Physics Lab

Release 1.3.1

Martin Brajer

Feb 27, 2022

CONTENTS:

Physics Lab is a free and open-source python package. Helps with evaluation of common experiments.

Source code can be found on Github.

CHAPTER

ONE

FEATURES

- Van der Pauw & Hall method
- Magnetization measurements
- Batch measurement evaluation
- Plot the data / analysis results
- Using pandas.DataFrame

CHAPTER

TWO

INSTALLATION

```
pip install physicslab
```

For usage examples see *Get started* section.

CHAPTER

THREE

COMPLIANCE

Versioning follows Semantic Versioning 2.0.0.
Following PEP8 Style Guide coding conventions.
Testing with `unittest` and pycodestyle.
Using Python 3 (version >= 3.6).

CHAPTER

FOUR

LICENSE

Physics Lab is licensed under the MIT license.

4.1 Physics Lab overview

Physics Lab is a free and open-source python package. Helps with evaluation of common experiments.

Source code can be found on Github.

4.1.1 Features

- Van der Pauw & Hall method
- Magnetization measurements
- Batch measurement evaluation
- Plot the data / analysis results
- Using pandas.DataFrame

4.1.2 Installation

```
pip install physicslab
```

For usage examples see *Get started* section.

4.1.3 Compliance

Versioning follows Semantic Versioning 2.0.0.
Following PEP8 Style Guide coding conventions.
Testing with `unittest` and pycodestyle.
Using Python 3 (version >= 3.6).

4.1.4 License

Physics Lab is licensed under the MIT license.

4.2 Get started

Use cases.

4.2.1 Load data

```
# Read *.ods or excel (spreadsheet).
data = pd.read_excel('D:\...\data2.ods', sheet_name=0,
                     skiprows=1).dropna(how='all')
data.name = 'data2'
```

4.2.2 Experiment subpackage

You can either use subpackages directly (`physicslab.experiment.van_der_pauw`) or utilize the following batch function.

```
# Example: Van der Pauw
import pandas as pd

def load(filename):
   data = pd.read_csv(filename + '.csv')
   data.name = filename
   return data

thickness = 1.262e-6  # meters
samples = ['sample#1', 'sample#2', ...]
data_list = [load(sample) for sample in samples]

results = physicslab.experiment.process(
   data_list,
   by_module=physicslab.experiment.van_der_pauw,
   thickness=thickness
)
print(results)
```

```
           sheet_resistance  ratio_resistance  sheet_conductance  resistivity␣
↪conductivity
units       ohms per square                 1      1/ohms square        ohm m       1/
↪ohm/m
sample#1       1.590823e+05          1.168956       6.286055e-06     0.200762      4.
↪981026
sample#2       1.583278e+05          1.185031       6.316009e-06     0.199810      5.
↪004762
...
```

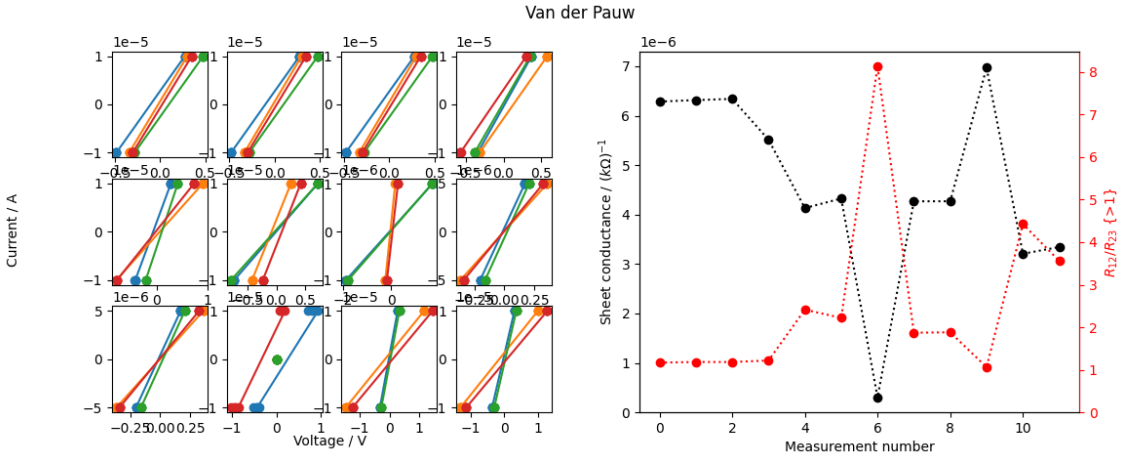
4.2.3 Van der Pauw

Handling Geometry enum.

```
def get_geometry(orientation, direct):
    """
    :param int orientation: Contacts rotation in multiples of 90°.
    :param bool direct: Contacts counter-clockwise (True) or not.
    """
    geometry = van_der_pauw.Geometry.R1234  # Default configuration.
    geometry = geometry.shift(number=orientation)
    if not direct:
        geometry = geometry.reverse_polarity()
    return geometry
```

