parameters needed here.

Allows you to define how the uncertainties are calculated.

The following is an example of how to call this function using `bakeddata`. In this example the uncertainties are estimated from the range of measurements obtained over the range of DMs that include all bursts after exclusions are performed.

```
ax, _ = driftlaw.plotSlopeVsDuration(bakeddata['frames'], bakeddata['labels'],␣
↪title="My Favorite Repeating FRB", logscale=True, errorfunc=driftlaw.
↪limitedDMrangeerror, fiterrorfunc=driftlaw.log_error)
```

See here for a (nontrivial) usage example.

**Parameters**

- **frames** (*list[pd.DataFrame]*) – list of dataframes of measurements. Produce this with *bakeMeasurements()*
- **labels** (*list[str]*) – Labels to use on plot. One per source dataframe.
- **title** (*str, optional*) – title of plot
- **logscale** (*bool, optional*) – if True use a logscale
- **annotatei** (*int or list[int], optional*) – index or list of indices of `frames`. Indices which frames' bursts you would like to see annotated. Useful for identifying outliers for further review.
- **markers** (*str or list[str], optional*) – matplotlib plot markers
- **hidefit** (*int or list[int], optional*) – index or indices of fits to hide. Starts from 0.
- **hidefitlabel** (*bool, optional*) – If True will not display a label in the legend of fits
- **fitlines** (*str or list[str], optional*) – matplotlib linestyles used for fits
- **fitextents** (*tuple, optional*) – (min duration, max duration) extents over which to display the fits
- **figsize** (*tuple, optional*) – matplotlib figure size (width, height)
- **errorfunc** (*function, optional*) – the function you want to evaluate your measurement uncertainties. Used as part of pd.scatter. If unsure, use *driftlaw.limitedDMrangeerror()* or the default value.

- **fiterrorfunc** (*function, optional*) – function for evaluating fit uncertainties. If unsure use *driftlaw.log_error()* or the default value.
- **dmtrace** (*bool, optional*) – if True create an additional plot of norm. slope vs. duration for each `frame` showing the trace of the point over the range of DMs used to measure.
- **ax** (*matplotlib.axes.Axes, optional*) – the axis to plot on.

**Returns**
(ax, fits). `ax` is the figure axes and can be used to make additional modifications to the figure. `fits` is a list of [label, param, err, red_chisq, residuals, len(frame)].

**Return type**
tuple

driftlaw.**rangeerror**(*frame*)

These ranges are not errors in the statistical sense. they are the min/max values, which should be larger than the real errors. So this is extremely conservative while also being easier to compute.

The strange shape of the returned value is due to a quirk in the way pandas handles asymmetric errors.

driftlaw.**rangeerror_odr**(*frame*)

The range errors are asymmetric. Take the largest error

driftlaw.**rangelog_error**(*frame*)

The range errors are asymmetric. Average the error

driftlaw.**reciprocal**(*x*, *a*)

driftlaw.**reciprocal_log**(*x*, *b*)

driftlaw.**reciprocal_odr**(*B*, *x*)

driftlaw.**reciprocal_odr_log**(*B*, *x*)

driftlaw.**sloperanges**(*source*)

Given all burst and model data at different trial DMs, computes the range of slopes durations across the range of trial DMs

Used as part of estimating uncertainties in *bakeMeasurements()*.

CHAPTER

# FIVE

# OUTPUT CSVS

FRBGui outputs measurements as a comma-separated-value (CSV) spreadsheet along with a corresponding PDF that contains plots of each burst waterfall with its measurements overlaid.

The output spreadsheet is arranged as one measurement per row under the following columns:

**Note:**

- Plaintext columns like "slope (mhz/ms)" indicate a measurement.
- *Italicized* columns indicate information about how the measurement was taken, such as "time_res (s)".
- ~~Stricken~~ column names indicate a deprecated column and normally should not be used.

