PyHaHa2 Documentation

Release 1.0.1

Michael Hufschmidt

Feb 04, 2019

CHAPTER

ONE

GENERAL INTRODUCTION

PyHaHa2 is a software project within the Particle Physics and Detector Development Group of the Universität Hamburg, Institut für Experimentalpyhsik, http://wwwiexp.desy.de/groups/pd/ . It is the upwards compatible successor of the PyHaHa project.

The objective is to provide universal Python classes to analyze the data files created by several Labview programs, usually these are located somewhere under m_data on the uh2usnmserver. These classes can easily be adapted to specific needs and analyses. Future expansions will be possible.

Recommended Software Stack

- Linux Ubuntu 16.04 (also tested on Ubuntu 14.04)
- Python 3.5.x (also tested with Python 2.7.6) with numpy, scipy, matplotlib.pyplot and others

Warning: Some classes require numpy version $\geq$ 1.11.

- Git (version control system for distributed software development)
- Sphinx (tool to create intelligent and beautiful documentation like this one)

Helpful, but not absolutely neccessary:

- Spyder3 (IDE for Python 3.5.x, alternatively Spyder for Python 2.7.x or any other editor.)
- Pylint for checking your source with respect to the PEP 8 style guide for Python code

Source code analysis with Pylint can be done within Spyder (menu "Source" / "Run static code analysis"). You should target for an overall rating > 7.0 of your programs.

Installlation and Update

With `git` it is very easy to obtain the package and the complete history of all sources including documentation. The complete reference to `git` is published at https://git-scm.com/doc, a simple introduction can be found here: http://rogerdudler.github.io/git-guide/index.de.html .

First time users go to a directory where they collect their own Python libraries (for instance `~/my_Phyton_libs/` and create a sub-directory `~/my_Python_libs/PyHaHa2` under that directory with the command:

```
git clone https://github.com/MiHuf/PyHaHa2.git
```

After that, you will find this document under `PyHaHa2/doc/latex/PyHaHa2.pdf`

After some time it may be neccessary to update your local `PyHaHa2` directory with the latest project version. Therefore you should move into your local `PyHaHa2` directory and use the command:

```
git pull
```

To and create / update this documentation, move to the local `PyHaHa2` directory and use the command:

```
make latexpdf
```

If that does not work "Extension error: Could not import extension sphinx.ext.imgmath (exception: No module named imgmath)", you will need to install python-pip or python3-pip from the software center and then:

```
pip3 install --user pip --upgrade
pip3 install --user sphinx
pip3 install --user sphinx --upgrade
pip3 install --user pymysql
pip3 install --user pymysql --upgrade
```

The last two lines are only needed for directly accessing data from the database `uh2detlab` at `uh2usnmfile01.desy.de`.

For active participation on the project you will need an own git account with collaborator access to this project. Before doing any changes, be sure to first update your local `PyHaHa2` directory with a `git pull`. You can then modify the source code or add new files in your local directory. At the end of the day, after finishing your modifications, you should publish your work with:

```
git commit -a -m "Short description of my modifications"
git push
```

Very useful are these git commands:

```
git --help
git status
```

Usage

For using PyHaHa2 all the modules `xxx.py` defined in the directory PyHaHa2/ need to be accessable by your programs. Currently these are:

- `pyhaha.py` (Main module, imports all othe modules)
- `cold_chuck_tools.py`
- `transient_tools.py`
- `pyhaha_plots.py`
- `cold_chuck_plots.py`
- `transient_plots.py`
- `file_utils.py`
- `conf.py` (needed for Sphinx)

Rather than copying these files to your local development directory, you should ensure, that you can use the modules from anywhere. You can achieve this by modifying your local file `~/.bashrc` or `~/.profile` and add the line:

```
export PYTHONPATH=$PYTHONPATH:'~/my_Python_libs/PyHaHa2/'
```

to the end of `~/.bashrc` or `~/.profile`. When using the PyHaHa2 tools, you then only need to add the code line:

```
from pyhaha import *
```

to the beginning of your Python program.

- Your datafile(s) `xxx.iv` or `xxx.cv`

The datafiles can be located in your working directory or in a separate directory (`data_dir`) provided as a parameter to the constructor of the class `ColdChuckData` or `ScopeData`. or elsewhere.

- A copy the file `pyhaha_plot_defaults` in your woking directory

This file is requiued in your woking directory for all plot programs. You can edit this file for using personal plot defaults.

In order to enable instances of `ColdChuckData` or `ScopeData` to load the files directly from the GVFs shares `m_data/macroscopic/` or `scratch_nmsamba/rd_data/` on uh2usnmserver, it is recommended to create symlinks `~/m_data/` and `~/scratch_nmsamba/` in your home directory pointing to the shares "smb://uh2usnmserver/xxx".

This can be done by mounting the shares m_data and scratch_nmsamba with the file server (Nautilus) before: Connect to "smb://uh2usnmserver/xxx" as registered user. Make sure to chose to save the password forever. After that you can create the symlinks.

See also the function mount_gvfs() in the module file_utils.

CHAPTER

TWO

HOW TO MAKE COLD_CHUCK_PLOTS

The module cold_chuck_plots.py contains classes to read, plot and analyse data-files from CV- and IV-measurements of silicon particle detectors created by one the cold chuck setups. The following plots are pre-defined:

IV-plot

Here is an example how to create a simple IV-plot with only 12 lines of Python code (source code in file `demo_iv.py`, result in Fig. 2.1 left).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'FTH200N_04_DiodeS_14_2015-11-06_7.iv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
p = IVPlot(ccd)                         # create plot object
p.make_plot()                           # create the plot
# plt.semilogy()                        # optional
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

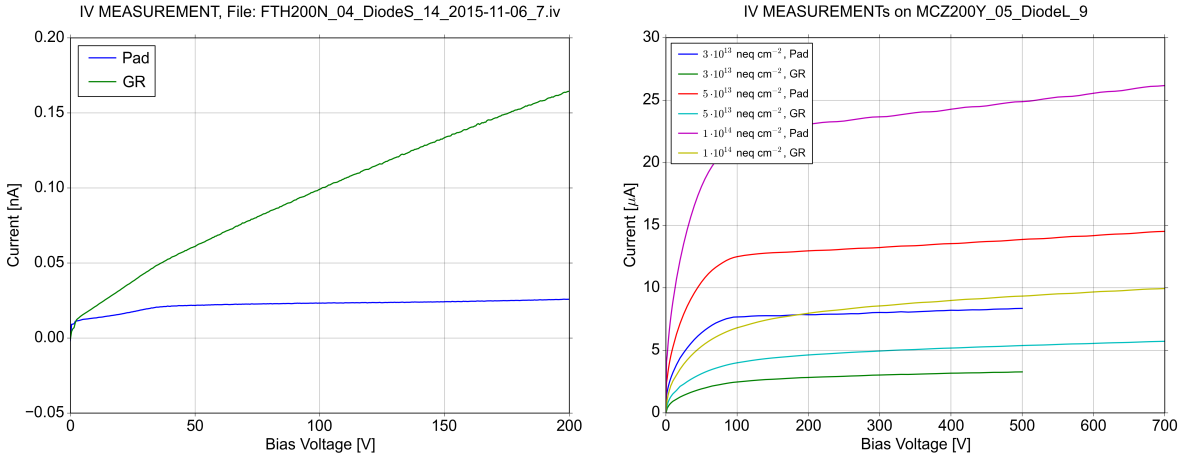


Fig. 2.1: IV-plots based on single (left) and mutilple (right) datafiles

All the built-in xxxPlot-classes can also handle mutiple datafiles, see the following example (source code in file `demo_multi_iv.py`, result in Fig. 2.1 right).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
my_files  = ['MCZ200Y_05_DiodeL_9_2012-08-08_4.iv',
             'MCZ200Y_06_DiodeL_11_2012-08-09_4.iv',
             'MCZ200Y_07_DiodeL_8_2012-08-13_4.iv']
my_labels = [r'$3\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$5\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$1\cdot 10^{14}$ neq cm${}^{-2}$']
ccds = []                               # list of data file objects
for my_file in my_files:
    ccd = ColdChuckData(my_file, directory=my_dir) # create an object for each
    ccds.append(ccd)                    # and collect in a list
    print(ccd.get_filepath())           # where the file finally was found
p = IVPlot(ccds, params={'legend.fontsize': 'small'}) # create plot object
p.make_plot(my_labels, 1e6, r'$\mu$A')  # create the plot
# plt.semilogy()                        # optional
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

