Protection Engineering CONSULTANTS

PEC Spectra Software

A New Library & Application for PDV Signal Analysis

2024-10-16

Alex Lakocy (Presenter)
Duncan McGeehan

Matt Barsotti (PI)
Eddie O'Hare, Ph.D.

Approved for PDV Workshop 2024

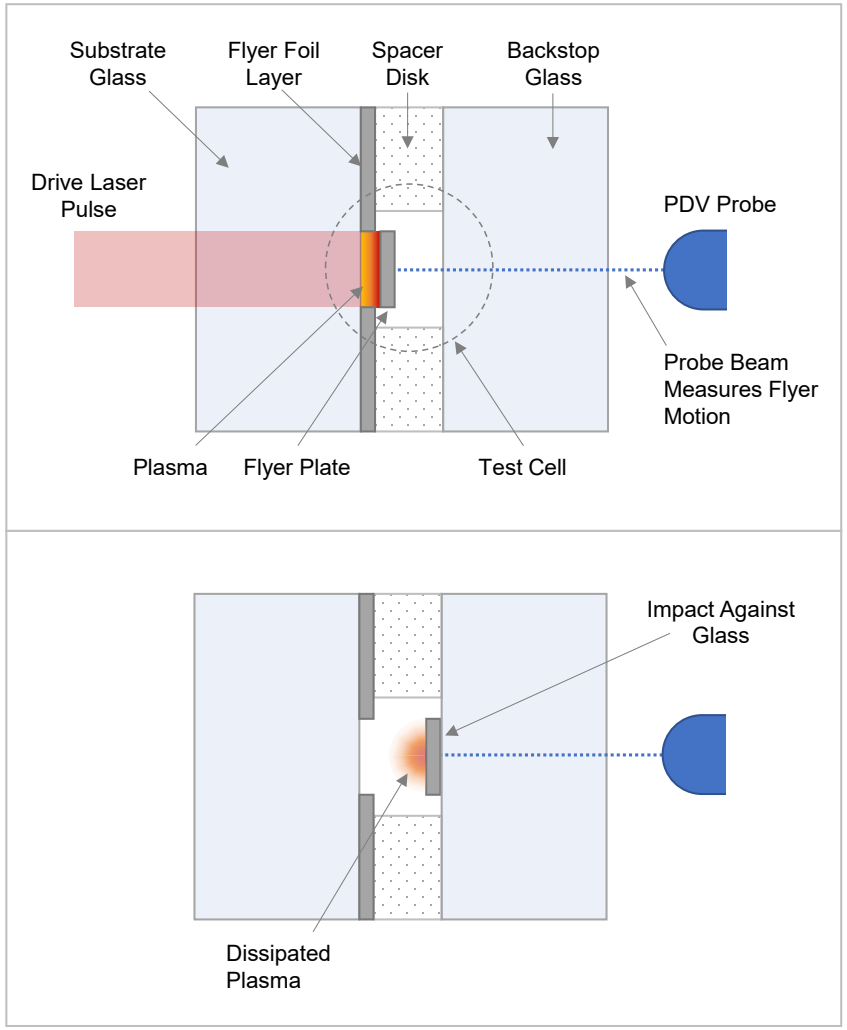
Organization

- Background & Motivation
- PEC Spectra Components
- Library Description
- Application Description
- Application Examples

BACKGROUND & MOTIVATION

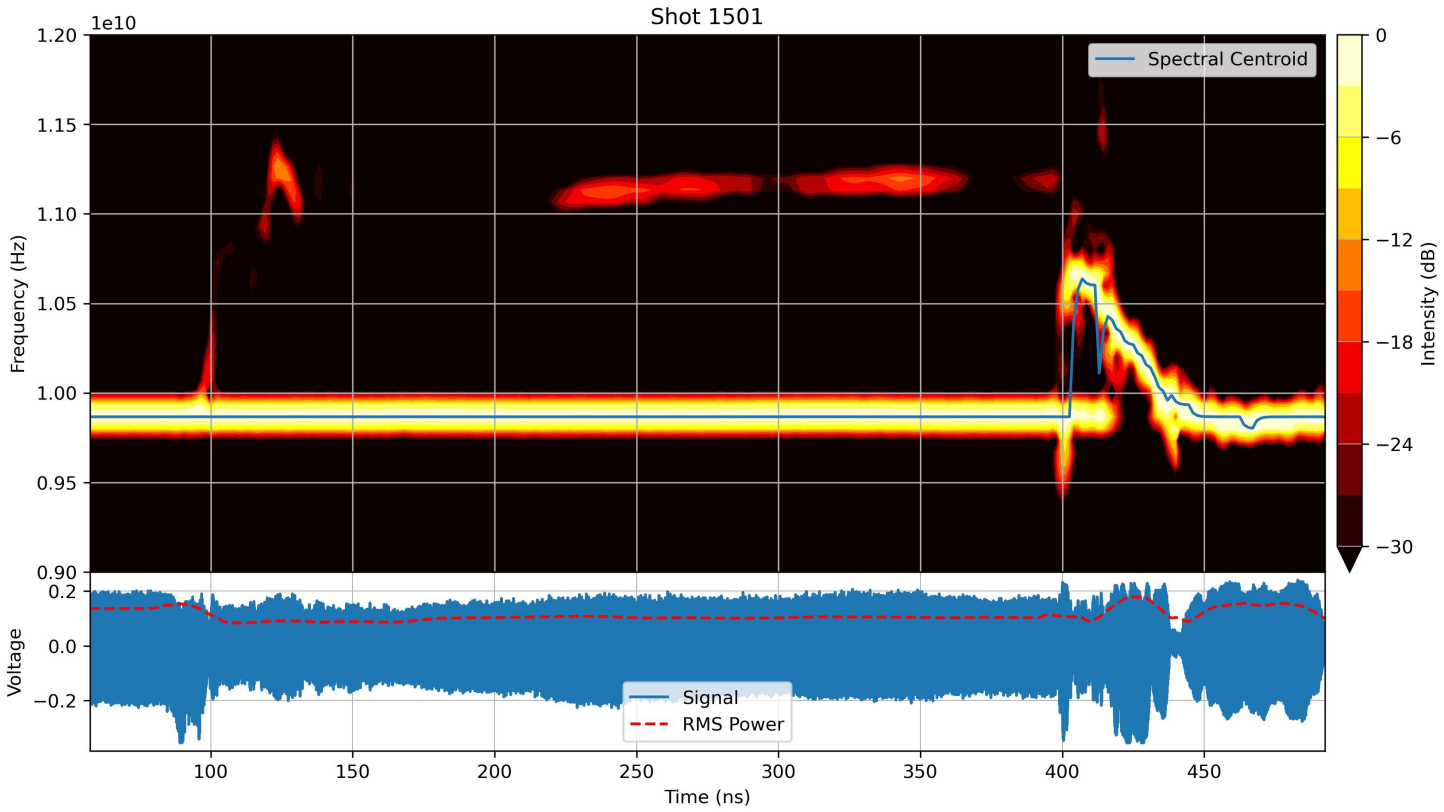
Background

- PEC conducted a multi-year research effort with SwRI involving laser driven flyer testing.
- Metallic foil flyers were launched into glass and metallic targets.
- Material response was measured using PDV probes.



Motivation

- Low-quality signals
  - High baseline leakage
  - Dynamic reflections
  - Nano-scale resolution
- High volume testing
  - Automation
    - Monitor Mode
    - Batch Mode
- User-Friendly GUI



PEC Spectra Software

- Free and available for all government users
- Working through approvals for academic & industry users
  - Still contact us if interested
- Send requests to: alakocy@protection-consultants.com

Spectra Library & Application

Library

- For Python users
- Modular toolbox for PDV signal analysis
- Simple, straightforward, extensible

Application

- For all users
- Windows application
- Built on Library backend

LIBRARY

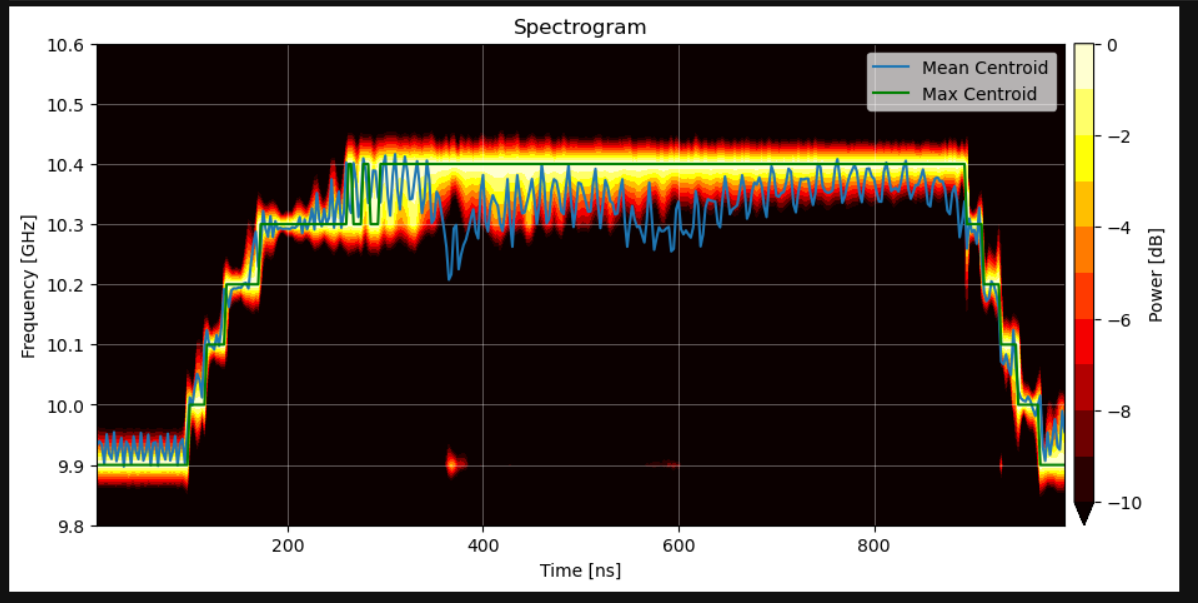
Spectra Library

- ***Library***
  - Python package (wheel file)
  - Modular, Extensible
  - Matlab-style figures (matplotlib)
  - Requires Python environment & familiarity with coding

```
[47]: stft_transform = StftTransform(twindow=10 * unit.nanosecond, overlap_ratio=0.75)
      stft_transformed_signal = stft_transform.apply(filtered_signal)

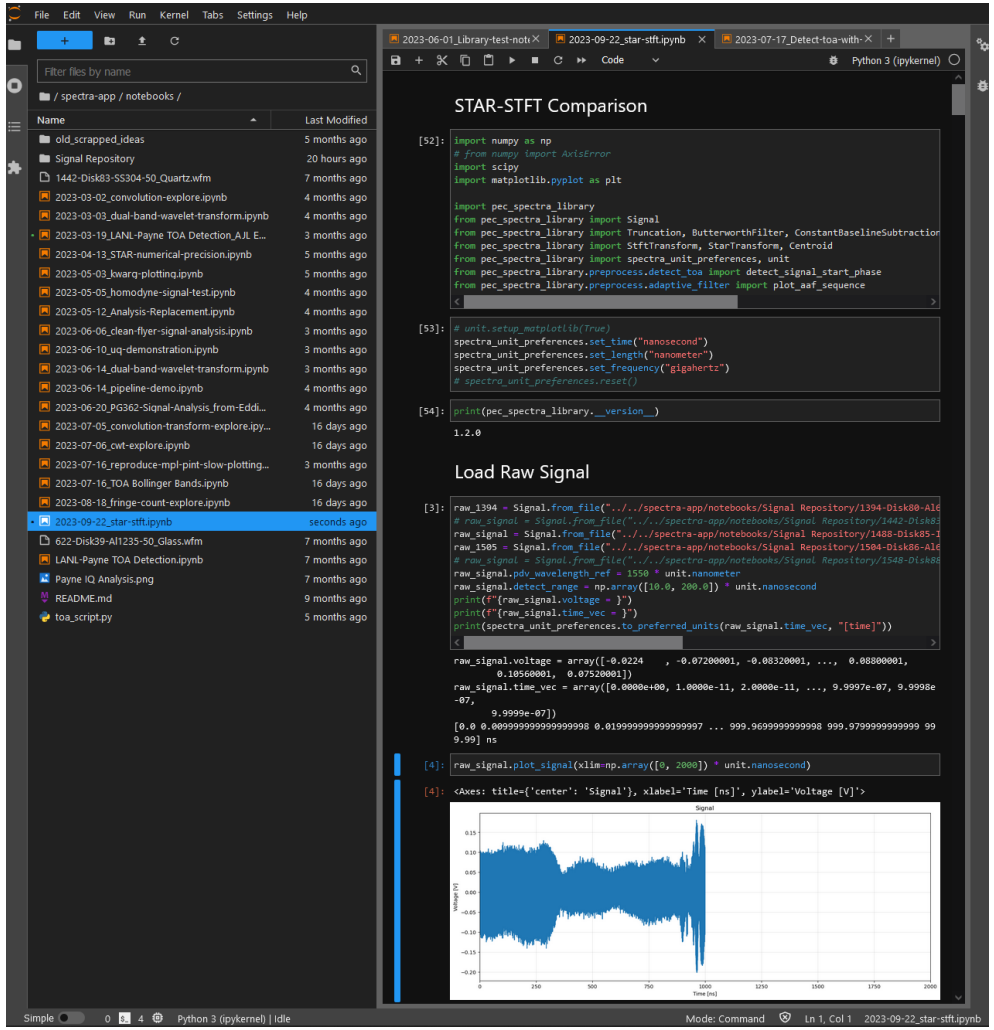
[48]: ax = stft_transformed_signal.plot_spectrogram(ylim=np.array((9.8, 10.6)) * unit.gigahertz, y_value="f", figsize=(10, 5))
      mean_centroid = Centroid("mean")
      stft_mean_centroid = mean_centroid.apply(stft_transformed_signal)
      stft_mean_centroid.plot_trace(ax=ax, y_value="f", label="Mean Centroid", legend=True)

      max_centroid = Centroid("max")
      stft_max_centroid = max_centroid.apply(stft_transformed_signal)
      stft_max_centroid.plot_trace(ax=ax, y_value="f", label="Max Centroid", c="green", legend=True)
      plt.show()
```



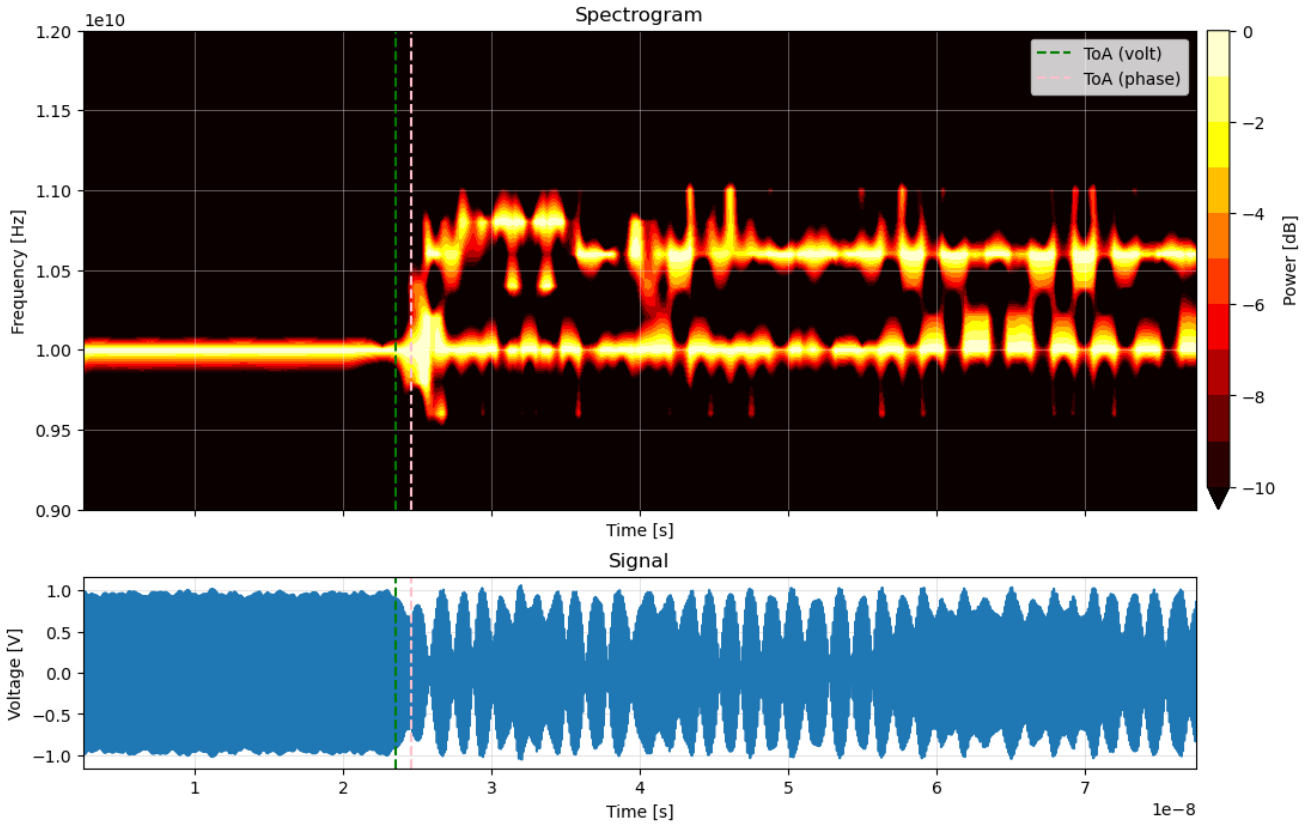
Library Features

- Modular source code
  - Data Classes:
    - Signal
    - TransformedSignal
    - Trace
  - Process Classes:
    - Preprocess
    - Transform
    - Postprocess
  - Workflow Classes:
    - Protocol
    - Analysis
    - UnitPreferences



