PySP2

Contents

PySP2 is a Python package for processing data from Droplet Measurement Technologies' Single Particle Soot Photometer (SP2).

The goals (and advantages) of PySP2 are:

```
* Bring DMT's processing code, originally written in IGOR, to open source in Py

* Be interactable with Dask for parallelizing SP2 processing on clusters such a
```

PySP2 is capable of:

```
* Processing raw SP2 data from DMT in .sp2b format

* Generating wave properties from each channel

* Genetering particle mass and size distributions from the particle properties
```

```
import act
import pysp2
import matplotlib.pyplot as plt
%pylab inline
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries
Populating the interactive namespace from numpy and matplotlib
```

Skip to main content

PySP2 has two procedures for reading in the probe configuration in a .ini file as well as reading an .sp2b file. Here, we load a sample time period from the SP2.

```
config = pysp2.io.read_config('20181110/20181110114046.ini')
in_sp2b = pysp2.io.read_sp2('20181110/20181110x001.sp2b', arm_convention=False
```

```
list(config.keys())
```

```
['DEFAULT',
 'Versions',
 'Program',
 'Controllers',
 'Alarms',
 'Alicat Flow Controller',
 'Control',
 'SPAT',
 'Acquisition',
 'Digital',
 'Laser',
 'Analog Input',
 'Housekeeping',
 'Streaming Data',
 'Flow Meters',
 'Calculated Channels',
 'Calculations',
 'Missing Value']
```

The SP2 has 8 different photodiode detector arrays that measure the scattering and incandescence signals from the laser shining onto a partilcle that enters the beam. A refractory black carbon particle would, first, scatter the laser beam back to the detector. Applying Mie theory to the scattering signal gives us the approximate diameter of the particle entering the beam. Channels 0 (high gain) and 1 (low gain) measure the strength of this signal along a photodiode array that is aligned along the direction of the sample flow. In essence, the position along the photodiode array is proportional to the time at which the particle scatters light after it enters the laser beam. The low gain channel signal is useful for when the high gain channel is saturated from large particles entering the SP2 sample volume.

However, after the non-black carbon material coating the particle is evaporated by the heat from the laser, the black carbon proceeds to incandesce. The SP2 then records the signal into ch4 (high gain channel) and ch5 (low gain channel). This is done for one of every *x*

Skip to main content

time, this is set to 5, since recording every particle would take up too much storage and processing time.

These raw waveforms for each particle are stored into an xarray dataset, shown below.

```
in_sp2b
```

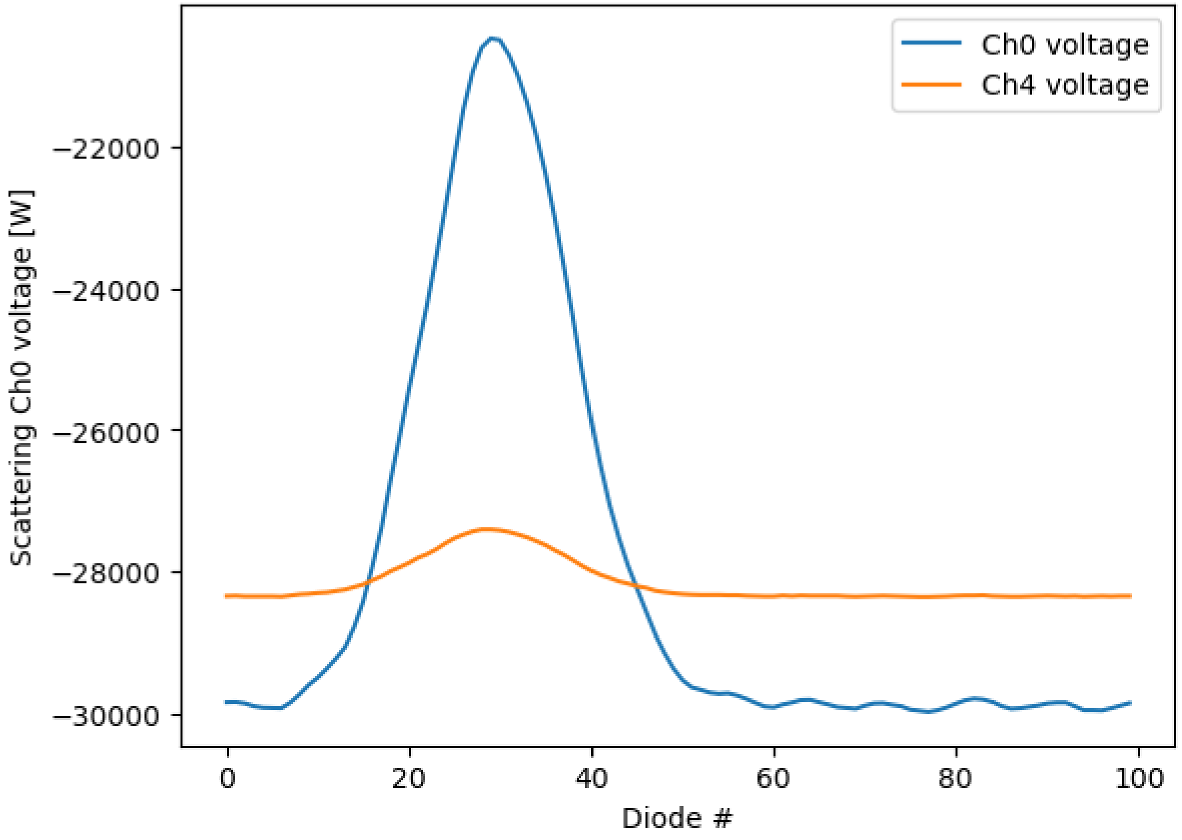
```
xarray.Dataset
▸ Dimensions:      (event_index: 50014, columns: 100)
▸ Coordinates:  (0)
▸ Data variables:
 (19)
▸ Indexes:  (0)
▸ Attributes:  (0)
```