Voltage-plot

Here is an example how to create a simple Voltage-plot. Voltage-plots are useful to get an overview over the indexes of the numpy-arrays (source code in file `demo_voltage.py`, result in Fig. 2.2).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'Ketek-2018_01_PM3315-WB-C0_3_2018-08-23_1.iv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
p = VoltagePlot(ccd)                    # create plot object
p.make_plot()                           # create the plot
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

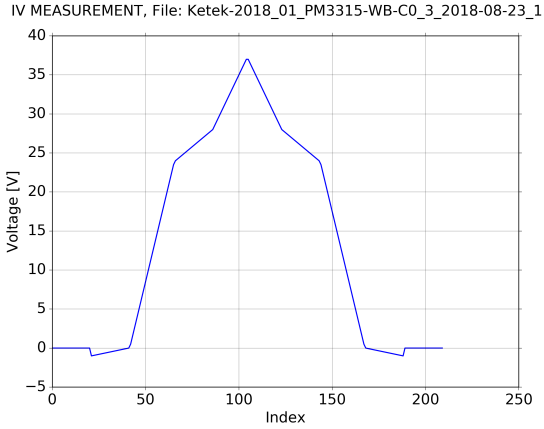


Fig. 2.2: Voltage plot: Voltage versus index of the numpy-array

CV-plot

Here an example for a standard CV-plot $\frac{1}{C^2}$ versus V_{Bias} with few code lines (source code in file `demo_cv.py`, result in Fig. 2.3 left).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'FTH200N_04_DiodeS_14_2015-11-05_4.cv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
p = CVPlot(ccd)                         # create plot object
p.make_plot()                           # create the plot
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

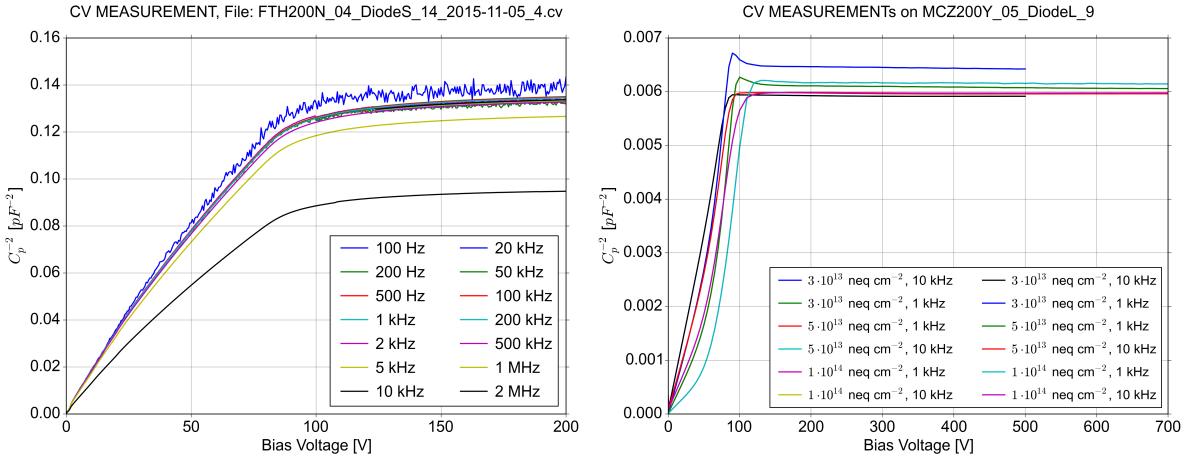


Fig. 2.3: CV-plots based on single (left) and mutilple (right) datafiles

Similarly, a standard CV-plot with multiple datafiles (source code in file `demo_multi_cv.py`, result in Fig. 2.3 right):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
my_files  = ['MCZ200Y_05_DiodeL_9_2012-08-08_4.cv',
             'MCZ200Y_06_DiodeL_11_2012-08-09_4.cv',
             'MCZ200Y_07_DiodeL_8_2012-08-13_4.cv']
my_labels = [r'$3\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$5\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$1\cdot 10^{14}$ neq cm${}^{-2}$']
ccds = []                               # list of data file objects
for my_file in my_files:
    ccd = ColdChuckData(my_file, directory=my_dir) # create an object for each
    ccds.append(ccd)                    # and collect in a list
    print(ccd.get_filepath())           # where the file finally was found
p = CVPlot(ccds)                        # create plot object
p.update_params({'legend.fontsize': 'medium'})
p.make_plot(my_labels)                  # create the plot
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

Cf-plot

And here another example how to create the standard Cf-plot C_p versus f for a few voltages provided as parameter (source code in file `demo_cf.py`, result in Fig. 2.4 left).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'FTH200N_04_DiodeS_14_2015-11-05_4.cv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
p = CfPlot(ccd)                         # create plot object
p.make_plot([5, 20, 50, 100, 200])      # create the plot for some voltages
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

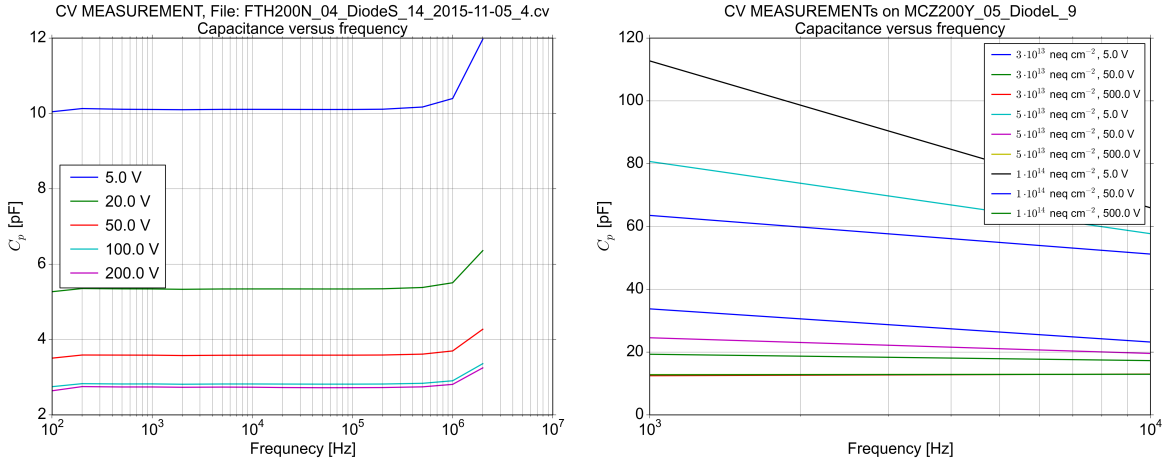


Fig. 2.4: Cf-plots based on single (left) and mutilple (right) datafiles

The same with multiple datafiles (source code in file `demo_multi_cf.py`, result in Fig. 2.4 right):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
my_files  = ['MCZ200Y_05_DiodeL_9_2012-08-08_4.cv',
             'MCZ200Y_06_DiodeL_11_2012-08-09_4.cv',
             'MCZ200Y_07_DiodeL_8_2012-08-13_4.cv']
my_labels = [r'$3\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$5\cdot 10^{13}$ neq cm${}^{-2}$',
             r'$1\cdot 10^{14}$ neq cm${}^{-2}$']
ccds = []                               # list of data file objects
for my_file in my_files:
    ccd = ColdChuckData(my_file, directory=my_dir) # create an object for each
    ccds.append(ccd)                    # and collect in a list
    print(ccd.get_filepath())           # where the file finally was found
p = CfPlot(ccds, params={'legend.fontsize': 'small'}) # create plot object
p.make_plot([5, 20, 50, 100, 200], my_labels) # create the plot for some voltages
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