Time of Arrival Detection

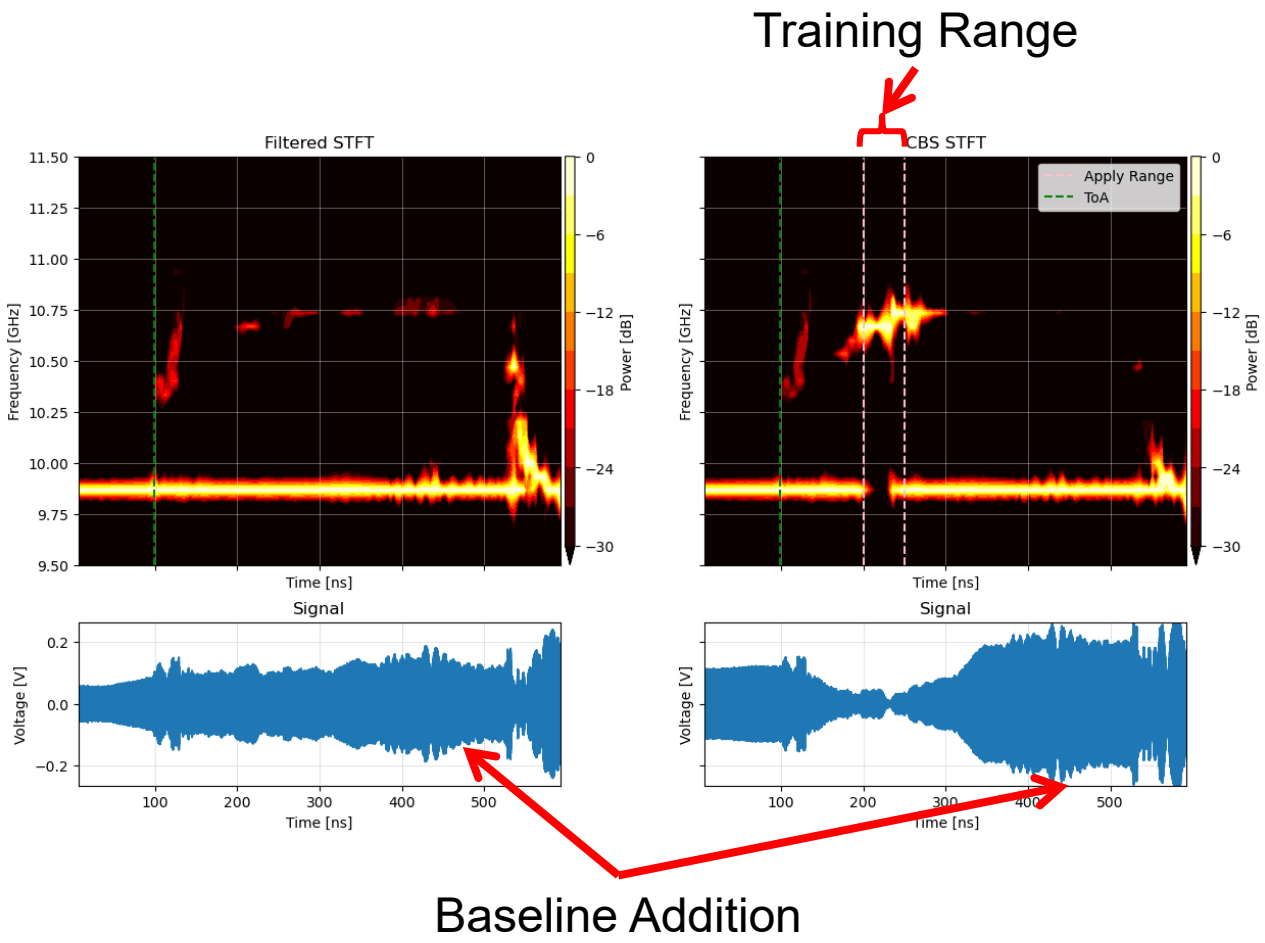
- Voltage
  - Signal strength
- Phase
  - Baseline switch (heterodyne)




```
detect_v, _ = detect_signal_start_voltage(filtered_signal, training_range=training_range, t0=16*unit.nanosecond, window_length_pts=150, sigma_threshhold=3)
detect_p, _ = detect_signal_start_phase(filtered_signal, training_range=training_range, t0=16*unit.nanosecond, window_length_pts=150, sigma_threshhold=3)
```

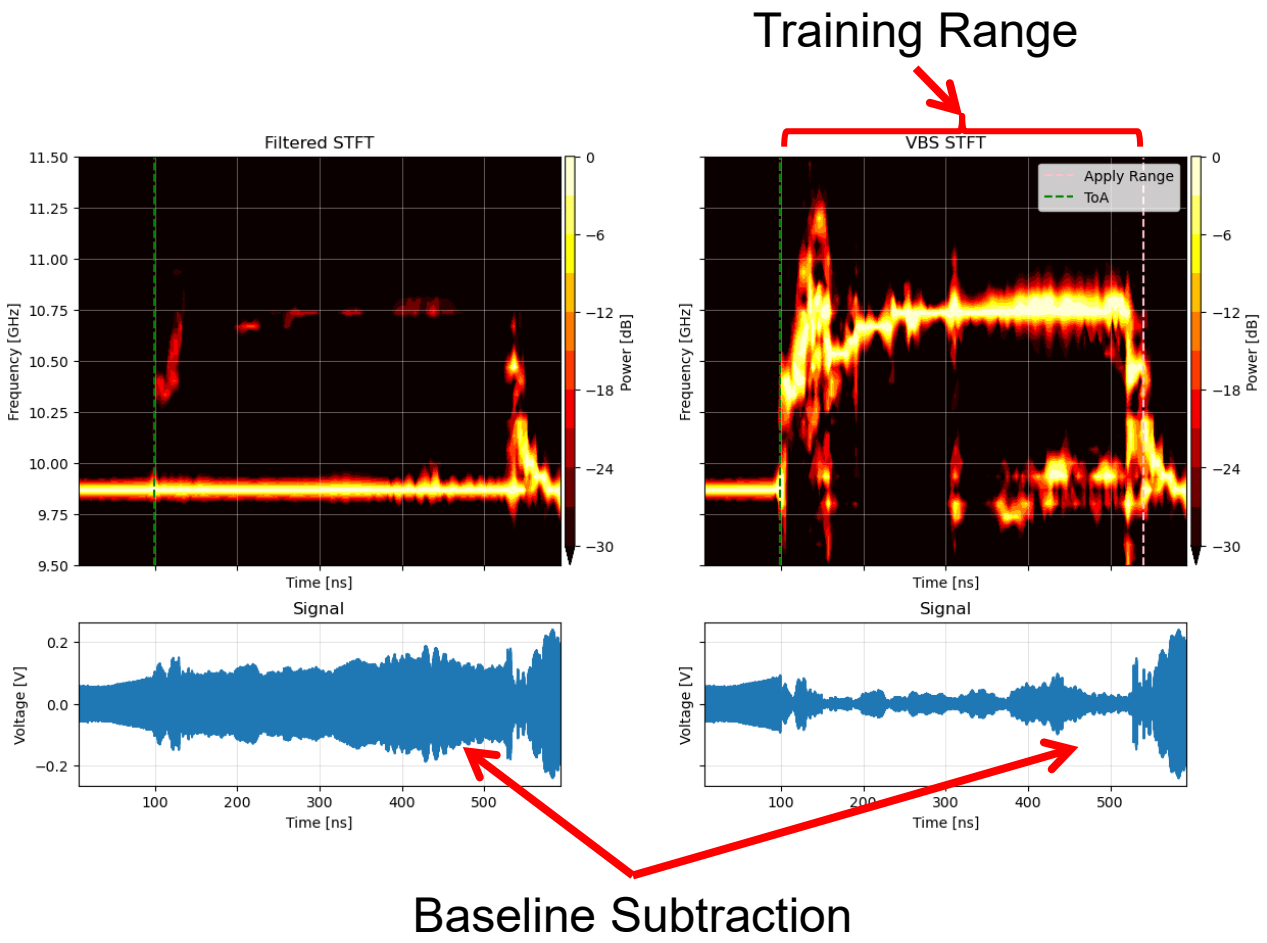
Constant Baseline Subtraction (CBS)

- Trains on one range
- Subtracts from entire signal
- Struggles with changing baseline amplitude or phase

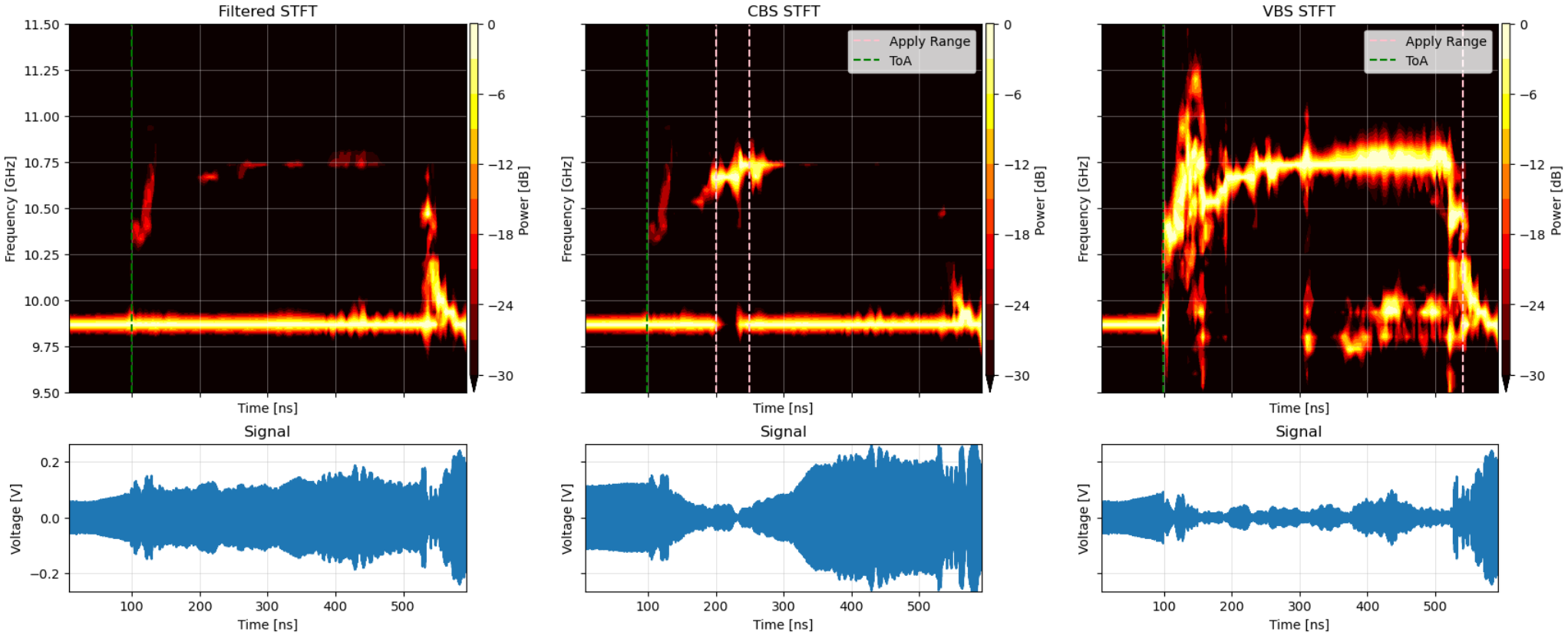


Variable Baseline Subtraction (VBS)

- Segregates signal into blocks
- Trains and subtracts each block individually
- Accounts for variable baseline amplitude and/or phase



Baseline Subtraction



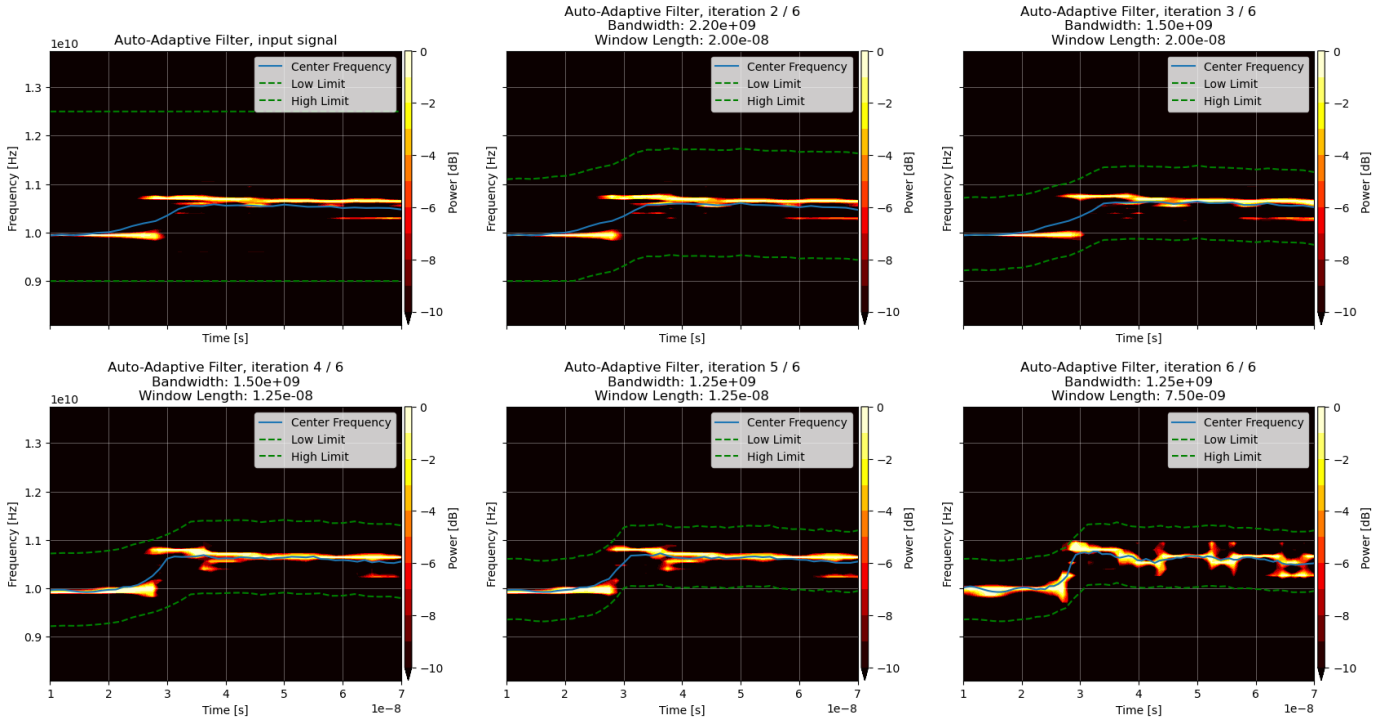

```
cbs = ConstantBaselineSubtraction(apply_range=np.array([200, 250]) * unit.nanosecond)
vbs = VariableBaselineSubtraction(apply_range=np.array([toa.to("ns").magnitude, 540]) * unit.nanosecond, block_length=12.5*unit.nanosecond)
```

```
cbs_1505 = cbs.apply(filtered_1505)
vbs_1505 = vbs.apply(filtered_1505)
cbs_1505_stft = temp_stft.apply(cbs_1505)
vbs_1505_stft = temp_stft.apply(vbs_1505)
```

Auto-Adaptive Filter

- Applies series of dynamic filters to input signal:

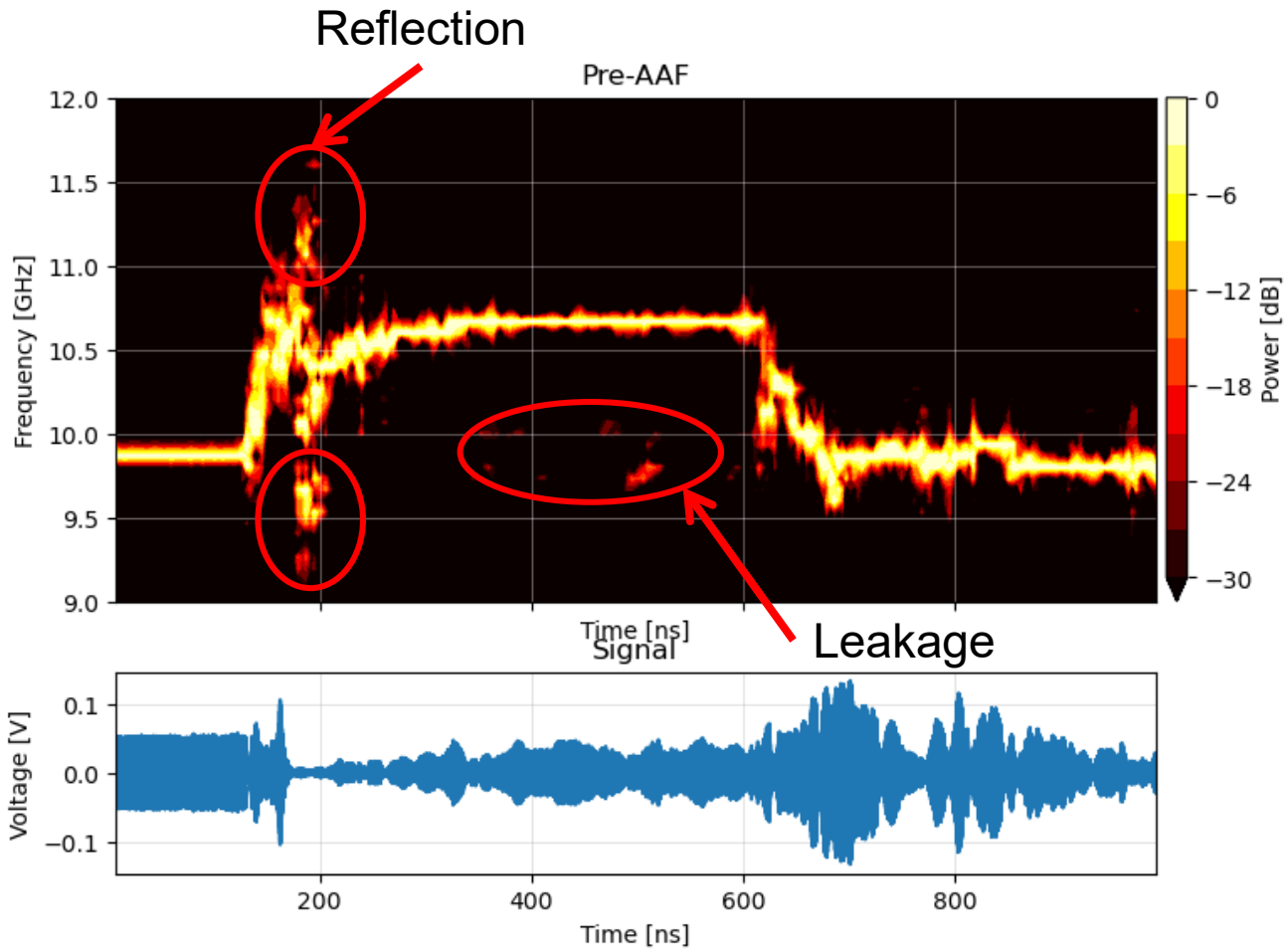
1. Find centroid
2. Determine limits for dynamic band-pass filter
3. Apply filter



