monashspa Documentation

Release 1.8.0

School of Physics & Astronomy, Monash University

Sep 08, 2022

Contents

This Python library contains helpful Python functions and utilities that will be used in Physics & Astronomy classes at Monash University.

This documentation is not meant to replace your study resources for individual activities (which you can find on the Monash LMS) but does document every function and every option available to use, which you might find useful as a reference.

You are viewing documentation for version: 1.8.0

CHAPTER 1

Contents

1.1 Installation

These installation instructions assume you already have Python installed. If you do not already have a copy of Python, we recommend you install Anaconda Python 3.

1.1.1 PyPi

We recommend installing *monashspa* from the Python Package Index. To do this, open Spyder and in the console window (bottom right) at the *In [1]* prompt type:

```
!pip install monashspa
```

followed by return. The package should now install. You only need to do this once. Note the exclamation mark at the beginning.

1.1.2 Upgrading monashspa

To upgrade to the latest version of *monashspa*, open Spyder and in the console window (bottom right) at the *In [1]* prompt type:

```
!pip install -U monashspa
```

To upgrade to a specific version of *monashspa* (or, alternatively, if you wish to downgrade), run:

```
!pip install -U monashspa==<version>
```

where `<version>` is replaced by the version you wish (for example `!pip install -U monashspa==0.1.0`).

Note: After upgrading, you will need to **restart** any open Python or IPython terminals in order to ensure you are using the new version. This includes terminals within Spyder (restart the entire Spyder application if you are unsure how to restart a terminal). You can check which version of monashspa you are using by running `import monashspa` followed by `print(monashspa.__version__)` from with a Python script or terminal.

1.1.3 Run inside Google Colab

If for some reason you have trouble installaing *Anaconda* on your local computer, it is possible to run everything inside your web browser using what is called *Google Colab*. Follow the directions below to get this up and working.

1. Go to *Google Colab https://research.google.com/colaboratory/*
2. In the window that comes up select *GitHub*
3. At the line that asks for a *GitHub URL* type *Monash-University-Physics-Astronomy* and press return
4. Pick one of the files that shows up (like the *PHS2062 one*)
5. Select *Copy to Drive* at the top of the code to get your own copy to work on
6. If your python code requires some data file, then pick the small folder icon to the left and then select the little icon with an up arrow to upload a file.
7. You are now ready to run. If you want to come back to your code later, simply go to your Google Drive, find the file inside the *Colab Notebooks* folder and double click it. Note that you might have to upload your data file again though.

1.2 PHS1011

This documents the `monashspa.PHS1011` library that you will import into code used in the PHS1011 unit.

`monashspa.PHS1011.`**`get_fit_parameters`**(*fit_result*)

Returns the parameters from a fit result as a dictionary.

This function takes a `lmfit.model.ModelResult` object from the lmfit Python library and extracts the parameters of the fit along with their uncertainties. These are returned to you in a Python dictionary format. The format of the dictionary depends on the model used to perform the fit. For example, a linear fit would result in the following dictionary:

```
parameters = {
    'slope': <value>,
    'u_slope': <value>,
    'intercept': <value>,
    'u_intercept': <value>,
}
```

The parameter names always match those of the lmfit model, and the uncertainties are always the identical parameter name prefixed with `u_`.

Parameters **`fit_result`** – A `lmfit.model.ModelResult` object from the lmfit Python library.

Returns A dictionary containing the fit parameters and their associated uncertainties.

monashspa.PHS1011.**linear_fit**(*x*, *y*, *u_y=None*, *slope_guess=None*, *intercept_guess=None*)

General purpose linear fit function.

This function takes your x and y data (as numpy arrays) and returns a `lmfit.model.ModelResult` object from the lmfit Python library. It attempts to fit your data to a model define by:

$$y = mx + c$$

where $m = slope$ and $c = intercept$. If guesses for the slope and intercept are not explicitly provided when calling this function, they will be inferred from the provided data arrays.

Parameters

- **x** – A 1D numpy array of x data points
- **y** – A 1D numpy array of y data points

Keyword Arguments

- **u_y** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points
- **slope_guess** – An optional argument for providing an initial guess for the value of the slope parameter
- **intercept_guess** – An optional argument for providing an initial guess for the value of the intercept parameter

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

monashspa.PHS1011.**savefig**(*fname*, *dpi=600*, *bbox_inches='tight'*, ***kwargs*)

A wrapper for `matplotlib.pyplot.savefig()` with sensible defaults.