Let's take the first particle in this file and plot the raw signals recorded by the SP2. First, we will look at the high gain scattering (ch0) and low gain scattering (ch4) signals.

```
in_sp2b["Data_ch0"].sel(event_index=10).plot(label='Ch0 voltage')
in_sp2b["Data_ch4"].sel(event_index=10).plot(label='Ch4 voltage')
plt.ylabel('Scattering Ch0 voltage [W]')
plt.xlabel('Diode #')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x15fe23710>
```

Skip to main content



As we can see here, the peak of the scattering signal (ch0) is at around diode 32 with a Gaussian shape. In the low gain scattering channel (ch4), we see a similar peak with a much smaller magnitude since these diodes are less sensitive to the amount of scattered light from the laser beam.

Obtaining waveform statistics

The next step is to obtain the characteristics of these waveforms such as the peak amplitude, noise floor, and half width for each channel. In addition, Gaussian fits are generated for ch0 and ch4 since these waveforms are Gaussian if there is a single particle in the sample volume. These are the characteristics that PySP2 uses to derive particle mass and size distributions.

```
particle_statistics = pysp2.util.gaussian_fit(in_sp2b, config)
```

```
/Users/rjackson/PySP2/pysp2/util/peak_fit.py:452: RuntimeWarning: divide by ze
  peak2area = np.max(data2, axis=1)/denominator
```

Skip to main content

```
Processing record 0
```

```
/Users/rjackson/mambaforge/envs/act_env/lib/python3.11/site-packages/scipy/opt
  warnings.warn('Covariance of the parameters could not be estimated',
```

```
Processing record 1000
Processing record 2000
Processing record 3000
Processing record 4000
Processing record 5000
Processing record 6000
Processing record 7000
Processing record 8000
Processing record 9000
Processing record 10000
Processing record 11000
Processing record 12000
Processing record 13000
Processing record 14000
Processing record 15000
Processing record 16000
Processing record 17000
Processing record 18000
Processing record 19000
Processing record 20000
Processing record 21000
Processing record 22000
Processing record 23000
Processing record 24000
Processing record 25000
Processing record 26000
Processing record 27000
Processing record 28000
Processing record 29000
Processing record 30000
Processing record 31000
Processing record 32000
Processing record 33000
Processing record 34000
Processing record 35000
Processing record 36000
Processing record 37000
Processing record 38000
Processing record 39000
Processing record 40000
Processing record 41000
Processing record 42000
Processing record 43000
Processing record 44000
```

Skip to main content

```
Processing record 46000
Processing record 47000
Processing record 48000
Processing record 49000
Processing record 50000
```

```
/Users/rjackson/PySP2/pysp2/util/peak_fit.py:76: RuntimeWarning: Mean of empty
  ratio = np.nanmean(
```

```
50014 records processed in 52.92566895484924 s
```

These particle statistics are then stored in an xarray dataset for each particle.

```
particle_statistics
```

```
xarray.Dataset
▸ Dimensions:          (event_index: 1106, columns: 100)
▸ Coordinates:   (0)
▸ Data variables:
 (81)
▸ Indexes:   (0)
▸ Attributes:   (0)
```

