fiducia Documentation

Release 0.2.0

Pawel M. Kozlowski, Daniel H. Barnak, Myles T. Brophy

Mar 19, 2021

FIRST STEPS

Fiducia is an open source package for unfolding spectral information from filtered diode array diagnostics (such as Dante) using the cubic splines analysis method . This method simply assumes that the underlying spectrum is smoothly varying, and does not impose any other constraints on the shapes of spectrum. See below for instructions on how to install Fiducia, and for examples on how to run an analysis using Fiducia.

CHAPTER

ONE

INSTALLING FIDUCIA

1.1 Requirements

Fiducia require Python version 3.7 or newer. Fiducia also require the following openly available packages for installation:

- NumPy — 1.15.0 or newer
- SciPy — 1.1.0 or newer
- pandas — 0.23.0 or newer
- matplotlib — 3.0.0 or newer
- xarray — 0.15.1 or newer
- Astropy — 3.1 or newer

1.2 Installation with pip

Official releases of Fiducia are published to pypi.org and can simply be pip installed like so:

```
pip install fiducia
```

1.3 Building and installing from source (for contributors)

1.3.1 Make sure you have python installed, preferably via Anaconda

Here is where you get Anaconda, and make sure to get the Python 3 version. https://www.anaconda.com/distribution/

1.3.2 Setup installation directory

Make a directory called "fiducia" in a sensible place on your system. Preferably in a directory where none of the higher level directory names have spaces in them.

1.3.3 Setup a virtual environment

If you have python installed via Anaconda, then create your virtual environment like this

```
conda create --name fiducia
```

1.3.4 Clone the repository using git

In the fiducia directory you created, run the following on the command line

```
git clone https://github.com/lanl/fiducia.git
```

1.3.5 Activate your virtual environment

Still on the command line, run

```
source activate fiducia
```

1.3.6 Install requirements

```
pip install -r requirements.txt
```

1.3.7 Install fiducia

If you are a user then do

```
pip install .
```

If you wish to help in developing fiducia, then do

```
pip install -e .
```

1.3.8 Test if install was successful

Open a python and try doing `import fiducia`. If all went well then you shouldn't get any error messages.

CHAPTER

TWO

EXAMPLES

2.1 General examples

General-purpose and introductory examples from Fiducia

2.1.1 Test Example for Sphinx Docs

We create a test plot to see if sphinx gallery works

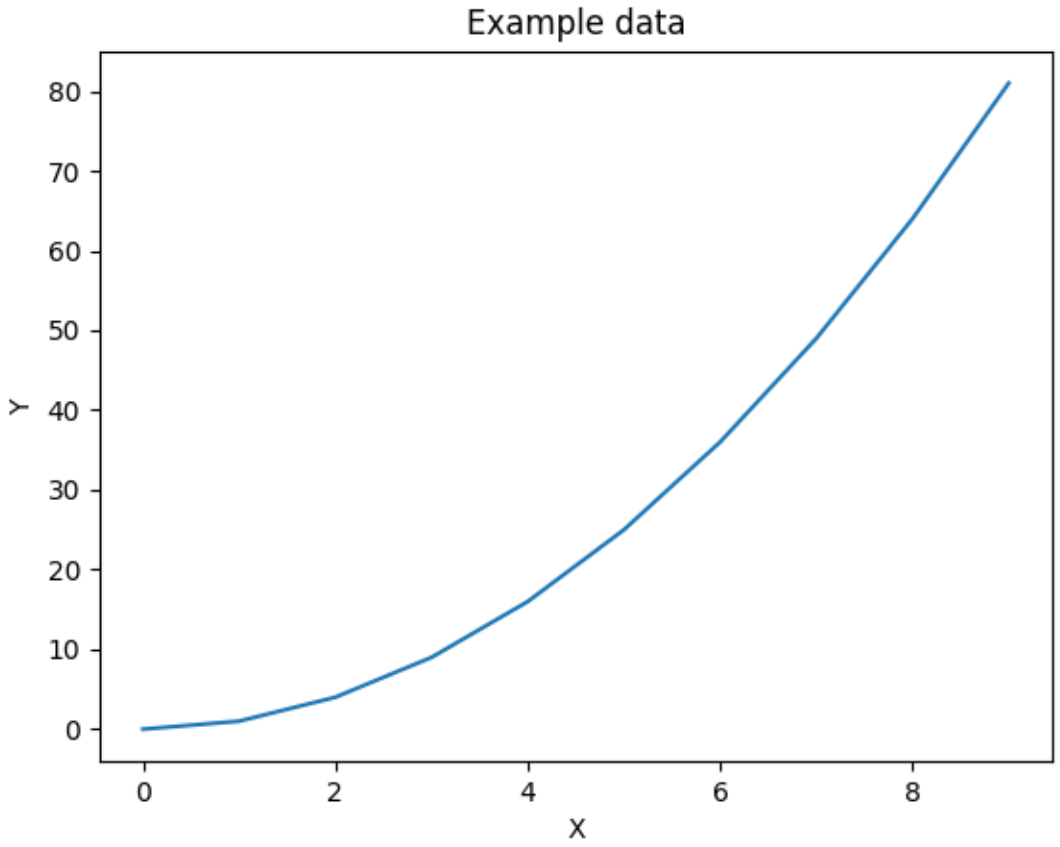
```
import numpy as np
import fiducia
import matplotlib.pyplot as plt
```

