pyOSA

Thorlabs

Apr 24, 2024

CONTENTS

1 Introduction

pyOSA is intended for controlling Thorlabs OSA20X and OSA30X from python, and for loading and saving Thorlabs .spf2 files. This is an SDK for ThorSpectra, and it will not work without it.

2 Quickstart

2.1 Installation

You will need an installation of ThorSpectra, compatible with the pyOSA version. Please make sure you have NumPy available for your python setup.

To install pyOSA You can either place the pyOSA directory in the directory of your python script, or you can install it with pip. To install with pip go to the folder with setup.py and then run:

```
pip install .
```

2.2 Simplest example

Reading a spectrum from an OSA

```
import pyOSA
# Get the connection to the OSA
osa = pyOSA.initialize()
# In setup() you can e.g. resolution and sensitivity
osa.setup()

acquisitions = osa.acquire(number_of_acquisitions=3)

# We have measured 3 acquisitions, but here we only use the last
acquisition = acquisitions[-1]
spectrum = acquisition["spectrum"]

# Now the x- and y-values can be obtained from spectrum.
x = spectrum.get_x()
y = spectrum.get_y()

# Getting the labels for the different axes
xlabel = spectrum.get_xlabel()
ylabel = spectrum.get_ylabel()

print(xlabel, x)
print(ylabel, y)
```

2.3 spectrum_t datastructure

ThorSpectra and pyOSA uses the spectrum_t datastructure for storing spectra and interferograms. Common things you will use are:

```
spectrum.convert_spectrum(x_unit="nm (vac)", y_unit="mW")
x = spectrum.get_x()
y = spectrum.get_y()
dateobj = spectrum.get_datetime()
```

3 Advanced Functionality

3.1 Reading an interferogram

You can change the simple example above to also acquire the interferograms

```
acquisitions = osa.acquire(number_of_acquisitions=3,
                                        spectrum=True,
                                        interferogram=True)
```

Then accessing the data with by index, where the 0 indicates the first measurement out of 3.

```
acquisition = acquisitions[0]
spectrum = acquisition["spectrum"]
interferogram = acquisition["interferogram"]
```

you may consider looping over the acquisitions

```
for acquisition in acquisitions:
        spectrum = acquisition["spectrum"]
        interferogram = acquisition["interferogram"]
```

3.2 Accessing data from different detectors

The OSA30X series have two different detectors, by default `acquistion["spectrum"]` returns the Stitched spectrum. However for interferograms it is necessary to specify from what detector to get the data. In the OSA30X you can access the data in the followinw ways:

```
acquisition["spectrum"] # which returns the same as Stitched
acquisition["spectrum", "Stitched"]
acquisition["spectrum", "Detector 1"]
acquisition["spectrum", "Detector 2"]
acquisition["interferogram", "Detector 1"]
acquisition["interferogram", "Detector 2"]
```

3.3 Continuous acquisition

Rather than waiting for the 3 measurements to finish before processing the data we can use "acquire_continuous"

```
for acquisition in osa.acquire_continuous(number_of_acquisitions=3,
                                          spectrum=True,
                                          interferogram=True):
        # Handle data in the loop
        spectrum = acquisition["spectrum"]
```

By setting number_of_acquisitions=-1, or omitting the keyword, the loop will continue forever, or until you set

```
osa.stop = True
```

It is important to stop a continuous acquisition cleanly, otherwise it might be necessary to restart the OSA instrument.

Beware that the operations handling the data needs to be quicker than the measurement, otherwise the readout will lag behind.

3.4 Checking validity of measurement

When writing a console scripts you will be notified by text if there are problems with the data quality due to any of the follwing reasons:

1. The reference laser was locked during the measurement
2. The interferogram signal was not clipped (also checked for a spectrum)
3. The interferogram was not nonlinear (tested for OSA205, OSA207, and Redstone305)
4. Only for OSA200: The optimal gain was used during the measurement (only tested if autogain is enabled)

However if you are running headless with a script or GUI, it is of very high importance to handle this check. You can perform the check like this:

```
spectrum = acquisition["spectrum"]
if not spectrum.is_valid():
        print("There are issues with the data quality of this measurement")
```

or to handle it in a more detailed way:

```
validity = spectrum.check_validity()
# Handle each different validity state
if not validity["ref_laser_locked"]:
        print("Reference laser not locked")
if not validity["interferogram_within_detector_range"]:
        print("Interferogram is clipped")
if not validity["interferogram_is_linear"]:
        print("Interferogram is non-linear")
# Only available for OSA20X
if not validity["autogain_satisfied"]:
        print("Autogain was not finished adjusting")
```

The method is also available to use for measured interferograms:

```
interferogram = acquisition["interferogram", "Detector 1"]
if not interferogram.is_valid():
        raise Exception("Problem with data quality")
```

3.5 Problems loading FTSLib.dll

In case FTSlib cannot find FTSLib.dll, you might have to edit pyOSA/FTSLib.py and point it to the .dll Set the last row to, for example:

```
FTSLib = load_dll(r"C:\SomeFolder\FTSLib.dll")
```

3.6 Reference warmup exception

If your OSA takes long time to lock its reference laser, you can force pyOSA to let you perform measurements anyway. When acquiring include the following parameter

```
acquisition = osa.acquire(ignore_errors=["Reference Warmup"])
```

Beware that this affect your data quality.