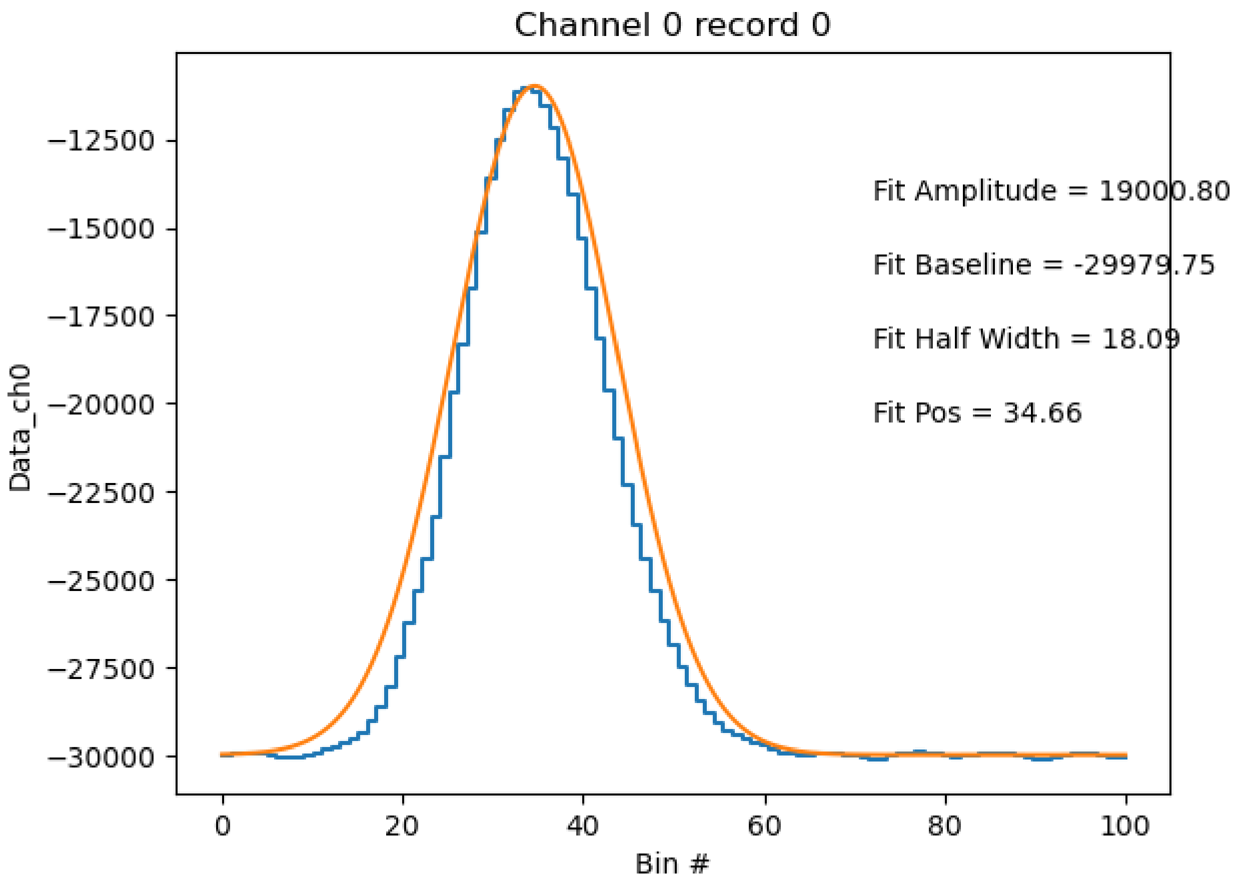
PySP2 has a handy function to check your particle fits called *pysp2.vis.plot_wave*. We will plot the fit characteristics for the particle we plotted above. As we can see, the particle wave-form is narrower than a Gaussian fit. However, the peak amplitude is represented well by the fit. Let's replot the ch0 and ch4 waves using this capability.

```
pysp2.vis.plot_wave(particle_statitics, 0, 0)
```

```
/Users/rjackson/ACT/act/plotting/plot.py:81: UserWarning: Could not discern da
  warnings.warn(
```

Skip to main content

```
<act.plotting.histogramdisplay.HistogramDisplay at 0x15fe39f10>
```