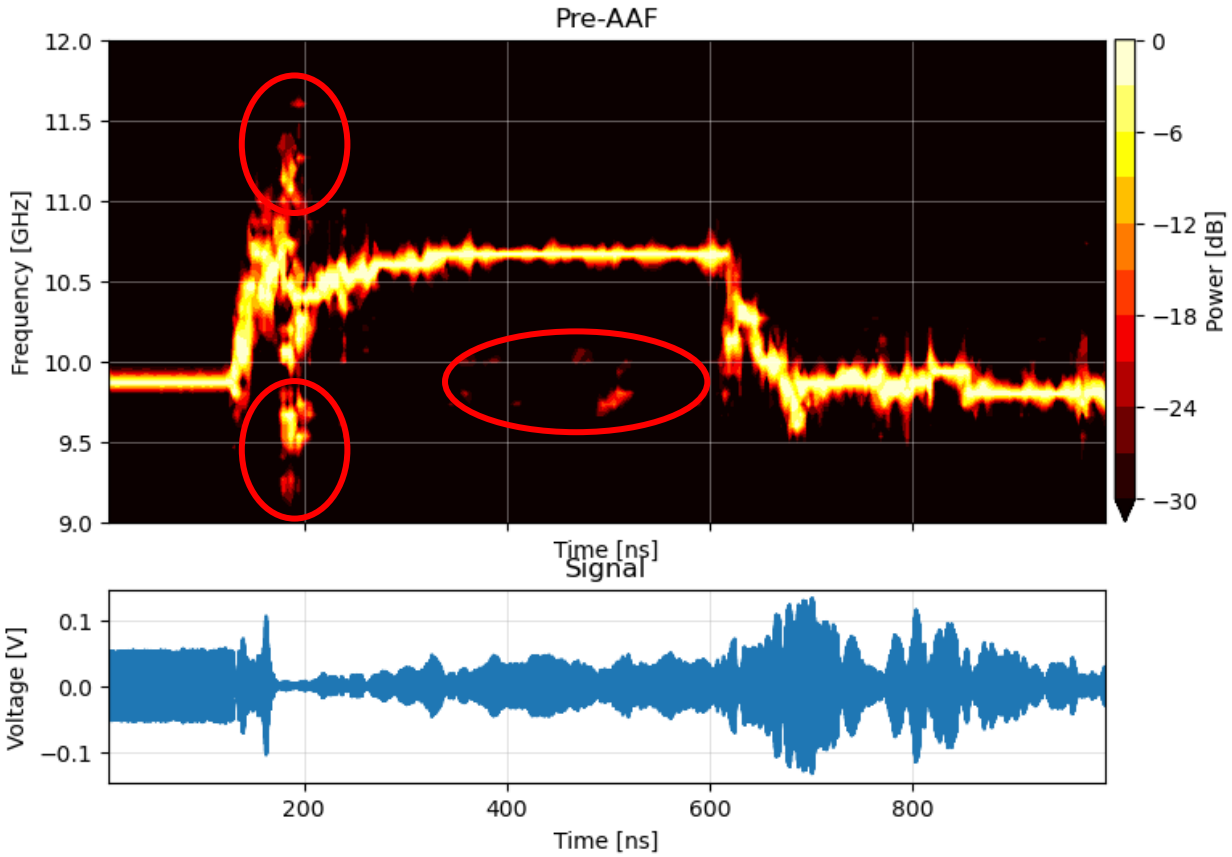

```
abs_limits = np.array([9, 12.5]) * unit.gigahertz
bandwidths = np.array([2.2, 1.50, 1.25]) * unit.gigahertz
windows = np.array([20, 12.5, 7.5]) * unit.nanosecond
aaf = AutoAdaptiveFilter(
    absolute_limits=abs_limits, bandwidths=bandwidths,
    window_lengths=windows, transform_cls=StftTransform, overlap_ratio=0.90
)
aaf_signal = aaf.apply(vbs_signal)
```

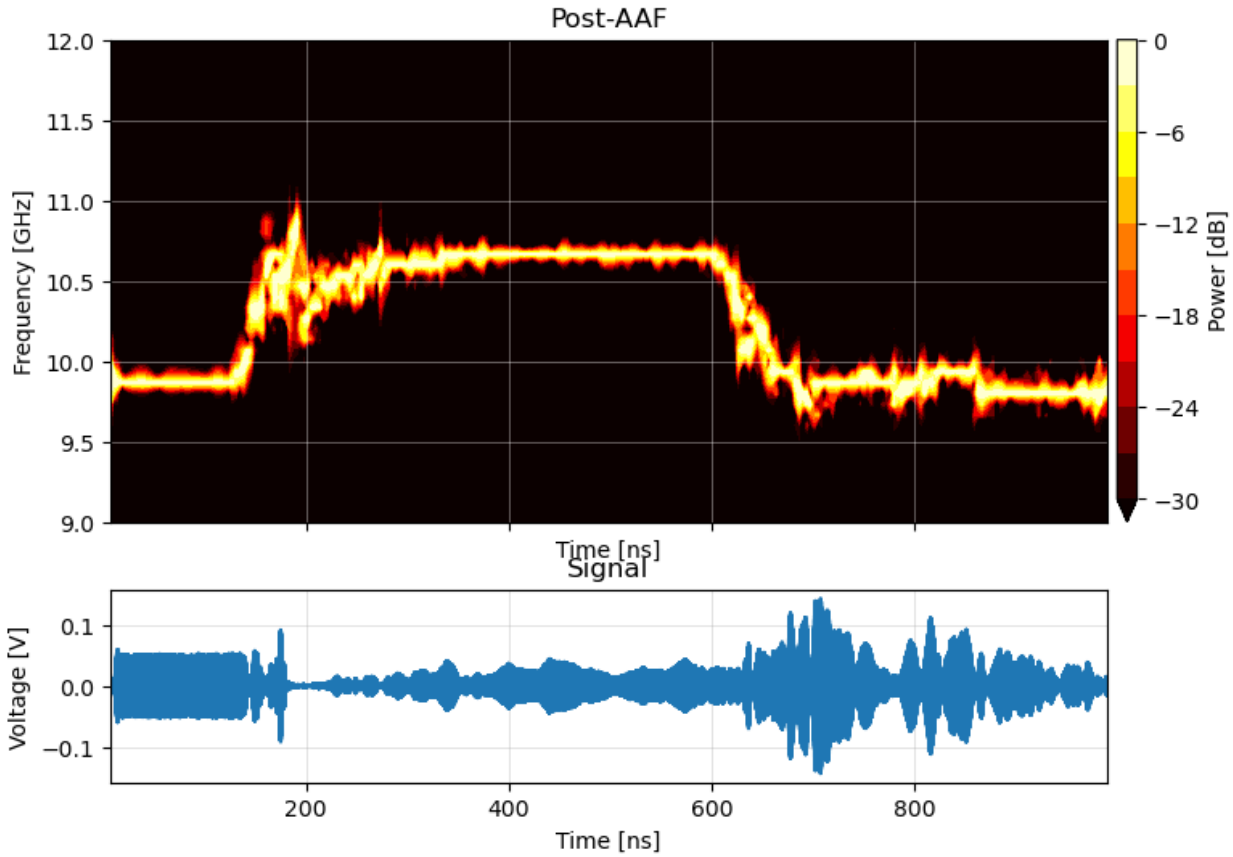
Auto-Adaptive Filter

- Signal contains unwanted artifacts
  - Reflections/ Harmonics
  - Baseline leakage
- Small, but impact centroid finding methods



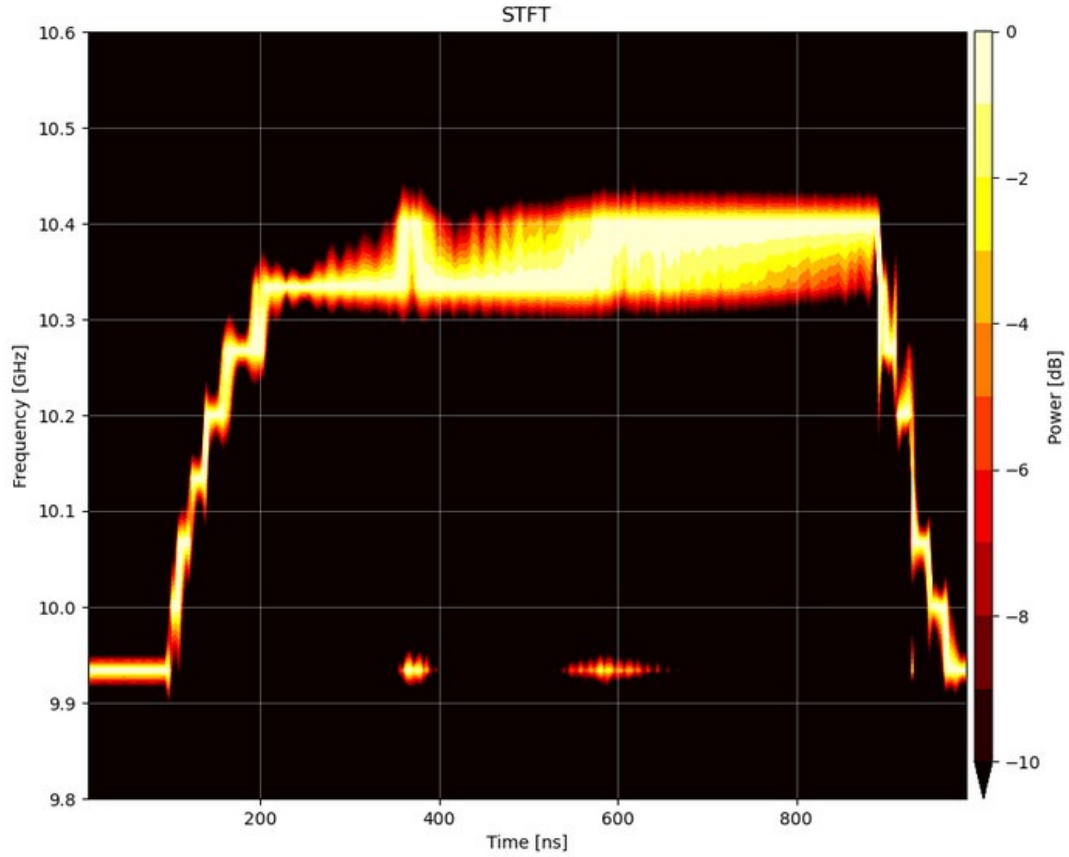
Auto-Adaptive Filter Improvement



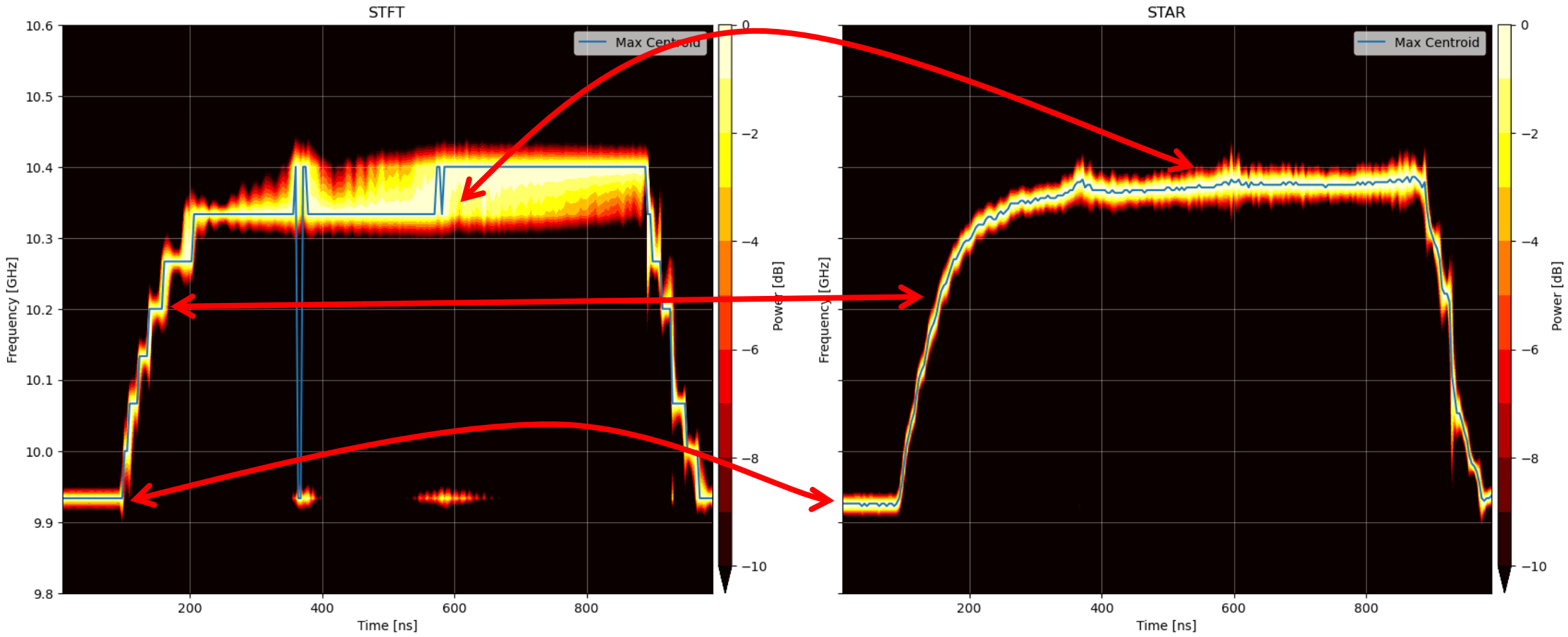


STFT & STAR

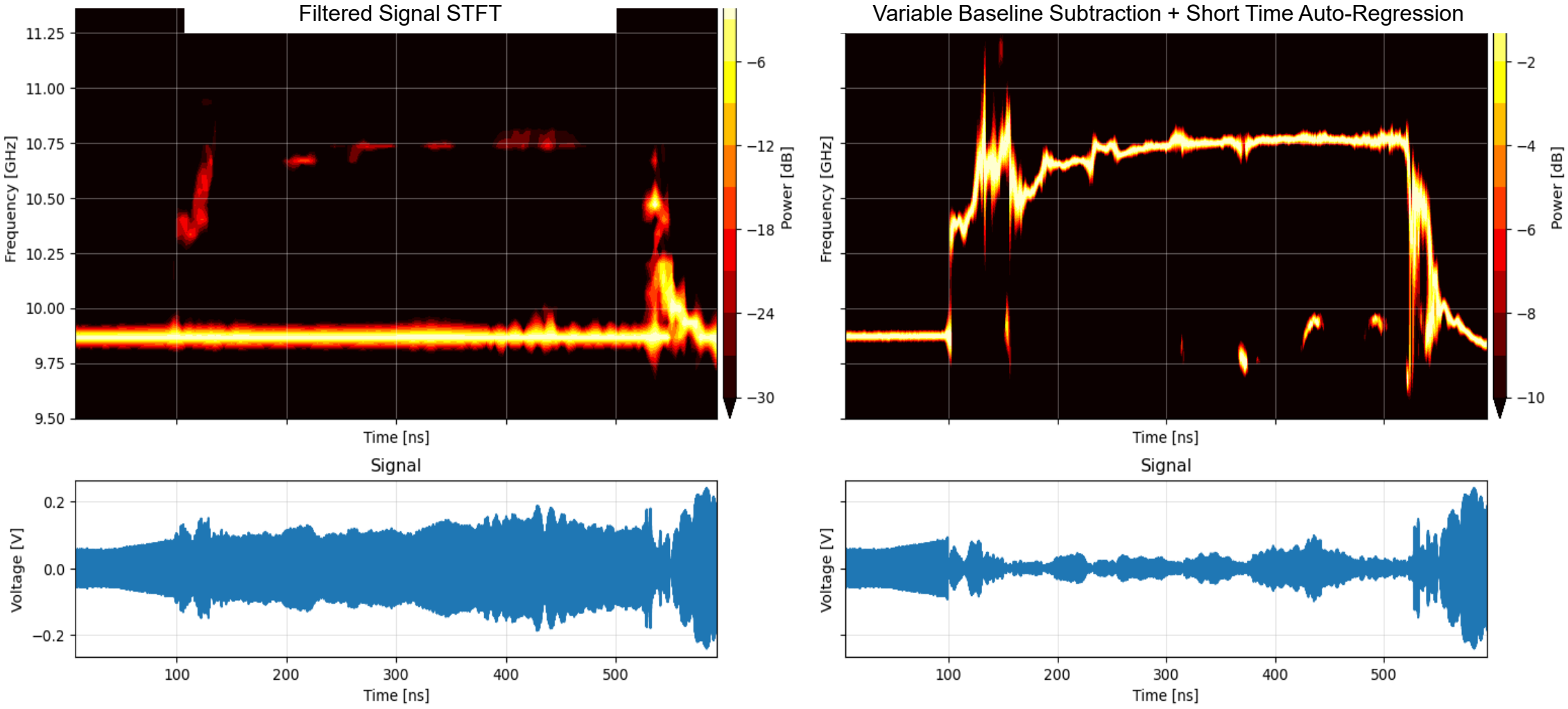
- Short Time Fourier Transform (STFT)
  - Faster runtime
  - Limited resolution by definition
- Short Time Auto-Regression (STAR)
  - Slower runtime
  - Greater resolution



STFT & STAR

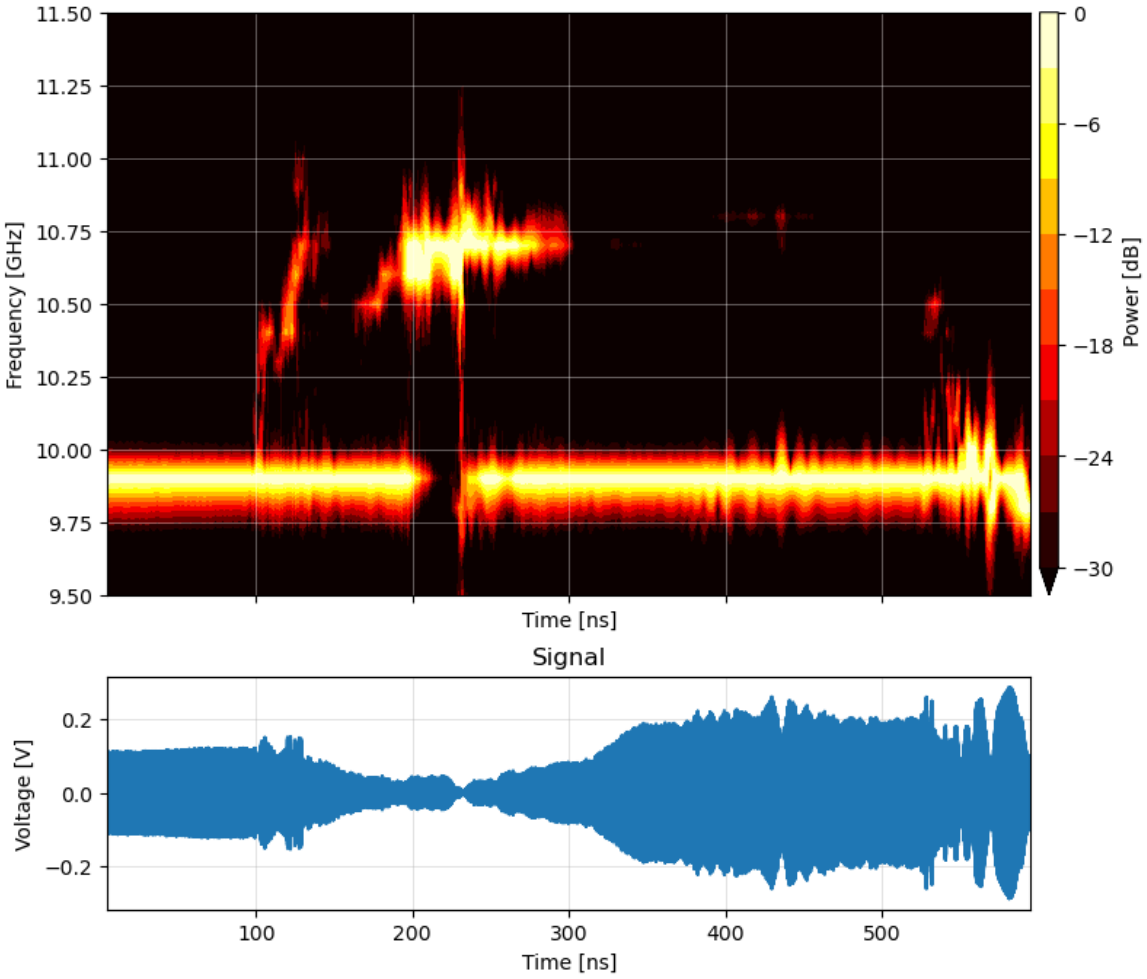

STFT
STAR
Max Centroid
Frequency [GHz]
Time [ns]
Power [dB]
10.6
10.5
10.4
10.3
10.2
10.1
10.0
9.9
9.8
200
400
600
800
0
−2
−4
−6
−8
−10

