pyLabLib Documentation

Release 0.4.1

Alexey Shkarin

May 29, 2021

Contents:

Note: This is the last **0.x** version of the library. The new library (**1.x**) is located at https://github.com/AlexShkarin/pyLabLib/, and its documentation can be found at https://pylablib.readthedocs.io/. It has large changes in the interface and code organization, rendering almost all previous code partially incompatible (see the new documentation for details).

CHAPTER 1

Installation

You can install the old version of the library (0.4.2) described here using pip:

```
pip install "pylablib<1"
```

This will install only the minimal subset of dependencies. To add packages needed for device communication, you can specify `devio` extra (on non-Windows systems use `devio-basic`, as some of the packages are not available there). To add packages needed for GUI, you can specify `gui` extra (note that one of the required packages is `PyQt5`, which is not available on pip for Python 2.7; hence, it needs to be installed prior to installing pyLabLib). To grab full set of required packages, call:

```
pip install "pylablib[devio,gui]<1"
```

Note: It is recommended tp can install the new version and, if you need it, use the legacy code package there.

1.1 Usage

To access to the most common functions simply import the library:

```
import pylablib as pll
data = pll.load("data.csv","csv")
```

1.2 Requirements

The package requires numpy, scipy, matplotlib, pandas and numba modules for computations. Note that when installed directly from pip, `numpy` comes with the OpenBLAS version of the linear algebra library; if other version (e.g., Intel MKL) is preferred, it is a good idea to `numpy` already installed before installing `pyLabLib`. All other packages can be safely installed from pip.

PyVISA and pySerial are the main packages used for the device communication. For some specific devices you might require `pyft232`, `pywinusb`, `websocket-client`, or nidaqmx (keep in mind that it's different from the `PyDAQmx` package). Some devices have additional requirements (devices software or drivers installed, or some particular dlls), which are specified in their description.

The package has been tested with Python 3.6 and Python 3.7. Python 2.7 might not be fully compatible anymore (although effort is made to preserve the compatibility, testing with Python 2.7 is far less extensive). The last version officially supporting Python 2.7 is 0.4.0.

1.3 Installing from GitHub

The library is available on GitHub at https://github.com/AlexShkarin/pyLabLib-v0/. To simply get all the source code, you can download it as a zip-file and unpack it into any appropriate place (can be folder of the project you're working on, Python site-packages folder, or any folder added to Python path variable).

Keep in mind that required packages will not be automatically installed, so this has to be done manually:

```
pip install future numpy scipy matplotlib pandas numba rpyc
pip install pyft232 pyvisa pyserial nidaqmx pywinusb websocket-client
pip install pyqt5 sip pyqtgraph
```

CHAPTER 2

Data processing utilities

2.1 Fitting

Class `fitting.Fitter` is a user-friendly wrapper around `scipy.optimize.least_squares()` routine. Dealing with fitting is made more convenient in a couple of ways:

- it is easy to specify the x-parameter name (in the case it is not the first parameter), or specify multiple x-parameters;
- all of the fit and fixed parameters are specified by name; it is easy to switch between any parameter being fit or fixed;
- the wrapper automatically handles complex parameters (split into real and imaginary parts), numpy arrays, lists, ot tuples (including nested structures);
- the final parameters (fit and fixed) are returned in a single dictionary indexed by their names;
- the wrapper also returns the fit function with all of the parameters bound to the final fit and fixed values;
- the fit function result is flattened during fitting, so it, for example, works for functions returning 2D arrays.