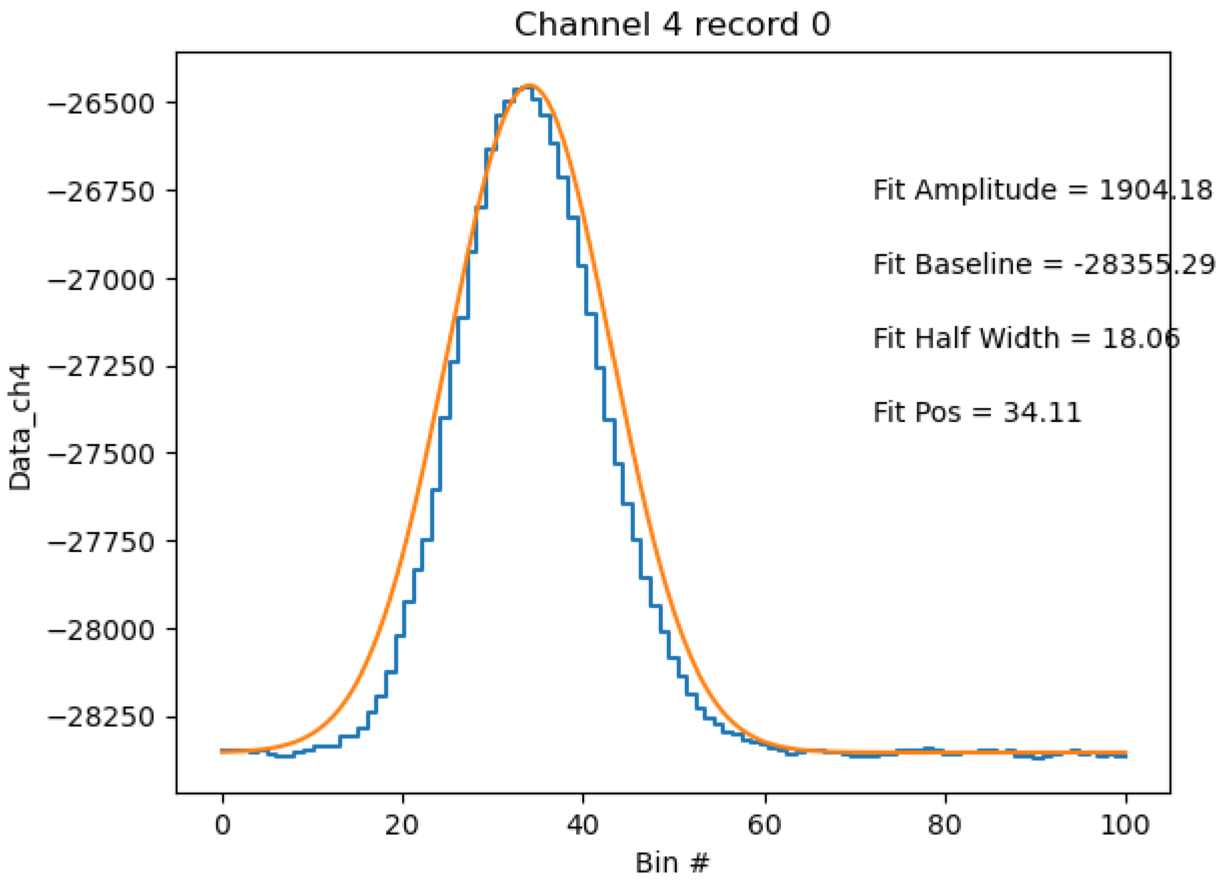


```
pysp2.vis.plot_wave(particle_statitics, 0, 4)
```

```
/Users/rjackson/ACT/act/plotting/plot.py:81: UserWarning: Could not discern da
  warnings.warn(
```

```
<act.plotting.histogramdisplay.HistogramDisplay at 0x16609bf50>
```

Skip to main content



Finally, we can save these particle statistics to either a netCDF file or a .dat file that is readable by DMT's IGOR-based processing software. PySP2 also supports loading .dat files made by IGOR's software, enabling users to seamlessly read already-processed data from the IGOR-based code.

```
# Save to the IGOR .dat format
pysp2.io.write_dat(particle_statistics, '20181110x001.dat')
```

We use the pysp2.io.read_dat function to read in IGOR .dat files. Wildcards are supported, enabling concatenation of particle datasets for further processing. Let's load in some previously processed data!

```
part_stats = pysp2.io.read_dat('20181110/20181110x*.dat', type='particle')
part_stats
```

```
xarray.Dataset
▸ Dimensions:        (index: 750446)
```

Skip to main content

```
index            (index)  int64  0 1 2 3 ... 50031 50032 50033 50034
▸ Data variables:
 (69)
▸ Indexes:  (1)
▾ Attributes:
  _datastream :       20181110x002.dat
  _site :             201
  _arm_standar...     0
```

Reading and plotting housekeeping data

PySP2 also supports reading the housekeeping variables saved by the SP2 for diagnostics. If these files are saved in a .hk file that is output by the SP2, PySP2 will read these characteristics into an xarray dataset that can be viewed.

```
hk_ds = pysp2.io.read_hk_file('20181110/20181110114047.hk')
hk_ds
```