Generating some data

```
dataX = np.arange(10)
dataY = dataX ** 2
```

Plotting the data

```
plt.plot(dataX, dataY)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Example data')
plt.show()
```



Total running time of the script: (0 minutes 0.156 seconds)

CHAPTER

THREE

HOW TO CONTRIBUTE

Visit our GitHub repository and look through the list of open issues to see how you can contribute. If you find a bug, or would like to see a feature enhancement, then open up an issue and describe it detail.

CHAPTER

FOUR

ACKNOWLEDGING AND CITING

If you use Fiducia for work/research presented in a publication (whether directly, or as a dependency to another package), we encourage the following acknowledgement:

> This research made use of Fiducia, a community-developed Python package for analysis of filtered diode array signals.

and that you cite the following paper(s):

```
@article{barnak2020soft,
  title={Soft x-ray spectrum unfold of K-edge filtered x-ray diode arrays using cubic␣
↪splines},
  author={Barnak, DH and Davies, JR and Knauer, JP and Kozlowski, Pawel Marek},
  journal={Review of Scientific Instruments},
  volume={91},
  number={7},
  pages={073102},
  year={2020},
  publisher={AIP Publishing LLC}
  url={https://doi.org/10.1063/5.0002856}
}
```

CHAPTER

FIVE

FIDUCIA LICENSE (BSD 3-CLAUSE)

This program is open source under the BSD-3 License. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3.Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER

SIX

CUBIC SPLINE MATRICES (*FIDUCIA.CSPLINE*)

Created on Fri Mar 8 09:41:36 2019

Functions for working with cubic spline equation in matrix form.

@author: Pawel M. Kozlowski

6.1 Functions

splineCoords(energy, energyStart, energyEnd)	Convert photon energy value into normalized coordinates for a particular spline region.
splineCoordsInv(energyNorm, energyStart, ...)	Given a normalized energy value and the bounds of a spline segment, return the un-normalized photon energy value.
yCoeffArr(energyNorm, chLen)	Returns the matrix M_y(t) for a given value of t in:
dCoeffArr(energyNorm, chLen)	Returns the matrix M_D(t) for a given value of t in:
dToyArr(chLen)	Construct matrix for converting from D_i to y_i vector.
responseInterp(energyNorm, energyMin, ...)	Given a DANTE detector response as a function of energy, convert the response to normalized photon energy, t, over a given spline segment, and return interpolated response values for a given value of t.
yChiCoeffArr(energyNorm, chLen, dToY)	This is the matrix corresponding to:
yChiCoeffArrEnergies(energyNorms, chLen, dToY)	This is the matrix corresponding to:
fancyTrapz2(energyNorms, yChis, segments, ...)	Trap rule integration of the folding between our $M_{y\chi}$ matrix and response function matrix, with respect to normalized photon energy, for each channel.
segmentsArr(knots)	Returns the bounds of each spline segment, given the spline knot points.
detectorArr(channels, knots, responseFrame)	Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum.
knotSolve(signals, detArr, ... [, ...])	Get knot points y_i from measured DANTE signals S_d.
reconstructSpectrum(chLen, knots, knotsY[, ...])	Reconstruct the inferred DANTE spectrum given the knot points y_i obtained from knotSolve().

6.1.1 splineCoords

fiducia.cspline.**splineCoords**(*energy*, *energyStart*, *energyEnd*)

Convert photon energy value into normalized coordinates for a particular spline region.

Parameters

- **energy** (*float, numpy.ndarray*) – Energy value to be converted into normalized spline coordinate.
- **energyStart** (*float*) – Lower bound energy for the spline region based on knot points.
- **energyEnd** (*float*) – Upper bound energy for the spline region based on knot points.
- **normCoord** (*float*) – Return value of energy converted into normalized spline coordinates.

Returns **normCoord** – Normalized energy coordinate(s).