Yf- or Zf-plot

Here a double logarithmic plot of impedance versus frequency for some voltages (source code in file `demo_yzf.py`, result in Fig. 2.5).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'FTH200N_04_DiodeS_14_2015-11-05_4.cv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
p = YZfPlot(ccd)                        # create plot object
p.make_plot([5, 20, 50, 100, 200], plot_type='Z') # create impedance plot
p.save_plot()                           # save plot as .pdf
plt.show()                              # show plot
```

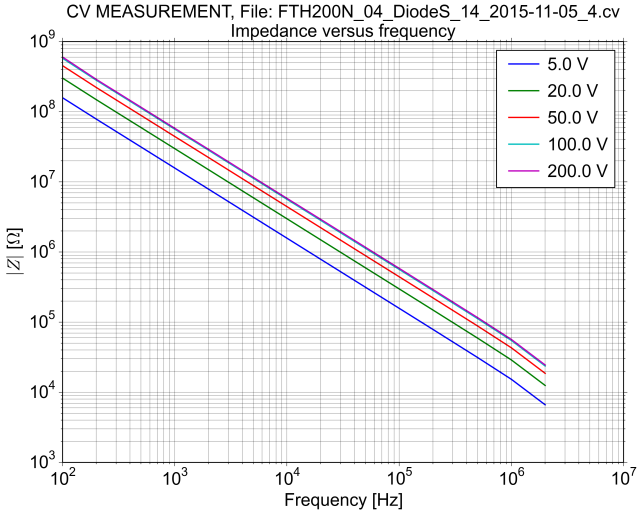


Fig. 2.5: Impedance versus frequency plot

More sophisticated plots

Here a plot consisting of two double logarithmic subplots for serial and parallel resistance versus frequency (source code in file `demo_rs_rp.py`, result in Fig. 2.6).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
my_dir = './data_dir'                   # here are all my .iv and .cv files
datafile = 'w1-pm1125-2_2013-07-24_4.cv'
ccd = ColdChuckData(datafile, my_dir)   # create a data file object
print(ccd.get_filepath())               # where the file finally was found
rs = ccd.get_rs()                       # serial resistance
rp = 1.0 / ccd.get_gp()                 # parallel resistance from admittance
freqs = ccd.get_frequencies()           # all frequencies
voltages = [4, 8, 12, 14, 16, 20, 24, 26] # plot only these voltages
p = CVPlot(ccd)                         # create plot object
volt_indices = ccd.v_index(voltages)
fig = plt.figure(figsize=[11.0, 12.0])
```

```
fig.text(0.5, 0.98, p.make_title(), ha='center', va='top')
# First Subplot
plt.subplot(2, 1, 1)
plt.title('Serial Resistance versus Frequency')
for vi in volt_indices:
    v_label = "{} V".format(ccd.get_volts()[vi])
    plt.plot(freqs, rs[vi, :], label=v_label)
plt.ylabel(r'Resistance $R_s$ [$\Omega$]')
plt.loglog()
plt.legend(loc='best')
# Second Subplot
plt.subplot(2, 1, 2)
plt.title('Parallel Resistance versus Frequency')
for vi in volt_indices:
    v_label = "{} V".format(ccd.get_volts()[vi])
    plt.plot(freqs, rp[vi, :], label=v_label)
plt.xlabel('Frequency [Hz]')
plt.ylabel(r'Resistance $R_p$ [$\Omega$]')
plt.loglog()
plt.legend(loc='best')
# Show and save all
p.save_plot()                          # save plot as .pdf
plt.show()                             # show plot
```

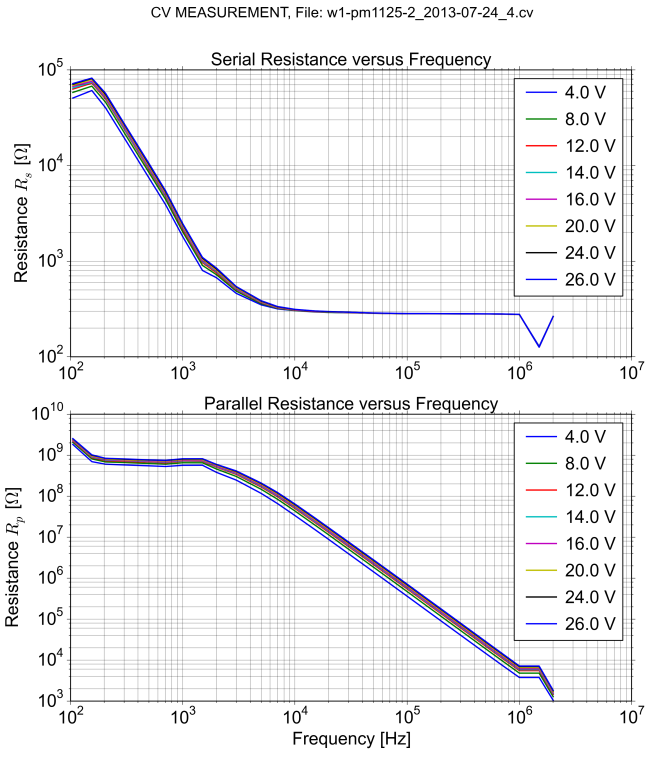


Fig. 2.6: Log-log plot of serial and parallel resistance versus frequency

Limited Voltage Range

Many of the return values of the methods in class ColdChuckData and many of the plots can be restriced to a limited range of voltages. For details have a look at the documentation of the methods `set_voltage_index_range` and `set_voltage_range` in the class `ColdChuckData`.

CHAPTER

THREE

HOW TO MAKE TRANSIENT_PLOTS

The module transient_plots.py contains classes to read, plot and analyse data-files from scope data. The following plots are predefined:

Waveform - Plot

Here is an example how to create wafeform plots. (Source code in file `demo_waveform.py`, result in Fig. 3.1).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
data_dir = './data_dir'
filename = '03_20GSs_400ns_49807_raw'   # the Base-Name
sd = ScopeData(filename, data_dir)      # create a ScopeData object
sd.print_filenames()                    # call a ScopeData method
sd.print_props()                        # call a ScopeData method