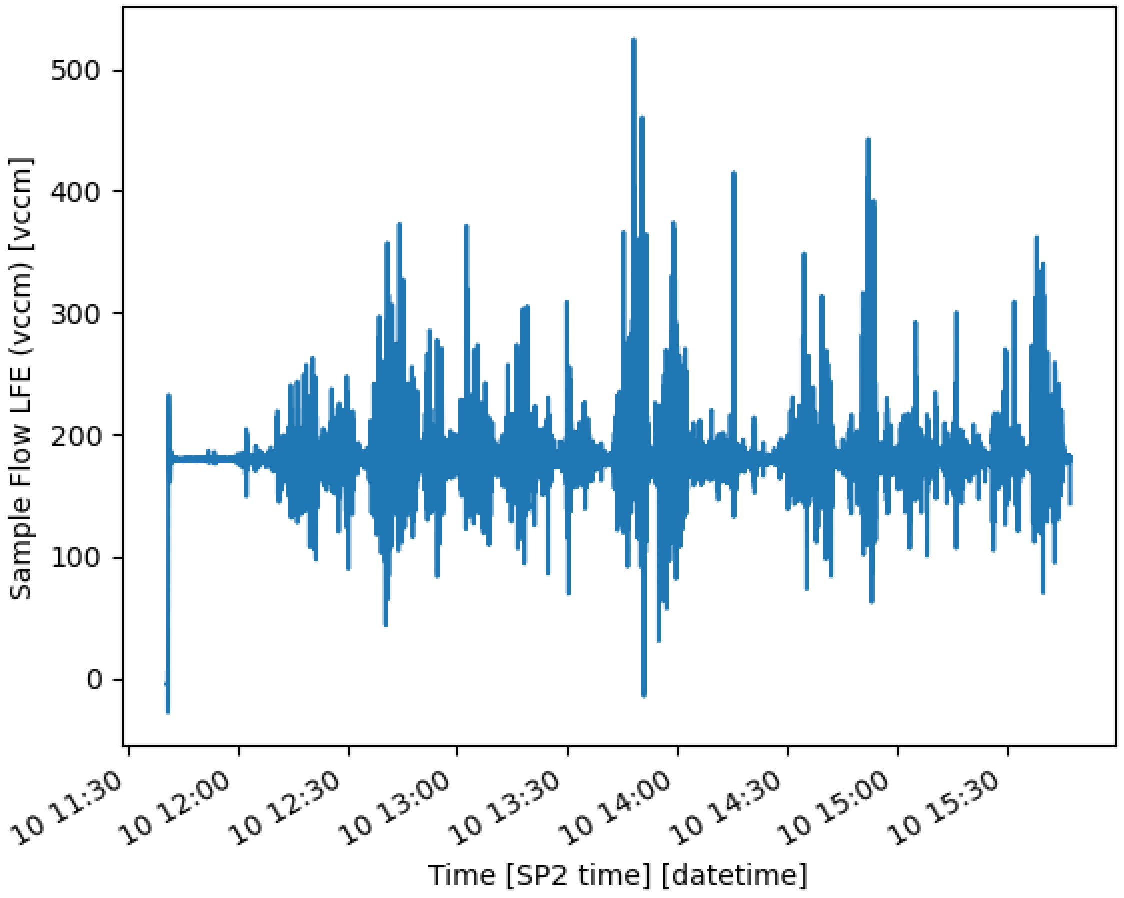
```
xarray.Dataset
▸ Dimensions:     (time: 14799)
▾ Coordinates:
  time            (time)  datetime64[ns]  2018-11-10T11:40:48.300000 ... 2...
▸ Data variables:
 (46)
▸ Indexes:  (1)
▾ Attributes:
  _datastream :       20181110114047.hk
  _site :             201
  _arm_standar...     0
```

Let’s use xarray’s plotting tools to view the Sample Flow Rate into the SP2 for the day.

Skip to main content

```
hk_ds["Sample Flow LFE"].plot()
```

```
[<matplotlib.lines.Line2D at 0x166287c50>]
```



Obtaining particle size distributions

In this section, we'll go over how to calculate and plot mass and number concentrations and size distributions. Often, the SP2 will need to be calibrated in order to ensure that mass and number size distributions are properly calculated. Thankfully, PySP2 has a DMTGlobals class that enables seamless loading of calibration files.

```
DGlbals = pysp2.util.DMTGlobals('20181110/Unit24CAL_Aldine_final.txt')
```

Let's process the particle sizes and masses using the calibration we loaded.

```
psd = pysp2.util.calc_diams_masses(part_stats)
```

Skip to main content

```
/Users/rjackson/PySP2/pysp2/util/particle_properties.py:38: RuntimeWarning: Al
  PkHt_ch0 = np.nanmax(np.stack([input_ds['PkHt_ch0'].values, input_ds['FtAmp_
/Users/rjackson/PySP2/pysp2/util/particle_properties.py:39: RuntimeWarning: Al
  PkHt_ch4 = np.nanmax(np.stack([input_ds['PkHt_ch4'].values, input_ds['FtAmp_
```

```
Number of scattering particles accepted = 638017
Number of scattering particles rejected for min. peak height = 26642
Number of scattering particles rejected for peak width = 12482
Number of scattering particles rejected for fat peak = 0
Number of scattering particles rejected for peak pos. = 30819
```

```
psds = pysp2.util.process_psds(psd, hk_ds, config, num_bins=199)
psds
```