By default, if a matplotlib legend is located outside of the plot axes, then `matplotlib.pyplot.savefig()` may cut off the legend when saving the figure. This function fixes this issue by setting `bbox_inches='tight'` and setting `bbox_extra_artists` to be a list of the current figure legends, unless the user chooses to define them otherwise. This function also sets the default for `dpi` to 600, which is large enough for most purposes.

This function accepts any keyword argument from `matplotlib.pyplot.savefig()` and returns the result of the call to that function. See `matplotlib.pyplot.savefig()` for usage.

1.3 PHS2061

This documents the `monashspa.PHS2061` library that you will import into code used in the PHS2061 unit.

monashspa.PHS2061.**get_fit_parameters**(*fit_result*)

Returns the parameters from a fit result as a dictionary.

This function takes a `lmfit.model.ModelResult` object from the lmfit Python library and extracts the parameters of the fit along with their uncertainties. These are returned to you in a Python dictionary format. The format of the dictionary depends on the model used to perform the fit. For example, a linear fit would result in the following dictionary:

```
parameters = {
    'slope': <value>,
    'u_slope': <value>,
    'intercept': <value>,
    'u_intercept': <value>,
}
```

The parameter names always match those of the lmfit model, and the uncertainties are always the identical parameter name prefixed with `u_`.

Parameters **fit_result** – A `lmfit.model.ModelResult` object from the lmfit Python library.

Returns A dictionary containing the fit parameters and their associated uncertainties.

`monashspa.PHS2061.`**`linear_fit`**(*x*, *y*, *u_y=None*, *slope_guess=None*, *intercept_guess=None*)

General purpose linear fit function.

This function takes your x and y data (as numpy arrays) and returns a `lmfit.model.ModelResult` object from the lmfit Python library. It attempts to fit your data to a model define by:

$$y = mx + c$$

where $m = slope$ and $c = intercept$. If guesses for the slope and intercept are not explicitly provided when calling this function, they will be inferred from the provided data arrays.

Parameters

- **x** – A 1D numpy array of x data points
- **y** – A 1D numpy array of y data points

Keyword Arguments

- **u_y** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points
- **slope_guess** – An optional argument for providing an initial guess for the value of the slope parameter
- **intercept_guess** – An optional argument for providing an initial guess for the value of the intercept parameter

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

`monashspa.PHS2061.`**`make_lmfit_model`**(*expression*, *independent_vars=None*, *allow_constant_model=False*, ***kwargs*)

A convenience function for creating a lmfit Model from an equation in a string

This function takes an expression containing the right hand side of an equation you wish to use as your fitting model, and generates a `lmfit.model.Model` object from the lmfit Python library

For example, the expression `"m*x+c"` would create a model that would fit to linear data modelled by the equation $y = m * x + c$. Note that the expression does not contain the `y` or = symbols

Standard numpy and scipy.special functions are also available for use in your expression. For example, this is also a valid expression: `"sin(x)+c"`.

The expression must always be valid Python code, and must be able to be evaluated with every parameter set to a random floating point number.

The independent variable is assumed to be `x` unless otherwise specified. All other variables are assumed to be parameters you wish the fitting routine to optimise. These parameters will be given an initial hint of 1 in the returned model, but can be overridden later using `lmfit.model.Model.set_param_hint()`, `lmfit.model.Model.make_params()`, or `lmfit.parameter.Parameters.add()`.

Note: Additional keyword arguments are passed directly to `lmfit.model.Model`.

Parameters **expression** – A string containing the right-hand-side of the equation you wish to model (assumes the left hand side is equal to "y").

Keyword Arguments

- **independent_vars** – a list of independent variable names that should not be varied by lmfit. If set to `None` (the default) it assumes that the independent variables is just `["x"]`
- **allow_constant_model** – A Boolean to indicate whether to suppress the exception raised if you don't use the independent variable(s) in your model equation. Defaults to `False` (raise the exception). If you do wish to use a constant model, we recommend leaving this as "False" and modifying your model equation to include an "x*0" (or similar) term as this also ensures the component can be plotted using `lmfit.model.ModelResult.eval_components()` without additional modification. However, you can also set this to `True` to suppress the Exception and restore the default lmfit behaviour.

Returns A `lmfit.model.Model` object to be used for fitting with the lmfit library.

`monashspa.PHS2061.`**`model_fit`**(*model*, *parameters*, *x*, *y*, *u_y=None*, ***kwargs*)

A wrapper for fitting to an arbitrary model using lmfit.

This function automatically inverts the array of standard errors to weights (which lmfit expects) and disables the scaling of the covariant matrix is the array of standard errors are provided.

Note: Any additional keyword arguments passed to this function will be passed directly to `model.fit`.