Plotting.

```
from physicslab.experiment import van_der_pauw

data_list = load(sample_name)  # Custom function.
results = physicslab.experiment.process(data_list, by_module=van_der_pauw)
van_der_pauw.plot(data_list, results)
plt.show()
```



4.2.4 Magnetism type

```
results = physicslab.experiment.magnetism_type.process(measurement)
print(results)

cols = physicslab.experiment.magnetism_type.Columns
B = measurement[cols.MAGNETICFIELD]
plt.plot(B, measurement[cols.MAGNETIZATION], 'ko')  # Original data.
plt.plot(B, measurement[cols.DIAMAGNETISM], 'r-')  # Separated DIA contribution.
plt.plot(B, measurement[cols.FERROMAGNETISM], 'b-')  # Separated FM contribution.
plt.plot(B, measurement[cols.RESIDUAL_MAGNETIZATION], 'g-')  # Residual (unseparated)␣
↪data.
plt.show()
```

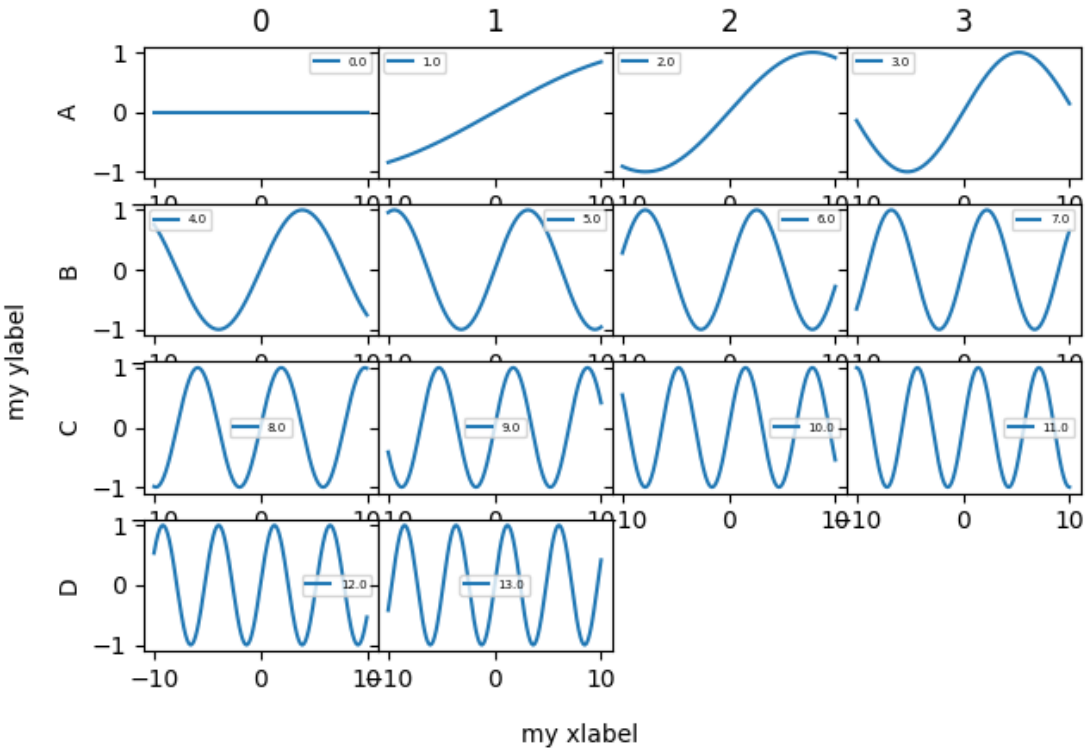
4.2.5 curves.Line

```
line1 = Line(3, -2)  # Line: y = 3 - 2x
line2 = Line(slope=2)  # Line: y = 0 + 2x
line1(4.3)  # -5.6
line1 - 5.3 + 2.4 * line2  # Line: y = -2.3 + 2.8x
line1.zero()  # 1.5
Line.Intersection(line1, line2)  # (0.75, 1.5)
```

4.2.6 ui.plot_grid & utility.squarificate

```
import matplotlib.pyplot as plt
import numpy as np
import physicslab

x = np.linspace(-10, 10, num=1000)

def plot_value(ax, value):  # Sine.
   ax.plot(x, np.sin(x * value / 10), label=value)

def alphabet(num):  # ['A', 'B', ...]
   return [(chr(ord('A') + i)) for i in range(num)]

data = np.arange(14, dtype=float)  # E.g. a list of measurements.
data = physicslab.utility.squarificate(data)  # Squarish 2D array distribution.
df = pd.DataFrame(data, index=alphabet(data.shape[0]))  # Naming.
df.name = 'My title'

print(df)
physicslab.ui.plot_grid(
   df, plot_value, xlabel='my xlabel', ylabel='my ylabel',
   subplots_adjust_kw={'hspace': 0}, sharey=True, legend_size=5)
```

```
      0     1     2     3
A   0.0   1.0   2.0   3.0
B   4.0   5.0   6.0   7.0
C   8.0   9.0  10.0  11.0
D  12.0  13.0   NaN   NaN
```



4.2.7 Show images

```
import matplotlib.image as mpimg

# Show pictures (like SEM images). Parameter value is then e.g. a filename.
def plot_value(ax, value):
    img = mpimg.imread(filepath)
    ax.imshow(img, cmap='gray')
```

4.3 Library

Physics Lab package helps with evaluation of common experiment data.