Return type float, numpy.ndarray

Notes

Examples

6.1.2 splineCoordsInv

fiducia.cspline.**splineCoordsInv**(*energyNorm*, *energyStart*, *energyEnd*)

Given a normalized energy value and the bounds of a spline segment, return the un-normalized photon energy value. This is the inverse of splineCoords().

Parameters

- **energyNorm** (*float, numpy.ndarray*) – Normalized photon energy
- **energyStart** (*float*) – Lower bound energy for the spline region based on knot points.
- **energyEnd** (*float*) – Upper bound energy for the spline region based on knot points.

Returns **energy** – Absolute photon energy (un-normalized).

Return type float, numpy.ndarray

Notes

Examples

6.1.3 yCoeffArr

fiducia.cspline.**yCoeffArr**(*energyNorm*, *chLen*)

Returns the matrix M_y(t) for a given value of t in:

$$Y_i(t) = M_y(t)y_i + M_D(t)D_i$$

Parameters

- **energyNorm** (*float*) – normalized photon energy for a spline section.
- **chLen** (*int*) – Number of DANTE channels (equal to number of spline knots).

Returns **mArr** – Sparse matrix $M_y(t)$.

Return type scipy.sparse.lil.lil_matrix

Notes

Examples

6.1.4 dCoeffArr

fiducia.cspline.**dCoeffArr**(*energyNorm*, *chLen*)

Returns the matrix M_D(t) for a given value of t in:

$$Y_i(t) = M_y(t)y_i + M_D(t)D_i$$

Parameters

- **energyNorm** (*float*) – normalized photon energy for a spline section.
- **chLen** (*int*) – Number of DANTE channels (equal to number of spline knots).

Returns **dArr** – Sparse matrix $M_D(t)$.

Return type scipy.sparse.lil.lil_matrix

Notes

Examples

6.1.5 dToyArr

fiducia.cspline.**dToyArr**(*chLen*)

Construct matrix for converting from D_i to y_i vector.

Parameters **chLen** (*int*) – Number of DANTE channels (equal to number of spline knots).

Returns **diToyi** – Matrix for converting from D_i to y_i vector.

Return type numpy.ndarray

Notes

The matrix is given by:

$$D_i = 3\chi_1^{-1}\chi_3 y_i$$

Examples

6.1.6 responseInterp

fiducia.cspline.**responseInterp**(*energyNorm*, *energyMin*, *energyMax*, *responseFrame*, *channels*)

Given a DANTE detector response as a function of energy, convert the response to normalized photon energy, t, over a given spline segment, and return interpolated response values for a given value of t. Returns an array of interpolated responses corresponding to the number of channels.

Parameters

- **energyNorm** (*float, numpy.ndarray*) – normalized photon energy
- **energyMin** (*float*) – Lower bound photon energy of the spline segment over which we are normalizing.
- **eneryMax** (*float*) – Upper bound photon energy of the spline segment over which we are normalizing.
- **responseFrame** (*pandas.core.frame.DataFrame*) – DANTE channel responses as a function of photon energy (not normalized).
- **channels** (*numpy.ndarray*) – numpy array of DANTE channel numbers.

Returns **responsesInterpd** – Returns a matrix of (energyNorms, channels) of response functions.

Return type numpy.ndarray

Notes

Examples

6.1.7 yChiCoeffArr

fiducia.cspline.**yChiCoeffArr**(*energyNorm*, *chLen*, *dToY*)

This is the matrix corresponding to:

$$M_y(t) + 3M_D(t)\chi_1^{-1}\chi_3$$

which describes the cubic spline interpolation of the x-ray spectrum.

Parameters

- **energyNorm** (*float*) – normalized photon energy
- **chLen** (*int*) – Number of DANTE channels (equal to number of spline knots).
- **dToY** (*numpy.ndarray*) – Matrix for converting from D_i to y_i values in cubic spline interpolation. See dToyArr().

Returns **yChiArr** – Returns a 2D matrix for a particular value of energyNorm.

Return type numpy.ndarray

Notes

The matrix is given by:

$$M_y(t) + 3M_D(t)\chi_1^{-1}\chi_3$$

Examples

6.1.8 yChiCoeffArrEnergies

fiducia.cspline.**yChiCoeffArrEnergies**(*energyNorms*, *chLen*, *dToY*)

This is the matrix corresponding to:

$$M_y(t) + 3M_D(t)\chi_1^{-1}\chi_3$$

which describes the cubic spline interpolation of the x-ray spectrum.

energyNorms: numpy.ndarray Vector of normalized photon energies.

chLen: int Number of DANTE channels (equal to number of spline knots).

dToY: numpy.ndarray Matrix for converting from D_i to y_i values in cubic spline interpolation. See dToyArr().