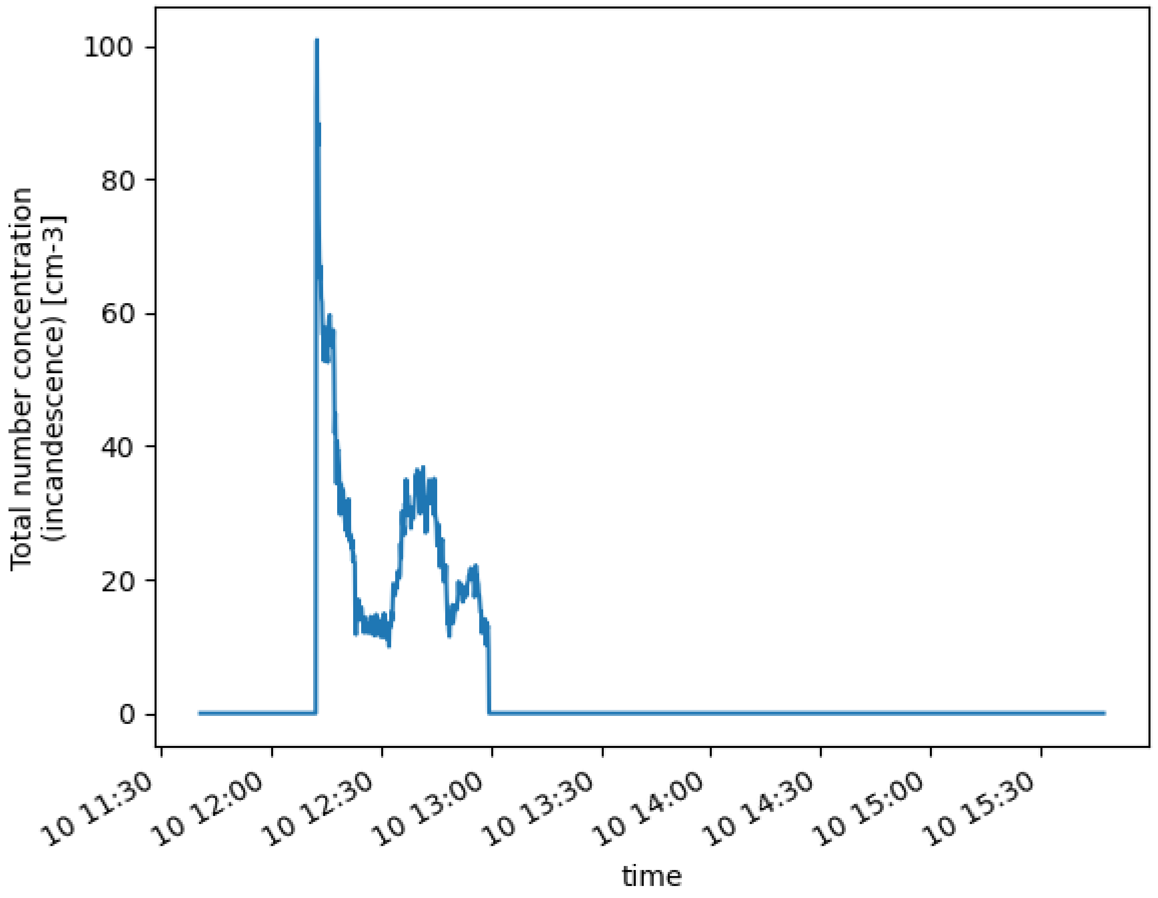
```
xarray.Dataset
▸ Dimensions:      (time: 1479, num_bins: 199)
▾ Coordinates:
   time          (time)       datetime64[ns]  2018-11-10T11:40:48.300000 ... 2...
   num_bins      (num_bins)          float64  0.01 0.015 0.02 ... 0.99 0.995 1.0
▸ Data variables:
 (17)
▸ Indexes:   (2)
▸ Attributes:   (0)
```

Let’s view the number concentrations. Since we have only processed about five minutes of data for this example, we will zoom into the processed data.

```
psds.NumConcIncan.plot()
```

```
[<matplotlib.lines.Line2D at 0x15b356e50>]
```