Source code (and link to the documentation) can be found at GitHub

4.3.1 Experiment subpackages

Main

Modules for particular experiments and general functions.

physicslab.experiment.**UNITS = 'units'**
pandas.Dataframe.attrs tag.

physicslab.experiment.**process**(*data_list: list*, *by_module: module*, ***kwargs*) → pandas.core.frame.DataFrame
Genereal process function calling appropriate *process()* function from selected experiment module.

Include units attribute.

Parameters

- **data_list** (*list (pandas.DataFrame)*) – List of measurements, which are passed to the appropriate *process()* function
- **by_module** (experiment submodule) – Submodule of experiment by which the individual measurements are to be processed
- **kwargs** – Additional keyword arguments are forwarded to by_module.process() function

Returns Collection of results indexed by measurement's name

Return type pandas.DataFrame

physicslab.experiment.**print_**(*df: Union[pandas.core.frame.DataFrame, pandas.core.series.Series]*) → None
Print the data including units row if available.

Does not change the input DataFrame.
If Series is supplied, it's printed in the DataFrame format.

Parameters **df** (*pandas.DataFrame or pandas.Series*) – Data to be printed

Curie temperature

Curie temperature.

Find Curie temperature from magnetization vs temperature measurement.

physicslab.experiment.curie_temperature.**process**(*data*)
Bundle method.

Parameter data must include temperature and magnetization. See *Columns* for details and column names.

Supplying *None* for data returns pandas.Series of the same columns with values being units.

Parameters **data** (*pandas.DataFrame*) – Measured data. If None, return units instead

Returns Derived quantities listed in *Columns.process()* or units

Return type pandas.Series

class physicslab.experiment.curie_temperature.**Columns**
Bases: *physicslab.utility._ColumnsBase*

Column names.

classmethod mandatory()
Get the current values of the mandatory column names.

Return type set(str)

classmethod process()
Get the current values of the *process()* output column names.

Return type lits(str)

class physicslab.experiment.curie_temperature.**Measurement**(*data*)
Magnetization vs temperature measurement.

Parameters **data** (*pandas.DataFrame*) – Magnetization and temperature data.

Raises **ValueError** – If `data` is missing a mandatory column

analyze(*p0=None*)
Find Curie temperature.

Parameters **p0** (*tuple, optional*) – Initial guess of spontaneous magnetization curve parameters. If None, the parameters will be estimated automatically, defaults to None

Returns Curie temperature

Return type float

fit(*T*, *M*, *p0=None*, *high_temperature_focus=False*)
Fit spontaneous magnetization curve to the data.

Save the fit into `Columns.HIGHTEMPERATUREFIT`.

Parameters

- **T** (*numpy.ndarray*) – Temperature
- **M** (*numpy.ndarray*) – Magnetization
- **p0** (*tuple, optional*) – Initial guess of spontaneous magnetization curve parameters. If None, the parameters will be estimated automatically, defaults to None
- **high_temperature_focus** (*bool, optional*) – Give high temperature data more weight, defaults to False

Returns Curie temperature, fit

Return type tuple(float, numpy.ndarray)

physicslab.experiment.curie_temperature.**plot**(*data_list*)
Simple plot data and fit for all measurement at once.

Parameters **data_list** (*list[pandas.DataFrame]*) –

Returns Same objects as from `matplotlib.pyplot.subplots()`

Return type tuple[Figure, Axes]

Hall module

Hall measurement.

Induced voltage perpendicular to both current and magnetic field.

physicslab.experiment.hall.**process**(*data*, *thickness=None*, *sheet_resistance=None*)

Bundle method.

Parameter `data` must include voltage, current and magnetic field. See *Columns* for details and column names.

The optional parameters allows to calculate additional quantities: *concentration* and *mobility*.

Supplying *None* for `data` returns `pandas.Series` of the same columns with values being units.

Parameters

- **data** (*pandas.DataFrame or None*) – Measured data. If None, return units instead
- **thickness** (*float, optional*) – Sample dimension perpendicular to the plane marked by the electrical contacts, defaults to None
- **sheet_resistance** (*float, optional*) – Defaults to None

Returns Derived quantities listed in *Columns.process()* or units

Return type pandas.Series

class physicslab.experiment.hall.**Columns**

Bases: *physicslab.utility._ColumnsBase*

Column names.

classmethod mandatory()

Get the current mandatory column names.