Returns **yChiArrEnergies** – Returns a 3D matrix composed of a series of 2D yChiCoeff matrices corresponding to the given energyNorm values. This matrix is indexed as (energyNorms, knotIndex, knotIndex).

Return type numpy.ndarray

Notes

The matrix is given by:

$$M_y(t) + 3M_D(t)\chi_1^{-1}\chi_3$$

Examples

6.1.9 fancyTrapz2

fiducia.cspline.**fancyTrapz2**(*energyNorms*, *yChis*, *segments*, *responseFrame*, *channels*)

Trap rule integration of the folding between our $M_{y\chi}$ matrix and response function matrix, with respect to normalized photon energy, for each channel. The result should be a matrix with shape (*len(channels)*, *len(segments)*, *len(knotIndex)*).

Parameters

- **energyNorms** (*numpy.ndarray*) – 1D array of normalized photon energies
- **yChis** (*numpy.ndarray*) – 3D array of $M_{y\chi}$ values corresponding to (energyNorms, segments, knotIndex).
- **responses** (*numpy.ndarray*) – 2D array of DANTE channel response functions corresponding to (energyNorms, channels).
- **channels** (*numpy.ndarray*) – Array of DANTE channel numbers.

Returns **integArr** – A matrix containing the folded integration of the $M_{y\chi}$ matrix and response function matrix, with respect to normalized photon energy. Has shape (*len(channels)*, *len(segments)*, *len(knotIndex)*).

Return type xarray.DataArray

Notes

Examples

6.1.10 segmentsArr

fiducia.cspline.**segmentsArr**(*knots*)

Returns the bounds of each spline segment, given the spline knot points.

Returns an array of tuples of (energyMin, energyMax) describing the bounds of each spline segment, given an array of spline knots (photon energies corresponding to K-edges).

Parameters **knots** (*numpy.ndarray*) – numpy array of photon energies describing positions of spline knots.

Returns **segments** – A 1D array of tuples of (energyMin, energyMax), corresponding to the bounds of each spline segment. Has length of 'len(knots) - 1'.

Return type numpy.ndarray

Notes

Examples

6.1.11 detectorArr

fiducia.cspline.**detectorArr**(*channels*, *knots*, *responseFrame*, *boundary='y0'*, *npts=1000*)

Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. This is applied to the measured DANTE channel signals to recover knot points y_i of the cubic spline, which can then be used to reconstruct the inferred x-ray spectrum.

Parameters

- **channels** (*numpy.ndarray*) – Array of DANTE channel numbers.
- **knots** (*numpy.ndarray*) – Array of photon energies describing positions of spline knots.
- **responseFrame** (*pandas.core.frame.DataFrame*) – DANTE channel responses as a function of photon energy (not normalized).
- **boundary** (*str, optional*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are 'y0' or 'yn+1'. Default 'y0'.
- **npts** (*int*) – Number of points used in computing the integral

Returns **detArr** – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. 2D array of channels and knot points of shape (n, n).

Return type numpy.ndarray

Notes

For each spline segment we have:

$$M_{stuff} = \int_0^1 (M_y(t) + 3M_D(t)\chi_1^{-1}\chi_3)R_d(t)dt$$

Each spline is then summed to form the full detector matrix for recovering the knot points.

Examples

6.1.12 knotSolve

fiducia.cspline.**knotSolve**(*signals*, *detArr*, *detArrBoundaryCol*, *detArrVarianceBoundaryCol*, *detArrInv*, *stdDetArrInv*, *signalsUncertainty=None*, *yGuess=1e-77*, *npts=1000*)

Get knot points y_i from measured DANTE signals S_d.

Parameters

- **signals** (*numpy.ndarray*) – numpy array of DANTE measured signal for each channel at a particular point in time.
- **detArr** (*numpy.ndarray*) – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. 2D array of channels and knot points of shape (n, n).
- **detArrBoundaryCol** (*xarray.DataArray*) – Column of cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrVarianceBoundaryCol** (*xarray.DataArray*) – Column of variances in the cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrInv** (*xarray.DataArray*) – Inversion of detArr, with the column corresponding to boundary removed so detArr is invertible.
- **stdDetArrInv** (*xarray.DataArray*) – Array of the standard deviation of each element in detArrInv based on variance using the 'responseUncertaintyFrame' propagated with Monte Carlo.
- **signalsUncertainty** (*xarray.DataArray, optional*) – numpy array of the uncertainty of the DANTE measured signal for each channel at a particular point in time. The default is None.
- **yGuess** (*float, optional*) – Guess for position of boundary knot point. Default is 1e-77.
- **npts** (*int, optional*) – Number of points used in computing the integral. Default is 1000.