Examples

Fitting a Lorentzian:

```
def lorentzian(frequency, position=0., width=1., height=1.):
    return height/(1.+4.*(frequency-position)**2/width**2)

## creating the fitter
# fit_parameters dictionary specifies the initial guess
fit_par = {"position":0.5, "height":1.}
fitter = pll.Fitter(lorentzian, xarg_name="frequency", fit_parameters=fit_par)
# additional fit parameter is supplied during the call
fit_par, fit_func = fitter.fit(xdata, ydata, fit_parameters={"width":1.0})
plot(xdata, ydata)  # plot the experimental data
plot(xdata, fit_func(xdata))  # plot fit result
```

Fitting a sum of complex Lorentzians with the same width:

```
def lorentzian_sum(frequency, positions, width, amplitudes):
    # list of complex lorentzians
    #   positions and amplitudes are lists, one per peak
    lorentzians = [a/(1.+2j*(frequency-p)/width) for (a,p) in zip (amplitudes,
↪positions)]
    return np.sum(lorentzians, axis=0)

## creating the fitter
# fit_parameters dictionary specifies the initial guess
#      (complex initial guess for the "amplitude" parameter hints that this parameter␣
↪is complex)
fit_par = {"positions":[0.,0.5,1.], "amplitudes":[1.+0.j]*3}
fitter = pll.Fitter(lorentzian_sum, xarg_name="frequency", fit_parameters=fit_par)
# fixed parameter is supplied during the call (could have also been supplied on␣
↪Fitter initialization)
fit_par, fit_func = fitter.fit(xdata, ydata, fixed_parameters = {"width":0.3})
plot(xdata, ydata.real)  # plot the experimental data
plot(xdata, fit_func(xdata).real)  # plot fit result
```

Fitting 2D Gaussian and getting the parameter estimation errors:

```
def gaussian(x, y, pos, width, height):
    return np.exp( -((x-pos[0])**2+(y-pos[1])**2)/(2*width**2) )*height

## creating the fitter
# fit_parameters dictionary specifies the initial guess
fit_par = {"pos":(100,100), "width":10., "height":5.}
fitter = pll.Fitter(gaussian, xarg_name=["x","y"], fit_parameters=fit_par)
xs, ys = np.meshgrid(np.arange(img.shape[0]), np.arange(img.shape[1]), indexing="ij")
↪# building x and y coordinates for the image
# fit_stderr is a dictionary containing the fit error for the corresponding parameters
fit_par, fit_func, fit_stderr = fitter.fit([xs,ys], img, return_stderr=True)
imshow(fit_func(xs, ys))  # plot fit result
```

The full module documentation is given at `pylablib.core.dataproc.fitting`.

CHAPTER 3

Utilities

3.1 Multi-level dictionary

`dictionary.Dictionary` is an expansion of the standard *dict* class which supports tree structures (nested dictionaries). The extensions include:

- handling multi-level paths and nested dictionaries, with several different indexing methods
- iteration over the immediate branches, or over the whole tree structure
- some additional methods: mapping, filtering, finding difference between two dictionaries
- combined with `pylablib.core.fileio` allows to save and load the content in a human-readable format.

Creating and indexing:

```
>>> d = pll.Dictionary()
>>> d['d/0/x'] = 5
>>> d
Dictionary('d/0/x': 5)
>>> d['d/0/x']  # string path indexing
5
>>> d['d']['0']['x']  # nested indexing
5
>>> d['d','0','x']  # multi-level path indexing
5
>>> d['d',0,'x']  # all path elements are converted into strings
5
>>> d['d/0']['x']  # indexing styles can be freely mixed
5
>>> d['d','0/x']
5
>>> b = d['d']  # indexing a branch yields another Dictionary object
>>> b
Dictionary('0/x': 5)
>>> b['0/x'] = 10  # the branch shares the data with the main dictionary
```

(continues on next page)

(continued from previous page)

```
>>> d
Dictionary('d/0/x': 10)
```

A dictionary can be build from a Python `dict`, which automatically normalizes paths and nested dictionaries:

```
>>> d = pll.Dictionary({ 'a':1, 'b/i':2, 'c':{'i':3, 'ii':4}, 'd/0/x':5 })
>>> d
Dictionary('b/i': 2
'c/i': 3
'c/ii': 4
'd/0/x': 5
'a': 1)
```

Note: There are several limitations on the dictionary structure (mostly they involve possible paths and keys):