Spectra Capability

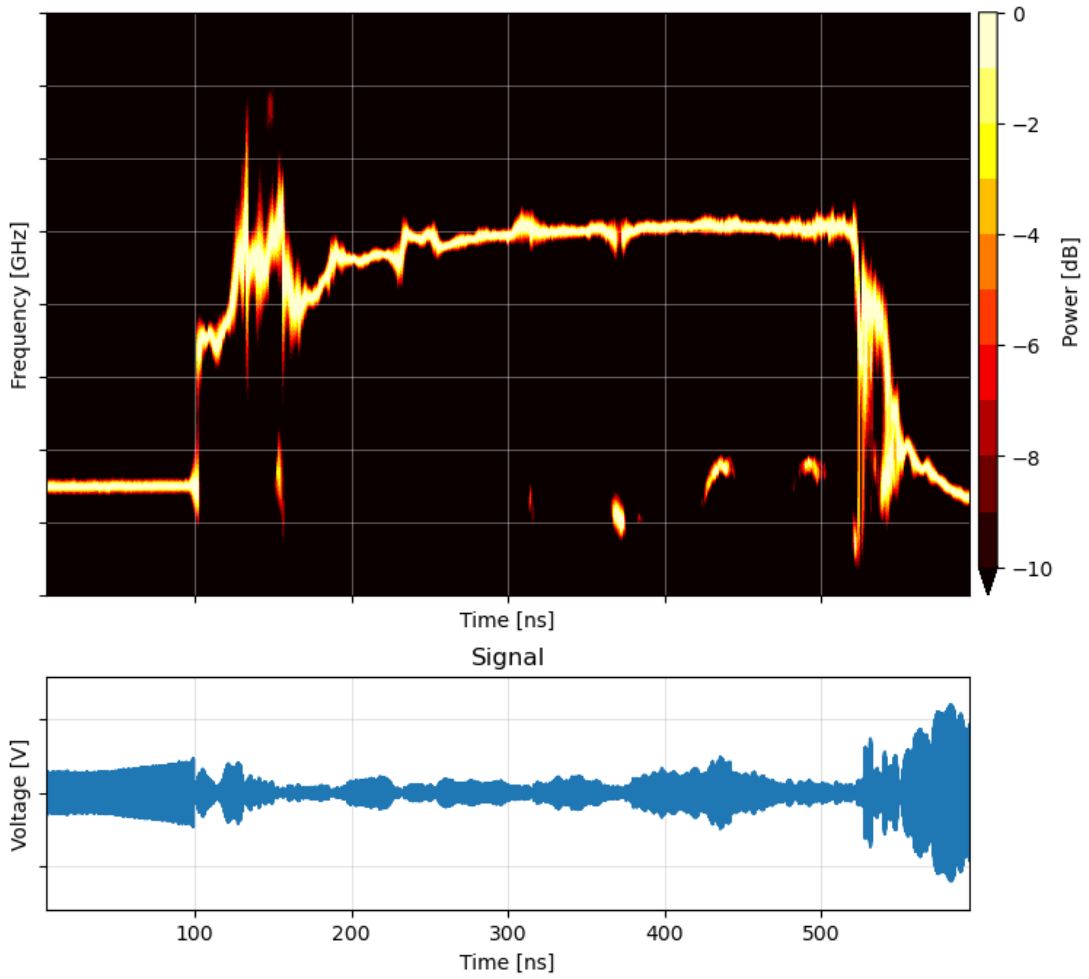


Spectra Capability (2)

Constant Baseline Subtraction + Short Time Fourier Transform



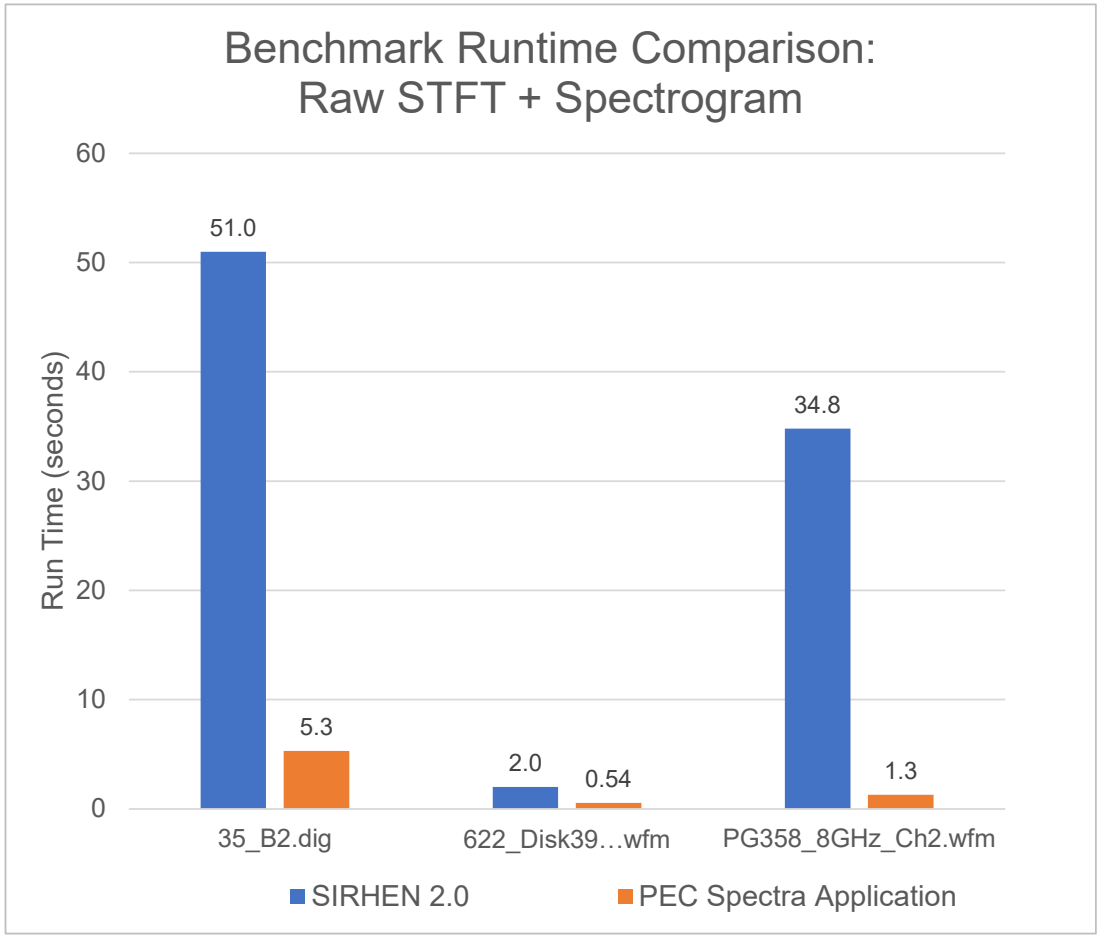
Variable Baseline Subtraction + Short Time Auto-Regression



Runtime Comparison

- Runtime includes:
  - STFT on Raw Signal
  - Render of spectrogram plot
- Vanilla Matlab & Spectra
  - No parallelization
  - No multi-threading
  - CPU only
  - Interpreted Matlab, no compile

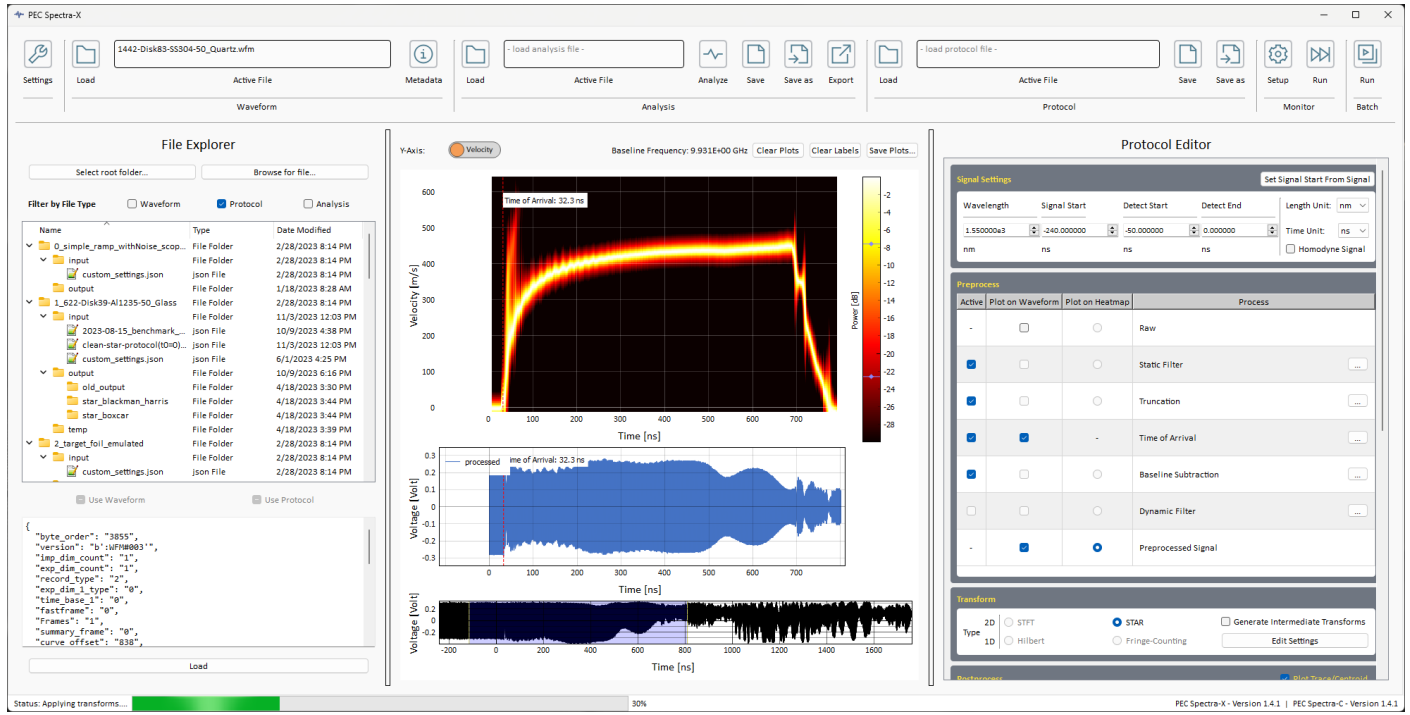
File	35_B2.dig	622_Disk39…wfm	PG358_8GHz_Ch2.wfm
num_pts	5,000,000	100,000	1,000,000
overlap_ratio	0.95	0.95	0.95
spectrogram_time_pts	40,000	4,000	80,000
points_per_window	2,500	500	250



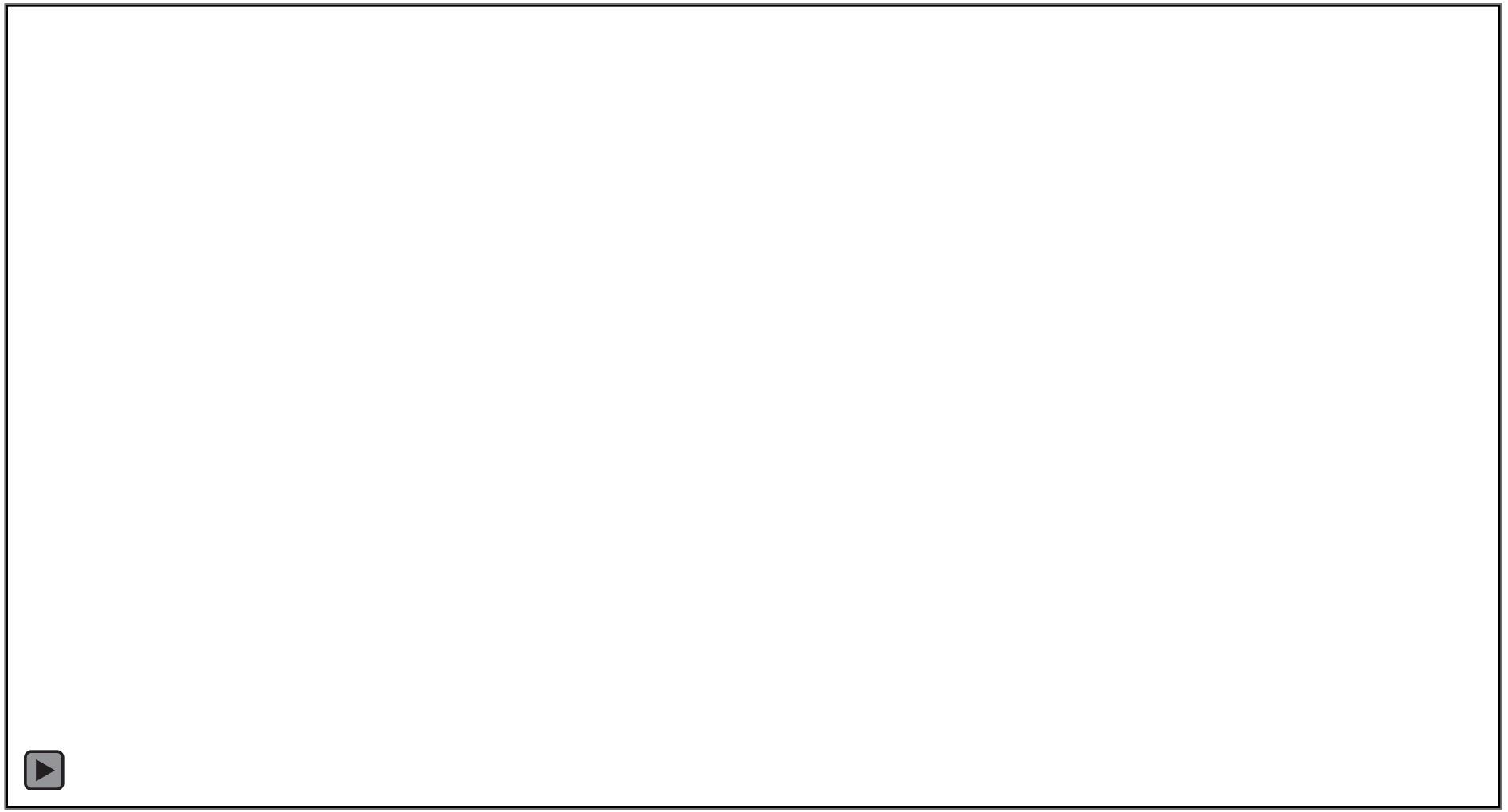
APPLICATION

Spectra Application

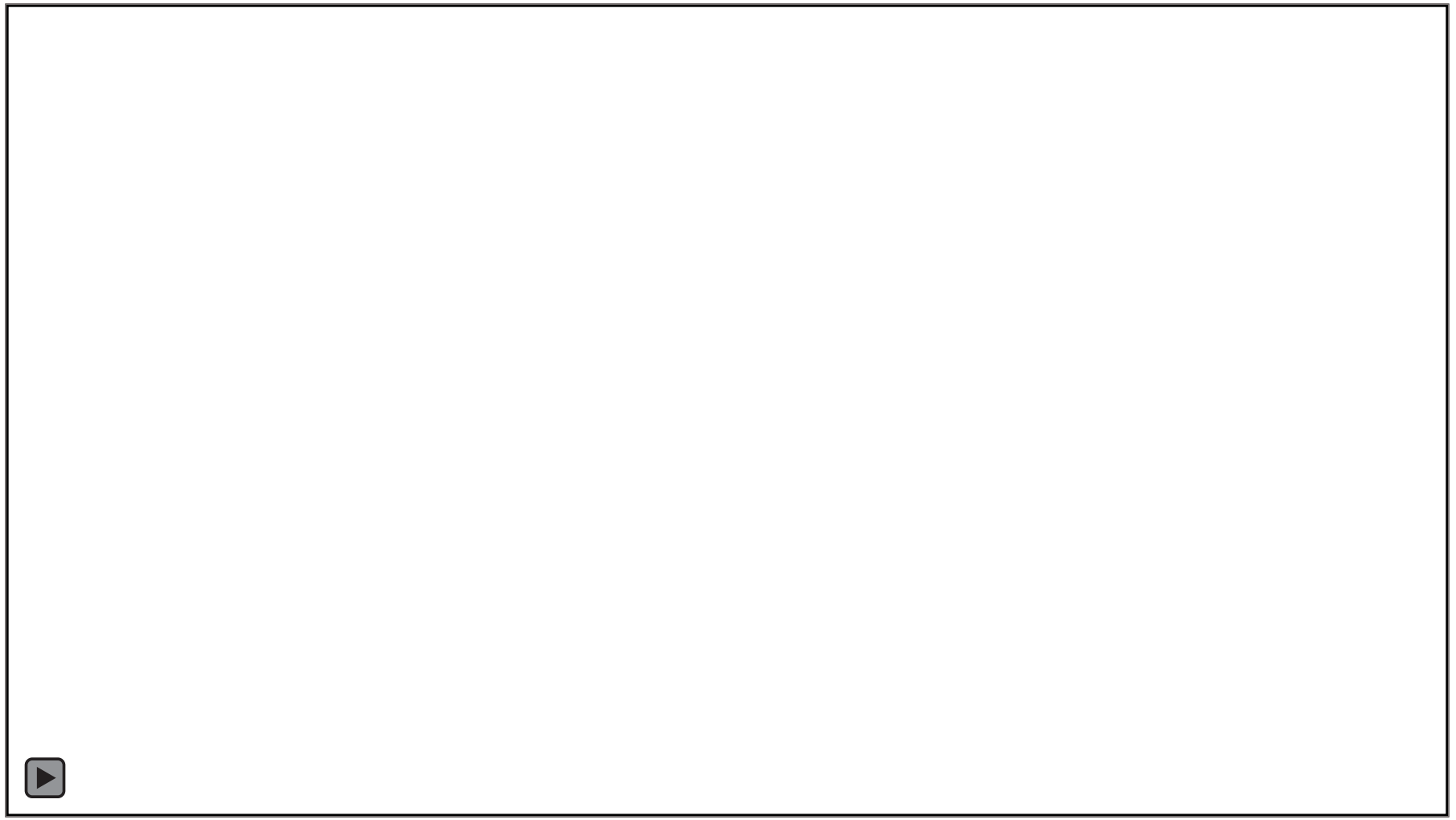
- ***Application***
  - Bundled Executable
  - Graphical User Interface
  - Fast, Seamless
  - Interactive Plotting
  - Requires Windows 7+