p = WaveformPlot(sd)                    # create a standard plot object
p.make_plot(data_slot=50)               # create a single-waveform plot
p.save_plot('plot_waveform_single')     # save plot
plt.close()                             # clear plot, prepare for new one
p.make_plot()                           # create a mulit-waveform plot
p.save_plot('plot_waveform_multi')      # save plot
plt.show()                              # show plot
```

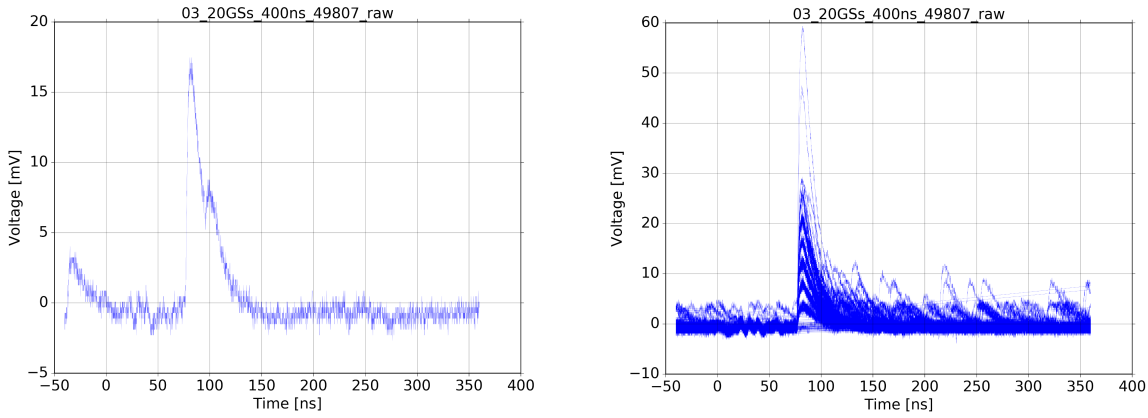


Fig. 3.1: Wafeform-plots based on single (left) and mutilple (right) waveforms

Spectrum - plot

Here is an example how to create a simple spectrum plot. (Source code in file `demo_spectrum.py`, result in Fig. 3.2).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pyhaha import *                    # Environment variable $PYTHONPATH
                                        # points to relevant folder
data_dir = './data_dir'
filename = '03_20GSs_400ns_49807_raw'   # the Base-Name
sd = ScopeData(filename, data_dir)      # create a ScopeData object
sd.print_filenames()                    # call a ScopeData method
sd.print_props()                        # call a ScopeData method

p = SpectrumPlot(sd)                    # create a standard plot object
p.make_plot()                           # create a plots with default params
#p.make_plot(2200, 3500)                # create all plots with index range
p.save_plot('plot_spectrum')            # save plot
plt.show()                              # show plot
```

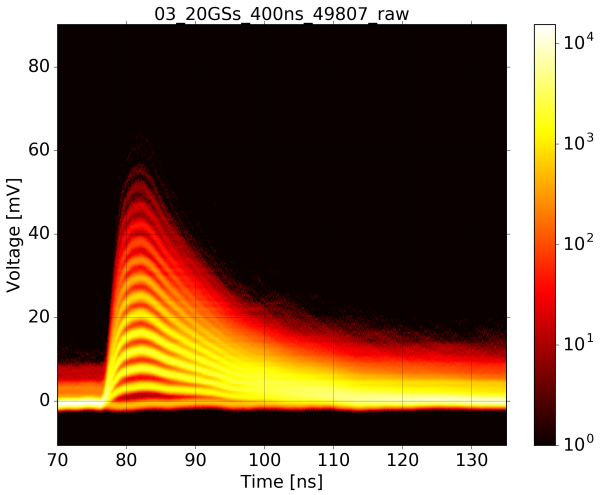


Fig. 3.2: Spectrum Plot

CHAPTER

FOUR

MODULE: PYHAHA

pyhaha.py is the base module. Here all other modules and imports are loaded, and some general constants are defined. For usage, simply add the following code to your Python program:

```
from pyhaha import *
```

The following modules are required and will always be loaded:

- os
- sys
- re (for regular expressions)
- csv (for CSV files)
- numpy as np (for multidimensional arrays, linear algebra, ...)
- cmath
- collections (for OrderedDict)
- lxml.etree as et (for XML parsers)
- matplotlib.pyplot as plt
- matplotlib.colors as colors
- string
- io
- datetime
- time

The following project modules are imported:

- cold_chuck_tools.py
- transient_tools.py
- pyhaha_plots
- cold_chuck_plots.py
- transient_plots.py
- file_utils.py

CHAPTER

FIVE

MODULE: COLD_CHUCK_TOOLS

The module cold_chuck_tools.py contains classes to read and analyse data-files from CV- and IV-measurements of silicon particle detectors created by the cold chuck setup. For usage, simply add the following code to your Python program:

```
from cold_chuck_tools import *
```

The module defines the global constants:

- MYHOME (my home directory)
- M_DATA_DIR (= '~/m_data/' which shold be s symlink to a gvfs share)
- RD_DATA_DIR (= '/scratch_nmsamba/' which shold be s symlink to a gvfs share)
- DATE_REGEX (regular expression for dates in ISO format)

The module contains the classes:

- class ColdChuckData

class ColdChuckData

class cold_chuck_tools.**ColdChuckData**(*filename=''*, *directory=''*, *fullpath=''*, *logfile=None*)

This class reads the data-files `xxx.cv` or `xxx.iv` created by the cold chuck lab setup. When using Python 3.x, the character-set will automatically be converted from iso-8859-1 (used by Windows XP) to utf-8 (used by Python). Suitable methods to obtain data are provided.

To create an instance of ColdChuckData you can use as an exmaple:

```
ccd = ColdChuckData('FTH200N_04_DiodeS_14_2015-11-05_4.cv', 'myDataDir')
```

The parameter(s) provided to the constructor have the following meaning:

filename [string] Filename to read, e.g. `FTH200N_04_DiodeS_14_2015-11-05_4.cv`

directory [string, optional] The directory which contains the data-file (if not it is not in the current directory).

fullpath [string, optional] If *filename* is an empty string, searching of the file in different directories as described below is skipped, and *fullpath* ist assumed to contain the absolute directory and filename, instanciation would then be:

```
ccd = ColdChuckData(fullpath=<complete path + filename>)
```

logfile [Instance of an open text-file or None] If logfile points ot an open writeable text- file, error messages will be sent to the file, otherwise they will be printed to stdout.

The file `FTH200N_04_DiodeS_14_2015-11-05_4.cv` is looked for in directories according to the following order:

1.) In Python's current directory `./`

2.) In the directory provided by the parameter *directory*

3.) In `<md>/macroscopic/FTH200N_04_DiodeS_14/`

4.) In `<md>/macroscopic/FTH200N/FTH200N_04_DiodeS_14/`

`<md>` is the `m_data`-directory within `afs`. Under Linux it can be accessed from Python with `/run/user/<uuu>/gvfs/smb-share:server=uh2usnmserver,share=m_data`. `<uuu>` is the numerical user-id (e.g. 26356) of the current user. This requires the smb-shares to be mounted, for instance by using a call to the function "mount_gvfs() " when starting the program.

If the file is not found in either of these directories, Python will exit with an error message.

`get_Y` (*factor=1.0*)

This method returns `None` if not a .cv file. Otherwise:

Returns the complex admittance $Y = G_p + i\,\omega \cdot C_p$ directly from the values provided by the Agilent LRC meter (assuming the default setting to C_p-mode). For the norm $|Y|$ and the phase ϕ_Y, the function `get_Yabs_Phi` should be used. The complex impedance `Z` can be calcluated from that using Python's complex arithmetic $Z = 1/Y$.

factor [float, optional] multiplies the Y-values with *factor* when returning the complex array.

return [2-dimensional numpy array of complex] Y-values, one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

`get_Yabs_Phi` (*factor=1.0*)

This method returns `None` if not a .cv file. Otherwise:

Returns the norm $|Y|$ and the phase ϕ_Y of the complex admittance $Y = G_p + i\,\omega \cdot C_p$ directly from the values provided by the Agilent LRC meter (assuming the default setting to C_p-mode). Corresponding values $|Z|, \phi_Z$ for the impedance can easily be calcluated with $|Z| = 1.0/|Y|$ and $\phi_Z = \pi + \phi_Y$.