| Column Name | Description |
|---|---|
| name | The name of the burst file measured, potentially suffixed with a letter (e.g., "a", "b", "c") denoting the sub- |
| DM | The particular trial DM used to perform measurements |
| center_f | The center frequency of the burst as measured by frbgui |
| center_f_err | The propagated uncertainty on center_f. For sub-pulses, this value will be the same across DMs, as the wat |
| slope (mhz/ms) | The sub-burst slope, obtained from the cotangent of the fit angle |
| slope error (mhz/ms) | The propagated uncertainty on the slope |
| theta | The angle of the semi major axis of the fit ellipse measured counter-clockwise from the positive y-axis |
| red_chisq | Reduced chi-squared indicating the goodness-of-fit of the 2D Gaussian to the 2D ACF |
| amplitude | Amplitude of the 2D Gaussian fit. Not normalized. |
| xo | Central x-position (time) of the 2D gaussian fit to the 2D-ACF |
| yo | Central y-position (frequency) of the 2D gaussian fit to the 2D-ACF |
| sigmax | Standard deviation in x (time if vertical) of the 2D gaussian fit to the 2D-ACF |
| sigmay | Standard deviation in y (frequency if vertical) of the 2D gaussian fit to the 2D-ACF |
| angle | The fit angle. Equivalent to theta up to a difference of pi/2 due to the ambiguity between axes when fitting |
| amp_error | The fit uncertainty on the amplitude |
| xo_error | The fit uncertainty on xo |
| yo_error | The fit uncertainty on yo |
| sigmax_error | The fit uncertainty on sigmax |
| sigmay_error | The fit uncertainty on sigmay |
| angle_error | The fit uncertainty on angle |
| slope_abs | The absolute value of slope (mhz/ms). If negative, this indicates a potential measurement issue |
| slope_over_nuobs | 'slope_abs' / 'center_f'. In the TRDM, this is proportional to 1/'tau_w_ms' |
| slope_over_nuobs_err | The propagated 2D gaussian fit uncertainty for 'slope_over_nuobs'. |
| recip_slope_over_nuobs | The reciprocal of slope_over_nuobs |
| slope_abs_nuobssq | 'slope_abs' / 'center_f**2'. In the TRDM, this is proportional to a constant. |
| min_sigma | Same as either sigmax or sigmay, whichever is smaller |

Table 1 – continued from previous pa

| Column Name | Description |
|---|---|
| max_sigma | Same as either sigmax or sigmay, whichever is larger |
| min_sigma_error | Same as either sigmax_error or sigmay_error, associated with whichever sigma is smaller |
| max_sigma_error | Same as either sigmax_error or sigmay_error, associated with whichever sigma is larger |
| tau_w | The sub-burst duration in milliseconds as defined in Chamma et al. (2022). |
| tau_w_error | The propagated uncertainty on tau_w (ms) |
| tau_w_ms | The sub-burst duration in milliseconds as defined in Chamma et al. (2022). An alias of tau_w. Units are im |
| bandwidth (mhz) | Best-fit bandwidth for the sub-pulse in MHz |
| bandwidth error (mhz) | Propagated uncertainty for the bandwidth in MHz |
| *f_res (mhz)* | The frequency resolution (MHz) of the final waterfall used for the FRBGUI measurements |
| *time_res (s)* | The time resolution (seconds) of the final data array used for the FRBGUI measurements |
| *downf* | The factor by which the file was downsampled in frequency from FRBGUI from the original waterfall |
| *downt* | The factor by which the file was downsampled in time from FRBGUI from the original waterfall |
| *fchans* | The number of frequency channels remaining after the downsample in FRBGUI |
| *tchans* | The number of time channels remaining after the downsample in FRBGUI |
| *tsamp_width* | The number of time channels set by the user in FRBGUI used for measurement (i.e., the width of the water |
| *subbg_start (ms)* | The time, in ms, from the start of the file that the user-defined background sample begins |
| *subbg_end (ms)* | The time, in ms, from the start of the file that the user-defined background sample ends |
| *sksigma* | The sigma chosen for the SK-SG filter, which performs RFI removal |
| *skwindow* | The window chosen for the SK-SG filter, which performs RFI removal |
| *regstart_a* | When using regions, the start of the "a" subpulse region in ms |
| *regend_a* | When using regions, the end of the "a" subpulse region in ms |
| *regstart_b* | When using regions, the start of the "b" subpulse region in ms |
| *regend_b* | When using regions, the end of the "b" subpulse region in ms |
| *background* | The size of the background subpulse region in ms. Used to pad sub-pulses with zeroes, which improves me |
| ~~sigma_t~~ | The product of min_sigma and time_res (s). This is a (poor) measure of burst duration. Use tau_w or tau_v |
| ~~tau_w_old~~ | The sub-burst duration as defined and used in Chamma et al. (2021). Due to the choice of coordinates whe |
| ~~t_from_sigma~~ | The product of min_sigma and sin(theta). This is a (poor) measure of burst duration when finding fits with |
| ~~sigma_t_ms~~ | 'sigma_t' in ms. See sigma_t |
| ~~tau_w_ms_old~~ | tau_w_old in milliseconds |

Special thanks to Dr. Sofia Sheikh for contributions to this table.

CHAPTER

SIX

# 2D AUTOCORRELATION METHOD

This page will detail how spectro-temporal measurements are obtained in FRBGui using the 2D autocorrelation method. This material is adapted from Chamma et al. (2021, 2023).

This documentation is still being written. For more information consult the repository or feel free to email me. The information that will be presented here is also available in the publications mentioned above.

# PYTHON MODULE INDEX

# INDEX

## T



## U