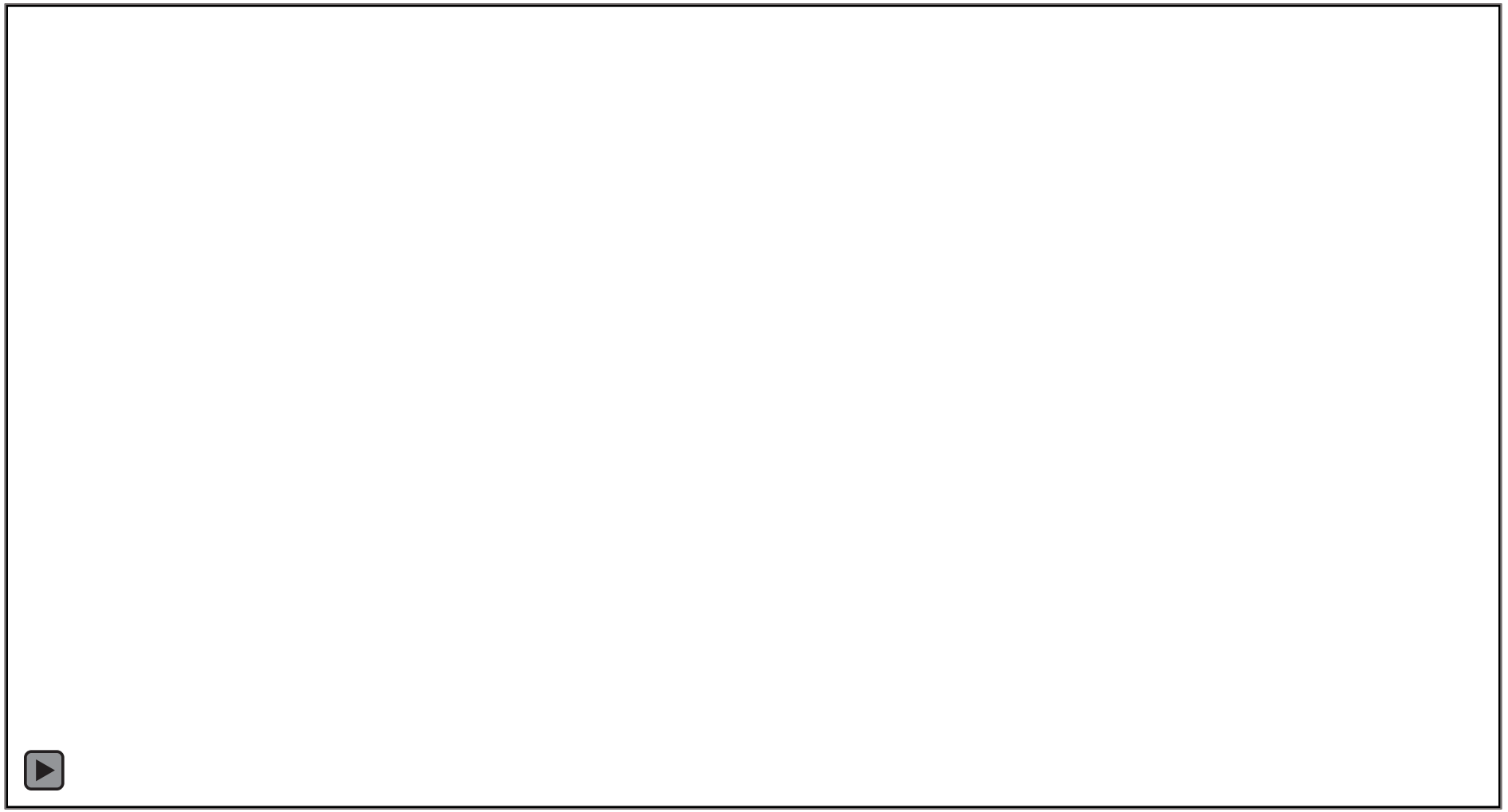
Plot Interaction – Pan & Zoom



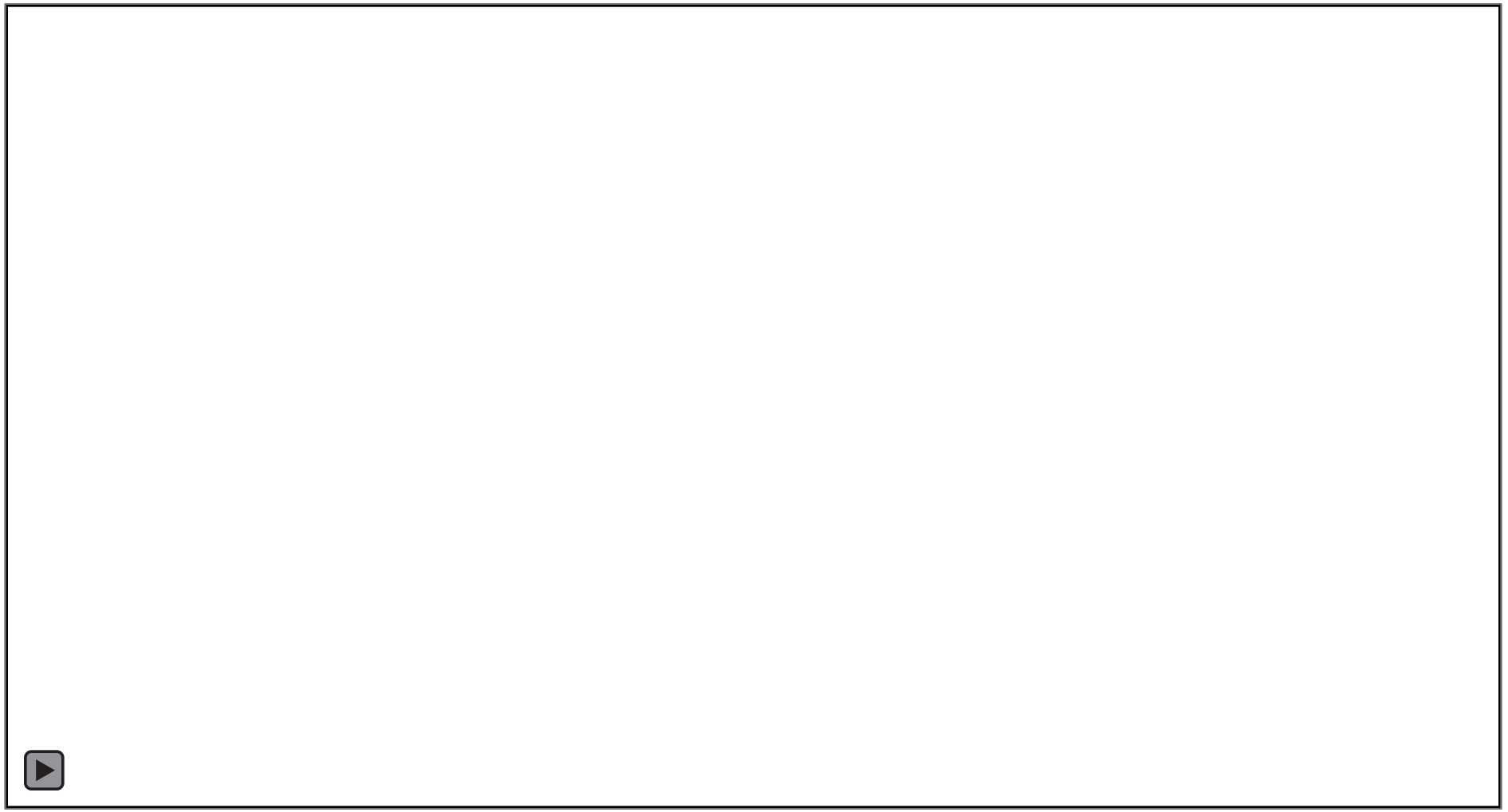
Persistent Zoom Extents



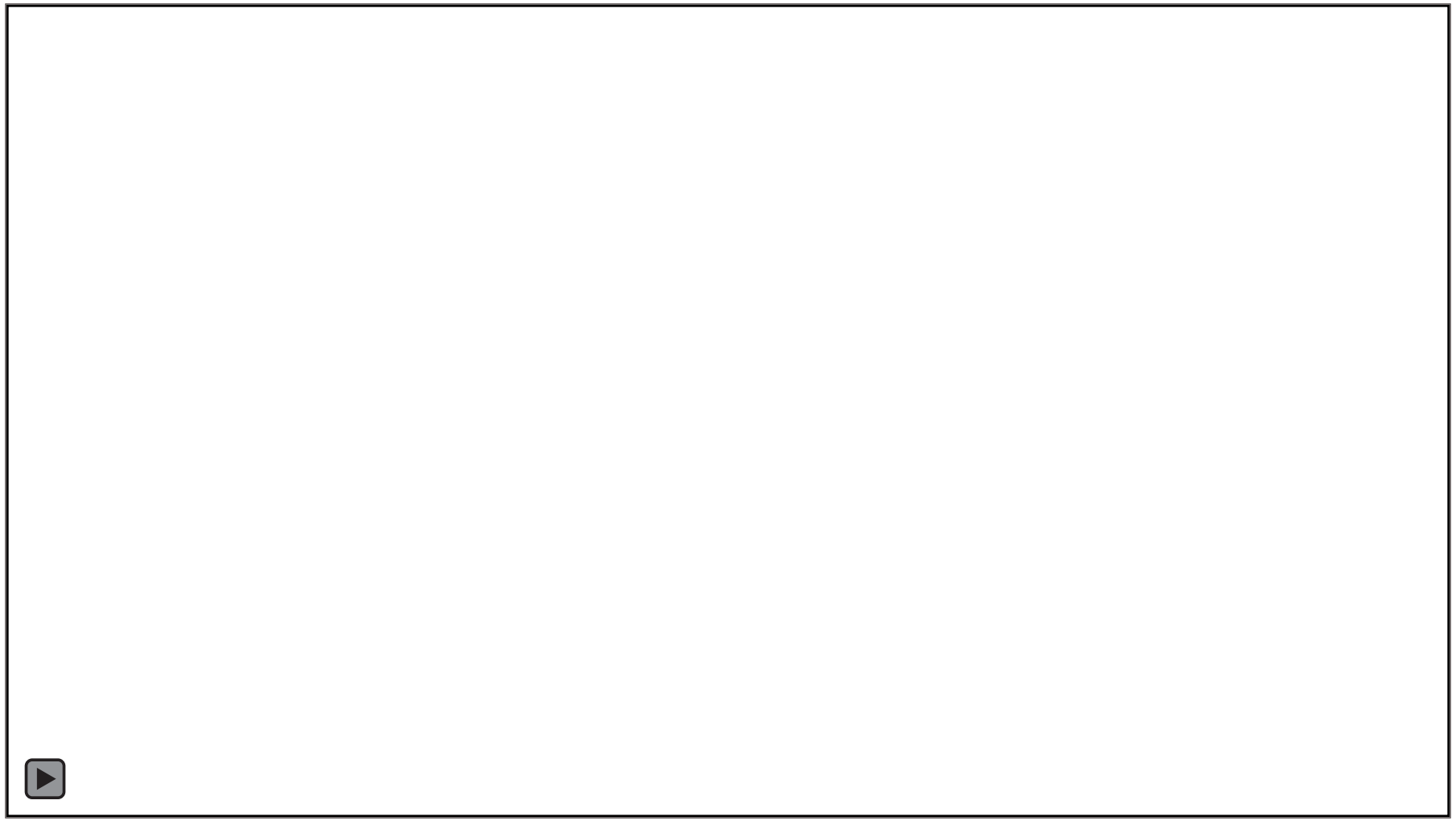
Click & Drag Levels Colorbar



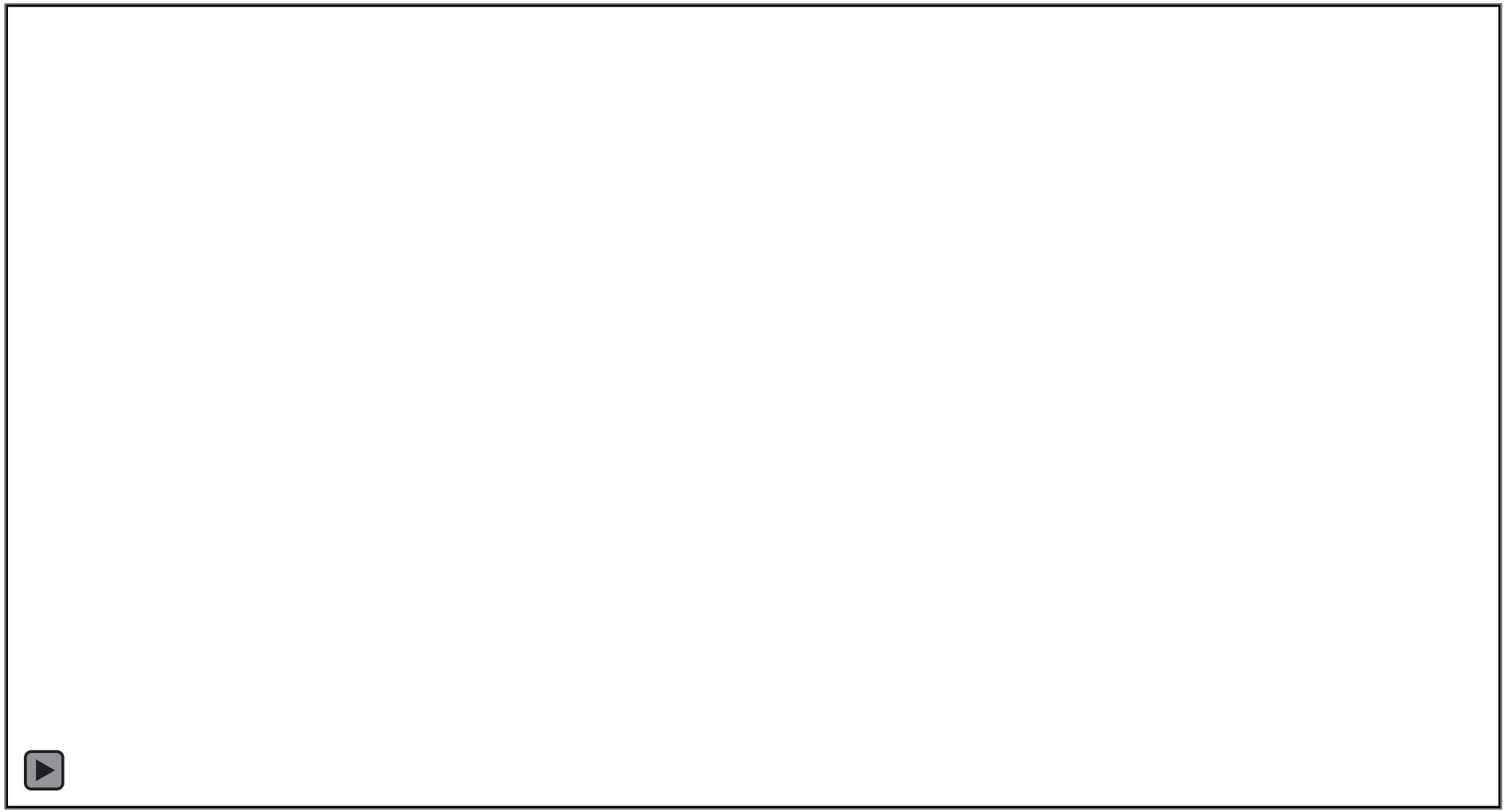
Plotting with Units



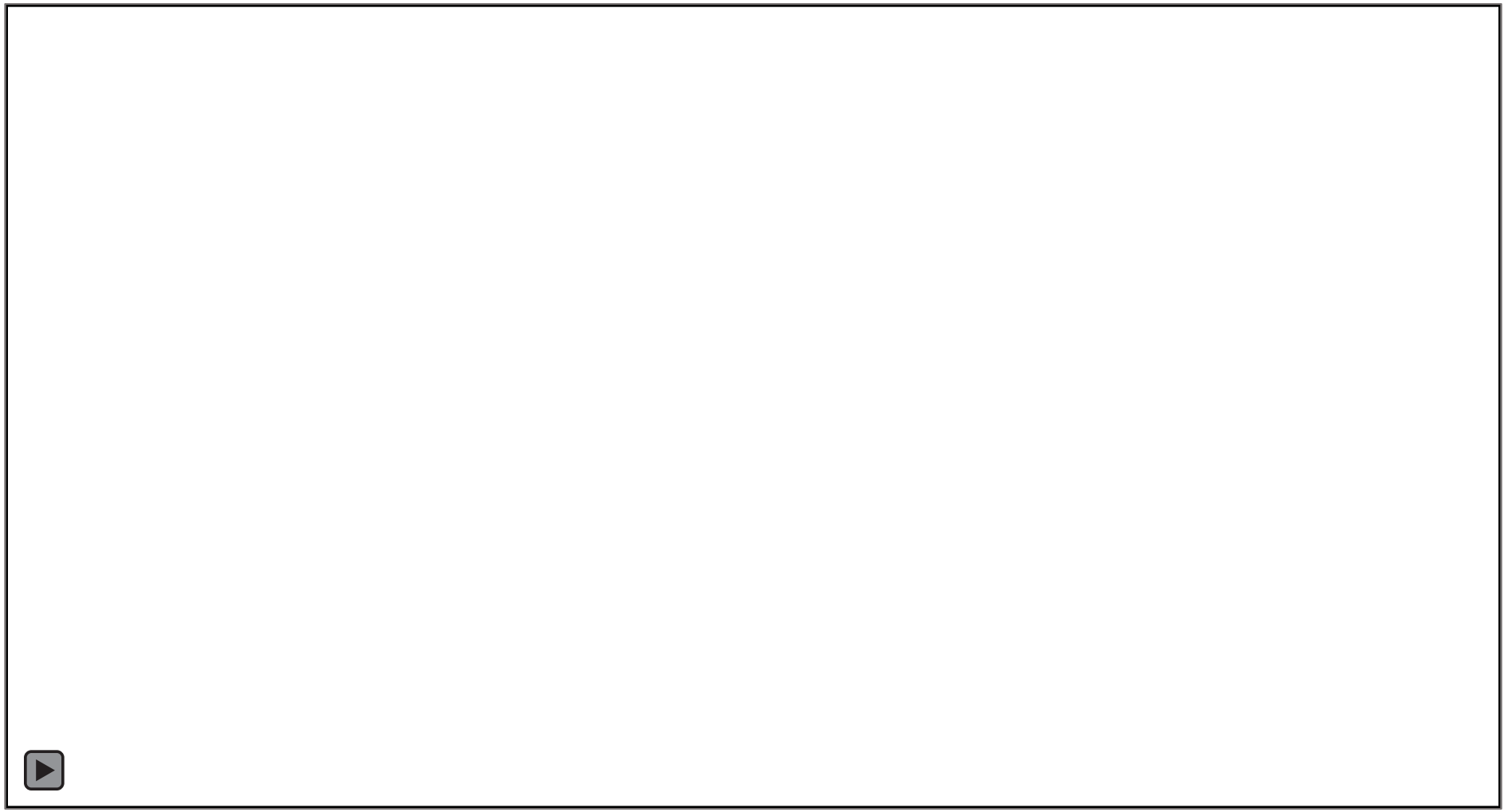
Track Processing History



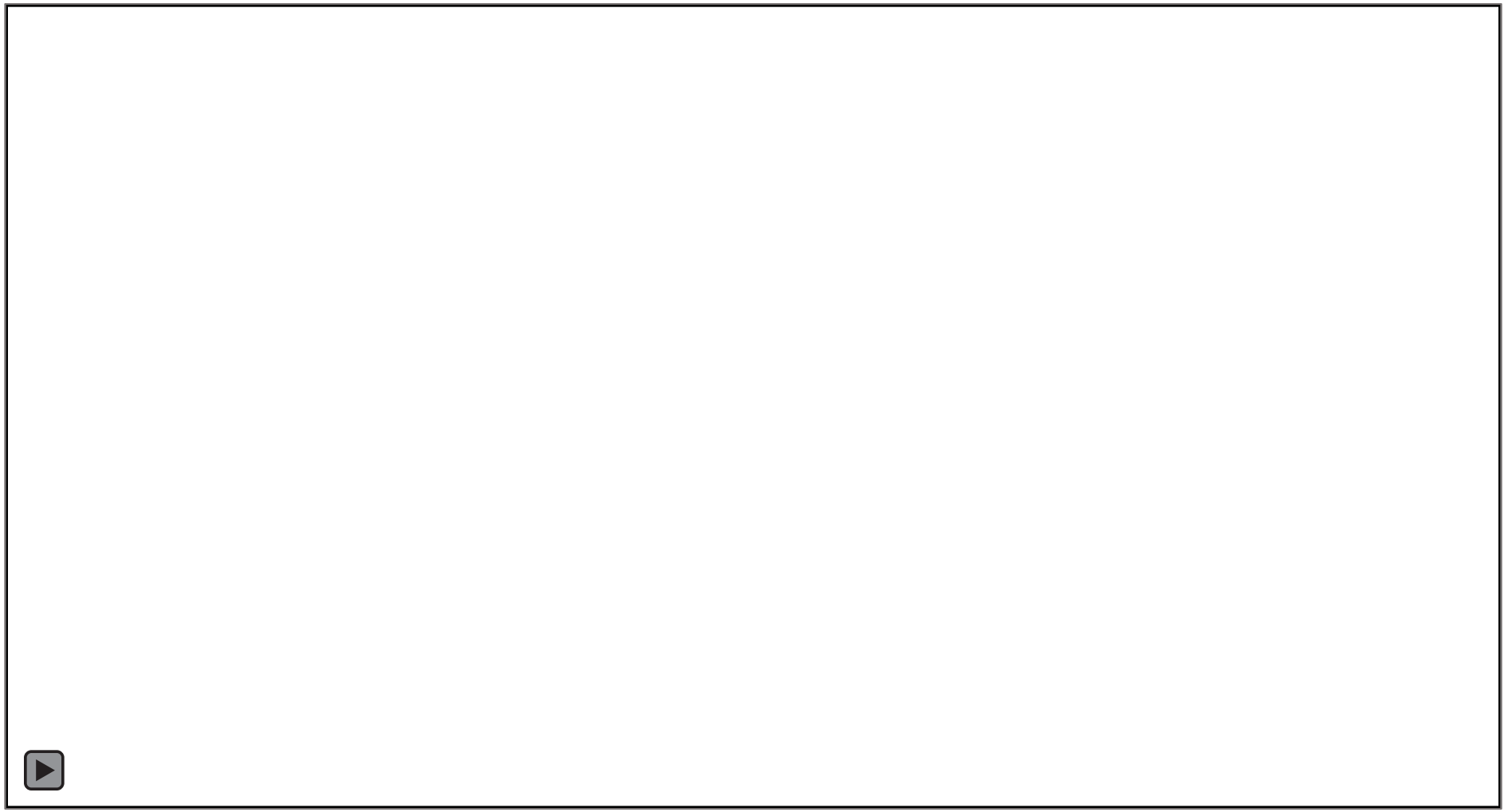
Mark Data Points



Reusable Protocol

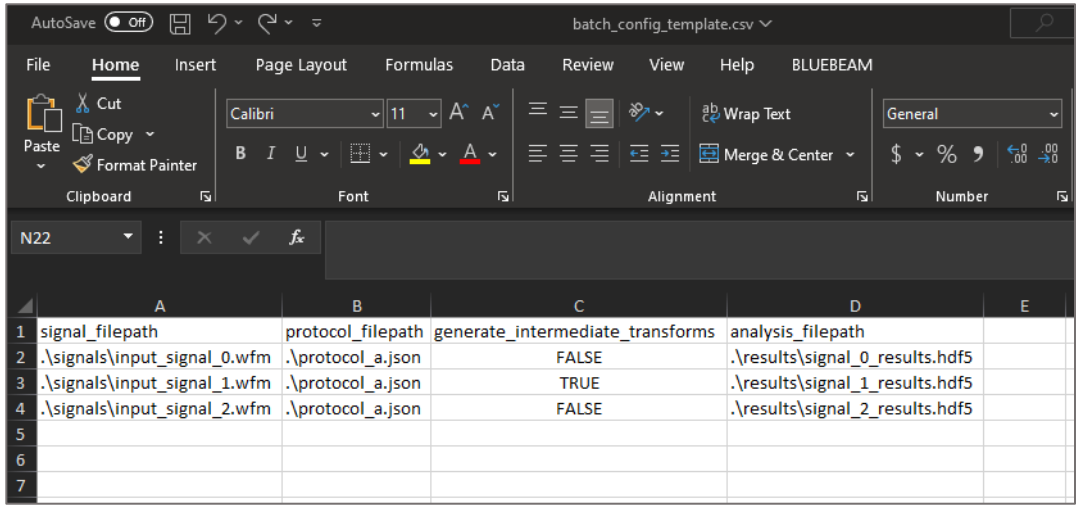


Save & Load Analysis File



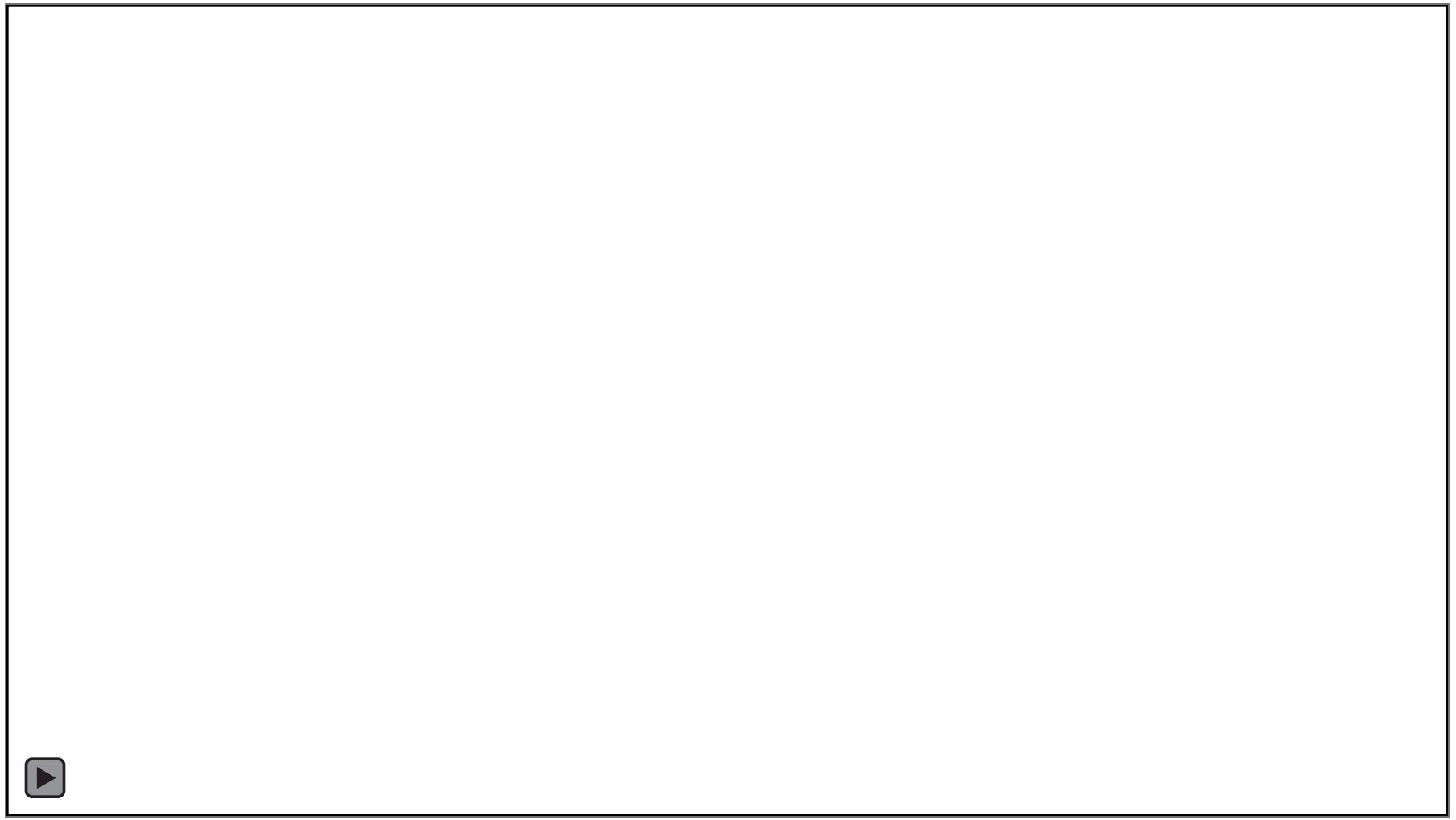
Batch Analysis

- Configure batch job with a csv file:
  - Input signal (path)
  - Protocol (path)
  - Generate intermediate transforms