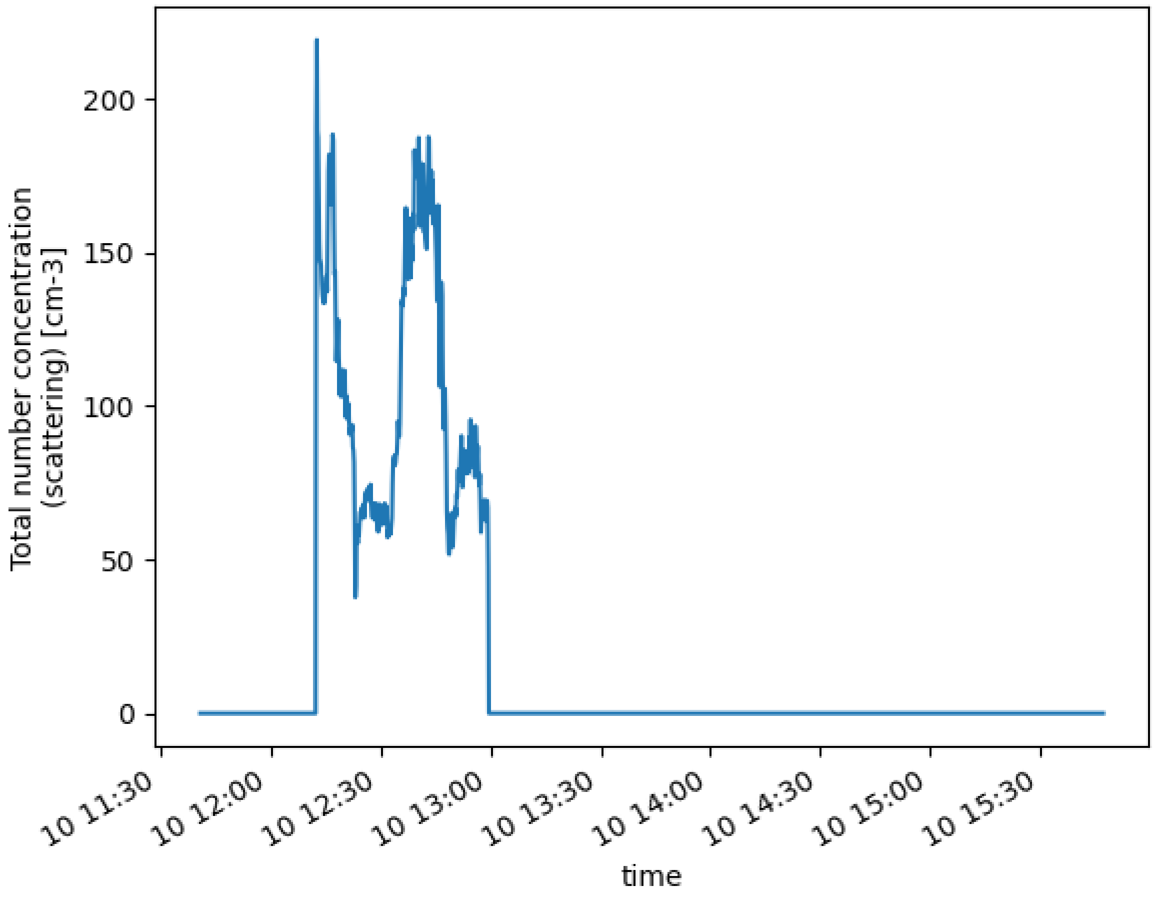
Skip to main content



```
psds.NumConcScat.plot()
```

```
[<matplotlib.lines.Line2D at 0x165e82a50>]
```