- As mentioned above, the keys are converted into strings to get the path; therefore, different Python object can merge together (e.g., number `0` and string literal `'0'`). This also discourages use of some of the objects with “underdefined” (implementation dependent) representations, for example, floating point numbers.
- Since the `'/'` symbol is used to split different path entries, it can’t be used inside a single-level key. It is possible to re-define this symbol on dictionary creation; however, it might lead to compatibility issues.
- Using spaces is in principle allowed; however, it leads to problems if the dictionary is saved to or loaded from a text file using standard methods, since there space is used to separate the path and the value (so a part of the path after the first space would become a part of the value). The same concerns other whitespace characters (`'\n'`, `'\r'`, `'\t'`).
- Empty keys are not allowed. When building a path, they are automatically dropped, so `'a/b'`, `'a/b/'`, `'a///b//'` all correspond to the same path.
- One path can either correspond to a branch node, or a leaf node. In other words, one path can’t be a prefix of other paths and also contain data: structures like `pll.Dictionary({ 'a':1, 'a/b':2})` are not allowed. To get around this, one can define a specific “data key” not used anywhere else, and store data in a node under that key (e.g., with the data key `'#'` the example before turns into a valid structure `pll.Dictionary({ 'a/#':1, 'a/b/#':2}))`.

Thus, it is generally recommended to only use strings or non-negative integers as keys, and apply the same restrictions to them as to the Python variable names (with the addition of names starting with a digit).

CHAPTER 4

Specific device classes

4.1 General concepts

Most devices share common methods and approach to make them more predictable and easier to use.

First, the device identifier / address needs to be provided by user during the device object creation, and it is automatically connected. The devices have `open` and `close` methods but the device also works as a resource (with Python `with` statement), so these usually aren't used explicitly.

The devices usually have `get_settings` and `apply_settings` methods which return Python dictionaries with the most common settings or take these dictionaries and apply them. In addition, there are `get_full_status` and `get_full_info` functions, which return progressively more information (`get_full_status` adds variable status information which cannot be changed by user, and `get_full_info` adds constant device information, such as model name and serial number). `get_full_info` can be particularly useful to check the device status and see if it is connected and working properly.

Devices of the same kind (e.g., cameras or translation stages) aim to have consistent overlapping interfaces (where it makes sense), so different devices are fairly interchangeable in simple applications.

4.2 Examples

Connecting to a Cryomagnetics LM500 level meter and reading out the level at the first channel:

```
from pylablib.aux_libs.devices import Cryomagnetics  # import the device library
# Next, create the device object and connect to the device;
#   the connection is automatically opened on creation, and closed when the ``with``␣
↪block is ended
with Cryomagnetics.LM500("COM1") as lm:
    level = lm.get_level(1)  # read the level
```

Stepping the M Squared laser wavelength and recording an image from the Andor IXON camera at each step:

```
from pylablib.aux_libs.devices import M2, Andor  # import the device libraries
with M2.M2ICE("192.168.0.1", 39933) as laser, Andor.AndorCamera() as cam:  # connect␣
↪to the devices
    # change some camera parameters
    cam.set_shutter("open")
    cam.set_exposure(50E-3)
    cam.set_amp_mode(preamp=2)
    cam.set_EMCCD_gain(128)
    cam.setup_image_mode(vbin=2, hbin=2)
    # setup acquisition mode
    cam.set_acquisition_mode("cont")
    cam.setup_cont_mode()
    # start camera acquisition
    cam.start_acquisition()
    wavelength = 740E-9  # initial wavelength (in meters)
    images = []
    while wavelength < 770E-9:
        laser.tune_wavelength_table(wavelength)  # tune the laser frequency (using␣
↪coarse tuning)
        time.sleep(0.5)  # wait until the laser stabilizes
        cam.wait_for_frame()  # ensure that there's a frame in the camera queue
        frame = cam.read_newest_image()
        images.append(frame)
        wavelength += 0.5E-9
```