Return type set(str)

classmethod process()

Get the current values of the *process()* output column names.

Return type lits(str)

class physicslab.experiment.hall.**Measurement**(*data*)

Hall measurement.

Parameters **data** (*pandas.DataFrame*) – Voltage-current-magnetic field triples.

Raises **ValueError** – If `data` is missing a mandatory column

analyze()

Compute sheet density and determine conductivity type.

Returns Sheet density, conductivity type, fit residual

Return type tuple(float, str, float)

physicslab.experiment.hall.**plot**(*data_list*, *results*)

Plot all the measurements in a grid

Parameters

- **data_list** (*list[pandas.DataFrame]*) –
- **results** (*pandas.DataFrame*) – Analysis data from *physicslab.experiment.process()*

Returns Same objects as from `matplotlib.pyplot.subplots()`

Return type tuple[Figure, numpy.ndarray[Axes]]

Magnetism type module

Magnetization measurement.

Separate diamagnetic and ferromagnetic contributions.

physicslab.experiment.magnetism_type.**process**(*data*, *diamagnetism=True*, *ferromagnetism=True*)

Bundle method.

Parameter `data` must include magnetic field and magnetization. See *Columns* for details and column names.

Output `ratio_DM_FM` compares max values - probably for the strongest magnetic field.

Supplying *None* for `data` returns `pandas.Series` of the same columns with values being units.

Parameters

- **data** (*pandas.DataFrame or None*) – Measured data. If None, return units instead
- **diamagnetism** (*bool, optional*) – Look for diamagnetism contribution, defaults to True
- **ferromagnetism** (*bool, optional*) – Look for ferromagnetism contribution, defaults to True

Returns Derived quantities listed in *Columns.process()* or units

Return type pandas.Series

class physicslab.experiment.magnetism_type.**Columns**

Bases: *physicslab.utility._ColumnsBase*

Column names.

classmethod mandatory()

Get the current mandatory column names.

Return type set(str)

classmethod process()

Get the current values of the *process()* output column names.

Return type lits(str)

class physicslab.experiment.magnetism_type.**Measurement**(*data*)

Magnetization vs magnetic field measurement.

Copy magnetization column as `Columns.RESIDUAL_MAGNETIZATION`, so individual magnetic effects can be subtracted.

Parameters **data** (*pandas.DataFrame*) – Magnetic field and magnetization data.

Raises **ValueError** – If `data` is missing a mandatory column

diamagnetism(*from_residual=False*)

Find diamagnetic component of overall magnetization.

Simulated data are subtracted from residue column (making it centred).

Parameters **from_residual** (*bool, optional*) – Use residual data instead of the original data, defaults to False

Returns Magnetic susceptibility and magnetization offset

Return type tuple

ferromagnetism(*from_residual=False*, *p0=None*)
Find ferromagnetic component of overall magnetization.

Simulated data are subtracted from residue column.
Hysteresis loop shape can be found in *magnetic_hysteresis_loop()*.

Parameters

- **from_residual** (*bool, optional*) – Use residual data instead of the original data, defaults to False
- **p0** (*tuple, optional*) – Initial guess of hysteresis loop parameters. If None, the parameters will be estimated automatically, defaults to None

Returns Saturation, remanence and coercivity

Return type tuple

physicslab.experiment.magnetism_type.**plot**(*data*)
Plot single magnetization measurement separated data

Parameters **data** (*list[pandas.DataFrame]*) –

Returns Same objects as from `matplotlib.pyplot.subplots()`

Return type tuple[Figure, Axes]

Profilometer

Profile measurement.

physicslab.experiment.profilometer.**process**(*data*, ***kwargs*)
Bundle method.

Parameter `data` must include position and height. See *Columns* for details and column names.

Output *histogram* column (type *Histogram*) stores histogram data and fit data.

Supplying *None* for `data` returns `pandas.Series` of the same columns with values being units.

Parameters

- **data** (*pandas.DataFrame*) – Measured data. If None, return units instead
- **kwargs** – All additional keyword arguments are passed to the *Measurement.analyze()* call.

Returns Derived quantities listed in *Columns.process()* or units

Return type pandas.Series

class physicslab.experiment.profilometer.**Columns**
Bases: *physicslab.utility._ColumnsBase*

Column names.

classmethod mandatory()
Get the current mandatory column names.

Return type set(str)

classmethod process()
Get the current values of the *process()* output column names.

Return type lits(str)

class physicslab.experiment.profilometer.**Measurement**(*data*)
Profile measurement.

Parameters **data** (*pandas.DataFrame*) – Position and height data.