Skip to main content

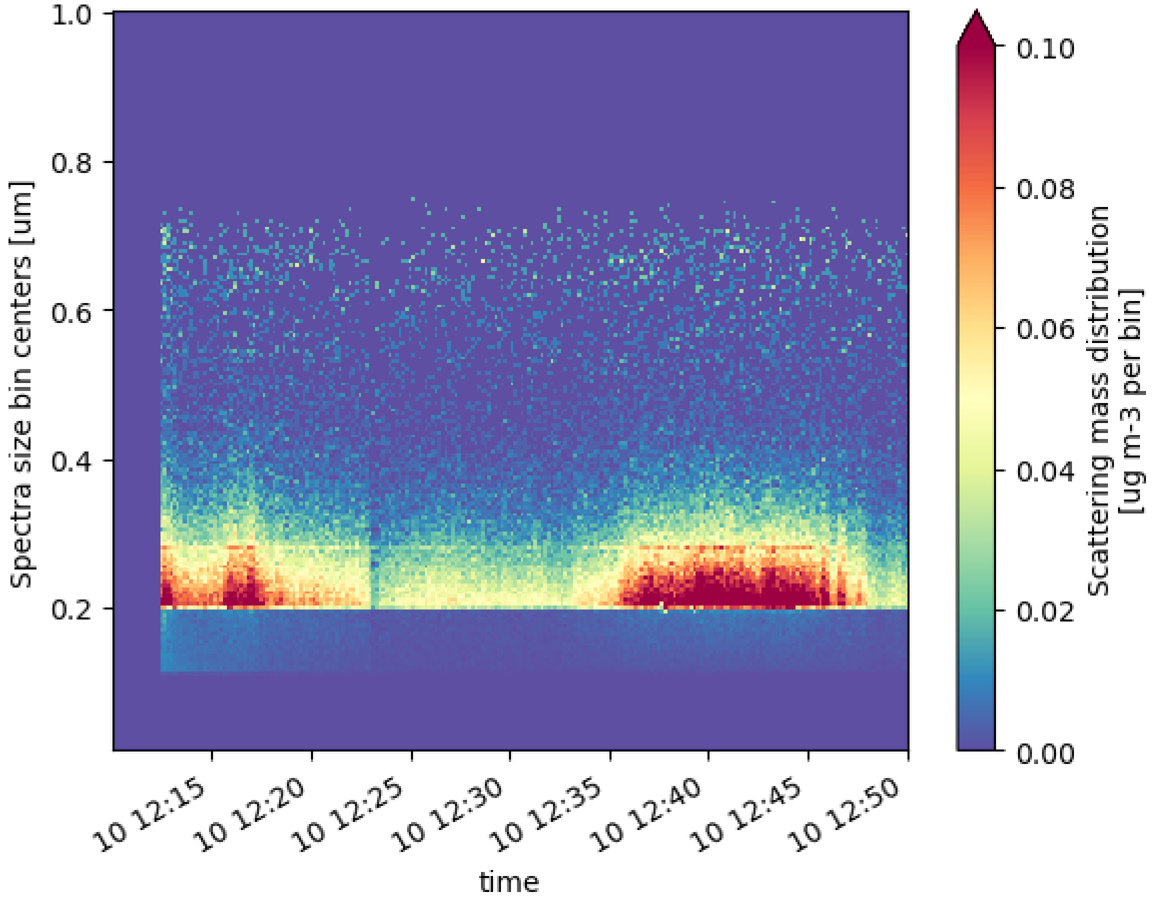


Let’s take a look at the particle size distributions!

```
psds.ScatMassEnsemble.T.sel(time=slice('2018-11-10T12:10:00', '2018-11-10T12:5
```

```
<matplotlib.collections.QuadMesh at 0x166b06190>
```

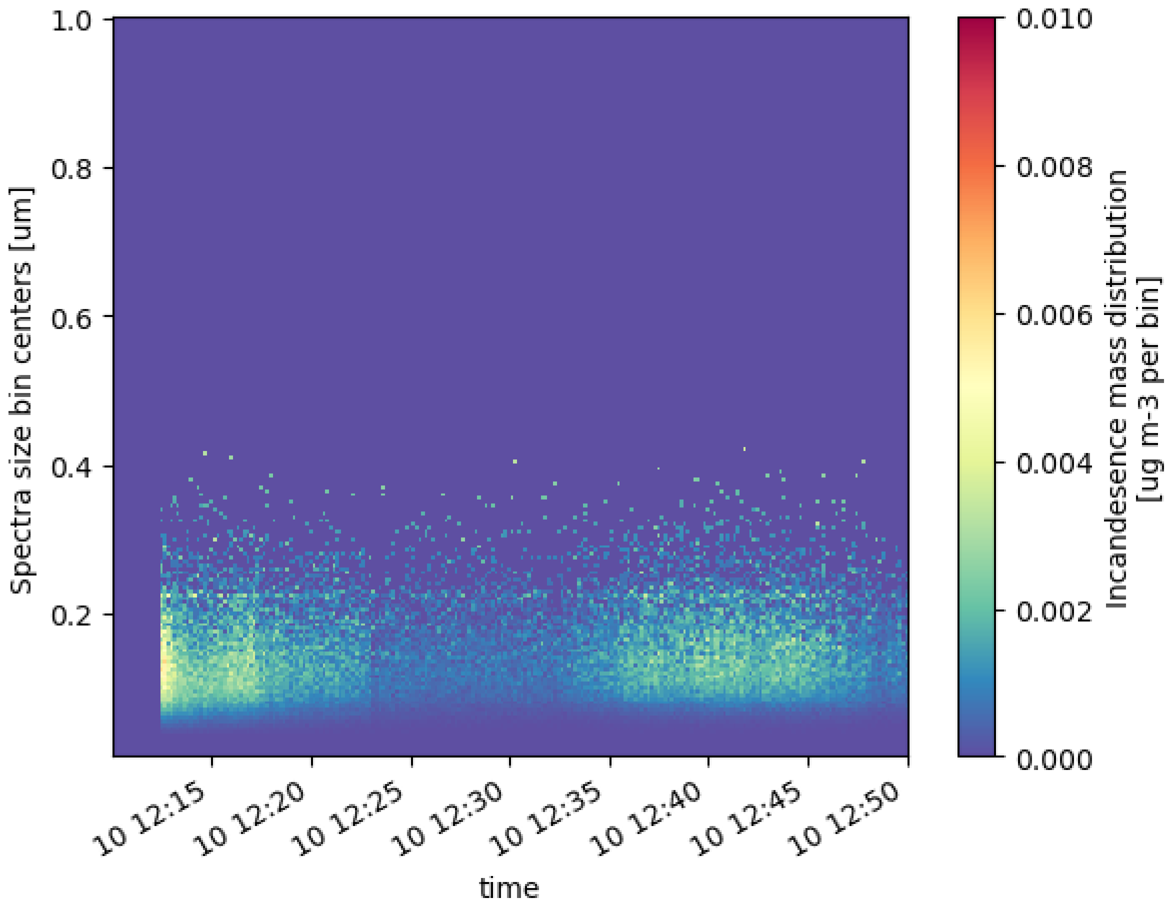
Skip to main content



```
psds.IncanMassEnsemble.T.sel(time=slice('2018-11-10T12:10:00', '2018-11-10T12:
```

```
<matplotlib.collections.QuadMesh at 0x1665c6610>
```

Skip to main content


1.0
0.8
0.6
0.4
0.2
Spectra size bin centers [um]
10 12:15
10 12:20
10 12:25
10 12:30
10 12:35
10 12:40
10 12:45
10 12:50
time
0.010
0.008
0.006
0.004
0.002
0.000
Incandesence mass distribution
[ug m-3 per bin]