3.7 Testing with a virtual OSA

For testing and development purposes a virtual OSA can instead be used by changing the line `osa = pyOSA.initialize()`. For a virtual Redstone change it to

```
pyOSA.create_virtual_Redstone(spectrometer_index=0, model='Redstone305', source_type=0)
osa = pyOSA.initialize(virtual_nr=1)
```

and for a virtual OSA20X to

```
pyOSA.create_virtual_OSA20X(spectrometer_index=0,
                  model='OSA201C',
                  source_type=0,
                  peak_num=1,
                  centerWavelengths_nm=[650],
                  FWHMs_nm=[1],
                  peak_amplitudes=[1])
osa = pyOSA.initialize(virtual_nr=1)
```

3.8 Loading and saving .spf2 files

.spf2 files can be loaded with:

```
measurements = pyOSA.core.load_spf2_file(filename)
```

This will return a list of spectrum_t datasets in the file.

A single spectrum_t or a list with spectrum_t can be saved to a .spf2 file with

```
pyOSA.core.save_spf2_file(spectra, filename)
```

3.9 Multiple OSAs

You can control and acquire from multiple OSAs, change your initializtion to

osas will now be a list of instruments.

4 Analysis

pyOSA offers Python access to some analysis functions within ThorSpectra.

4.1 For Spectra:

- **Peak Track** finds and measures peak heights and widths within a spectrum.
- **Valley Track** measures valley depths and widths in a spectrum.
- **Optical Power Measurement** determines the optical power within a specified region of a spectrum.

4.2 For Interferograms:

- **Wavemeter Analysis** of an interferogram to extract wavelength data.
- **Coherence Length** measurement the coherence length of the interferogram.

5 Examples

The attached examples are:

- Example 1 - Measure a spectrum
- Example 2 - Measure spectra using all different sensitivities and resolutions, save them to files
- Example 3 - Load a spectrum or interferogram from an .spf2 file, perform wavemeter analysis on interferogram, or peak track on spectrum
- Example 4 - Continuous acquisition with wavemeter and peak detection. Displayed live using matplotlib
- Example 5 - Continuous peak track, plotting peakposition over time

6 Units

The available units are:

x_units:

```
"cm", "cm^-1", "THz", "nm (vac)", "nm (air)", "eV," "Index", "Seconds", "Pixel", "Hz",
↪"cm^-1 (Raman)"
```

y_units:

```
"a.u.", "Counts", "dBm", "dBm (norm)", "mW", "mW (norm)", "Percent", "Radians", "Degrees",
↪ "Celsius", "Kelvin", "hPa", "Log", "dB", "Pp", "Mixed", "Intensity", "Logintensity",
↪"Cal I", "Cal Logi", "Atmosphere", "Torr", "Psi", "Log Counts", "Cm2 Molec", "Molec Cm2
↪", "Grams Cm2", "Molec Cm3", "Grams Cm3", "Mol Dm3", "Transmittance"
```

7 API Documentation

8 pyOSA core

pyOSA.core()	pyOSA core, takes care of instrument parameters, initialization of instruments and loading/saving of spf2 files

8.1 pyOSA.core

class pyOSA.**core**

pyOSA core, takes care of instrument parameters, initialization of instruments and loading/saving of spf2 files

__init__()

Methods

__init__()	
close(osa)	Closes the connection to the OSA.
create_virtual_OSA20X(spectrometer_index, ...)	Creates a virtual OSA20X instrument with a specified virtual source.
create_virtual_Redstone(spectrometer_index, ...)	Creates OSA30X Redstone virtual instrument.
initialize([virtual_nr, multiple])	Initializes and opens a connection to a single or all OSAs connected to the system.
load_spf2_file(filename)	Loads an spf2 file containing spectra or interferograms.
save_spf2_file(specs, filename)	Saves a list of spectra/interferograms, or one single spectrum/interferogram, to file.

class pyOSA.**core**

pyOSA core, takes care of instrument parameters, initialization of instruments and loading/saving of spf2 files

static close(*osa:* Instrument)

Closes the connection to the OSA.

Parameters:
osa (Instrument): The OSA instrument to close the connection for.

Raises:
RuntimeError: If an error occurs while closing the connection.

static create_virtual_OSA20X(*spectrometer_index: int*, *model: str*, *source_type: int*, *peak_num: int*, *centerWavelengths_nm: List[float]*, *FWHMs_nm: List[float]*, *peak_amplitudes: List[float]*) → None

Creates a virtual OSA20X instrument with a specified virtual source.

Inputs:
spectrometer_index (int): Spectrometer index of the created instrument model (String): String specifying instrument model, eg 'OSA201C', 'OSA203B'... source_type (int): Either monochromatic (0) or broadband (1) peak_num (int): Number of peaks, must be less than VIRTUAL_SOURCE_MAX_PEAK_NUM centerWavelengths_nm (list(doubles)): List

of center wavelengths, peak_num elements FWHMs_nm (list(doubles)): List of Full Width at Half Maximum for each peak, peak_num elements peak_amplitudes (list(doubles)): List of each peak's amplitude in mW, peak_num elements

static create_virtual_Redstone(*spectrometer_index: int*, *model: str*, *source_type: int*) → None

Creates OSA30X Redstone virtual instrument.

Inputs:
spectrometer_index (int): Spectrometer index of the created instrument model (String): String specifying instrument model, eg 'REDSTONE305' source_type (int): Indicates what virtual source is created:

0: Laser emitting at 1530 nm 1: Superluminescent diode (SLD) with center wavelength at 1550 nm 2: Laser emitting at 4600 nm 3: Blackbody with a temperature of 1500 K

classmethod initialize(*virtual_nr: int = 0*, *multiple: bool = False*) → Union[*Instrument*, List[*Instrument*]]

Initializes and opens a connection to a single or all OSAs connected to the system.