  - Analysis output (path)
- One row per analysis



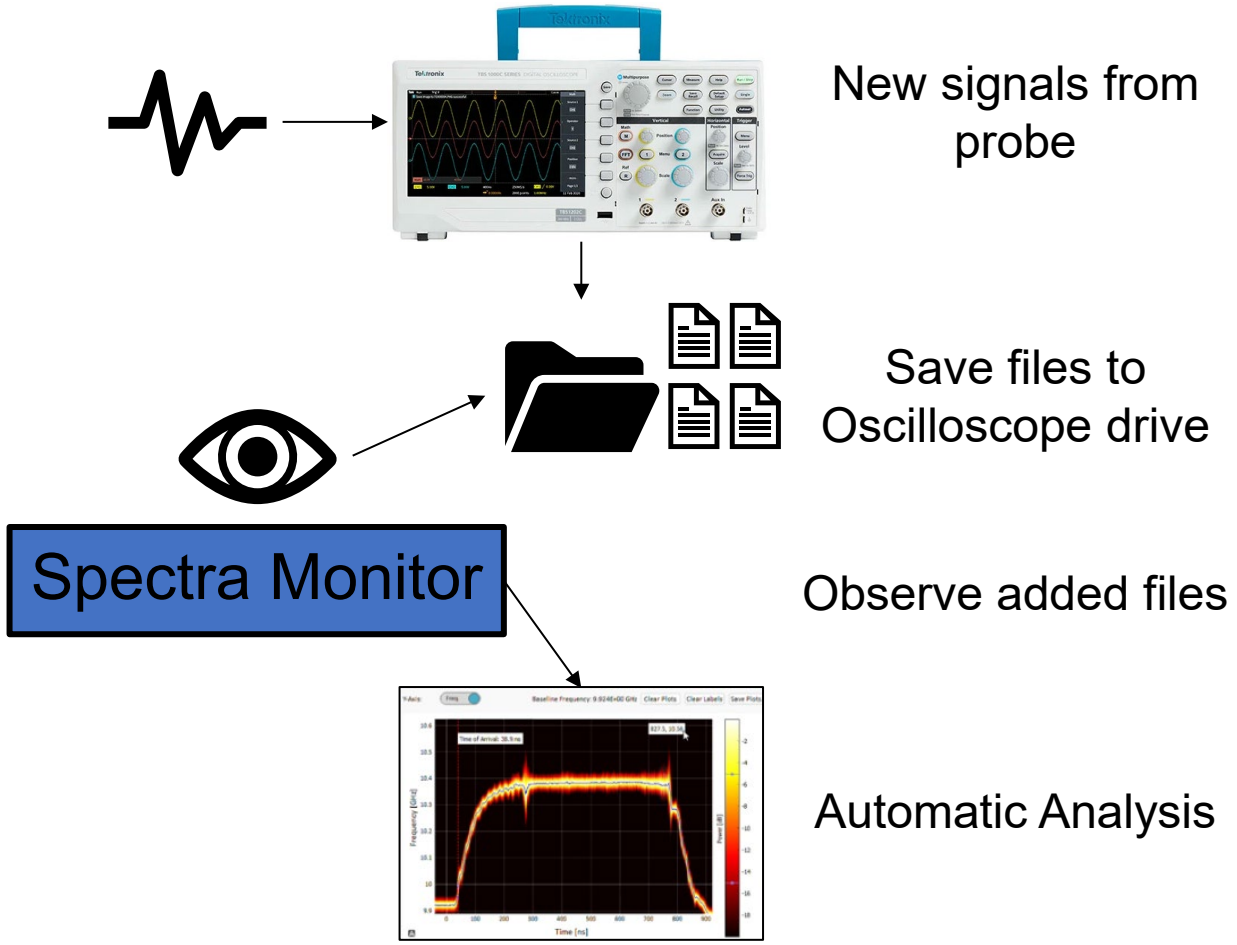
	A	B	C	D	E
1	signal_filepath	protocol_filepath	generate_intermediate_transforms	analysis_filepath	
2	.\signals\input_signal_0.wfm	.\protocol_a.json	FALSE	.\results\signal_0_results.hdf5	
3	.\signals\input_signal_1.wfm	.\protocol_a.json	TRUE	.\results\signal_1_results.hdf5	
4	.\signals\input_signal_2.wfm	.\protocol_a.json	FALSE	.\results\signal_2_results.hdf5	
5					
6					
7					

Batch Analysis

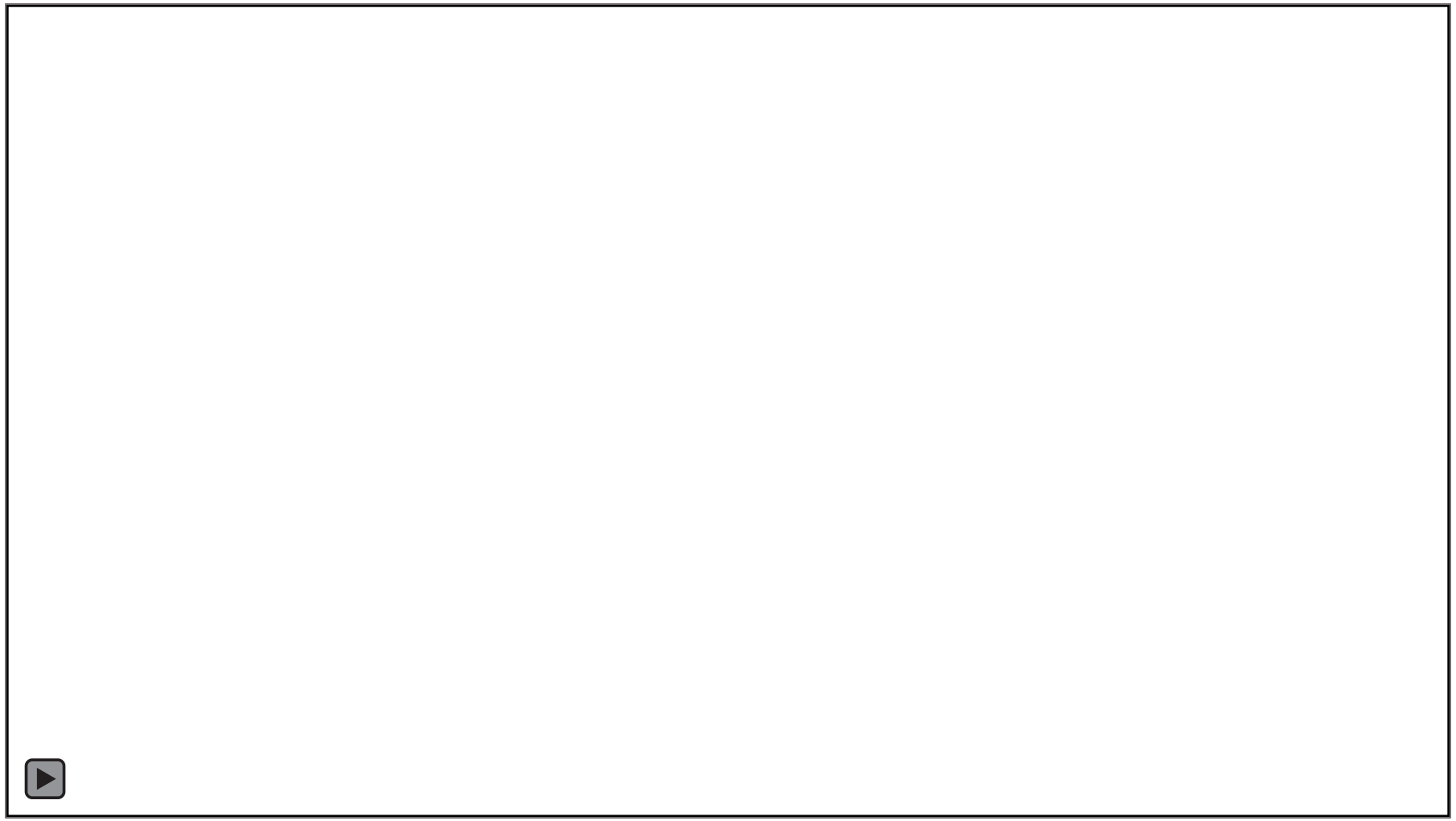


Monitor Mode

- Spectra can be set up to monitor a file directory on an oscilloscope (Windows 7+)
- Enables quick, automated analyses for rapid feedback