4.3 List of devices

Device	Kind	Module	Comments
M Squared ICE BLOC	Laser	`M2`	
Pure Photonics PPCL200	Laser	`PurePhotonics`	In CBDX1 chassis
Lighthouse Photonics SproutG	Laser	`LighthousePhotonics`	
LaserQuantum Finesse laser	Laser	`LaserQuantum`	
Agilent HP8168F	Laser	`AgilentLasers`	
Nuphoton NP2000	EDFA	`NuPhoton`	
HighFinesse WS/6 and WS/7	Wavemeter	`HighFinesse`	
Andor Shamrock	Spectrometer	`Andor`	Tested with Andor SR-
Andor SDK2 interface	Camera	`Andor`	Tested with Andor IXC
Andor SDK3 interface	Camera	`Andor`	Tested with Andor Zyl
Hamamatsu DCAM interface	Camera	`DCAM`	Tested with ORCA-Fla
NI IMAQdx interface	Camera	`IMAQdx`	Tested with Photon Fo
NI IMAQ interface	Camera	`IMAQ`	Tested with NI PCI-14
Photon Focus PFCam interface	Camera	`PhotonFocus`	Tested with MV-D102
PCO SC2 interface	Camera	`PCO_SC2`	Tested with PCO.edge
Ophir Vega	Optical power meter	`Ophir`	
Thorlabs PM100D	Optical power meter	`Thorlabs`	
OZ Optics TF100	Tunable optical filter	`OZOptics`	
OZ Optics DD100	Variable optical attenuator	`OZOptics`	
OZ Optics EPC04	Polarization controller	`OZOptics`	
Agilent AWG33220A	Arbitrary waveform generator	`AgilentElectronics`	
Agilent N9310A	Microwave generator	`AgilentElectronics`	
Vaunix LMS (Lab Brick)	Microwave generator	`Vaunix`	

Table 1 – continued from previous page

Device	Kind	Module	Comments
Thorlabs MDT693/4A	High voltage source	`Thorlabs`	
Agilent AMP33502A	DC amplifier	`AgilentElectronics`	
Rigol DSA1030A	Microwave spectrum analyzer	`Rigol`	
Agilent HP8712B, HP8722D	Vector network analyzers	`AgilentElectronics`	
Tektronix DPO2014, TDS2000, MDO3000	Oscilloscopes	`Tektronix`	
NI DAQ interface	NI DAQ devices	`NI`	Wrapper around the ni
Zurich Instruments HF2 / UHF	Lock-in amplifiers	`ZurichInstruments`	
Arcus PerforMax	Translation stage	`Arcus`	Tested with PMX-4EX
SmarAct SCU3D	Translation stage	`SmarAct`	
Attocube ANC300	Piezo slider controller	`Attocube`	Only tested with Ether
Attocube ANC350	Piezo slider controller	`Attocube`	Only tested with USB
Trinamic TMCM1110	Stepper motor controller	`Trinamic`	
Thorlabs KDC101	DC servo motor controller	`Thorlabs`	
Thorlabs K10CR1	Motorized rotation mount	`Thorlabs`	
Thorlabs FW102/202	Motorized filter wheel	`Thorlabs`	
Thorlabs MFF	Motorized flip mount	`Thorlabs`	
Cryomagnetics LM500/510	Cryogenic level meter	`Cryomagnetics`	
Lakeshore 218 and 370	Temperature controllers	`Lakeshore`	
MKS 9xx	Pressure gauge	`MKS`	
Pfeiffer TPG261	Pressure gauge	`Pfeiffer`	

All the modules are located in `pylablib.aux_libs.devices`.

4.4 Additional requirements

First, any device using `PyVISA` require NI VISA to be installed. See PyVISA for details.