Parameters:

virtual_nr (int, default 0): Indicates how many virtual machines are in use.
Note that virtual and real machines can't be used concurrently.

multiple (bool, default False): If True, initializes connections to all available
OSAs. If False, initializes connection to a single OSA.

Returns:

Union[Instrument, List[Instrument]]: Either a single OSA_instrument instance or
a list of OSA_instrument instances opened and initialized.

static load_spf2_file(*filename: str*) → List[*spectrum_t*]

Loads an spf2 file containing spectra or interferograms.

Parameters:
filename (str): The filename, including path, to the file to load.

Returns:
list of spectrum_t: A list of spectra if filename is given.

static save_spf2_file(*specs: Union[*spectrum_t*, List[*spectrum_t*], Dict[str,* spectrum_t*]]*, *filename: str*)

Saves a list of spectra/interferograms, or one single spectrum/interferogram, to file.

Inputs:
specs (List(spectrum_t) | spectrum_t): List of spectrum_t structs to save, or a single spectrum_t filename (String): Full file path with file name

9 pyOSA instrument

pyOSA.instrument.Instrument(spectrometer_index)	This class serves as a representation of an Optical Spectrum Analyzer (OSA) instrument.

9.1 pyOSA.instrument.Instrument

class pyOSA.instrument.**Instrument**(*spectrometer_index: int*, *virtual: bool = False*)

This class serves as a representation of an Optical Spectrum Analyzer (OSA) instrument. It provides users with the capability to configure instrument settings and retrieve data.

__init__(*spectrometer_index: int*, *virtual: bool = False*)

Methods

__init__(spectrometer_index[, virtual])	
acquire([spectrum, interferogram, ...])	Acquire spectra and/or interferogram, returns after all measurements are complete.
acquire_continuous([spectrum, ...])	Acquire spectra and/or interferograms, yielding data continuously.
get_apodization()	Returns the used apodization setting
get_attenuation_filter(detector)	Retrieve information about the attenuation filter for a given detector channel.
get_autogain()	Retrieve the current autogain setting for the spectrometer.
get_available_gain_levels([detectors])	Returns the available gain levels (depends on instrument and sensitivity setting)
get_available_resolutions()	Returns the available resolutions (depends on instrument)
get_available_sensitivities()	Returns the available sensitivities (depends on instrument)
get_detector_offsets([detectors])	Returns the detector offsets for each specified detector
get_formatted_detector_range(detector_name ...)	Returns a string containing the range of the detector:
get_formatted_model_range(desired_unit)	Returns a formatted string of the range of the model
get_gain_level([detectors])	Returns the used gain level setting for the supplied detectors For Redstone, this depends on which channel/detector is used
get_model()	Returns the model of the instrument
get_resolution()	Returns the used resolution setting
get_sensitivity()	Returns the used sensitivity setting
get_serial_number()	Returns the serial number of the instrument
get_zerofill()	Returns the used zerofill setting
is_OSA200()	Returns true if the instrument belongs to the OSA200 series
is_Redstone()	Returns true if the instrument belongs to the Redstone series
set_apodization(apodization)	Sets the apodization setting
set_attenuation_filter(detector[, active, ...])	Set the attenuation filter state for a specified detector channel.
set_autogain([autogain])	Sets whether autogain should be used or not.
set_detector_offsets(detector_offsets[, ...])	Set the detector offset for Redstone detectors
set_gain_level(gain_level[, detectors])	Sets the used gain level
set_resolution(resolution)	Sets the resolution setting
set_sensitivity(sensitivity)	Sets the sensitivity setting
set_zerofill(zerofill)	Sets the zerofill setting
setup([autosetup, autogain, resolution, ...])	Applies chosen settings to an instrument, or runs auto-setup.

class pyOSA.**Instrument**(*spectrometer_index: int*, *virtual: bool = False*)

This class serves as a representation of an Optical Spectrum Analyzer (OSA) instrument. It provides users with the capability to configure instrument settings and retrieve data.

acquire(*spectrum: bool = True*, *interferogram: bool = False*, *number_of_acquisitions: int = 1*, *spectrum_averaging: int = 1*, *zerofill: int = 0*, *apodization: str = 'Hann'*, *ignore_errors: List[str] = []*, *power_type='Absolute power'*, *x_unit='nm (vac)'*, *y_unit='mW'*, ***kwargs*) → List[TupleDict]

Acquire spectra and/or interferogram, returns after all measurements are complete.

Parameters:

spectrum (bool, default True):
Indicates if spectra should be returned.

interferogram (bool, default False):
Indicates if interferograms should be returned.

number_of_acquisitions (int, default 1):
Indicates the number of acquisitions. -1 will start an infinite acquisition (stopped by setting osa.stop = True).

spectrum_averaging (int, Default: 1):
Sets the number of spectra to average over when retrieving a spectrum.

zerofill (int, Default: 0):
Sets the zero-fill factor. Valid values are 0, 1, 2

apodization (str, Default: "Hann"):
Sets the apodization type. Valid values are listed below.

- "None"
- "Norton beer weak"
- "Norton beer medium"
- "Norton beer strong"
- "Triangular"
- "Cosine"
- "Hann"
- "Hamming"
- "Blackmanharris3"
- "Blackmanharris4"
- "Gaussian"
- "Two pass hann"

ignore_errors: (list[str], default False):
List of errors to ignore, for examples ["Reference Warmup"] to ignore cold reference laser.

power_type (str, default "Absolute power"):
The spectrum intensity can be either "Absolute power" or "Power Density".

x_unit (str, default "nm (vac)"):
The x unit for the spectrum. Refer to the manual for possible units.

y_unit (str, default "mW"):
The y unit for the spectrum. Refer to the manual for possible units.