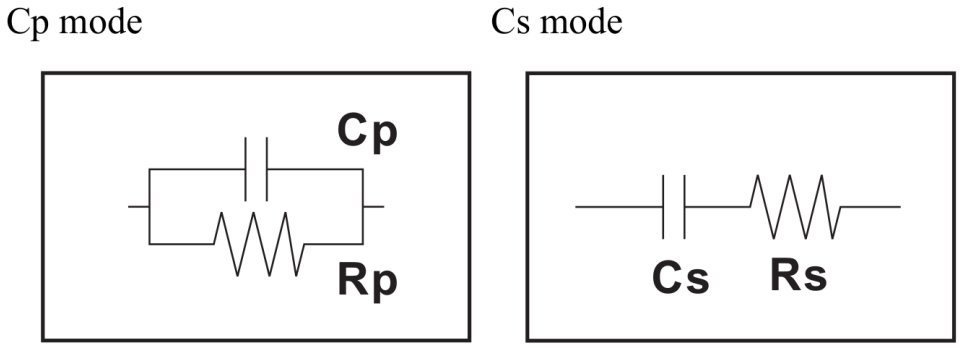
factor [float, optional] multiplies the $|Y|$-values with *factor* when returning the array, has no effect on the ϕ_Y-values.

return [Two 2-dimensional numpy arrays of real] $|Y|$-values and ϕ_Y-values, each having one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

`get_cp` (*factor=1.0*)

This method returns `None` if not a .cv file. Otherwise:

Returns the C_p-values (capacitance of an equivalent parallel circuit) from the data. These are directly provided by the Agilent LRC meter (assuming the default setting to C_p-mode).



factor [float, optional] multiplies the C_p-values with *factor* when returning the array. For instance a `get_cp(1e12)` will return capacitances in pF rather than F.

return [2-dimensional numpy array of float] C_p-values, multiplied by *factor*, one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_cs(*factor=1.0*)
This method returns `None` if not a .cv file. Otherwise:

Returns the C_s-values (capacitance of an equivalent serial circuit, see picture above). These are calculated using the C_p and G_p data together with the frequencies f from the meta-data line 'List of frequencies' with the following formula:

$$\omega = 2\pi f \; ; \quad C_s = \frac{G_p^2 + (\omega C_p)^2}{\omega^2 C_p}$$

factor [float, optional] multiplies the C_s-values with *factor* when returning the array. For instance a `get_cs(1e12)` will return capacitances in pF rather than F.

return [2-dimensional numpy array of float] C_s-values, one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_data(*i_range=None*)

return [numpy array of float] Data lines, interpreted as 2-dimesional array of floats. Note: After a previous call of set_voltage_index_range or set_voltage_range only the restricted part of the measured data is returned.

get_data_lines()

return [list of string] All lines in the data-file between 'BEGIN' and 'END' (the data)

get_file_ext()

return [string] The extension of the data-file (mostly .cv or .iv)

get_file_name()

return [string] The filename of the data-file as provided to the constructor.

get_filepath()

return [string] The path where the data-file was finally found

get_frequencies()
This method returns `None` if not a .cv file. Otherwise:

return [list of float] All frequencies as from the meta-data line 'List of frequencies' for further calculations. Example: `[490.0,1010.0,1900.0,5000.0]`

get_frequency_labels(*decimal=0*)
This method returns `None` if not a .cv file. Otherwise:

decimal [int, optional] Number of digits after the decimal point, defaults to zero.

return [list of string] All frequencies as from the meta-data line 'List of frequencies', reformated with <*decimal*> digits after the decimal point, together with a suitable unit (e.g. 'Hz', 'kHz', 'MHz'). The units are chosen so that the numbers will always be < 1000. Example: `['490 Hz','1 kHz','2 kHz','5 kHz']`. Useful for labeling data-lines in a plot.

get_gp(*factor=1.0*)
This method returns `None` if not a .cv file. Otherwise:

Returns the G_p-values (admittance of an equivalent parallel circuit, $G_p = \frac{1}{R_p}$, see figure above) directly from the data, as they are provided by the Agilent LRC meter (assuming the default setting to C_p-mode). Note: The G_p-values can be negative!

factor [float, optional] multiplies the G_p-values with *factor* when returning the array.

return [2-dimensional numpy array of float] G_p-values, one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_header()

return [string] The first line of the data-file

get_i_gr(*factor=1.0*)
This method returns `None` if not a .iv file. Otherwise:

Returns the guard ring current , optionally multiplied by *factor*, hence `get_i_gr(1e9)` will return the current in nA.

factor [float, optional] multiplies current-values with *factor* when returning the array.

return [list of float] The fourth column of the data array (the guard ring current). Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_i_pad(*factor=1.0*)
This method returns `None` if not a .iv file. Otherwise:

Returns the pad current , optionally multiplied by *factor*, hence `get_i_pad(1e9)` will return the current in nA.

factor [float, optional] multiplies current-values with *factor* when returning the array.

return [list of float] The third column of the data array (the pad current). Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_lines()

return [list of string] All lines in the data-file

get_meta_data(*with_line_number=False*)
This method builds a dictionary for all meta_data. The index is taken from lines beginning or ending with ':' excluding the ':' itself (e.g.'tester' or 'Annealing time [min]'). The content is given by subsequent line(s) (e.g. 'Michael') and is either a string or a list of strings. For instance `print(ccd.get_meta_data()['tester'])` may print `Michael`.

New in Version 0.9.9.3: Lines ending with a ':' are also recognized as an index. Empty lines are ignored. Parameters:

with_line_number [bool, optional] If True, the line number is added to the index (e.g. '07: tester').

If *with_line_number* = True, the complete meta-data can then be printed by:

```
for k in sorted(ccd.get_meta_data(True)):
    print(k, ccd.get_meta_data(True)[k])
```

return [dictionary] containing all meta-data as key / value pairs of strings

get_meta_lines()

return [list of string] All lines in the data-file containing meta-data (up to the first 'BEGIN')

get_rs(*factor=1.0*)
This method returns `None` if not a .cv file. Otherwise:

Returns the R_s-values (resistance of an equivalent serial circuit, see picture above). These are calculated using the C_p and G_p data together with the frequencies f from the meta-data line 'List of frequencies' with the following formula:

$$\omega = 2\pi f \; ; \quad R_s = \frac{1}{G_s} = \frac{G_p}{G_p^2 + (\omega C_p)^2}$$

Returns an array of floats with a column for each frequency and a line for each voltage.

factor [float, optional] multiplies the R_s-values with *factor* when returning the array.

return [2-dimensional numpy array of float] R_s-values, one line for each voltage, one column for each frequency. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_temps()

return [list of float] The second column of the data array (the temperature). Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

get_volts()

return [list of float] The first column of the data array (the voltage) Note: After a previous call of set_voltage_index_range or set_voltage_range only the restricted part of the measured data is returned.

set_voltage_index_range(*i_range=None*)

This method sets an index range for voltages in the data array so that the return values from **all subsequent calls** to the get_xxx - methods will be restricted to the given voltage indices.

Particularly all subsquent plots and fits will only use that index range. To reset the index range to full range use `set_voltage_index_range()`

i_range [list of exactly two integers] Sets the first and the last voltage index for data output, for instance `set_voltage_index_range([10,40])`

set_voltage_range(*v_min=None*, *v_max=None*)

This method calls `set_voltage_index_range(...)` (see asbove) to set an index range for voltages in the data array so that the return values from **all subsequent calls** to the get_xxx - methods will be restricted to the given voltage range. Particularly all subsquent plots and fits will only use that voltage range.

The indices are calculated by calling the method `v_index(...)`, hence it will work properly only if the voltages in the data-file are ordered monotonic an ascending order. (No hysteresis measurements!)

A valid call could be `set_voltage_range(5.0,100.0)`

To reset the voltage range to full range use `set_voltage_range()`

v_min [float, optional] Sets the lower voltage limit, defaults to the minimal voltage in the data array.

v_max [float, optional] Sets the upper voltage limit, defaults to the maximal voltage in the data array.

v_index(*volt_in*)

For a given voltage or a given list of voltages provided as parameter *volt_in*, this method searches for an item in the numpy-list of voltages from that data-file which is closest to that voltage(s) and either returns the index or a list of indices. The voltages in the data-file **must** be ordered monotonic in ascending order. (Hence not suitable for hysteresis measurements!)

volt_in [float or list of floats] Voltage to search for in the data-file.

return [int or list of int] Index (indices) of the item(s) in the data-file. Note: After a previous call of set_voltage_index_range or set_voltage_range the return values are restricted to that range.