Parameters

- **model** – a reference to a lmfit model.
- **parameters** – a reference to a `lmfit.parameters.Parameters` object for your model
- **x** – A 1D numpy array of x data points
- **y** – A 1D numpy array of y data points

Keyword Arguments **u_y** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

`monashspa.PHS2061.`**`savefig`**(*fname*, *dpi=600*, *bbox_inches='tight'*, ***kwargs*)

A wrapper for `matplotlib.pyplot.savefig()` with sensible defaults.

By default, if a matplotlib legend is located outside of the plot axes, then `matplotlib.pyplot.savefig()` may cut off the legend when saving the figure. This function fixes this issue by setting `bbox_inches='tight'` and setting `bbox_extra_artists` to be a list of the current figure legends, unless the user chooses to define them otherwise. This function also sets the default for `dpi` to 600, which is large enough for most purposes.

This function accepts any keyword argument from `matplotlib.pyplot.savefig()` and returns the result of the call to that function. See `matplotlib.pyplot.savefig()` for usage.

1.4 PHS2062

This documents the `monashspa.PHS2062` library that you will import into code used in the PHS2062 unit. If not working on you local computer, you can alternatively start the exercise completely online by clicking on the badge below

1.5 PHS2081

This documents the `monashspa.PHS2081` library that you will import into code used in the PHS2081 unit.

`monashspa.PHS2081.`**`get_fit_parameters`**(*fit_result*)

Returns the parameters from a fit result as a dictionary.

This function takes a `lmfit.model.ModelResult` object from the lmfit Python library and extracts the parameters of the fit along with their uncertainties. These are returned to you in a Python dictionary format. The format of the dictionary depends on the model used to perform the fit. For example, a linear fit would result in the following dictionary:

```
parameters = {
    'slope': <value>,
    'u_slope': <value>,
    'intercept': <value>,
    'u_intercept': <value>,
}
```

The parameter names always match those of the lmfit model, and the uncertainties are always the identical parameter name prefixed with `u_`.

Parameters **`fit_result`** – A `lmfit.model.ModelResult` object from the lmfit Python library.

Returns A dictionary containing the fit parameters and their associated uncertainties.

`monashspa.PHS2081.`**`linear_fit`**(*x*, *y*, *u_y=None*, *slope_guess=None*, *intercept_guess=None*)

General purpose linear fit function.

This function takes your x and y data (as numpy arrays) and returns a `lmfit.model.ModelResult` object from the lmfit Python library. It attempts to fit your data to a model define by:

$$y = mx + c$$

where $m = slope$ and $c = intercept$. If guesses for the slope and intercept are not explicitly provided when calling this function, they will be inferred from the provided data arrays.

Parameters

- **`x`** – A 1D numpy array of x data points
- **`y`** – A 1D numpy array of y data points

Keyword Arguments

- **`u_y`** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points
- **`slope_guess`** – An optional argument for providing an initial guess for the value of the slope parameter
- **`intercept_guess`** – An optional argument for providing an initial guess for the value of the intercept parameter

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

`monashspa.PHS2081.`**`make_lmfit_model`**(*expression*, *independent_vars=None*, *allow_constant_model=False*, ***kwargs*)

A convenience function for creating a lmfit Model from an equation in a string

This function takes an expression containing the right hand side of an equation you wish to use as your fitting model, and generates a `lmfit.model.Model` object from the lmfit Python library

For example, the expression `"m*x+c"` would create a model that would fit to linear data modelled by the equation $y = m * x + c$. Note that the expression does not contain the `y` or = symbols

Standard numpy and scipy.special functions are also available for use in your expression. For example, this is also a valid expression: `"sin(x)+c"`.

The expression must always be valid Python code, and must be able to be evaluated with every parameter set to a random floating point number.

The independent variable is assumed to be `x` unless otherwise specified. All other variables are assumed to be parameters you wish the fitting routine to optimise. These parameters will be given an initial hint of 1 in the returned model, but can be overridden later using `lmfit.model.Model.set_param_hint()`, `lmfit.model.Model.make_params()`, or `lmfit.parameter.Parameters.add()`.

Note: Additional keyword arguments are passed directly to `lmfit.model.Model`.

Parameters **`expression`** – A string containing the right-hand-side of the equation you wish to model (assumes the left hand side is equal to "y").