Returns:

Returns a list of acquisitions, where each acquisition is a TupleDict.

Possible keys for OSA200:

- acquisition["spectrum"]
- acquisition["interferogram"]

Possible keys for Redstone:

- acquisition["spectrum"] which returns the same as Stitched
- acquisition["spectrum", "Stitched"]
- acquisition["spectrum", "Detector 1"]
- acquisition["spectrum", "Detector 2"]
- acquisition["interferogram", "Detector 1"]
- acquisition["interferogram", "Detector 2"]

Each value in the TupleDict is a spectrum_t

acquire_continuous(*spectrum: bool = True*, *interferogram: bool = False*, *number_of_acquisitions: int = -1*, *spectrum_averaging: int = 1*, *zerofill: int = 0*, *apodization: str = 'Hann'*, *ignore_errors: List[str] = []*, *power_type='Absolute power'*, *x_unit='nm (vac)'*, *y_unit='mW'*, ***kwargs*) → Iterator[TupleDict]

Acquire spectra and/or interferograms, yielding data continuously.

To stop the acquistion set osa.stop = True in your code

get_apodization() → str

Returns the used apodization setting

get_attenuation_filter(*detector: str*) → Tuple[bool, bool, bool, int]

Retrieve information about the attenuation filter for a given detector channel.

Parameters:

detector (str):
The name of the detector for which to retrieve attenuation filter information.

Returns:

tuple[bool, bool, bool, str]:
A tuple containing the following information:

- **available (bool):**
  True if the attenuation filter is available for the specified detector channel.
- **active (bool):**
  True if the attenuation filter is currently active for the specified detector channel.
- **automatic (bool):**
  True if the attenuation filter is set to automatic mode for the specified detector channel.
- **bits (int):**
  A binary representation of the attenuation filter status for additional details.

get_autogain() → int

Retrieve the current autogain setting for the spectrometer.

Returns:

int:
The current autogain setting used by the spectrometer.

get_available_gain_levels(*detectors: Optional[Tuple[str]] = None*) → List[List[float]]

Returns the available gain levels (depends on instrument and sensitivity setting)

For Redstone, this depends also on which channel/detector is used It will return a list of gain_levels, with one list for each supplied detector

Args:
detectors (Tuple[str], optional): Detector names. Defaults based on instrument.

Returns:
List[List[float]]: Gain levels for each specified detector.

get_available_resolutions() → List[str]

Returns the available resolutions (depends on instrument)

get_available_sensitivities() → List[str]

Returns the available sensitivities (depends on instrument)

get_detector_offsets(*detectors: Optional[Tuple[str]] = None*) → List[int]

Returns the detector offsets for each specified detector

If no detector is specified then it will be returned for all

get_formatted_detector_range(*detector_name: str*, *desired_unit: Union[int, str]*) → str

Returns a string containing the range of the detector:

The format is: x_min-x_max x_unit, e.g., '1000-2500 nm' desired_unit can be given either in the indices or the corresponding names found in OSA_constants.x_units. Only the spectral units are allowed.

get_formatted_model_range(*desired_unit: Union[int, str]*) → str

Returns a formatted string of the range of the model

The format is on
x_min-x_max x_unit, e.g., '1000-2500 nm'

desired_unit can be given either in the indices or the corresponding names found in OSA_constants.x_units. Only the spectral units are allowed.

get_gain_level(*detectors: Optional[Tuple[str]] = None*) → List[int]

Returns the used gain level setting for the supplied detectors For Redstone, this depends on which channel/detector is used

get_model() → str

Returns the model of the instrument

get_resolution() → str

Returns the used resolution setting

get_sensitivity() → str

Returns the used sensitivity setting

get_serial_number() → str

Returns the serial number of the instrument

get_zerofill() → int

Returns the used zerofill setting

is_OSA200() → bool

Returns true if the instrument belongs to the OSA200 series

is_Redstone() → bool

Returns true if the instrument belongs to the Redstone series

set_apodization(*apodization: str*) → None

Sets the apodization setting

set_attenuation_filter(*detector: str*, *active: bool = False*, *automatic: bool = False*) → None

Set the attenuation filter state for a specified detector channel.

Parameters:

detector (str):
The name of the detector for which to set the attenuation filter.

active (bool):
True to activate the attenuation filter, False to deactivate.

automatic (bool):
True to set the attenuation filter to automatic mode, False for manual mode.

set_autogain(*autogain: bool = True*)

Sets whether autogain should be used or not.

Parameters:

autogain (bool, optional):
If True, autogain is enabled; if False, autogain is disabled. Defaults to True.

set_detector_offsets(*detector_offsets: List[int]*, *detectors: Optional[Tuple[str]] = None*) → None

Set the detector offset for Redstone detectors

Example:

```
osa.set_detector_offsets([1472,1472], ["Detector 1", "Detector 2"])
```

set_gain_level(*gain_level: int*, *detectors: Optional[Tuple[str]] = None*)

Sets the used gain level

For an OSA200, detector can be omitted since the OSA200 only has one detector For a Redstone, this function needs to be called for each detector/channel or with multiple detectors specified as argument

set_resolution(*resolution: str*)

Sets the resolution setting

set_sensitivity(*sensitivity: str*)

Sets the sensitivity setting

set_zerofill(*zerofill: int*)

Sets the zerofill setting

setup(*autosetup: bool = False*, *autogain: bool = True*, *resolution: str = 'low'*, *sensitivity: str = 'low'*) → None

Applies chosen settings to an instrument, or runs auto-setup.