Returns

- **knotsY** (*numpy.ndarray*) – Array of knot point intensity values with yGuess appended.
- **knotsYVariance** (*numpy.ndarray*) – Array with each element corresponding to the variance of the same element in 'knotsY'.

Notes

Examples

6.1.13 reconstructSpectrum

fiducia.cspline.**reconstructSpectrum**(*chLen*, *knots*, *knotsY*, *knotsYUncertainty=None*, *npts=1000*, *plot=False*)

Reconstruct the inferred DANTE spectrum given the knot points y_i obtained from knotSolve().

Parameters

- **chLen** (*int*) – Number of DANTE channels (equal to number of spline knots).
- **knots** (*list, numpy.ndarray*) – List or array of knot point photon energy value. See knotFind().
- **knotsY** (*numpy.ndarray*) – Array of knot point intensity values with yGuess appended. See knotSolve() and analyzeSpectrum().
- **knotsYUncertainty** (*numpy.ndarray*) – Array of knot point intensity uncertainty values with yGuess appended. See knotSolve() and analyzeSpectrum(). The default is None.
- **npts** (*int*) – Number of points used in computing the integral. The default is 1000.
- **plot** (*Bool*) – Flag for plotting unfolded spectrum. The default is False.

Returns

- **photonEnergies** (*numpy.ndarray*) – Photon energy axis of unfolded spectrum.
- **intensities** (*numpy.ndarray*) – Spectral intensity axis of unfolded spectrum.
- **intensitiesVariance** (*numpy.ndarray*) – Uncertaitny (1 σ) on spectral intensity values.

Notes

Examples

CHAPTER

SEVEN

SPLINE UNCERTAINTY PROPAGATION (*FIDUCIA.ERROR*)

Created on Tues June 16 13:48:21 2020

Utilities for calculating response uncertainty

@author: Myles Brophy

7.1 Functions

detectorErrMC(detArr, detArrVariance[, ...])	Monte Carlo simulation and statistics to determine cubic spline uncertainty.
knotVarianceFind(channels[, ...])	Modification of response.knotFind()
responseInterpVariance(energyNorm, ...)	Given a DANTE detector response as a function of energy, convert the response to normalized photon energy, t, over a given spline segment, and return interpolated response values for a given value of t.
fancyTrapz2Variance(energyNorms, yChis, ...)	Calculate the variance when propogating uncertainties through *fiducia.cspline.fancyTrapz2()*.
detectorArrVariance(channels, knots, ...[, ...])	Propagates uncertanity through `cspline.detectorArr()` to find the variance in :math:`M_{int}()`.
detectorUncertainty(channels, responseFile)	Finds the cspline detector matrix, it`s inverse matrix and std matrix using Monte Carlo uncertainty propagation.

7.1.1 detectorErrMC

fiducia.error.**detectorErrMC**(*detArr*, *detArrVariance*, *samples=10000*, *boundary='y0'*, *MChistogram=False*)

Monte Carlo simulation and statistics to determine cubic spline uncertainty.

Calculate the cubic spline matrix uncertainty using a Monte Carlo simulation and statistics on the MC's output.

Parameters

- **detArr** (*numpy.ndarray*) – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. See 'cspline.detectorArr()'.
- **detArrVariance** (*numpy.ndarray*) – A DataFrame containing the uncertainty for each Dante channel for the photon energy range that detArr spans.
- **samples** (*int, optional*) – Number of MCs amples to run. Default is *10000*.

- **boundary** (*str, optional*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are *y0* or *yn+1*. Default is 'y0'.
- **MChistogram** (*bool, optional*) – Plot histograms corresponding to each variant of detArr generated with Monte Carlo uncertainty propagation. Default is *False*.

Returns **stdErrorMatrix** – A numpy.ndarray with the standard deviation of the inverted matrices generated using random weights based on the channel uncertainty.

Return type numpy.ndarray

Raises

- **Exception** – If *boundary* doesn't equal *y0* or *yn+1*.
- **ValueError** – If the shapes of *detArr* and *detUncertaintyArr* aren't equal.

Notes

Examples

7.1.2 knotVarianceFind

fiducia.error.**knotVarianceFind**(*channels*, *responseUncertaintyFrame=None*, *forceKnot=array([], dtype=float64)*, *knotBoundaryY=1e-77*, *boundary='y0'*)

Modification of response.knotFind()

Parameters

- **channels** (*numpy.ndarray*) – Array of DANTE channel numbers.
- **responseUncertaintyFrame** (*pandas.core.frame.DataFrame, optional*) – DataFrame holding percent uncertainties of DANTE channel responses as a function of photon energy (not normalized). The default is *None*.
- **forceKnot** (*TYPE, optional*) – DESCRIPTION. The default is *np.array([])*.
- **knotBoundaryY** (*float, optional*) – Guess for position of y_0 or y_{n+1} knot point. Default is 1e-77.
- **boundary** (*str, optional*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are *y0* or *yn+1*. Default *y0*.