Keyword Arguments

- **`independent_vars`** – a list of independent variable names that should not be varied by lmfit. If set to `None` (the default) it assumes that the independent variables is just `["x"]`
- **`allow_constant_model`** – A Boolean to indicate whether to suppress the exception raised if you don't use the independent variable(s) in your model equation. Defaults to `False` (raise the exception). If you do wish to use a constant model, we recommend leaving this as "False" and modifying your model equation to include an "x*0" (or similar) term as this also ensures the component can be plotted using `lmfit.model.ModelResult.eval_components()` without additional modification. However, you can also set this to `True` to suppress the Exception and restore the default lmfit behaviour.

Returns A `lmfit.model.Model` object to be used for fitting with the lmfit library.

`monashspa.PHS2081.`**`model_fit`**(*model*, *parameters*, *x*, *y*, *u_y=None*, ***kwargs*)

A wrapper for fitting to an arbitrary model using lmfit.

This function automatically inverts the array of standard errors to weights (which lmfit expects) and disables the scaling of the covariant matrix is the array of standard errors are provided.

Note: Any additional keyword arguments passed to this function will be passed directly to `model.fit`.

Parameters

- **`model`** – a reference to a lmfit model.
- **`parameters`** – a reference to a `lmfit.parameters.Parameters` object for your model
- **`x`** – A 1D numpy array of x data points
- **`y`** – A 1D numpy array of y data points

Keyword Arguments **`u_y`** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

`monashspa.PHS2081.`**`savefig`**(*fname*, *dpi=600*, *bbox_inches='tight'*, ***kwargs*)

A wrapper for `matplotlib.pyplot.savefig()` with sensible defaults.

By default, if a matplotlib legend is located outside of the plot axes, then `matplotlib.pyplot.savefig()` may cut off the legend when saving the figure. This function fixes this issue by setting

bbox_inches='tight' and setting bbox_extra_artists to be a list of the current figure legends, unless the user chooses to define them otherwise. This function also sets the default for dpi to 600, which is large enough for most purposes.

This function accepts any keyword argument from matplotlib.pyplot.savefig() and returns the result of the call to that function. See matplotlib.pyplot.savefig() for usage.

1.6 PHS3000

This documents the monashspa.PHS3000 library that you will import into code used in the PHS3000 unit.

1.6.1 Submodules

PHS3000 Optical Tweezers

This documents the monashspa.PHS3000.optical_tweezers library that you will import into code used in the PHS3000 unit when performing experiment 1.3 Optical Tweezers.

monashspa.PHS3000.optical_tweezers.**cf_linearised**(*f*, *ps*, *initial_fc*, *call_show=True*)

Finds the corner frequency value for a lorentzian power spectrum

Finds the corner frequency (f_c) of a power spectrum of the form: $y = \frac{a}{(f_c^2+f^2)}$

by transforming into the logarithmic domain and determining when the spectrum is linearised.

Parameters

- **f** – A 1D numpy array containing the frequency values associated with the power spectrum
- **ps** – A 1D numpy array containing the power spectrum data (must be the same length as f)
- **initial_fc** – An initial guess for the corner frequency

Keyword Arguments **call_show** – Whether to call matplotlib.pyplot.show() at the end of the function (prior to returning). Defaults to True. Set this to False if you are not using Spyder/IPython and wish your entire script to complete before showing any plots. Note, you will need to explicitly call matplotlib.pyplot.show() if you set this to False.

Returns The best estimate for the corner frequency f_c.

monashspa.PHS3000.optical_tweezers.**ps_load**(*filepath*)

Imports the power spectrum data from optical tweezers file

Parameters **filepath** – The path to the .lvm file produced by the optical tweezers acquisition software

Returns A tuple (f, psx, psy) where f is a 1D numpy array containing the frequency values associated with the power spectrums in psx and psy (which are also both 1D numpy arrays).

monashspa.PHS3000.optical_tweezers.**trap_k_theory**(*r*, *w*, *alpha*, *eccentricity*, *I*)

Calculates the theoretical spring constant (k) for an optical tweezers trap for specified parameters

Calculates using the equation: $k = \alpha\, I_0\, \omega\, \frac{2\,\pi\,\epsilon^3}{c\,\xi^3} \left(\sqrt{\frac{\pi}{2}} \left(\left(\frac{\xi\, a}{\epsilon} \right)^2 - 1 \right) \exp\left[-\frac{a^2}{2} \right] \operatorname{erf}\left[\frac{\xi\, a}{\sqrt{2}\,\epsilon} \right] + \frac{\xi\, a}{\epsilon} \exp\left[-\frac{a^2}{2\,\epsilon^2} \right] \right)$

from Bechhoefer 2002.

If the input arguments are numpy arrays, then the output will also be an array of the appropriate dimension. Otherwise a single number will be returned.