Raises **ValueError** – If `data` is missing a mandatory column

class Histogram(*bin_centers*, *count*, *x_fit*, *y_fit*)
Histogram and fit data.

analyze(*zero=0*, *background_degree=None*, *edge_values=None*)
Analyze

Parameters

- **zero** (*int, optional*) – Assumed position of the main peak, defaults to 0
- **background_degree** (*int or None, optional*) – Degree of polynomial used to subtract background. None to disable background subtraction, defaults to None
- **edge_values** (*tuple(float, float), optional*) – Background subtraction will happen inside those bounds. None means left half of the positions, defaults to None

Returns Expected values, variances, amplitudes, FWHMs, thickness and histogram. The last one is of type *Histogram*) and store histogram data and fit data.

Return type tuple

static background(*pos*, *height*, *background_degree*, *edge_values*)
Find best fit given the constrains.

Parameters

- **pos** (*numpy.ndarray*) – Position
- **height** (*numpy.ndarray*) – Height
- **background_degree** (*int*) – Degree of polynomial used
- **edge_values** (*tuple(float, float)*) – Background subtraction will happen inside those bounds

Returns Background

Return type numpy.ndarray

physicslab.experiment.profilometer.**plot**(*data*, *results*)
Plot both the data analysis parts and the results histogram.

Units are shown in nanometers.

Parameters

- **data** (*pandas.DataFrame*) –
- **results** (*pandas.Series*) – Analysis data from *physicslab.experiment.process()*

Returns Same objects as from `matplotlib.pyplot.subplots()`

Return type tuple[Figure, numpy.ndarray[Axes]]

Van der Pauw module

Van der Pauw resistivity measurement.

Four-point measurement bypass resistance of ohmic contacts.
To find resistivity from sheet resistance, use
physicslab.electricity.Resistivity.from_sheet_resistance method.
Pay special attention to enum *Geometry*.

physicslab.experiment.van_der_pauw.**process**(*data*, *thickness=None*)
Bundle method.

Parameter `data` must include geometry voltage and current. See *Columns* for details and column names.

The optional parameter allows to calculate additional quantities: *resistivity* and *conductivity*.

Supplying *None* for `data` returns `pandas.Series` of the same columns with values being units.

Parameters

- **data** (*pandas.DataFrame or None*) – Measured data. If None, return units instead
- **thickness** (*float, optional*) – Sample dimension perpendicular to the plane marked by the electrical contacts, defaults to None

Returns Derived quantities listed in *Columns.process()* or units

Return type pandas.Series

class physicslab.experiment.van_der_pauw.**Solve**
Van der Pauw formula and means to solve it.

static implicit_formula(*Rs*, *Rh*, *Rv*)
Van der Pauw measurement implicit function.

The function reads:
$func(R_s) = exp(-\pi R_v/R_s) + exp(-\pi R_h/R_s) - 1.$
This function's roots give the solution.

Parameters

- **Rs** (*float*) – Sheet resistance. Independent variable - MUST be first
- **Rh** (*float*) – Horizontal resistance
- **Rv** (*float*) – Vertical resistance

Returns Quantification of this formula is meant to be zero

Return type float

static square(*Rh*, *Rv*)
Compute sheet resistance from the given resistances.

Accurate only for square sample: $R_h = R_v$.

Parameters

- **Rh** (*float*) – Horizontal resistance

- **Rv** (*float*) – Vertical resistance

Returns Sheet resistance

Return type float

classmethod universal(*Rh*, *Rv*, *Rs0*)

Compute sheet resistance from the given resistances.

Universal formula. Computation flow for square-like samples is as follows:

```
Rs0 = van_der_pauw.Solve.square(Rh, Rv)
Rs = van_der_pauw.Solve.universal(Rh, Rv, Rs0)
```

Parameters

- **Rh** (*float*) – Horizontal resistance
- **Rv** (*float*) – Vertical resistance
- **Rs0** (*float*) – Approximate value to start with.

Returns Sheet resistance

Return type float

classmethod analyze(*Rh*, *Rv*)

Solve *Solve.implicit_formula()* to find sample's sheet resistance. Also compute resistance symmetry ratio (always greater than one). The ratio assess how squarish the sample is, quality of ohmic contacts (small, symmetric) etc.

Parameters

- **Rh** (*float*) – Horizontal resistance
- **Rv** (*float*) – Vertical resistance

Returns Sheet resistance and symmetry ratio

Return type tuple(float, float)

class physicslab.experiment.van_der_pauw.**Columns**

Bases: *physicslab.utility._ColumnsBase*

Column names.

classmethod mandatory()

Get the current mandatory column names.

Return type set(str)

classmethod process()

Get the current values of the *process()* output column names.

Return type lits(str)

class physicslab.experiment.van_der_pauw.**Measurement**(*data*)

Van der Pauw resistances measurements.

Parameters **data** (*pandas.DataFrame*) – Voltage-current pairs with respective geometries.

Raises **ValueError** – If `data` is missing a mandatory column

analyze()
Classify geometries into either `Geometry.Horizontal` or `Geometry.Vertical`. Then average respective hall resistances.

Additionally save Hall resistances to `data`.