Second, some devices need dlls supplied by the manufacturer:

- Andor SDK2 cameras: require *atmcd.dll* (currently supplied for x64 and x86). Can be obtained with Andor Solis software, or Andor SDK. If Andor Solis is installed in the default location (*C:/Program Files/Andor Solis*), these dlls are accessed automatically. It might be called *atmcd64d_legacy.dll* or *atmcd32d_legacy.dll* (depending on the Solis version and Python bitness), but it needs to be renamed to *atmcd.dll* when placed into *aux_libs/devices/libs/x64* (or *x32*) folder.
- Andor SDK3 cameras: require several *at*.dll*: *atcore.dll*, *atblkbx.dll*, *atcl_bitflow.dll*, *atdevapogee.dll*, *atdevregcam.dll*, *atusb_libusb.dll*, *atusb_libusb10.dll* (currently supplied only for x64). Has potential incompatibilities between different versions of Windows; tested with Windows 7 x64 and Andor Solis 4.30.30034.0. Can be obtained with Andor Solis software. If Andor Solis is installed in the default location (*C:/Program Files/Andor Solis*), these dlls are accessed automatically.
- PCO SC2 cameras: require several *SC2_*.dll*: *SC2_Cam.dll*, *sc2_cl_me4.dll*, *sc2_cl_mtx.dll*, *sc2_cl_nat.dll*, *sc2_cl_ser.dll*, *sc2_clhs.dll*. These are provided with pco.sdk, which can be obtained on PCO website.
- Arcus PerforMax translation stages: require *PerformaxCom.dll* and *SiUSBXp.dll*. Can be obtained from Arcus website, either at USB 64-bit DLL (for 64-bit systems), or inside Python USB source (for 32-bit systems)
- HighFinesse WS/6 and WS/7 wavemeters: require *wlmData.dll*. Each device needs a unique dll supplied by the manufacturer. One can either supply DLL path on creation of the device class, or place it into *aux_libs/devices/libs/x64* (or *x32*) folder; in the latter case, it should be renamed to *wlmData6.dll* or *wlmData7.dll* depending on the wavemeter model (WS/6 or WS/7).

- SmarAct SCU3D translation stage controller: requires *SCU3DControl.dll*. Should be supplied with the device by the manufacturer.

For the application to have access to them, they need to be placed into the package folder (correspondingly, into `aux_libs/devices/libs/` inside the main package folder, which is usually something like `Python36/Lib/site-packages/pylablib/`).

Third, some devices need additional software installed:

- IMAQ cameras: National Instruments IMAQ library.
- IMAQdx cameras: National Instruments IMAQdx library.
- Photon Focus cameras: Photon Focus PFRemote software.
- Hamamatsu DCAM cameras: DCAM software (Hamamatsu HOKAWO) and drivers.
- Andor cameras: Andro Solis software and drivers
- NI DAQs: National Instruments NI-DAQmx library (with C support; just Runtime is sufficient).
- HighFinesse: manufacturer-provided drivers and software (specific to the particular wavemeter).
- Thorlabs Kinesis devices (KFF, KDC101, K10CR1): Kinesis/APT software.
- Trinamic hardware: Trinamic TMCL-IDE (needed to install device drivers)
- Arcus PerforMax software: Arcus Drivers and Tools, Arcus USB Series and Arcus Performax Series software (needed to install device drivers).
- Zurich Instruments: manufacturer provided software and Python libraries.

The list might be incomplete, and it does not include drivers for all USB devices.

CHAPTER 5

Change log

This is a list of changes between each version.

5.1 0.4.2

Final version of library with updated links to repositories and documentation. No significant code change.

5.2 0.4.1

Interface changes

- Slightly changed representations of complex number in to-string conversions depending on the conversion rules (`"python"` vs `"text"`).

Additions

- Devices
  - Added Thorlabs K10CR1 rotational stage (`devices.Thorlabs.K10CR1`)
  - Added Andor Shamrock spectrographs (`devices.AndorShamrock`)
  - Expanded Agilent AWG class
  - Added more 32bit dlls
  - Added `list_resources` method to every backend class, which lists available connections for this backend (not available for every backend; so far only works in `VisaDeviceBackend`, `SerialDeviceBackend`, and `FT232BackendOpenError`.
- GUI and threading
  - Added `TableAccumulatorThread.preprocess_data()` method to pre-process incoming data before adding it to the table

  - Added `update_only_on_visible` argument to `TracePlotter.setupUi()` method, and `TracePlotter.get_required_channels()` method.