Parameters

- **r** – Sphere radius (m)
- **w** – The $1/e^2$ radius (beam waist) of the trapping beam (m)
- **alpha** – $n_p^2/n_0^2 - 1$, where n_p is the refractive index of the microsphere and n_0 is the refractive index of water
- **eccentricity** – The eccentiricty of the trapping beam
- **I** – Trapping beam intensity (W/m^2)

Returns The theoretical spring constant (k)

Return type k

PHS3000 Muon Physics

This documents the `monashspa.PHS3000.muon` library that you will import into code used in the PHS3000 unit when performing experiment 1.4 muon physics.

`monashspa.PHS3000.muon.`**`histc`**(*x*, *bins*)

Counts the number of times an values in `x` fall between each bin in `bins`.

This uses the condition `if bins[i] <= x[j] < bins[i+1]:  result[i] += 1` for each element in `x`

Parameters

- **x** – The numpy array containing the data to count
- **bins** – A numpy array defining the bins

Returns A numpy array containing the number of elements between each bin.

`monashspa.PHS3000.muon.`**`read_data`**(*filepath*)

Imports data from the muon physics .data file

Parameters **filepath** – The path to the .data file produced by the muon acquisition software

Returns A 2D numpy array containing the data from the file. The first column contains the acquisition intervals and the second column contains the timestamps.

PHS3000 Reflex klystron

This documents the `monashspa.PHS3000.reflex_klystron` library that you will import into code used in the PHS3000 unit when performing experiment 1.5 Reflex klystron.

`monashspa.PHS3000.reflex_klystron.`**`micrometer_1_attenuation = array([[0.00e+00, 4.12e-01],`**

The calibration data for attenuator #1. Here, the first column corresponds to the attentuation at 9.15GHz in dB and the second column corresponds to the micrometer position in inches

`monashspa.PHS3000.reflex_klystron.`**`micrometer_2_and_3_attenuation = array([[ 0. , 0.412], [`**

The calibration data for attenuator #2 and #3. Here, the first column corresponds to the attentuation at 9.15GHz in dB and the second column corresponds to the micrometer position in inches

`monashspa.PHS3000.reflex_klystron.`**`micrometer_7925_attenuation = array([[ 2. , 12.14], [ 4.`**

The calibration data for attenuator #7925. Here, the first column corresponds to the attentuation at 9.15GHz in dB and the second column corresponds to the micrometer position in nm

monashspa.PHS3000.reflex_klystron.**wavemeter_4091 = array([[7.7024, 24.161], [7.8 , 22.6**
The calibration data for wavemeter S/N 4091 as a numpy array Here, the first column corresponds to the frequency in GHz and the second column corresponds to the wavemeter position

monashspa.PHS3000.reflex_klystron.**wavemeter_4929 = array([[7.7024, 24.215], [7.8 , 22.6**
The calibration data for wavemeter S/N 4929 as a numpy array Here, the first column corresponds to the frequency in GHz and the second column corresponds to the wavemeter position

monashspa.PHS3000.reflex_klystron.**wavemeter_4930 = array([[7.7024, 24.768], [7.8 , 23.2**
The calibration data for wavemeter S/N 4930 as a numpy array Here, the first column corresponds to the frequency in GHz and the second column corresponds to the wavemeter position

PHS3000 Rubidium hyperfine spectroscopy

This documents the `monashspa.PHS3000.rubidium_spectroscopy` library that you will import into code used in the PHS3000 unit when performing experiment 1.6 Rubidium hyperfine spectroscopy.

monashspa.PHS3000.rubidium_spectroscopy.**extract_frequency**(*michelson_data*, *MHz_per_fringe*, *num_dc_terms_to_remove=1*)
Returns a frequency calibration determined from Michelson interferometer data

Performs a Hilbert transform (minus the DC term) on the Michelson interferometer data and scales the resultant frequency array by the distance (in MHz) between constructive interference fringes.

Parameters

- **michelson_data** – A 1D numpy array containing the oscilloscope data from the michelson interferometer
- **MHz_per_fringe** – The distance (in MHz) between constructive interference fringes in the `michelson_data` calculated from the path length difference of the Michelson interferometer arms.

Keyword Arguments **num_dc_terms_to_remove** – The number of low frequency components to remove after performing the Fourier transform. The default is 1 (remove the DC term only). Higher integers remove low order frequency components

Returns A 1D numpy array (with the same length as `michelson_data`) containing a frequency calibration (in MHz) for the saturated absorption spectra. Use this array as the x-axis for plotting and fitting.

monashspa.PHS3000.rubidium_spectroscopy.**read_rigol_csv**(*filepath*)
Reads the csv file saved from the Rigol digital storage oscilloscope.