Monitor Mode

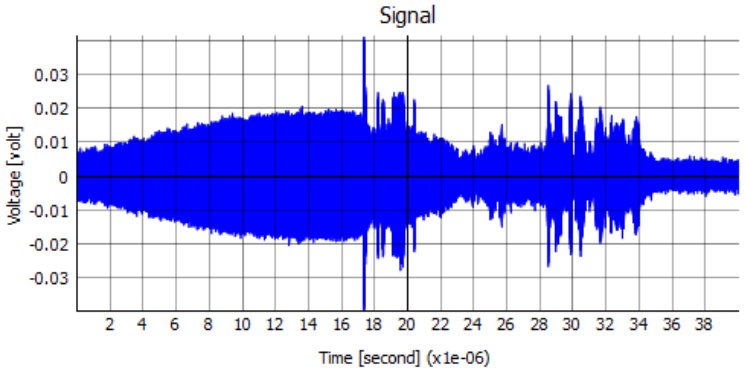


SPECTRA WORKFLOW

Data Components

Signal

- Input file
- Immutable



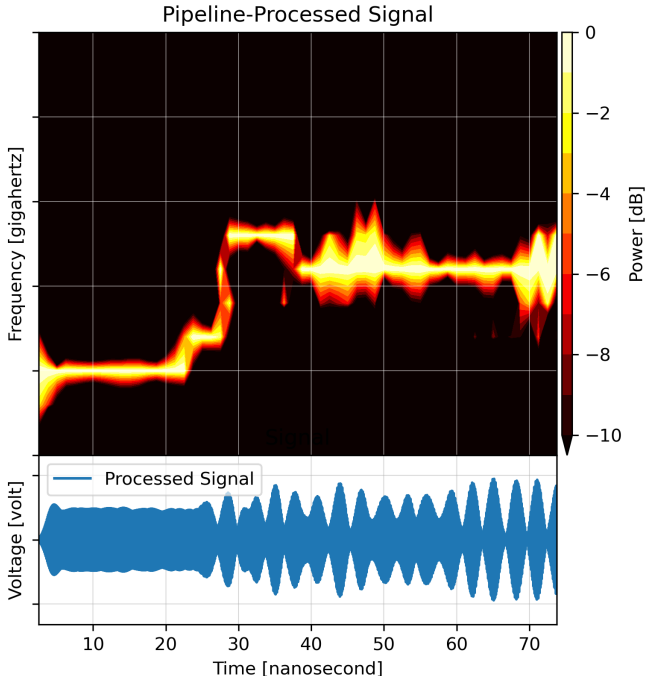
Protocol

- Processing Instructions
- Human-readable

- Truncation
- Pre-Filter(s)
- Arrival Time
- Transform
- Post-Filter(s)

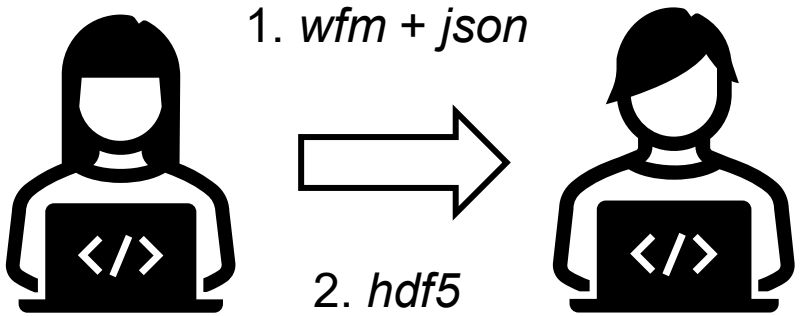
Analysis

- Output Data
- Hierarchical & Binary



Sharing Data & Results

- Option 1:
  - Share input *Signal* & *Protocol* files
  - Second user: opens both and presses “Analyze”
- Option 2:
  - Share output *Analysis* file
  - Second user: opens *Analysis* file



Availability

- Spectra is free but not open-source (limited distribution)
- Currently available to:
  - Federal Agencies
  - National Labs
- Potentially available to (case-by-case basis):
  - Corporations
  - Nonprofit Research Organizations
  - Universities
- Contact Alex to request the software

Thank You!

- Contact us:
  - For software questions or requests:
    Alex Lakocy: alakocy@protection-consultants.com
  - For PEC or programmatic matters:
    Matt Barsotti: mbarsotti@protection-consultants.com

Protection Engineering

CONSULTANTS

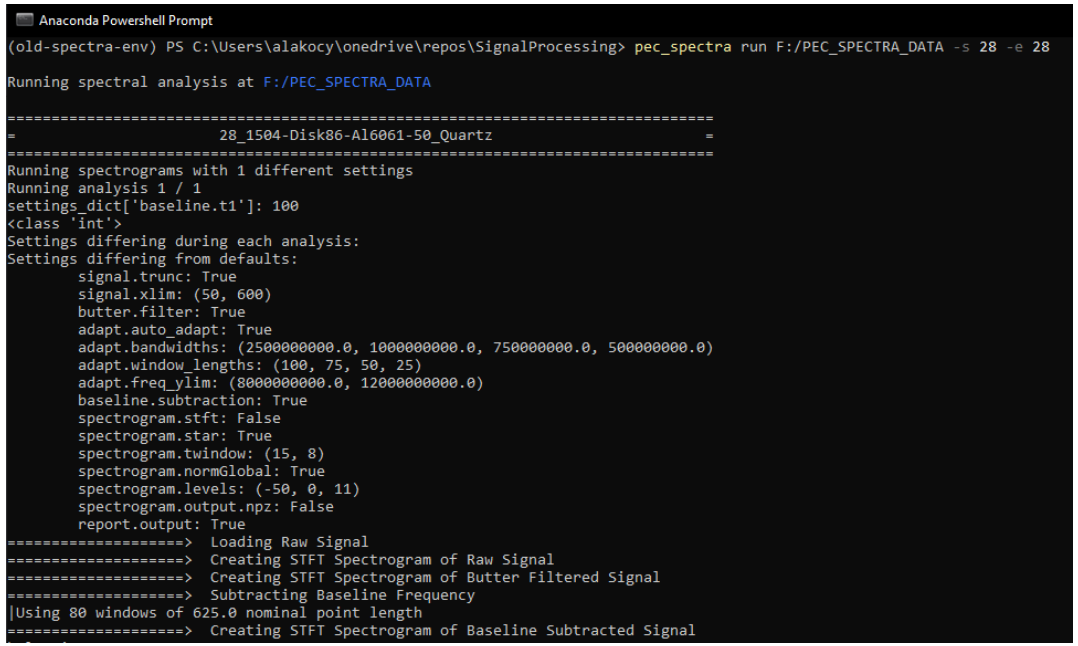
Approved for PDV Workshop 2024

BONUS SLIDES

Background

- 2017
  - Start of laser-driven flyer testing program
- 2018-2022
  - Developed Python scripts for signal processing
  - “Research Code”, based on project needs
- 2023
  - Finished development of production-level codebase
- 2024
  - Pursuing transition: open-source for government users

“Original” command-line *Spectra* software



Protocol

- Instruction for processing signal(s)
- Signal-Agnostic
  - Can be applied to any number of different Signals
- Human-readable & editable
  - JSON format



```
{
    "preprocess_pipeline": {
        "truncation": {
            "t_start": 1.8e-05,
            "t_end": 3.2e-05,
            "inclusive_end": false
        },
        "static_filter": {
            "low_limit": 2800000000.0,
            "high_limit": 4800000000.0,
            "order": 3
        },
        "baseline_subtraction": {
            "bs_type": "variable",
            "apply_range": [
                2.06e-05,
                2.9999999999999997e-05
            ],
            "block_length": 6.2499999999999995e-06
        },
        "adaptive_filter": null,
        "time_of_arrival": null,
        "autodetect_signal_start": false
    },
    "transform": {
        "transform_type": "STFT",
        "window": "blackmanharris",
        "twindow": 5.0000000000000004e-08,
        "nperseg": 2500,
        "overlap_ratio": 0.75,
        "noverlap": 1875,
        "offset": false
    },
    "postprocess_pipeline": {
        "centroid": {
            "method": "max"
        },
        "ema_filter": {
            "window": 5,
            "order": 1
        },
        "median_filter": null,
        "ema_apply_order": 1,
        "median_apply_order": 0
    },
```

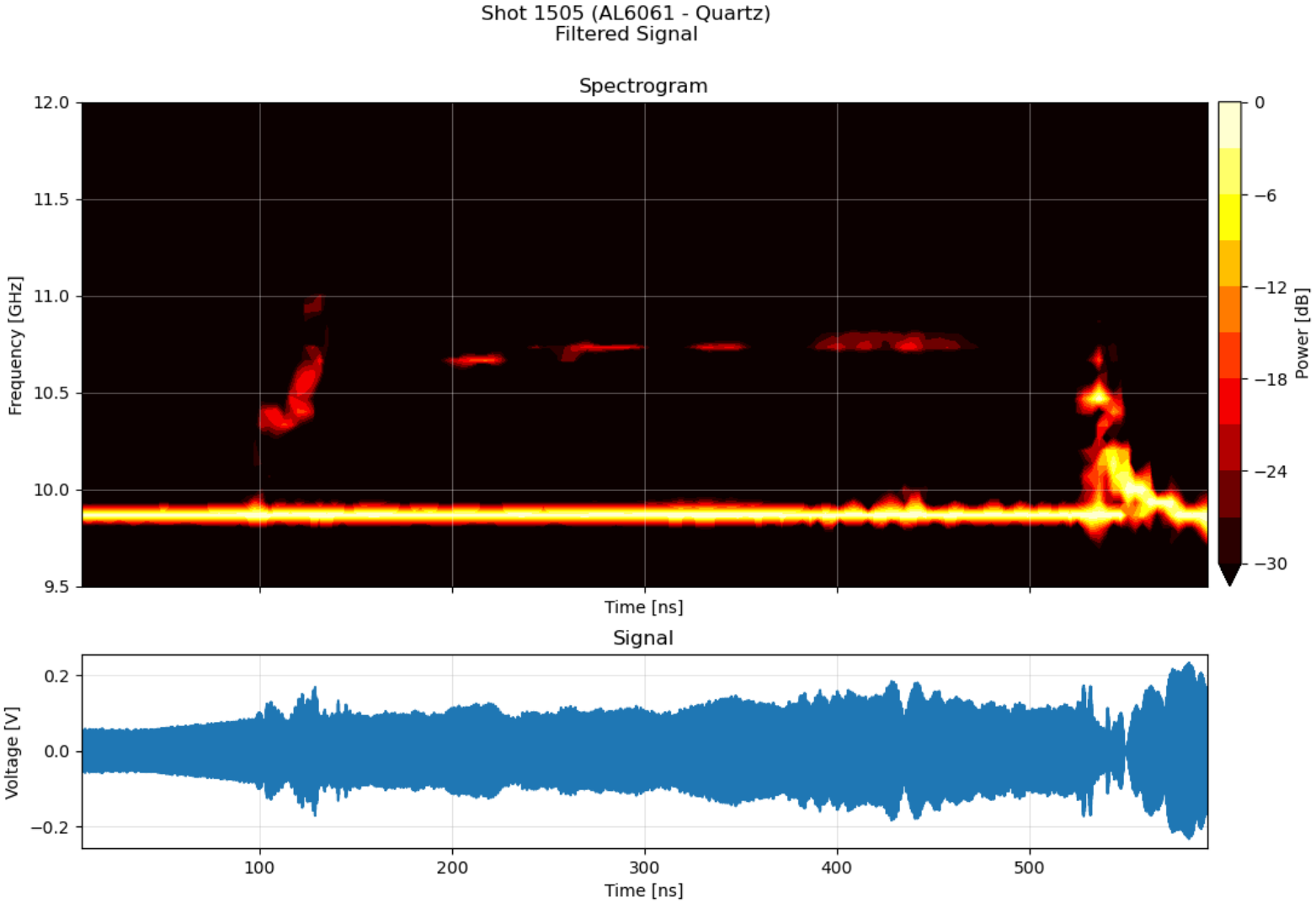
Analysis

- Binary (HDF5)
- Stores full history:
  - Signals (pre- and post- Transform)
  - Frequency & Velocity Traces
  - Protocol
  - Metadata
  - Calculated Values

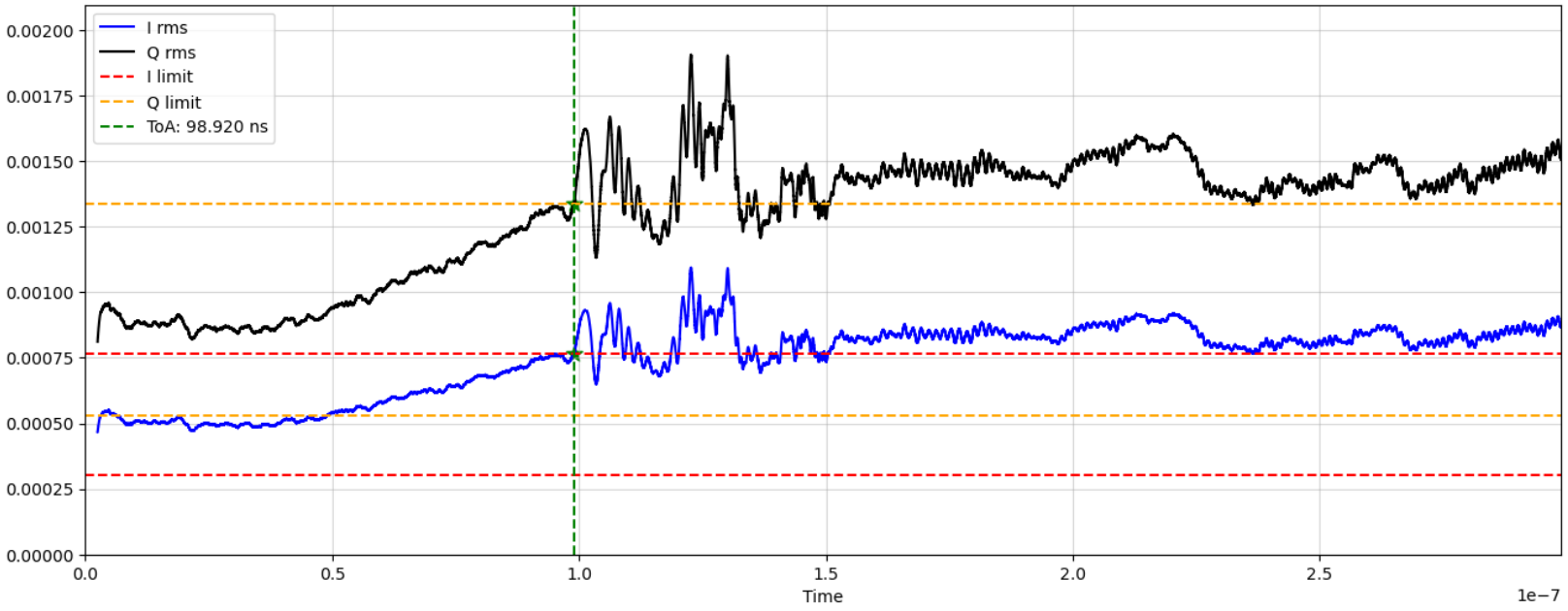
```
Analysis
        metadata:
                time_of_arrival
        protocol:
                protocol
        signals:
                raw
                truncated
                static_filtered
                baseline_subtracted
                adaptive_filtered
                processed
        transformed_signals:
                raw
                truncated
                static_filtered
                baseline_subtracted
                adaptive_filtered
                processed
        traces:
                raw_centroid
                truncated_centroid
                static_filtered_centroid
                baseline_subtracted_centroid
                adaptive_filtered_centroid
                processed_centroid
                postprocessed_ema
                postprocessed_med
                final
```

BONUS – LIBRARY

Time of Arrival Detection (2)

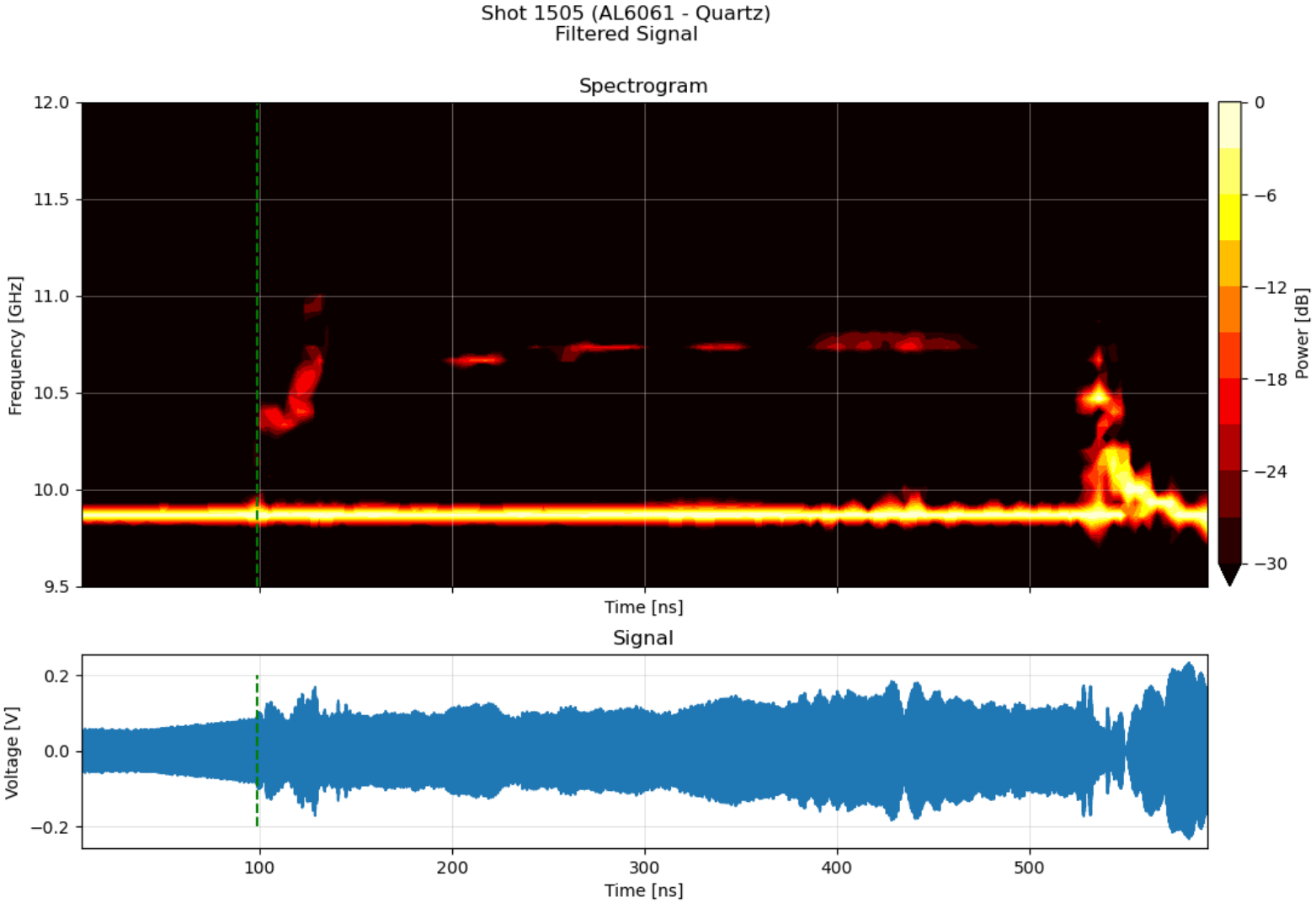


Time of Arrival Detection (3)