CHAPTER

SIX

MODULE: TRANSIENT_TOOLS

The module transient_tools.py contains classes to read and analyse data-files from scopes. For usage, simply add the following code to your Python program:

```
from transient_tools import *
```

The module defines the global constants:

- MYHOME (my home directory)
- M_DATA_DIR (= '~/m_data/' which shold be s symlink to a gvfs share)
- RD_DATA_DIR (= '/scratch_nmsamba/' which shold be s symlink to a gvfs share)
- DATE_REGEX (regular expression for dates in ISO format)

The module contains the classes:

- class ScopeData

class ScopeData

class `transient_tools.`**`ScopeData`**(*filename=''*, *directory=''*)

This class reads the header and data-files `xxx.bin` and `xxx.Wfm.bin` created by the setups with scopes. Suitable methods to obtain data are provided.

To create an instance of ScopeData you can use as an exmaple:

```
sd = ScopeData('03_20GSs_400ns_49807_raw', 'myDataDir')
```

The parameter(s) provided to the constructor have the following meaning:

filename [string] Filename to read, e.g. `'xxx_raw'` note that this parameter is only the basename, actually read will then be the files `xxx_raw.bin` and `xxx_raw.Wfm.bin`

directory [string, optional] The directory which contains the data-file (if not it is not in the current directory).

`get_basename`()

return [string] The basename used for calculating the headerfilename and datafilename.

`get_data`()

Parse the data-file and return data. The data-file will only be parsed once, if self._data already contains data, these will be returned.

return [array of numpy-arrays] data ready for creating plots

`get_props`()

Returns all properties as an odered dictrionary

return [OrderedDict] Important properties as evaluatec from the xml Header-File.

get_rawdata()
Parse the data-file and return raw data. The data-file will only be parsed once, if self._rawdata already contains data, these will be returned.

return [array of numpy-arrays] raw data for advanced calucations

getpropval(*proplist*, *propname*)
Returns the property *propname* from a propertylist. Parameters:

proplist **list of element props** Usually created with xml tools.

propname **string** The property to search for.

return [string] Value of that prorperty. In case of numerical properties you need to typecast this value.

print_filenames()
Prints the filenames

return : None

print_props()
Prints all properties and values as an odered dictrionary

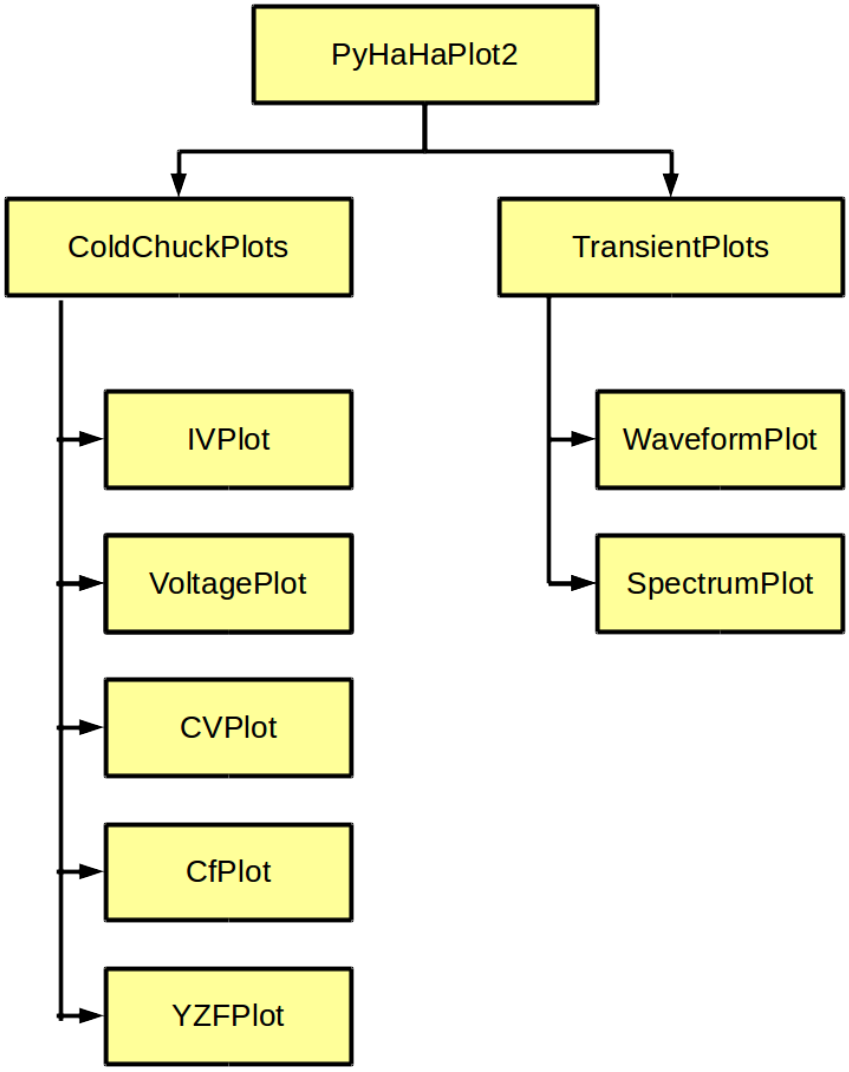
return : None

CHAPTER

SEVEN

MODULE: PYHAHA_PLOTS

The module pyhaha_plots.py only contains the class PyHaHaPlot2 which is the superclass class for other plot classes. The picture below illustrates the class hierarchy:



It is not intended to instanciate PyHaHaPlot2 directly.

class PyHaHaPlot2

class pyhaha_plots.**PyHaHaPlot2**(*plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

This class sets the plotting parameters and defines some defaults for inherited plot classes. The plotting parameters are read from a file (default filename is pyhaha_plot_defaults) which should be in the current directory. They can be adjusted with a dictionary provided as parameter *params* the constructor. It is not intended to use this module directly. If the parameter file is not found, Python will exit with an error message.

plotparams [dictionary, optional] Parameters to be provided to plt.rcParams.update(...), to overwrite some paramters loaded from pyhaha_plot_defaults for this specific plot. Check the file matplotlibrc.orig for the format and meaning of the entries.

plotparams_file [string, optional] If empty, the parameters dictionary is read from the file pyhaha_plot_defaults, which has to be present in the current directory. Otherwise you can define your onw filename. In any case, this file has to exist in the current directory.

get_plotparams()

return [dictionary] plot parameters as currently stored in this class

load_plotparams(*plotparams_file='pyhaha_plot_defaults'*)

Sets the default parameters to be provided to plt.rcParams.update(...), and stores them in a class variable. Check the file matplotlibrc.orig for the format and meaning of the entries.

plotparams_file [string, optional] If empty, the dictionary is read from the file pyhaha_plot_defaults, otherwise you can define your plot parameters in a file of your own.

update_plotparams(*plotparams*)

params [dictionary] parameters to update the plt.rcParams.update(...), and the parameters stored in this class. These parameters will be used for all subsequent plots. Check the file matplotlibrc.orig for the format and meaning of the entries.