Parameters **filepath** – A string containing the path to the csv file

Returns A tuple `(X, time, CH1, CH2)` where each element is a 1D numpy array containing an array of incrementing integers, an array of time points (x-axis of the oscilloscope), and the y-values for channels 1 and 2 respectively.

PHS3000 Mobility, diffusion and recombination of holes in Ge

This documents the `monashspa.PHS3000.holes_in_ge` library that you will import into code used in the PHS3000 unit when performing experiment 2.2 Mobility, diffusion and recombination of holes in Ge.

monashspa.PHS3000.holes_in_ge.**read_data**(*filepath*)
Imports the 'xls' file (actually just a tsv) save by the Rigol Oscilloscope software

Parameters **filepath** – The path to the .xls or .txt file produced by the Rigol oscilloscope software

Returns A tuple `(t, V)` where `t` is a 1D numpy array containing the time values associated with the oscilloscope trace and `V` is a 1D numpy array of the same length with the associated voltage readings for each time point.

PHS3000 Thermoelectricity

This documents the `monashspa.PHS3000.thermoelectricity` library that you will import into code used in the PHS3000 unit when performing experiment 2.3 Thermoelectricity.

`monashspa.PHS3000.thermoelectricity.`**`read_data`**(*filepath*)
Reads the csv file produced by the thermoelectricity software

Parameters **filepath** – The path to the .csv file

Returns A 2D numpy array containing the data from the file, with columns date-time, delta_t from row 0, T_Cold, T_Hot, T_Case, I_Heat, V_Heat, I_Pelt, V_Pelt, respectively.

PHS3000 Mossbauer

This documents the `monashspa.PHS3000.mossbauer` library that you will import into code used in the PHS3000 unit when performing experiment 2.4 Mossbauer.

`monashspa.PHS3000.mossbauer.`**`read_data`**(*filepath*)
Imports the acquired spectrum from the Mossbauer data file.

Parameters **filepath** – The path to the .ws5 file produced by the muon Mossbauer acquisition software.

Returns A 1D numpy array containing the data (counts) from the file.

PHS3000 Resistivity of Germanium

This documents the `monashspa.PHS3000.resistivity_germanium` library that you will import into code used in the PHS3000 unit when performing experiment 2.9 Resistivity of Germanium.

`monashspa.PHS3000.resistivity_germanium.`**`liquid_helium = array([[7.69e+01, 1.00e-01, 3.70e+`**
Resistance data for temperatures below 77K as a 2D numpy array. Columns are: temperature (K), u_temperature (K), resistance (Ohms), u_resistance(Ohms) respectively

PHS3000 Microwave transmission line

This documents the `monashspa.PHS3000.microwave_transmission` library that you will import into code used in the PHS3000 unit when performing experiment 4.1 Microwave transmission line.

`monashspa.PHS3000.microwave_transmission.`**`wavemeter_calibration = array([[24.215, 7.702], [`**
The calibration data for wavemeter #4929 as a 2D numpy array. Here the first column corresponds to wavemeter reading in mm and the second column corresponds to the frequency in GHz.

PHS3000 Measurement of betaray spectra

This documents the `monashspa.PHS3000.betaray` library that you will import into code used in the PHS3000 unit when performing experiment 4.4 Measurement of betaray spectra.

`monashspa.PHS3000.betaray.`**`modified_fermi_function_data = array([[0. , 6.591], [0.1 , 6.582]`**
The modified Fermi function, G, for $Z = 55$. Here, the first column corresponds to momentum, p, in relativistic units of m_0c^2 and the second column cooresponds to the value of the modified Fermi function G.

`monashspa.PHS3000.betaray.`**`read_data`**(*filepath*)
Reads the csv file produced by the betaray online experiment

Parameters **`filepath`** – The path to the .csv file

Returns A 2D numpy array containing the data from the file, with columns Acquisition time, Run mode, Vacuum pressure, Shutter Position, Duration (s), Count, Lens coil current (A), u(Lens coil current) (A), Bias coil current (A), u(Bias coil current) (A), Magnetometer x position, Magnetometer y position, Magnetometer x-axis field strength (μT), u(Magnetometer x-axis field strength) (μT), Magnetometer y-axis field strength (μT), u(Magnetometer y-axis field strength) (μT), Magnetometer z-axis field strength (μT), and u(Magnetometer z-axis field strength) (μT) respectively.

PHS3000 PET

This documents the `monashspa.PHS3000.PET` library that you will import into code used in the PHS3000 unit when performing experiment 4.6 PET.