Returns Horizontal and vertical sheet resistances

Return type tuple(float, float)

class physicslab.experiment.van_der_pauw.**Geometry**(*value*)
Resistance measurement configuration `Enum`.

Legend: `Rijkl` = $R_{ij,kl} = V_{kl}/I_{ij}$. The contacts are numbered from 1 to 4 in a counter-clockwise order beginning at the top-left contact. See Van der Pauw method at Wikipedia. There are two additional group values: `Vertical` and `Horizontal`.

classmethod parse(*text*)
Parse string to *Geometry*.

Parameters **text** (*str*) – Any valid permutation of digits 1-4, e.g. `4123` or `vertical` or `horizontal`

Raises **ValueError** – If there is no corresponding *Geometry*

Returns Enum corresponding to `text`

Return type *Geometry*

reverse_polarity()
Reverse polarity of voltage and current.

Returns Reversed geometry

Return type *Geometry*

rotate(*number=1*, *counterclockwise=True*)
Shift measuring pins counterclockwise.

Parameters

- **number** (*int, optional*) – Number of pins to jump, defaults to 1
- **counterclockwise** (*bool, optional*) – Direction of rotation, defaults to True

Returns Rotated geometry

Return type *Geometry*

is_horizontal()
Find whether the geometry describes horizontal configuration.

Returns Is horizontal?

Return type bool

is_vertical()
Find whether the geometry describes vertical configuration.

Returns Is vertical?

Return type bool

classify()
Sort the Geometry to either vertical or horizontal group.

Returns One of the two group configurations

Return type *Geometry*

physicslab.experiment.van_der_pauw.**plot**(*data_list*, *results*)
Plot individual measurements and results with quality coefficients.

For plotting details, see: https://matplotlib.org/stable/tutorials/intermediate/gridspec.html#a-complex-nested-gridspec-using-subplotspec

Parameters

- **data_list** (*list[pandas.DataFrame]*) –
- **results** (*pandas.DataFrame*) – Analysis data from *physicslab.experiment.process()*

Returns Same objects as from `matplotlib.pyplot.subplots()`. Axes are: grid axis array, right plot left axis, right plot right axis

Return type tuple[Figure, numpy.ndarray[Axes]]

4.3.2 Modules

Curves module

Curves.

physicslab.curves.**black_body_radiation**(*frequency*, *T*)
UNIT = W/sr/m^2/Hz

To express the law per unit wavelength, input `frequency = c / wavelength` (UNIT = W/sr/m^3).

Parameters

- **frequency** (*numpy.ndarray*) –
- **T** (*float*) – Temperature

Returns Spectral radiance

Return type numpy.ndarray

physicslab.curves.**gaussian_curve**(*x*, *expected_value*, *variance*, *amplitude=None*, *zero=0*)
Gauss curve function of given parameters.

Parameters

- **x** (*numpy.ndarray*) – Free variable
- **expected_value** (*float*) – Center
- **variance** (*float*) – Variance (not FWHM)
- **amplitude** (*float, optional*) – Amplitude (value at maximum relative to the baseline). None normalize as probability, defaults to None
- **zero** (*int*) – Baseline, defaults to 0

Returns Gaussian curve values

Return type numpy.ndarray

physicslab.curves.**gaussian_curve_FWHM**(*variance*)
Find FWHM from variance of a Gaussian curve.

Parameters **variance** (*float*) – Variance

Returns Full Width at Half Maximum

Return type float

physicslab.curves.**spontaneous_magnetization**(*T*, *M0*, *TC*, *a*, *b*, *zero*)

An empirical interpolation of the low temperature and the critical temperature regimes.

Parameters

- **T** (*numpy.ndarray*) – Temperature
- **M0** (*float*) – Spontaneous magnetization at absolute zero
- **TC** (*float*) – Curie temperature
- **a** (*float*) – Magnetization stays close to `M0` at higher temperatures
- **b** (*float*) – Critical exponent. Magnetization stays close to `zero` lower below `TC`
- **zero** (*float*) – Baseline

Returns Magnetization

Return type numpy.ndarray

physicslab.curves.**magnetic_hysteresis_branch**(*H*, *saturation*, *remanence*, *coercivity*, *rising_branch=True*)

One branch of magnetic hysteresis loop.

Parameters

- **H** (*numpy.ndarray*) – external magnetic field strength.
- **saturation** (*float*) – $max(B)$
- **remanence** (*float*) – $B(H = 0)$
- **coercivity** (*float*) – $H(B = 0)$
- **rising_branch** (*bool, optional*) – Rising (True) or falling (False) branch, defaults to True

Raises

- **ValueError** – If saturation is negative or zero
- **ValueError** – If remanence is negative
- **ValueError** – If coercivity is negative
- **ValueError** – If remanence is greater than saturation

Returns Resulting magnetic field induction B

Return type numpy.ndarray

physicslab.curves.**magnetic_hysteresis_loop**(*H*, *saturation*, *remanence*, *coercivity*)

Magnetic hysteresis loop.