Parameters:

autosetup (bool):
True if auto-setup should be run, False for manual setup. Auto-setup ignores all manual settings.

autogain (bool, Default: True):
If True, activates auto-gain for all channels.

resolution (str, Default: "low"):
Sets the resolution to the input value. Valid values are 'low', 'high' for OSA20X, and 'low', 'medium low', 'medium high', 'high' for OSA30X.

sensitivity (str, Default: "low"):
Sets the sensitivity to the input value. Valid values are 'low', 'medium low', 'medium high', 'high' for OSA20X, and 'low', 'medium', 'high' for OSA30X.

10 pyOSA analysis

pyOSA.analysis()	Collection of methods from FTSLib used to do analysis on interferograms and spectra

10.1 pyOSA.analysis

class pyOSA.**analysis**

Collection of methods from FTSLib used to do analysis on interferograms and spectra

__init__()

Methods

__init__()	
coherence(interferogram)	Measure the coherence length of an interferogram.
optical_power(spectrum[, ...])	Measure the optical power in part of a spectrum.
peak_track(spectrum[, start_index, ...])	Find and measure parameters of peaks in a spectrum.
valley_track(spectrum[, start_index, ...])	Find Valleys and their data from a spectrum.
wavemeter(interferogram[, spectral_unit])	Perform wavemeter analysis on an interferogram.

class pyOSA.**analysis**

Collection of methods from FTSLib used to do analysis on interferograms and spectra

static coherence(*interferogram:* spectrum_t) → Dict[str, float]

Measure the coherence length of an interferogram.

Returns coherence length data for an interferogram.

Parameters:

interferogram (spectrum_t):
Interferogram of type spectrum_t.

Returns:

dict:

A dictionary containing coherence data with the following keys:

- "coherence": The coherence length.
- "error": Error associated with the coherence measurement.

Example:

```
igram = acquisition["interferogram", "Detector 2"]
coherence_data = pyOSA.analysis.coherence(igram)
coherence_length = coherence_data["coherence"]
```

static optical_power(*spectrum:* spectrum_t, *optical_power_mode: int = 0*, *x_min: Optional[float] = None*, *x_max: Optional[float] = None*, *threshold=None*) → float

Measure the optical power in part of a spectrum.

Returns the optical power of the given spectrum according to the optical power mode.

Parameters:

spectrum (spectrum_t):
The spectrum for which to calculate the optical power.

optical_power_mode (int, optional):

Specifies how the optical power is calculated:

- 0: Measure over the entire spectrum.
- 1: Measure around the largest peak.
- 2: Manual mode, requires x_min and x_max inputs.

x_min (float, optional):
Starting point for calculating optical power. Should be provided in the same unit as the spectrum. Only applies if optical power mode is 2 (manual mode).

x_max (float, optional):
Stopping point for calculating optical power. Should be provided in the same unit as the spectrum. Only applies if optical power mode is 2 (manual mode).

Returns:

float:
The calculated optical power.

Example:

```
spectrum = acquisition["spectrum"]
power = pyOSA.analysis.optical_power(spectrum,
                                     optical_power_mode=2,
                                     x_min=1400,
                                     x_max=1600)
```

static peak_track(*spectrum:* spectrum_t, *start_index: int = 0*, *stop_index: Optional[int] = None*, *threshold: Optional[float] = None*, *min_peak_height_db: float = 3*, *sort_option: int = 1*, *max_peaks: int = 20*) → ndarray

Find and measure parameters of peaks in a spectrum.

Returns the number of peaks and peak data.

Parameters:

spectrum (spectrum_t):
Spectrum to read peaks from.

start_index (int, Default: 0):
Indicates the starting index for the peak search. Default 0 is the beginning of the spectrum.

stop_index (int, Default: None):
Indicates the stopping index for the peak search. Default None means that the spectrum length is used.

threshold (float, Default: None):
Sets the threshold for peak searching. Peaks below the threshold are not returned. Default None sets threshold to 0.1 for y-unit mW and -10 for y-unit dBm.

min_peak_height_db (float, Default: 3):
Sets the minimum height for a peak to be tracked, in dB.

sort_option (int, Default: 1):
Sets how the peaks are sorted along the rows in peak_data.

- 0: No sorting,
- 1: Sorted in order of increasing center position,
- 2: Sorted in order of increasing height,
- 3: Sorted in order of increasing width.

max_peaks (int, Default: 20):
The maximum number of peaks filled into the array.

Returns:
np.ndarray:

- The peak data read from the spectrum, organized in data type per row, peak number per column.
- Row one: centroid values of peaks.
- Row two: peak heights.
- Row three: peak widths.
- Row four: first value left of peak where intensity has dropped min_peak_height_db.
- Row five: same for right side.

Example:

```
spectrum = acquisition["spectrum"]
peaks = pyOSA.analysis.peak_track(spectrum, threshold=1.5e-3, sort_option=2)
peak_centers, peak_heights, peak_widths, peak_lefts, peak_rights = peaks
print(peak_centers[0])
```

static valley_track(*spectrum:* spectrum_t, *start_index: int = 0*, *stop_index: Optional[int] = None*, *threshold: Optional[float] = None*, *min_valley_depth_db: float = 3*, *sort_option: int = 1*, *max_valleys: int = 20*) → ndarray

Find Valleys and their data from a spectrum.