5.3 0.4.0

Interface changes

- Dictionary entries (`core.fileio.dict_entry`) system has been slightly redesigned: building entries from stored objects has been moved from `dict_entry.IDictionaryEntry.build_entry()` class method to a dedicated function `dict_entry.build_entry()`, and entry classes have been added.
- `aux_libs.gui.helpers.StreamFormerThread` architecture changes, so that it can accumulates several rows before adding them into the storage; this lead to replacement of `helpers.StreamFormerThread.prepare_new_row()` method by `helpers.StreamFormerThread.prepare_new_data()`.

Additions

- General
  - Added pandas support in a bunch of places: loading/saving tables and dictionaries; data processing routines in `core.dataproc`; conversion of `DataTable` and `Dictionary` object to/from pandas dataframes.
  - Expanded string conversion to support more explicit variable classes. For example, a numpy array `np.array([1,2,3])` can be converted into a string `'array([1, 2, 3])'` instead of a more ambiguous string `'[1, 2, 3]'` (which can also be a list). This behavior is controlled by a new argument `use_classes` in string conversion functions (such as `string.to_string()` and `string.from_string()`) and an argument `use_rep_classes` in file saving (`savefile.save()`)
  - Added general library parameters, which can be accessed via `pylablib.par` (works as a dictionary object). So far there's only one supported parameter: the default return type of the CSV file reading (can be `"pandas"` for pandas dataframe, `"table"` for `DataTable` object, or `"array"` for raw numpy array).
- Devices
  - Added LaserQuantum Finesse device class (`devices.LaserQuantum`)
  - NI DAQ now supports output of waveforms
  - Added `PCO_SC2.reset_api()` and `PCO_SC2.PCOSC2Camera.reboot()` methods for resetting API and cameras
  - Added `Thorlabs.list_kinesis_devices()` function to list connected Kinesis devices
  - Added serial communication methods for IMAQ cameras (`IMAQ.IMAQCamera`)
- GUI and threading
  - Added line plotter (`aux_libs.gui.widgets.line_plotter`) and trace plotter (`aux_libs.gui.widgets.trace_plotter`) widgets
  - Added virtual elements to value tables and parameter tables
  - Added `gui_thread_safe` parameter to value tables and parameter tables. Enabling it make most common methods thread-safe (i.e., transparently called from the GUI thread)
  - Added a corresponding `controller.gui_thread_method()` wrapper to implement the change above
  - Added functional thread variables (`controller.QThreadController.set_func_variable()`)

- File saving / loading
  - Added notation for dictionary files to include nested structures ('prefix blocks'). This lets one avoid common path prefix in stored dictionary files. For example, a file

    ```
    some/long/prefix/x  1
    some/long/prefix/y  2
    some/long/prefix/y  3
    ```

    can be represented as

    ```
    //some/long/prefix
        x   1
        y   2
        z   3
    ///
    ```

    The meaningful elements are `//some/long/prefix` line denoting that following elements have the given prefix, and `///` line denoting that the prefix block is done (indentation is only added for clarity).
  - New dictionary entries: `dict_entry.ExternalNumpyDictionaryEntry` (external numpy array, can have arbitrary number of dimensions) and `dict_entry.ExpandedContainerDictionaryEntry` (turns lists, tuples and dicts into dictionary branches, so that their content can benefit from the automatic table inlining, dictionary entry classes, etc.).
- Data processing
  - `fitting.Fitter` now takes default scale and limit as constructor arguments.
  - `feature.multi_scale_peakdet()` has new `norm_ratio` argument.
  - `image.get_region()` and `image.get_region_sum()` take `axis` argument.
- Miscellaneous
  - Functions introspection module now supports Python 3 - style functions, and added a new function `functions.funcsig()`
  - `utils.general.StreamFileLogger` supports multiple destination paths
  - New network function `utils.net.get_all_local_addr()` (return list of all local addresses on all interfaces) and `utils.net.get_local_hostname()`

CHAPTER 6

Indices and tables

- genindex
- modindex
- search