CHAPTER

EIGHT

MODULE: COLD_CHUCK_PLOTS

The module data_plots.py contains classes to create standard data plots, mainly from from CV- and IV-measurements of silicon particle detectors created by the cold chuck setup. For usage, simply add the following code to your Python program:

```
from cold_chuck_plots import *
```

The module contains the classes:

- class ColdChuckPlots (PyHaHaPlot2)
- class IVPlot (ColdChuckPlots)
- class VoltagePlot (ColdChuckPlots)
- class CVPlot (ColdChuckPlots)
- class CfPlot (ColdChuckPlots)
- class YZfPlot (ColdChuckPlots)

class ColdChuckPlots

class cold_chuck_plots.**ColdChuckPlots**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *pyhaha_plots.PyHaHaPlot2*

This class sets the plotting parameters and defines some defaults for other plot classes. The plotting parameters are read from a file (default filename is pyhaha_plot_defaults) which should be in the current directory. They can be adjusted with a dictionary provided as parameter *params* the constructor. If the parameter file is not found, Python will exit with an error message,

ccds [object or list of objects] One or several instances of class ColdChuckData, all should be based on either .cv or .iv - files, otherwise Python will exit with an error message.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

make_title()

return [string] The default plot title, based on the filename of the ccd object. If a list of filenames is provided to the constructor, the title is based on the device name of the first file.

save_plot(*filename=''*)

Save the current plot in a file (mostly a .pdf-file).

filename [string, optional] If empty, the filename is calcuated with ccd.get_file_name() or - if a list of filenames is provided to the constructor - with the device name of the first file.

class IVPlot

class cold_chuck_plots.**IVPlot**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *cold_chuck_plots.ColdChuckPlots*

This class creates simple IV-Plot(s) from measured data. The parameter provided to the constructor has the following meaning:

ccds [object or list of objects] One or several instances of class ColdChuckData, all should be based on .iv - files, otherwise Python will exit with an error message. A plot is created for each of of the ColdChuckData objects.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

make_plot(*labels=None*, *y_factor=1000000000.0*, *y_unit='nA'*, *with_GR=True*)

Create a standard IV-plot. Note: After a previous call of set_voltage_index_range or set_voltage_range the x-axis is restricted to that range.

labels [string or list of strings, optional] If the class is instanciated with several ColdChuckData objects, each of the plots is labeled with a matching label form that list.

y_factor [float, optional] Multiplies the currents with *y_factor* before creating the plot, defaults to 1.0e9.

y_unit [string, optional] Sets the unit for the y-axis, should be based on the inverse of *y_factor*, defaults to 'nA'. The string can contain Latex Code, a call to this method could be for instance `make_plot(1.0e6,r"$\mu$A")`

with_GR [bool, optional] If True, include IV for the guard ring. Default is True

class VoltagePlot

class cold_chuck_plots.**VoltagePlot**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *cold_chuck_plots.ColdChuckPlots*

This class creates simple Voltage-Plot(s) from measured data. The parameter provided to the constructor has the following meaning:

ccds [object or list of objects] One or several instances of class ColdChuckData. A plot is created for each of of the ColdChuckData objects.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

make_plot(*y_unit='[V]'*)

Create a standard plot voltage versus index. Note: After a previous call of set_voltage_index_range or set_voltage_range the x-axis is restricted to that range.

y_unit [string, optional] Sets the unit for the y-axis, defaults to '[V]'.

class CVPlot

class cold_chuck_plots.**CVPlot**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *cold_chuck_plots.ColdChuckPlots*

This class creates a simple CV-Plot from measured data. The parameter provided to the constructor has the following meaning:

ccds [object or list of objects] One or several instances of class ColdChuckData, all should be based on .cv - files, otherwise Python will exit with an error message. A plot is created for each of of the ColdChuckData objects.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

make_plot(*labels=None*, *y_factor=1e-24*, *y_unit='pF^{-2}'*)

Create a standard CV-plot by plotting $\frac{1}{C^2}$ versus V_{Bias} . Note: After a previous call of set_voltage_index_range or set_voltage_range the x-axis is restricted to that range.

labels [string or list of strings, optional] If the class is instanciated with several ColdChuckData objects, each of the plots is labeled with a matching label form that list.

y_factor [float, optional] Multiplies the $\frac{1}{C^2}$ values with *y_factor* before creating the plot, defaults to 1.0e-24.

y_unit [string, optional] Sets the unit for the y-axis, should be based on the inverse of *y_factor*, defaults to pF^{-2} . The string can contain Latex Code, a call to this method could be for instance `make_plot(1.0,r"$F^{-2}$")`

class CfPlot

class cold_chuck_plots.**CfPlot**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *cold_chuck_plots.ColdChuckPlots*

This class creates a simple Cf-Plot from measured data. The parameter provided to the constructor has the following meaning:

ccds [object or list of objects] One or several instances of class ColdChuckData, all should be based on .cv - files, otherwise Python will exit with an error message. A plot is created for each of of the ColdChuckData objects.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class PyHaHaPlot2. See documentation of that class.

make_plot(*volts*, *labels=None*, *y_factor=1000000000000.0*, *y_unit='pF'*)

Create a standard Cf-plot by plotting C_p versus frequency for a given voltage or list of voltages. Note: After a previous call of set_voltage_index_range or set_voltage_range the voltages are restricted to that range.

volts [float or list of floats] Bias voltage(s) to create the plot for. A call could be for instance `make_plot(5.0)` or `make_plot([5.0,50.0,200.0])`

labels [string or list of strings, optional] If the class is instanciated with several ColdChuckData objects, each of the plots is labeled with a matching label form that list.

y_factor [float, optional] Multiplies the C_p values with *y_factor* before creating the plot, defaults to 1.0e12.

y_unit [string, optional] Sets the unit for the y-axis, should be based on the inverse of *y_factor*, defaults to `pF` .

class YZfPlot

class `cold_chuck_plots.`**`YZfPlot`**(*ccds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *cold_chuck_plots.ColdChuckPlots*

This class creates a simple Yf-Plot or Zf-Plot (absolute value of the admittance or impedance) versus frequency from measured data. The parameter provided to the constructor has the following meaning:

ccds [object or list of objects] One or several instances of class ColdChuckData, all should be based on .cv - files, otherwise Python will exit with an error message. A plot is created for each of of the ColdChuckData objects.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

`make_plot`(*volts*, *labels=None*, *plot_type='Y'*)

Create a standard Yf-plot or Zf-plot by plotting the admittance $|Y|$ or impedance $|Z|$ versus frequency for a given voltage or list of voltages. Note: After a previous call of set_voltage_index_range or set_voltage_range the voltages are restricted to that range.

volts [float or list of floats] Bias voltage(s) to create the plot for. A call could be for instance `make_plot(5.0)` or `make_plot([5.0,50.0,200.0])`

labels [string or list of strings, optional] If the class is instanciated with several ColdChuckData objects, each of the plots is labeled with a matching label form that list.

plot_type [string, optional] Either 'Y', 'Z' or 'B'. Determines whether the admittance $|Y|$ (*plot_type* = 'Y'), or the impedance $|Z|$ (*plot_type* = 'Z') or both are plotted (*plot_type* = 'B'). Note the different y-scales of both are plotted. Default is 'Y'.

CHAPTER

NINE

MODULE: TRANSIENT_PLOTS

The module transient_plots.py contains classes to create standard data plots from scope data. **Warning:** Some of the classes in this module require numpy version $\geq$ 1.11.

For usage, simply add the following code to your Python program:

```
from transient_plots import *
```

The module contains the classes:

- class TransientPlots (PyHaHaPlot2)
- class WaveformPlot (TransientPlots)
- class SpectrumPlot (TransientPlots)

class TransientPlots

class `transient_plots.`**`TransientPlots`**(*sds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *pyhaha_plots.PyHaHaPlot2*

This class sets the plotting parameters and defines some defaults for other plot classes. The plotting parameters are read from a file (default filename is `pyhaha_plot_defaults`) which should be in the current directory. They can be adjusted with a dictionary provided as parameter *params* the constructor. If the parameter file is not found, Python will exit with an error message,

sds [object or list of objects] All have to be instances of class ScopeData otherwise Python will exit with an error message. **Note**: The current version only supports one instance.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

`make_title`()

return [string] The default plot title, based on the filename of the sds object. If a list of filenames is provided to the constructor, the title is based on the device name of the first file.