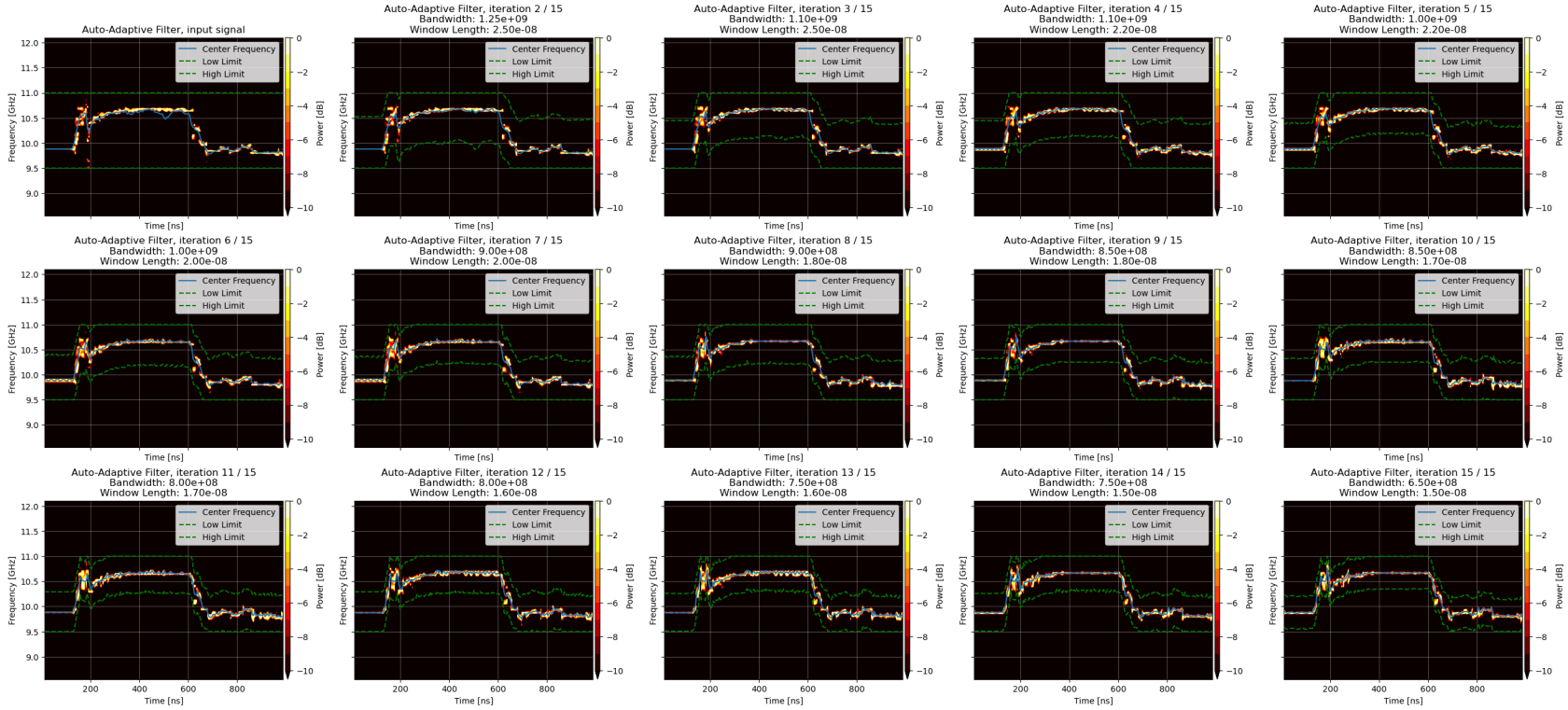



```
toa, ax = detect_signal_start_phase(
    filtered_1505,
    training_range=np.array([0, 75]) * unit.nanosecond,
    window_length_pts=250,
    sigma_threshhold=5.0,
    plot=True
)
```

Time of Arrival Detection (4)



Auto-Adaptive Filter




```
aaf = AutoAdaptiveFilter(
    absolute_limits=np.array([9.5, 11.0]) * unit.gigahertz,
    bandwidths=np.array([1.25, 1.10, 1.0, 0.90, 0.85, 0.80, 0.75, 0.65]) * unit.gigahertz,
    transform_cls=StftTransform,
    window_lengths=np.array([25, 22, 20, 18, 17, 16, 15]) * unit.nanoseconds,
    overlap_ratio=0.75,
)
aaf_1394 = aaf.apply(vbs_1394)
aaf_1394_stft = temp_stft.apply(aaf_1394)
aaf_result = aaf.process(vbs_1394)
axs = plot_aaf_sequence(aaf_result, ncols=5)

aaf_1394_centroid = max_centroid.apply(aaf_1394_stft)
```

Pipeline – Batch

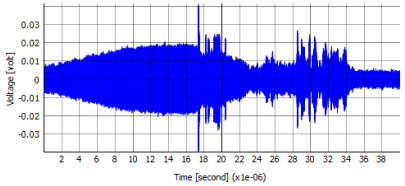
```
>>> analysis_a = protocol.analyze(signal_a)
>>> analysis_b = protocol.analyze(signal_b)
>>> analysis_c = protocol.analyze(signal_c)
```

Protocol

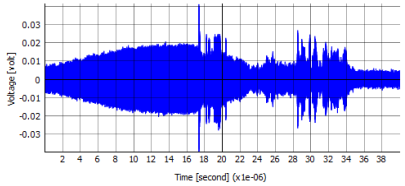
- Truncation
- Butterworth Filter
- STAR Transform
- Post-EMA Filter

Analyze()

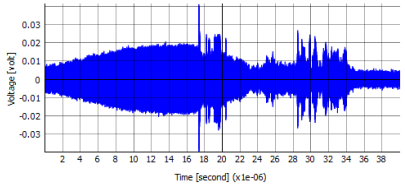
Signal A



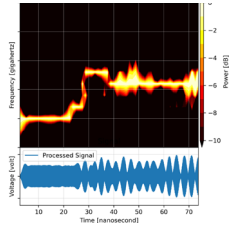
Signal B



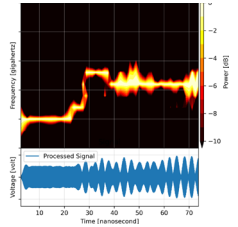
Signal C



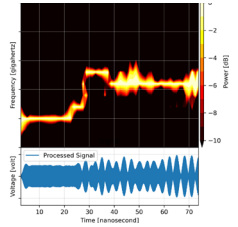
Analysis A



Analysis B



Analysis C



Pipeline – UQ

```
>>> analysis_1a = protocol_1.analyze(signal_a)
>>> analysis_2a = protocol_2.analyze(signal_a)
>>> analysis_3a = protocol_3.analyze(signal_a)
>>> analysis_4a = protocol_4.analyze(signal_a)
```

Analyze()

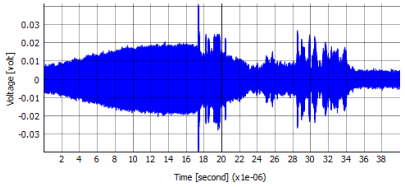
Protocol 1
- 5ns STFT window

Protocol 2
- 10ns STFT window

Protocol 3
- 15ns STFT window

Protocol 4
- 20ns STFT window

Signal A



Analysis 1A

Analysis 2A

Analysis 3A

Analysis 4A

Units

- Plot Labels
- Input Values
- Global Preferences

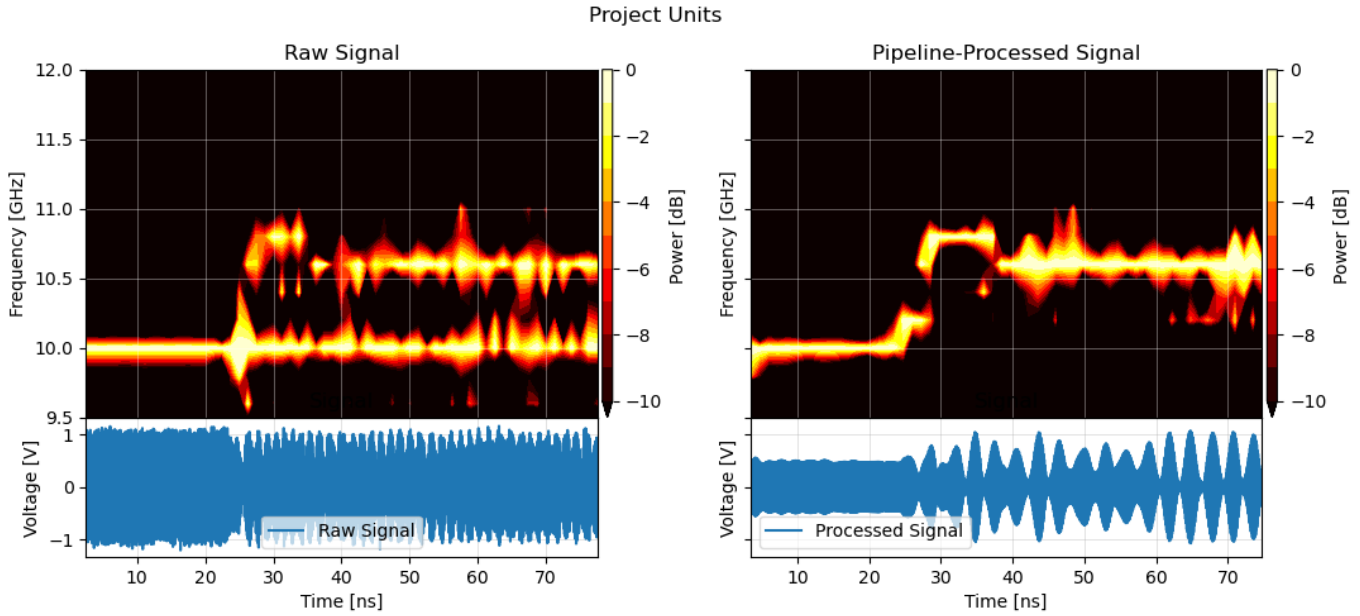
All numeric data within `Spectra` classes is stored in implicit MKS units (meter, kilogram, second).

```
>>> raw_signal.t0 = 10 * unit.nanosecond
>>> print(raw_signal.t0)  # Will print value in 'seconds'
1e-08
>>> raw_signal.pdv_wavelength_ref = 1550. * unit.nanometer
>>> print(raw_signal.pdv_wavelength_ref)  # Will print value in 'meters'
1.5500000000000002e-06
>>>print(raw_signal.detect_range)  # Will print value in 'seconds'
(0.0, 2e-08)
```

Convert implicit MKS units to desired units explicitly

```
>>> wavelength_quantity = raw_signal.pdv_wavelength_ref * unit.meter
>>> print(wavelength_quantity)
1.5500000000000002e-06 meter
>>> print(wavelength_quantity.to("micron"))
1.5500000000000003 micron
```

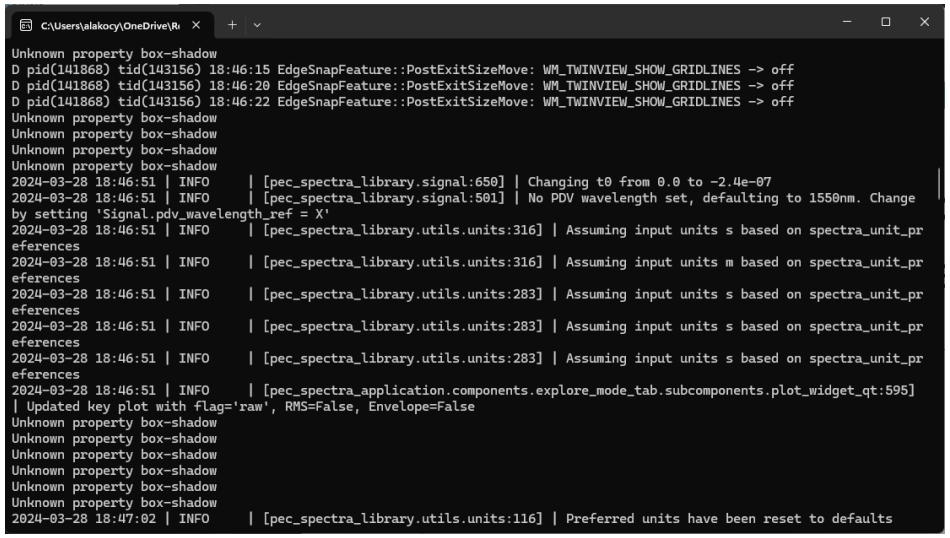
```
spectra_unit_preferences.set(time_unit="nanosecond", frequency_unit="gigahertz")
```



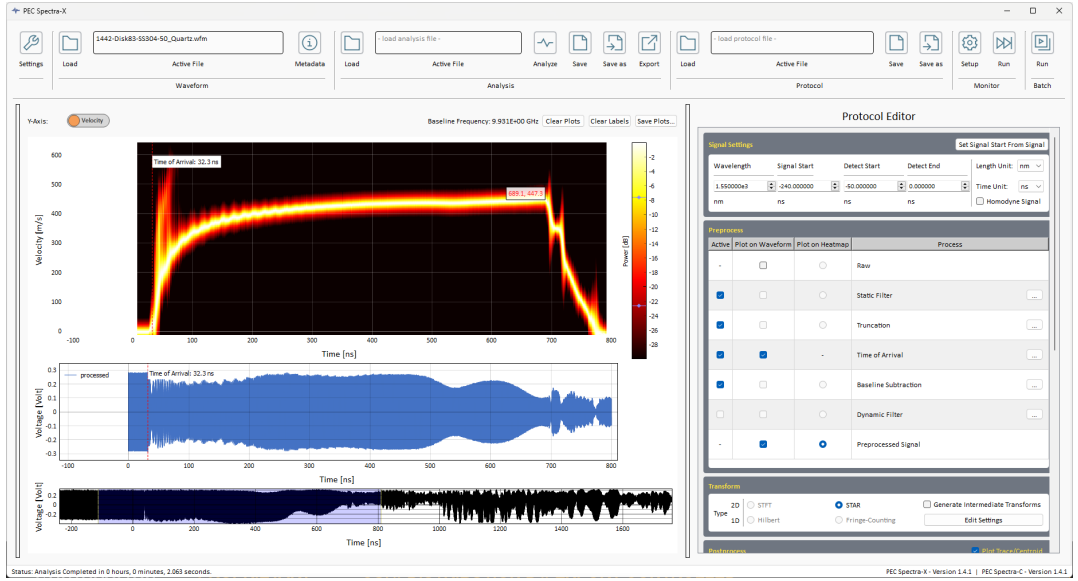
BONUS – APPLICATION

Application

- Requires: Windows 7 or later
- Bundled executable (.exe)
- Starting Application opens Console and Application windows

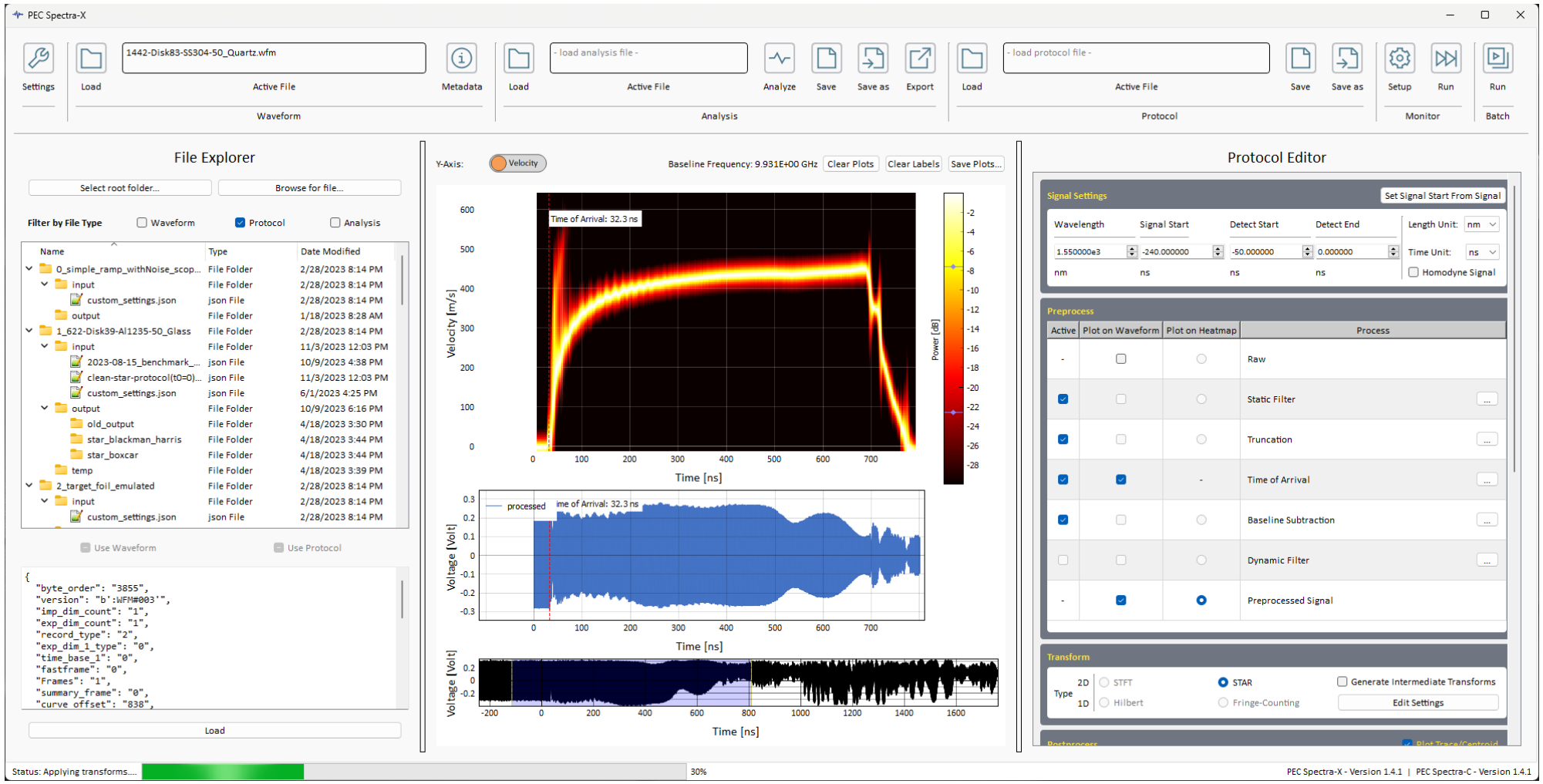


Console Window



Application Window

Application Window



Protection Engineering

CONSULTANTS

Approved for PDV Workshop 2024