Returns **knotUncertainty** – An array of uncertainty in knot points, with each element corresponding to a channel or boundary condition. See `response.knotFind()`.

Return type numpy.ndarray

Notes

Examples

7.1.3 responseInterpVariance

fiducia.error.**responseInterpVariance**(*energyNorm*, *energyMin*, *energyMax*, *responseUncertaintyFrame*, *channels*)

Given a DANTE detector response as a function of energy, convert the response to normalized photon energy, t, over a given spline segment, and return interpolated response values for a given value of t. Returns an array of interpolated responses corresponding to the number of channels.

Parameters

- **energyNorm** (*float, numpy.ndarray*) – normalized photon energy
- **energyMin** (*float*) – Lower bound photon energy of the spline segment over which we are normalizing.
- **eneryMax** (*float*) – Upper bound photon energy of the spline segment over which we are normalizing.
- **responseFrame** (*pandas.core.frame.DataFrame*) – DANTE channel responses as a function of photon energy (not normalized).
- **channels** (*numpy.ndarray*) – numpy array of DANTE channel numbers.

Returns **responsesInterpdVariance** – Returns a matrix of (energyNorms, channels) of response functions.

Return type numpy.ndarray

Notes

See also:

`cspline.repsonseInterp`

Examples

7.1.4 fancyTrapz2Variance

fiducia.error.**fancyTrapz2Variance**(*energyNorms*, *yChis*, *segments*, *responseUncertaintyFrame*, *channels*, *interpProp=True*)

Calculate the variance when propogating uncertainties through *fiducia.cspline.fancyTrapz2()*.

Parameters

- **energyNorms** (*numpy.ndarray*) – Array of normalized energies over which the integral is computed.
- **yChis** (*numpy.ndarray*) – 3D array corresponding to the $M_{y\chi}$ coefficients. Array shape corresponds to (*energyNorms*, *chLen*, *dToY*). See *fiducia.error.detectorArrVariance()*
- **segments** (*numpy.ndarray*) – Array of segments produced by `segmentsArr()` with the knots

- **responseUncertaintyFrame** (*pandas.core.frame.DataFrame*) – DataFrame holding uncertainty percentages of DANTE channel responses as a function of photon energy (not normalized).
- **channels** (*numpy.ndarray*) – Array of DANTE channel numbers.
- **interpProp** (*bool, optional*) – Boolean to decide if `error.responseInterpVariance()` should be used. If *False, :func:`cspline.responseInterp()* is used, speeding up the calculation. Note that the uncertainty is would not be propagated correctly if *False*. With future optimizations, this option to choose may be removed. Default is *True*.

Returns **integArrVariance** – A matrix containing the folded integration of the $M_{y\chi}$ matrix and response function uncertainty matrix, with respect to normalized photon energy. Has shape (*len(channels)*, *len(segments)*, *len(knotIndex)*).

Return type xarray.Dat

Notes

See also:

`cspline.fancyTrapz2`

Examples

7.1.5 detectorArrVariance

`fiducia.error.`**`detectorArrVariance`**(*channels*, *knots*, *responseUncertaintyFrame*, *boundary='y0'*, *npts=1000*)

Propagates uncertanity through `cspline.detectorArr()` to find the variance in :math:`M_{int}()`.

Parameters

- **channels** (*numpy.ndarray*) – Array of DANTE channel numbers.
- **knots** (*numpy.ndarray*) – Array of photon energies describing positions of spline knots.
- **responseUncertaintyFrame** (*pandas.core.frame.DataFrame*) – DataFrame holding uncertainty percentages of DANTE channel responses as a function of photon energy (not normalized).
- **npts** (*int, optional*) – Number of points used in computing the integral. The default is 1000.

Returns

- **detArrVariance** (*xarray.DataArray*) – 2D array of channels and knot points uncertainties of shape *(n, n+1)*.
- **detArrVarianceBoundaryCol** (*xarray.DataArray*) – Column of variances in the cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.

Notes

Covariances between segments is not currently accounted for. This covariance should be small compared to the other uncertainties, but should be noted.

See also:

`cspline.detectorArr`

Examples

7.1.6 detectorUncertainty

`fiducia.error.`**`detectorUncertainty`**(*channels*, *responseFile*, *responseUncertaintyFile=None*, *boundary='y0'*, *npts=1000*, *samples=1000*, *MChistogram=False*, *saveDataset=True*, *csplineDatasetFile=''*)

Finds the cspline detector matrix, it`s inverse matrix and std matrix using Monte Carlo uncertainty propagation.