`save_plot`(*filename=''*)

Save the current plot in a file (mostly a .pdf-file).

filename [string, optional] If empty, the filename is calcuated with ccd.get_file_name() or - if a list of filenames is provided to the constructor - with the device name of the first file.

class WaveformPlot

class transient_plots.**WaveformPlot**(*sds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *transient_plots.TransientPlots*

This class creates a waverform plot from measured data. The parameter provided to the constructor has the following meaning:

sds [object or list of objects] One or several instances of class ScopeData **Note**: The current version only supports one instance.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

make_plot(*data_slot=None*, *data_max=100*, *x_factor=1000000000.0*, *x_unit='ns'*, *y_factor=1000.0*, *y_unit='mV'*)

Create a ROHDE & SCHWARZ Multiwafeform Plot.

data_max [integer, optional] Only plot data with data-indexes up to *data_max*, default is 100.

data_slot [integer, optional] If set: Only plot data[data_slot], default is None, full spectrum will be plotted.

x_factor [float, optional] Multiplies the times with *x_factor* before creating the plot, defaults to 1.0e9.

x_unit [string, optional] Sets the unit for the x-axis, should be based on the inverse of *x_factor*, defaults to 'ns'. The string can contain Latex Code, a call to this method could be for instance `make_plot(x_factor=1.0e6,x_unit=r"$\mu$s")`.

y_factor [float, optional] Multiplies the voltages with *y_factor* before creating the plot, defaults to 1.0e3.

y_unit [string, optional] Sets the unit for the y-axis, should be based on the inverse of *y_factor*, defaults to 'mV'. The string can contain Latex Code, a call to this method could be for instance `make_plot(x_factor=1.0e6,x_unit=r"$\mu$V")`.

class SpectrumPlot

class transient_plots.**SpectrumPlot**(*sds*, *plotparams=None*, *plotparams_file='pyhaha_plot_defaults'*)

Bases: *transient_plots.TransientPlots*

This class creates a spectrum plot from measured data. The parameter provided to the constructor has the following meaning:

sds [object or list of objects] One or several instances of class ScopeData **Note**: The current version only supports one instance.

plotparams [dictionary, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

plotparams_file [string, optional] This parameter is passed to the constructor of the class `PyHaHaPlot2`. See documentation of that class.

make_plot(*idx_min=2200*, *idx_max=3500*, *x_factor=1000000000.0*, *x_unit='ns'*, *y_factor=1000.0*, *y_unit='mV'*)

Create a spectrum plot.

idx_min [integer, optional] Only plot data with an rawdata-index starting at *idx_min*, default is 2200.

idx_max [integer, optional] Only plot data with an rawdata-index ending at *idx_max*, default is 3500.

x_factor [float, optional] Multiplies the times with *x_factor* before creating the plot, defaults to 1.0e9.

x_unit [string, optional] Sets the unit for the x-axis, should be based on the inverse of *x_factor*, defaults to 'ns'. The string can contain Latex Code, a call to this method could be for instance `make_plot(x_factor=1.0e6,x_unit=r"$\mu$s")`.

y_factor [float, optional] Multiplies the voltages with *y_factor* before creating the plot, defaults to 1.0e3.

y_unit [string, optional] Sets the unit for the y-axis, should be based on the inverse of *y_factor*, defaults to 'mV'. The string can contain Latex Code, a call to this method could be for instance `make_plot(x_factor=1.0e6,x_unit=r"$\mu$V")`.

CHAPTER

TEN

MODULE: FILE_UTILS

The module file_utils.py contains some general utility funktions. For usage, simply add the following code to your Python program:

```
from file_utils import *
```

The module contains the utility functions:

- function mount_gvfs
- function date_stamp
- function datetime_stamp
- function strings_to_ints
- function log_to_file
- function logfile
- function collect_files
- function linuxpath_to_win
- function winpath_to_linux

The module defines the global constants:

- MYHOME (my home directory)
- M_DATA_DIR (= '~/m_data/' which shold be s symlink to a gvfs share)
- RD_DATA_DIR (= '/scratch_nmsamba/' which shold be s symlink to a gvfs share)
- DATE_REGEX (regular expression for dates in ISO format)

function mount_gvfs

`file_utils.`**`mount_gvfs`**`()`

This utility function calls the shell-script `~/bin/automount` which mounts shares on `uh2usnmserver`. No error will be sent if the script does not exist or the shares are already mounted. The content of this script could be for instance:

```
#!/bin/bash
# Script  ~/bin/automount
gvfs-mount smb://uh2usnmserver/m_data/
gvfs-mount smb://uh2usnmserver/scratch_nmsamba/
```

function date_stamp

file_utils.**date_stamp**()
Returns the current date, e.g. '2017-11-30'

return [string] date in ISO-Format

function datetime_stamp

file_utils.**datetime_stamp**()
Returns the current date, e.g. '2017-11-30 09:18:22'

return [string] date and time in ISO-Format

function strings_to_ints

file_utils.**strings_to_ints**(*in_string*)
This funcion converts a string with comma separated ints into a list of integers. Parameters:

in_string [string] input string

return [list of integers] converted string

function log_to_file

file_utils.**log_to_file**(*msg*, *logfile=None*)
This utility prints a message or sends it logfile which has to be opened before. Parameters:

msg [string] The error message to appear in the the logfile or on stdout.

logfile [instance of an open text-file or None] If logfile points to an open writeable text- file, error messages will be sent to that file, otherwise they will be printed to stdout.

function logfile

file_utils.**logfile**(*file_name*, *title*, *lines*, *separator='\t'*)
This function creates a file for logging csv data. Parameters:

file_name [string] Relative path of the output file

title [string or list of strings] The first line within the file

lines [list of strings or list of list of strings] The content, each string will be a separate line. If the line is a list of strings, each inner string will be printed as an item separated by *separator* (default ' ').

function collect_files

file_utils.**collect_files**(*base_dir*, *min_date='1990-01-01'*, *valid_ext=None*)
This funcion iterates recursively through all directories below *base_dir* and collects the filenames. Only files with a modification date later than *min_date* are collected. If the *fullname* contains 'macroscopic/', name-part before 'macroscopic/' will be removed.

Caveat! A call `collect_files(M_DATA_DIR + 'macroscopic/')` will run for 9 minutes an return a list of 163 463 files (total size = 224 663 331 076 Bytes ~ 209 GByte).

Parameters:

base_dir [string] Files below *base_dir* are scanned. Caution: If *base_dir* = M_DATA_DIR + 'macroscopic/' the reslut will be a list of ~80.000 items.

min_date [string] Files with a modification date < *min_date* will be ignored. Date format has to be YYYY-MM-DD.

valid_ext [list of strings or None] If *valid_ext* is not None, only files with an extesnion endinng with one of the entries in *valid_ext* will be processed. A call could for instance be `(...,valid_ext=['.cv','.iv'])`.

return [list of *items*] An *item* is a list of file properties: modification date [string], file-size [int], file-extension [string], filename [string] and fullname (= name including path below *base_dir*) [string]

function linuxpath_to_win

`file_utils.`**`linuxpath_to_win`**(*linux_path*, *drive=''*, *sql=False*)

This converts a Linux path with forward slashes to a windows path with backslashes.

Parameter: *linux_path* : string

Full path in Linux format, input

drive [string, optional] Optional a Windows drive to preceed the Windows path, could be for instance 'j:'.

sql [bool, optional] If True, the returned path can used in SQL statements. In this case the backslashes will to be escaped by double backslashes.

return [string] The windows path, optionally with drive, optionally with escaped backslases.

function winpath_to_linux

`file_utils.`**`winpath_to_linux`**(*winpath*)

This is a utility function to convert a Windows file path as stored in the column 'File', table 'Measurement' of the old database 'radhard' to a Linux path. For example `winpath_to_linux('j:\macroscopic\w6\w6-pncv-2\w6-pncv-2_2013-07-26_2.cv')` will return `M_DATA_DIR/macroscopic/w6/w6-pncv-2/w6-pncv-2_2013-07-26_2.cv'`. `winpath_to_linux` will try several linux directories for locating the file.

Parameter:

winpath [string] Filename in Windows format

return [string] Full path in Linux format where the file can be found. The function will return `None` if the file cannot automatically be found.

c

f

p

t

C

D

F

G

I

L

M

P

S

T

U

V

W

Y