Parameters:

spectrum (spectrum_t):
Spectrum to read valleys from.

start_index (int, Default: 0):
Indicates the starting index for the valley search. Default 0 is the beginning of the spectrum.

stop_index (int, Default: None):
Indicates the stopping index for the valley search. Default None means that the spectrum length is used.

threshold (float, Default: None):
Sets the threshold for valley searching. Valleys above the threshold are not returned. Default None sets threshold to 0.1 for y-unit mW and -10 for y-unit dBm.

min_valley_depth_db (float, Default: 3):
Sets the minimum depth for a valley to be tracked, in dB.

sort_option (int, Default: 1):
Sets how the valleys are sorted along the rows in valley_data:

- 0: No sorting,
- 1: Sorted in order of increasing center position,
- 2: Sorted in order of increasing depth,
- 3: Sorted in order of increasing width.

max_valleys (int, Default: 20):
The maximum number of valleys filled into the array.

Returns:

np.ndarray:
The valley data read from the spectrum, organized in data type per row, valley number per column.

- Row one: centroid values of valleys.
- Row two: valley depths.
- Row three: valley widths.
- Row four: first value left of valley where intensity has dropped min_valley_height_db.
- Row five: same for right side.

Example:

```
spectrum = acquisition["spectrum"]
valley_data = pyOSA.analysis.valley_track(spectrum,
                                          threshold=1.4,
                                          sort_option=1,
                                          max_valleys=4)
valley_centers = valley_data[0]
```

static wavemeter(*interferogram:* spectrum_t, *spectral_unit: str = 'nm (vac)'*) → Dict[str, float]

Perform wavemeter analysis on an interferogram.

Returns wavelength data for an interferogram.

Parameters:

interferogram (spectrum_t):
Interferogram of type spectrum_t.

spectral_unit (str, optional):
The unit of wavelength. Defaults to “nm (vac)”.

Returns:

dict:

The wavemeter data can be accessed as follows:

- wavemeter_dict[“wavelength”]
- wavemeter_dict[“error”]

Example:

```
igram = acquisition["interferogram"]
wavedata = pyOSA.analysis.wavemeter(igram)
igramstr = f"{wavedata['wavelength']:.4f} +- {wavedata['error']:.4f}"
```

11 spectrum_t

pyOSA.spectrum_t	Represents the internal data format used by ThorSpectra.

11.1 pyOSA.spectrum_t

class pyOSA.**spectrum_t**

Represents the internal data format used by ThorSpectra.

This Python object grants access to the internal attributes and offers some methods for convenient access.

__init__(**args*, ***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`check_validity`()	Returns a dictionary containing the "validity statuses" for the measured data.
`convert_spectrum`(x_unit, y_unit)	Converts the spectrum to specified x- and y-units.
`get_datetime`()	Returns a datetime object of when the data was acquired.
`get_gain_index`()	Returns the instrument detector gain index.
`get_gain_level`()	Returns the instrument detector gain.
`get_interferogram_signal_in_percent`([detec	Returns the interferogram signal (in percent) for the measured data
`get_model`()	Returns the instrument model used to measure the data as a string, e.g., 'OSA201C'
`get_resolution`()	Returns the instrument resolution.
`get_sensitivity`()	Returns the instrument sensitivity.
`get_serial_number`()	Returns the serial number of the instrument used to measure the data.
`get_x`()	Returns the x values from the measurement
`get_xlabel`()	Returns formatted xlabel e.g., 'Optical Frequency (THz)' or 'Wavelength (nm (vac))'
`get_y`()	Returns the intensity values from the measurement
`get_ylabel`()	Returns formatted ylabel e.g., 'Absolute Power (mW)' or 'Power Density (dBm/nm)'
`is_OSA200`()	Returns True if the measured data was captured using an OSA20X.
`is_Redstone`()	Returns True if the measured data was captured using a Redstone series spectrometer.
`is_autogain_satisfied`()	Returns True if the measured data was collected with the best possible gain setting.
`is_interferogram`()	Returns True if the measured data is an interferogram.
`is_spectrum`()	Returns True if the measured data is a spectrum.
`is_valid`()	Returns True if no problems were detected with the measured data.
`set_comment`(new_comment)	Set the comment of this spectrum_t.
`set_name`(new_name)	Set the name of this spectrum_t.
`set_operator`(new_operator)	Set the operator of this spectrum_t.

Attributes

`BBRefGainIndex`	Structure/Union member
`BBRefSignalInPercent`	Structure/Union member
`DoNotUse_usedToBeIsBroadband`	Structure/Union member
`I`	Structure/Union member
`UUIDTagClockSeqHiAndReserved_8bit`	Structure/Union member
`UUIDTagClockSeqLow_8bit`	Structure/Union member
`UUIDTagNode_48bit`	Structure/Union member
`UUIDTagTimeHiAndVer_16bit`	Structure/Union member