Propagates response uncertainties through

Parameters

- **`channels`** (*`numpy.ndarray`*) – Array of DANTE channel numbers.
- **`responseFile`** (*`str`*) – Path to the *.csv* holding DANTE channel responses as a function of photon energy (not normalized).
- **`responseUncertaintyFile`** (*`str, optional`*) – Path to the *.csv* holding DANTE channel response uncertainties as a function of photon energy. Uncertainty values provided as percentages.
- **`boundary`** (*`str, optional`*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are *y0* or *yn+1*. Default 'y0'.
- **`npts`** (*`int, optional`*) – Number of points used in computing the integral. Default is 1000.
- **`samples`** (*`int, optional`*) – Number of samples to generate during Monte Carlo propagation. See `error.detectorErrMC()`. Default is 1000.

Returns

- **detArr** (*xarray.DataArray*) – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. 2D array of channels and knot points of shape (n, n).
- **detArrBoundaryCol** (*xarray.DataArray*) – Column of cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrVarianceBoundaryCol** (*xarray.DataArray*) – Column of variances in the cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrInv** (*xarray.DataArray*) – Inversion of detArr, with the column corresponding to boundary removed so detArr is invertible.
- **stdDetArrInv** (*xarray.DataArray*) – Array of the standard deviation of each element in detArrInv based on variance using the *responseUncertaintyFrame* propagated with Monte Carlo.

Notes

Examples

CHAPTER

EIGHT

DATA LOADING UTILITIES (*FIDUCIA.LOADER*)

Created on Fri Mar 8 09:20:37 2019

Utilities for loading DANTE measurement and response function data.

@author: Pawel M. Kozlowski

8.1 Functions

cleanupHeader(dataFrame)	Strip whitespace and rename DataFrame headers.
loadResponses(channels, fileName[, solid])	Load DANTE measurement data from files given the channels and path to the directory containing the response function files.
loadResponseUncertainty(responseFrame, fileName)	Load uncertainty percentages into a DataFrame.
readDanProcessed(channels, directory)	Loads DANTE measurement data from files given the channels and path to the directory containing the reduced and aligned DANTE data.
signalsAtTime(time, measurementFrame, channels)	Get DANTE signals from each channel at a particular time.
signalInt(channels, measurementFrame, ...)	Get time-integrated Dante signals for a specified time interval.
readDanteData(filePath)	Reads Dante .dat file and returns header info and channel signals as two separate pandas dataframes.

8.1.1 cleanupHeader

fiducia.loader.**cleanupHeader**(*dataFrame*)

Strip whitespace and rename DataFrame headers.

Parameters **dataFrame** (*pandas.core.frame.DataFrame*) – DataFrame to be cleaned.

Returns **cleanedDataFrame** – DataFrame with stripped and renamed channel headers.

Return type pandas.core.frame.DataFrame

Notes

Examples

8.1.2 loadResponses

fiducia.loader.**loadResponses**(*channels*, *fileName*, *solid=True*)

Load DANTE measurement data from files given the channels and path to the directory containing the response function files. Returns a dataframe with the data.

Parameters

- **channels** (*list, numpy.ndarray*) – List or array of relevant channels
- **fileName** (*str*) – Full path and filename of .csv file containing DANTE respones functions.
- **solid** (*Bool, optional*) – Includes solid angle in response function value if true. The default is true.

Returns **responseFrame** – DataFrame with the response function data for the 'channels' requested

Return type pandas.core.frame.DataFrame

Notes

Examples

8.1.3 loadResponseUncertainty

fiducia.loader.**loadResponseUncertainty**(*responseFrame*, *fileName*)

Load uncertainty percentages into a DataFrame.

Parameters

- **responseFrame** (*pandas.core.frame.DataFrame*) – DataFrame to base the respones uncertainty frame on.
- **fileName** (*str*) – Full path and filename of .csv file containing DANTE response uncertainty percentages functions.

Returns **responseUncertaintyFrame** – DataFrame with each column being a channel and each element being the channel's uncertainty percentage. Extended to match the photon energy range in the response frame.

Return type pandas.core.frame.DataFrame

Notes

Examples

8.1.4 readDanProcessed

fiducia.loader.**readDanProcessed**(*channels*, *directory*)

Loads DANTE measurement data from files given the channels and path to the directory containing the reduced and aligned DANTE data. Returns a dataframe with the data. Note that this is *not* for raw data. It is for reading DANTE signals that have already been processed by Dan Barnak's scripts.