`monashspa.PHS3000.PET.`**`pet_rebuild`**(*filepath*, *filter_name=None*, *npoints=None*, *call_show=True*)
Perform inverse radon transform on acquired PET data and plots results

Parameters **`filepath`** – A string containing the path to the txt file containing the PET data

Keyword Arguments

- **`filter_name`** – A string containing the name of the filter to use during reconstruction. Defaults to `None` (no reconstruction). Options are:
  - None: don't reconstruct
  - 'none': Reconstruct with no filter
  - 'ramp': Reconstruct using the Ram-Lak filter
  - 'Shepp-Logan': Reconstruct using the Shepp-Logan filter
  - 'cosine': Reconstruct using the cosine filter
  - 'hamming': Reconstruct using the hamming filter
  - 'hann': Reconstruct using the hann filter
- **`npoints`** – The number of points to reconstruct
- **`call_show`** – Whether to call `matplotlib.pyplot.show()` at the end of the function (prior to returning). Defaults to `True`. Set this to `False` if you are not using Spyder/IPython and wish your entire script to complete before showing any plots. Note, you will need to explicitly call `matplotlib.pyplot.show()` if you set this to `False`.

Returns A 2D numpy array containing the coincidence counts (rows correspond to each linear stage position and columns to each rotation stage position)

1.6.2 General purpose functions

monashspa.PHS3000.**get_fit_parameters**(*fit_result*)

Returns the parameters from a fit result as a dictionary.

This function takes a `lmfit.model.ModelResult` object from the lmfit Python library and extracts the parameters of the fit along with their uncertainties. These are returned to you in a Python dictionary format. The format of the dictionary depends on the model used to perform the fit. For example, a linear fit would result in the following dictionary:

```
parameters = {
    'slope': <value>,
    'u_slope': <value>,
    'intercept': <value>,
    'u_intercept': <value>,
}
```

The parameter names always match those of the lmfit model, and the uncertainties are always the identical parameter name prefixed with `u_`.

Parameters **fit_result** – A `lmfit.model.ModelResult` object from the lmfit Python library.

Returns A dictionary containing the fit parameters and their associated uncertainties.

monashspa.PHS3000.**linear_fit**(*x*, *y*, *u_y=None*, *slope_guess=None*, *intercept_guess=None*)

General purpose linear fit function.

This function takes your x and y data (as numpy arrays) and returns a `lmfit.model.ModelResult` object from the lmfit Python library. It attempts to fit your data to a model define by:

$$y = mx + c$$

where $m = slope$ and $c = intercept$. If guesses for the slope and intercept are not explicitly provided when calling this function, they will be inferred from the provided data arrays.

Parameters

- **x** – A 1D numpy array of x data points
- **y** – A 1D numpy array of y data points

Keyword Arguments

- **u_y** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points
- **slope_guess** – An optional argument for providing an initial guess for the value of the slope parameter
- **intercept_guess** – An optional argument for providing an initial guess for the value of the intercept parameter

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

monashspa.PHS3000.**make_lmfit_model**(*expression*, *independent_vars=None*, *allow_constant_model=False*, ***kwargs*)

A convenience function for creating a lmfit Model from an equation in a string

This function takes an expression containing the right hand side of an equation you wish to use as your fitting model, and generates a `lmfit.model.Model` object from the lmfit Python library

For example, the expression `"m*x+c"` would create a model that would fit to linear data modelled by the equation $y = m * x + c$. Note that the expression does not contain the `y` or = symbols

Standard numpy and scipy.special functions are also available for use in your expression. For example, this is also a valid expression: `"sin(x)+c"`.

The expression must always be valid Python code, and must be able to be evaluated with every parameter set to a random floating point number.

The independent variable is assumed to be `x` unless otherwise specified. All other variables are assumed to be parameters you wish the fitting routine to optimise. These parameters will be given an initial hint of 1 in the returned model, but can be overridden later using `lmfit.model.Model.set_param_hint()`, `lmfit.model.Model.make_params()`, or `lmfit.parameter.Parameters.add()`.

Note: Additional keyword arguments are passed directly to `lmfit.model.Model`.

Parameters **`expression`** – A string containing the right-hand-side of the equation you wish to model (assumes the left hand side is equal to "y").

Keyword Arguments

- **`independent_vars`** – a list of independent variable names that should not be varied by lmfit. If set to `None` (the default) it assumes that the independent variables is just `["x"]`
- **`allow_constant_model`** – A Boolean to indicate whether to suppress the exception raised if you don't use the independent variable(s) in your model equation. Defaults to `False` (raise the exception). If you do wish to use a constant model, we recommend leaving this as "False" and modifying your model equation to include an "x*0" (or similar) term as this also ensures the component can be plotted using `lmfit.model.ModelResult.eval_components()` without additional modification. However, you can also set this to `True` to suppress the Exception and restore the default lmfit behaviour.