If more control is needed, use *magnetic_hysteresis_branch()*. To check whether the data starts with rising or falling part, first and middle element are compared.

Parameters

- **H** (*numpy.ndarray*) – external magnetic field strength. The array is split in half for individual branches.
- **saturation** (*float*) – $max(B)$
- **remanence** (*float*) – $B(H = 0)$

- **coercivity** (*float*) – $H(B = 0)$

Returns Resulting magnetic field induction B

Return type numpy.ndarray

class physicslab.curves.**Line**(*constant=0*, *slope=0*)

Represents a line function: $y = a_0 + a_1 x$.

Call the instance to enumerate it at the given x. You can do arithmetic with *Line*, find zeros, etc.

Parameters

- **constant** (*int, optional*) – Constant term (a_0), defaults to 0
- **slope** (*int, optional*) – Linear term (a_1), defaults to 0

__call__(*x*)

Find function values of self.

Parameters **x** (*numpy.ndarray*) – Free variable

Returns Function value

Return type numpy.ndarray

zero()

Find free variable (x) value which evaluates to zero.

Raises **ValueError** – If slope is zero.

root()

Alias for *zero()*.

invert()

Return inverse function of self.

Returns Inverted function

Return type *Line*

static Intersection(*line1*, *line2*)

Find intersection coordinates of the two given *Line*.

Parameters

- **line1** (Line) – First line
- **line2** (Line) – Second line

Returns Coordinates of the intersection of the two lines

Return type tuple

Electricity module

Electricity related properties.

Mainly mutual conversion and units.

class physicslab.electricity.**Carrier_concentration**

Number of charge carriers in per unit volume.

Also known as Charge carrier density.

UNIT = '1/m^3'

SI unit.

static from_sheet_density(*sheet_density*, *thickness*)
Find carrier concentration from sheet density.

Parameters

- **sheet_density** (*float*) – (1/m^2)
- **thickness** (*float*) –

(m)

Returns (1/m^3)

Return type float

class physicslab.electricity.**Carrier_sheet_concentration**
Number of charge carriers in per unit area.

Also known as Charge carrier sheet density.

UNIT = '1/m^2'
SI unit.

class physicslab.electricity.**Mobility**
Electrical mobility is the ability of charged particles (such as electrons or holes) to move through a medium in response to an electric field that is pulling them.

UNIT = 'm^2/V/s'
SI unit.

static from_sheets(*sheet_density*, *sheet_resistance*)
Find mobility from sheet density and sheet resistance

Parameters

- **sheet_density** (*float*) – (1/m^2)
- **sheet_resistance** (*float*) – (ohms per square)

Returns (m^2/V/s)

Return type float

class physicslab.electricity.**Resistance**
Object property.

UNIT = 'ohm'
SI unit.

static from_ohms_law(*voltage*, *current*)
Find resistivity from sheet resistance.

Parameters

- **voltage** (*float*) – (volt)
- **current** (*float*) – (ampere)

Returns (ohm)

Return type float

static from_resistivity(*resistivity*, *cross_sectional_area*, *length*)
Find resistivity from resistance.

Parameters

- **resistance** (*float*) – (ohm)

- **cross_sectional_area** (*float*) – (meter squared)
- **length** (*float*) – (meter)

Returns (ohm metre)

Return type float

class physicslab.electricity.**Conductance**

Object property.

UNIT = '1/ohm'

SI unit. Also "siemens"

static from_resistance(*resistance*)

Find conductance from resistance.

Parameters **resistance** (*float*) – (ohm)

Returns (1/ohm)

Return type float

class physicslab.electricity.**Sheet_Resistance**

Thin object property.

UNIT = 'ohms per square'

SI unit.

static from_resistivity(*resistivity*, *thickness*)

Find sheet resistance from resistivity.

Parameters

- **resistivity** (*float*) – (ohm m)
- **thickness** (*float*) –

(m)

Returns (ohms per square)

Return type float

class physicslab.electricity.**Sheet_Conductance**

Thin object property.

UNIT = '1/ohms square'

SI unit. Also "siemens square"

class physicslab.electricity.**Resistivity**

Material property.

UNIT = 'ohm m'

SI unit.

static from_sheet_resistance(*sheet_resistance*, *thickness*)

Find resistivity from sheet resistance.

Parameters

- **sheet_resistance** (*float*) – (ohms per square)
- **thickness** (*float*) –

(m)

Returns (ohm m)

Return type float

static from_resistance(*resistance*, *cross_sectional_area*, *length*)
Find resistivity from resistance.

Parameters

- **resistance** (*float*) – (ohm)
- **cross_sectional_area** (*float*) – (m^2)
- **length** (*float*) –

(m)

Returns (ohm m)

Return type float

class physicslab.electricity.**Conductivity**
Material property.