continues on next page

Table 1 – continued from previous page

UUIDTagTimeLow_32bit	Structure/Union member
UUIDTagTimeMid_16bit	Structure/Union member
acquisitionMode	Structure/Union member
adcBits	Structure/Union member
air_measureOption	Structure/Union member
air_press	Structure/Union member
air_relHum	Structure/Union member
air_temp	Structure/Union member
allocatedLengthI	Structure/Union member
allocatedLengthLog	Structure/Union member
allocatedLengthPhi	Structure/Union member
allocatedLengthx	Structure/Union member
amplifierSort	Structure/Union member
apodization	Structure/Union member
apodizationPowerCompensation	Structure/Union member
approximatePulseFrequency	Structure/Union member
approximateSamplesPerRefInRawIgram	Structure/Union member
attenuationFilterUsed	Structure/Union member
autogainStatus	Structure/Union member
averageNum	Structure/Union member
calculatedWithCUDA	Structure/Union member
calibCoeffNum	Structure/Union member
calibWaveNr	Structure/Union member
calib_Coeff	Structure/Union member
channelIndex	Structure/Union member
channelStatus	Structure/Union member
comment	Structure/Union member
commentLong	Structure/Union member
contrast	Structure/Union member
date	Structure/Union member
detectorCoolingState	Structure/Union member
detectorOffset	Structure/Union member
detectorSort	Structure/Union member
detectorType	Structure/Union member
detector_temperature_C	Structure/Union member
distanceFromStagePosAtoZPD_cm	Structure/Union member
dynamicRangeMedian_count	Structure/Union member
dynamicRangeUtil_count	Structure/Union member
dynamicRange_count	Structure/Union member
endPos_um	Structure/Union member
excitationWavelength_nm	Structure/Union member
external_pressure_hPa	Structure/Union member
external_relHum_percent	Structure/Union member
external_temperature_C	Structure/Union member
extra_for_temporary_debugging_double_1	Structure/Union member
extra_for_temporary_debugging_double_2	Structure/Union member
extra_for_temporary_debugging_double_3	Structure/Union member
extra_for_temporary_debugging_double_4	Structure/Union member
extra_for_temporary_debugging_double_5	Structure/Union member
extra_for_temporary_debugging_int_1	Structure/Union member
extra_for_temporary_debugging_int_2	Structure/Union member
extra_for_temporary_debugging_int_3	Structure/Union member
extra_for_temporary_debugging_int_4	Structure/Union member
extra_for_temporary_debugging_int_5	Structure/Union member

continues on next page

Table 1 – continued from previous page

firmwareVersion	Structure/Union member
firmwareVersionSensor	Structure/Union member
flagsEnvironment_64bit	Structure/Union member
flagsMath_64bit	Structure/Union member
flagsSample_64bit	Structure/Union member
fractionalIdxZeroPathDifference	Structure/Union member
fractionalIdxZeroPathDifference_BBRef	Structure/Union member
fractionalIdxZeroPathDifference_DUT1	Structure/Union member
fractionalIdxZeroPathDifference_DUT2	Structure/Union member
fractionalIdxZeroPathDifference_Ref	Structure/Union member
gainIndex	Structure/Union member
gainLevel	Structure/Union member
gainOrOffsetHasChangedSignificantly	Structure/Union member
gainPos	Structure/Union member
gmtTime	Structure/Union member
halfRange_DoubleSidedLowRes_cm	Structure/Union member
halfRange_SingleSidedFullRes_cm	Structure/Union member
hdrsize	Structure/Union member
hdrversion	Structure/Union member
hdrversionOriginal	Structure/Union member
idxAtMaxP2Pin200interferogramOr300RawInter	Structure/Union member
igramSmoothParam	Structure/Union member
igramSmoothType	Structure/Union member
importedFileformat	Structure/Union member
instr_model_str	Structure/Union member
instr_operator	Structure/Union member
instrumentSeriesIdentifier	Structure/Union member
instrumentStatus	Structure/Union member
instrumentType	Structure/Union member
instrument_model	Structure/Union member
integrationTime_ms	Structure/Union member
interferogramAverageNum	Structure/Union member
interferogramOffset	Structure/Union member
interferogramProperty	Structure/Union member
interferometerSerial	Structure/Union member
isPulsed	Structure/Union member
isStitched	Structure/Union member
latchedChannelStatus	Structure/Union member
latchedInstrumentStatus	Structure/Union member
latchedStageStatus	Structure/Union member
length	Structure/Union member
logData	Structure/Union member
maxControlLoopError	Structure/Union member
maxInterferogram	Structure/Union member
maxOPD_cm	Structure/Union member
minInterferogram	Structure/Union member
minOPD_cm	Structure/Union member
motorFirmwareVersion	Structure/Union member
name	Structure/Union member
noiseAmplitudeCutoffTimes10	Structure/Union member
offsetDarkLevel_double	Structure/Union member
offsetPlot	Structure/Union member
phaseCorrection	Structure/Union member
phasePolynomial	Structure/Union member

continues on next page

Table 1 – continued from previous page

phi	Structure/Union member
phiValueFormat	Structure/Union member
refGainIndex	Structure/Union member
refLaserHighPower	Structure/Union member
refLaserLocked	Structure/Union member
refLaserLowPower	Structure/Union member
refSignalInPercent	Structure/Union member
referenceWavelength_nm_vac	Structure/Union member
resolution	Structure/Union member
resolutionMode	Structure/Union member
rollingAverage	Structure/Union member
samplesPerReference	Structure/Union member
samplingDistance_cm_vac	Structure/Union member
samplingRate_MHz	Structure/Union member
sensitivityMode	Structure/Union member
sensorBlackLevel_count	Structure/Union member
sensorExposure_ms	Structure/Union member
sensorGain_dB	Structure/Union member
series	Structure/Union member
smoothParam	Structure/Union member
smoothType	Structure/Union member
softwareVersion	Structure/Union member
softwareVersionReconstructionAlgorithms	Structure/Union member
source	Structure/Union member
sourceType	Structure/Union member
speedFactor	Structure/Union member
stacking_count	Structure/Union member
stageStatus	Structure/Union member
startPos_um	Structure/Union member
stitched_amplifierSort	Structure/Union member
stitched_attenuationFilterUsed	Structure/Union member
stitched_autogainStatus	Structure/Union member
stitched_channelStatus	Structure/Union member
stitched_contrast	Structure/Union member
stitched_detectorCoolingState	Structure/Union member
stitched_detectorOffset	Structure/Union member
stitched_detectorSort	Structure/Union member
stitched_detectorType	Structure/Union member
stitched_detector_temperature_C	Structure/Union member
stitched_gainIndex	Structure/Union member
stitched_gainLevel	Structure/Union member
stitched_latchedChannelStatus	Structure/Union member
stitched_maxInterferogram	Structure/Union member
stitched_minInterferogram	Structure/Union member
stitched_x_maxWnr	Structure/Union member
stitched_x_minWnr	Structure/Union member
time	Structure/Union member
trigPos_um	Structure/Union member
triggerMode	Structure/Union member
type	Structure/Union member
wavelength_correction	Structure/Union member
wavelength_shift	Structure/Union member
wavenr_shift	Structure/Union member
x	Structure/Union member