Returns A `lmfit.model.Model` object to be used for fitting with the lmfit library.

`monashspa.PHS3000.`**`model_fit`**(*model*, *parameters*, *x*, *y*, *u_y=None*, ***kwargs*)

A wrapper for fitting to an arbitrary model using lmfit.

This function automatically inverts the array of standard errors to weights (which lmfit expects) and disables the scaling of the covariant matrix is the array of standard errors are provided.

Note: Any additional keyword arguments passed to this function will be passed directly to `model.fit`.

Parameters

- **`model`** – a reference to a lmfit model.
- **`parameters`** – a reference to a `lmfit.parameters.Parameters` object for your model
- **`x`** – A 1D numpy array of x data points
- **`y`** – A 1D numpy array of y data points

Keyword Arguments **`u_y`** – An optional argument for providing a 1D numpy array of uncertainty values for the y data points

Returns A `lmfit.model.ModelResult` object from the lmfit Python library

`monashspa.PHS3000.`**`read_csv`**(*filepath*, **args*, ***kwargs*)

A simple wrapper around the pandas csv reader.

This function wraps the `pandas.read_csv()` function and returns a numpy array. Any optional argument for the pandas function can be used here.

Examples

Standard csv files with a header row can be read using `read_csv(filepath)`.

If your csv file has no header row, use: `read_csv(filepath, header=None)`.

If your file is delimited by something other than a comma, you can specify the separator with the optional argument `sep`. For example, `sep='|'` if your file is delimited by the pipe symbol.

There are many options that may be useful, such as those that parse dates, ignore columns, or convert certain values to 'nan's. See the pandas documentation for `pandas.read_csv()` to see all optional arguments.

Parameters **`filepath`** – The path to the file to read

Keyword Arguments **`**kwargs`** – See the pandas documentation for `pandas.read_csv()`

Returns A numpy array. This array will be 1D if the csv file contains only a single row or column of data. Otherwise the array will be 2D.

`monashspa.PHS3000.`**`read_mca_file`**(*filepath*)
Imports data saved from the ADMCA software

Parameters **`filepath`** – The path to the .mca file produced by the ADMCA acquisition software

Returns A tuple `(header, data)` where `header` is a dictionary containing the key, value pairs from header rows of the .mca file and `data` is a 1D numpy array of the counts for each MCA channel.

`monashspa.PHS3000.`**`read_picoscope_csv`**(*filepath*)
Reads the csv file saved from the PicoScope software

Note: The picoscope software will save overrange values in the csv file as either "Infinity" or the unicode symbol for infinity (which will show up in some software as several random characters). This Python function converts those instances to `np.nan`.

Note: This function attempts to return values in S.I. units without a prefix. For example, if the csv file contains time in milliseconds, this function will attempt to detect that and return the data in units of seconds. If the function fails to make this conversion, a warning message will be printed.

Parameters **`filepath`** – A string containing the path to the csv file

Returns A tuple `(time, CHA, CHB)` where each element is a 1D numpy array containing an array of time points (x-axis of the oscilloscope), and the y-values for channels A and B respectively. The units of these should be seconds, Volts and Volts respectively unless otherwise stated via a warning message printed to your terminal.

`monashspa.PHS3000.`**`savefig`**(*fname*, *dpi=600*, *bbox_inches='tight'*, ***kwargs*)
A wrapper for `matplotlib.pyplot.savefig()` with sensible defaults.

By default, if a matplotlib legend is located outside of the plot axes, then `matplotlib.pyplot.savefig()` may cut off the legend when saving the figure. This function fixes this issue by setting `bbox_inches='tight'` and setting `bbox_extra_artists` to be a list of the current figure legends, unless the user chooses to define them otherwise. This function also sets the default for `dpi` to 600, which is large enough for most purposes.

This function accepts any keyword argument from `matplotlib.pyplot.savefig()` and returns the result of the call to that function. See `matplotlib.pyplot.savefig()` for usage.

1.7 PHS3302

This documents the `monashspa.PHS3302` library that you will import into code used in the PHS3302 unit.

The installation for this unit some extra packages are installed. You need to run:

```
pip install -U monashspa[PHS3302]
```

to get these extra modules in addition to the installation that you have already done. Alternatively, you can start the two exercises completely online by clicking on the two badges below

1.7.1 Calorimeter

1.7.2 Data selection

Python Module Index

m

Index

R

S

T

W