UNIT = '1/ohm/m'
SI unit. Also "siemens/metre"

static from_resistivity(*resistivity*)
Find conductivity from resistivity.

Parameters **resistivity** (*float*) – (ohm m)

Returns (1/ohm/m)

Return type float

Utility module

Utility functions.

physicslab.utility.**permutation_sign**(*array*)
Computes permutation sign of given array.

Relative to ordered array, see: $sgn(\sigma) = (-1)^{\sum_{0 \le i < j < n}(\sigma_i > \sigma_j)}$

Note: For permutation relative to another ordering, use the following identity: $sgn(\pi_1 \circ \pi_2) = sgn(\pi_1) \cdot sgn(\pi_2)$

Parameters **array** (*list*) – Input array (iterable)

Returns Permutation parity sign is either (+1) or (-1)

Return type int

physicslab.utility.**squarificate**(*iterable*, *filler=None*)
Reshape 1D `iterable` into squarish 2D array.

Mainly use with *physicslab.ui.plot_grid()*, if the positions are arbitrary.
Example: reshape `list` of 10 filenames into 3x4 array. The two new elements will be populated by `filler`.

Parameters

- **iterable** (*list, numpy.ndarray*) – Source 1D iterable.
- **filler** (*object, optional*) – Value to pad the array with, defaults to None

Raises

- **NotImplementedError** – If `iterable` is array-like
- **ValueError** – If `iterable` has more than one dimension

Returns Modified array

Return type numpy.ndarray

`physicslab.utility.`**get_name**(*df*)
Find `DataFrame` name.

Parameters **df** (*pandas.DataFrame or pandas.Series*) – Input

Returns Name or None if name does not exist

Return type str or None

class `physicslab.utility.`**_ColumnsBase**
Abstract base class for `physicslab.experiment.[Any].Columns` classes.

classmethod list_all_names()

Return type list[str]

IO module

Filesystem manipulation.

`physicslab.io.`**gather_files**(*extension*, *folder*, *key_edit=None*, *trim_extension=True*)
Gather all files of the given `extension` located in or under `folder`.

Parameters

- **extension** (*str*) – File extension to look for
- **folder** (*str*) – Look in that folder and all its subfolders
- **key_edit** (*callable, optional*) – If supplied, this function will be applied to the *filename* stem, defaults to None
- **trim_extension** (*bool, optional*) – Cut the extension from *filename* to be used as key, defaults to True

Returns Dictionary form {filename : path}

Return type dict

`physicslab.io.`**subfolder**(*folder*, *look_for*)
Look for a subfolder containing `look_for` in its name.

Parameters

- **folder** (*str*) – Search there
- **look_for** (*str*) – Part of the folder name to look for. Can be RegEx.

Raises

- **OSError** – If no subfolder found
- **OSError** – If multiple subfolders found

Returns Path to the found subfolder

Return type str

UI module

User interface.

physicslab.ui.**plot_grid**(*df*, *plot_value*, *fig_axs=None*, *skip=None*, *title=None*, *xlabel=None*, *ylabel=None*, *row_labels=True*, *column_labels=True*, *subplots_adjust_kw=None*, ***kwargs*)

Construct a figure with the same layout as the input.

For example, use it to display SEM images, where rows correspond to different magnifications and columns to samples.

If a `df` value is `None` or `numpy.nan`, skip the individual plot.

To display all figures, call `show()`.

Parameters

- **df** (*pandas.DataFrame*) – Data to drive plotting. E.g. filename to load and plot
- **plot_value** (*callable*) – Function to convert a `df` value into `ax.plot`.

```
def plot_value(ax: matplotlib.axes.Axes, value: object):
    ax.plot(value.x)
    ax.legend()  # Show a legend for each plot.
```

- **fig_axs** (*tuple(Figure, numpy.ndarray(Axes)), optional*) – Figure and axis array to draw to. Axis shape must match that of `df`. If None, create a new figure, defaults to None
- **skip** (*list, optional*) – Skip df values matching any of the listed items, defaults to None
- **title** (*str, optional*) – Common title. If `auto`, use `df.name` if available, defaults to None
- **xlabel** (*str, optional*) – Common x axis label, defaults to None
- **ylabel** (*str, optional*) – Common y axis label, defaults to None
- **row_labels** (*bool, optional*) – Annotate rows by `df.index`, defaults to True
- **column_labels** (*bool, optional*) – Annotate columns by `df.columns`, defaults to True
- **subplots_adjust_kw** (*dict, optional*) – Dict with keywords passed to the `subplots_adjust()` call. E.g. `hspace`, defaults to None
- **kwargs** – All additional keyword arguments are passed to the `figure()` call. E.g. `sharex`

Raises **ValueError** – If `df` and `axs` have different shapes

Returns Same objects as from `matplotlib.pyplot.subplots()`

Return type tuple[Figure, numpy.ndarray[Axes]]

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

F

G

I

L

M

P

R

S

U

Z