continues on next page

Table 1 – continued from previous page

xAxisUnit	Structure/Union member
xValueFormat	Structure/Union member
x_max	Structure/Union member
x_maxHz	Structure/Union member
x_maxWnr	Structure/Union member
x_min	Structure/Union member
x_minHz	Structure/Union member
x_minWnr	Structure/Union member
yAxisUnit	Structure/Union member
y_max	Structure/Union member
y_min	Structure/Union member
zeroFillFactor	Structure/Union member

class pyOSA.**spectrum_t**

Represents the internal data format used by ThorSpectra.

This Python object grants access to the internal attributes and offers some methods for convenient access.

check_validity() → Dict[str, bool]

Returns a dictionary containing the "validity statuses" for the measured data.

The measured data can be an interferogram, a spectrum, or a stitched spectrum The following parameters are checked:

1. The reference laser was locked during the measurement
2. The interferogram signal was not clipped (also checked for a spectrum)
3. The interferogram was not nonlinear (tested only for OSA205, OSA207, and Redstone305)
4. Only for OSA200: The optimal gain was used during the measurement (only tested if autogain is enabled)

Returns:

dict[str, bool]: A dictionary containing the validity statuses. The following keys are possible:

- 'ref_laser_locked': True if the reference laser was locked, False otherwise.
- 'interferogram_within_detector_range': True if the interferogram signal was not clipped, False otherwise.
- 'interferogram_is_linear': True if the interferogram was linear, False otherwise.
- 'autogain_satisfied': True if autogain was satisfied, False otherwise. This key is only available for OSA20X.

convert_spectrum(*x_unit: Union[int, str]*, *y_unit: Union[int, str]*) → None

Converts the spectrum to specified x- and y-units.

Parameters:

x_unit (int | str):
The target x-axis unit. Possible units are listed in the manual.

y_unit (int | str):
The target y-axis unit.

Examples:

```
spectrum.convert_spectrum(x_unit="nm (vac)", y_unit="dBm (norm)")

spectrum.convert_spectrum(x_unit="Thz", y_unit="mW")
```

get_datetime() → datetime

Returns a datetime object of when the data was acquired.

Example:

```
# Convert datetime object to a string
datestr = spectrum.get_datetime().strftime("%Y-%m-%d %H:%M:%S")
```

get_gain_index() → int

Returns the instrument detector gain index.

Typical values: 0, 1, 2, ...

get_gain_level() → float

Returns the instrument detector gain.

Typical values: 1.0, 2.154, ...

get_interferogram_signal_in_percent(*detector=None*) → float

Returns the interferogram signal (in percent) for the measured data

This function can be called for either an interferogram or a spectrum. If called for a spectrum, it is still the signal of the interferogram resulting in the spectrum which is tested. For a stitched Redstone spectrum, the detector ('Detector 1' or 'Detector 2') needs to be specified.

Examples:

```
interferogram.get_interferogram_signal_in_percent()

spectrum.get_interferogram_signal_in_percent()

stitched_spectrum.get_interferogram_signal_in_percent('Detector 1')
```

get_model() → str

Returns the instrument model used to measure the data as a string, e.g., 'OSA201C'

get_resolution() → str

Returns the instrument resolution.

get_sensitivity() → str

Returns the instrument sensitivity.

get_serial_number() → str

Returns the serial number of the instrument used to measure the data.

get_x() → List[float]

Returns the x values from the measurement

get_xlabel() → str

Returns formatted xlabel e.g., 'Optical Frequency (THz)' or 'Wavelength (nm (vac))'

get_y() → List[float]

Returns the intensity values from the measurement

get_ylabel() → str

Returns formatted ylabel e.g., 'Absolute Power (mW)' or 'Power Density (dBm/nm)'

is_OSA200() → bool

Returns True if the measured data was captured using an OSA20X.

is_Redstone() → bool

Returns True if the measured data was captured using a Redstone series spectrometer.

is_autogain_satisfied() → bool

Returns True if the measured data was collected with the best possible gain setting.

This is useful to get rid of measurements collected during the autogain process which typically occurs if the light level changed since the last measurement. Note that this function currently only works for the OSA200-series. (FTSLib does not yet handle autogainStatus for the Redstone series.)

is_interferogram() → bool

Returns True if the measured data is an interferogram.

is_spectrum() → bool

Returns True if the measured data is a spectrum.

is_valid() → bool

Returns True if no problems were detected with the measured data.

The following parameters are checked:

1. The reference laser was locked during the measurement
2. The interferogram signal was not clipped (also checked for a spectrum)
3. The interferogram was not nonlinear (tested for OSA205, OSA207, and Redstone305)
4. Only for OSA200: The optimal gain was used during the measurement (only tested if autogain is enabled)

set_comment(*new_comment: str*) → None

Set the comment of this spectrum_t.

set_name(*new_name: str*) → None

Set the name of this spectrum_t.

set_operator(*new_operator: str*) → None

Set the operator of this spectrum_t.

INDEX

Symbols

A

C

G

I

L

O

P

S

V

W