Parameters

- **channels** (*list, numpy.ndarray*) – List or array of relevant channels
- **directory** (*str*) – Path to channel response function files

Returns **dataFrame** – Dataframe of aligned signals from Dan's analysis.

Return type pandas.core.frame.DataFrame

Notes

Examples

8.1.5 signalsAtTime

fiducia.loader.**signalsAtTime**(*time*, *measurementFrame*, *channels*, *plot=False*, *method='interp'*)

Get DANTE signals from each channel at a particular time. Default is to return an interpolated value of the signal at the given time. Alternatively, this function can return the nearest value in the signal data array for the given time.

Parameters

- **time** (*float*) – Time for which we want DANTE signals (in ns).
- **measurementFrame** (*pandas.core.frame.DataFrame*) – Pandas dataframe containing DANTE measurement data. See readDanteData() and readDanProcessed().
- **plot** (*Bool*) – When True, plots DANTE signals vs channel index at a particular time.
- **method** (*str*) – Either 'nearest' or 'interp'. 'nearest' finds the nearest point in the DANTE signal to the given time. 'interp' returns an interpolated signal value for the given time. Default is 'interp'.

Returns **signals** – Dante signals for each channel at a particular time step.

Return type numpy.ndarray

Notes

Examples

8.1.6 signalInt

fiducia.loader.**signalInt**(*channels*, *measurementFrame*, *tStart*, *tEnd*)

Get time-integrated Dante signals for a specified time interval. Used in getting time-integrated spectrum from the unfold.

Parameters

- **measurementFrame** (*pandas.core.frame.DataFrame*) – Pandas dataframe containing DANTE measurement data. See loadDanteData().
- **tStart** (*float*) – Lower bound for time integration.
- **tEnd** (*float*) – Upper bound for time integration

Returns **signalInt** – Time integrated Dante signals for each channel.

Return type numpy.ndarray

Notes

Examples

8.1.7 readDanteData

fiducia.loader.**readDanteData**(*filePath*)

Reads Dante .dat file and returns header info and channel signals as two separate pandas dataframes.

Parameters **filePath** (*str*) – Full path to the Dante .dat file.

Returns

- **headerFrame** (*pandas.core.frame.DataFrame*) – Header of Dante data file. This typically include information about the various components used in each Dante channel, such as oscilloscopes, XRDs, etc.
- **dataFrame** (*pandas.core.frame.DataFrame*) – Dante data.

Notes

Examples

CHAPTER

NINE

FIDUCIA MAIN FILE (*FIDUCIA.MAIN*)

Created on Fri Jan 25 12:22:01 2019

FIDUCIA: Filtered Diode Unfolder (using) Cubic Spline Algorithm

DANTE spectrum deconvolver based on cubic splines method [1]. Translated from Dan Barnak's Mathematica code.

DANTE channels are bounded by edge absorption feature (knot point) due to filter for the respective channel. Cubic splines representing the estimated spectrum are fitted in each spectral region bounded by knot points. The detector signal for each channel is then equal to the response function of the detector folded with the matrix representation of the cubic spline. A triadiagonal matrix representation of the cubic spline equation is used to make the problem numerically tractable. This way a matrix inversion can be used to solve for the unknown coefficients in the cubic spline equation, using the measured signals. These coefficients are then plugged back into the cubic spline equation over each interval (between knot points) to make a piecewise reconstruction of the x-ray spectrum at each time step.

References

Cubic spline deconvolution method [1] J. P. Knauer and N. C. Gindele. Temporal and spectral deconvolution of data from diamond, photoconductive devices. Rev. Sci. Instrum. 75, 3714 (2004) https://doi.org/10.1063/1.1785274

Error propagation for cubic spline deconvolution method [2] D. L. Fehl and F. Briggs. Verification of unfold error estimates in the unfold operator code. Rev. Sci. Instrum. 68, 890 (1997) https://doi.org/10.1063/1.1147713

Useful description of cubic spline matrix representation [3] http://mathworld.wolfram.com/CubicSpline.html

Paper comparing cubic splines unfolds to other methods [4] D. H. Barnak, J. R. Davies, J. P. Knauer, and P. M. Kozlowski. Soft x-ray spectrum unfold of K-edge filtered x-ray diode arrays using cubic splines. Submitted to Review of Scientific Instruments in 2020.

@author: Pawel M. Kozlowski

9.1 Functions

simulateSignal()	Takes the inferred spectrum and folds it with the instrument function to retrieve the forward propagated signal for each DANTE channel.
inferRadTemp(power, area, angle[, ...])	Gets the inferred radiation temperature by calculating in from radiated power through the Stefan-Boltzmann Law.
inferPower(energies, spectra[, ...])	Gets the inferred total radiation power as a function of time.

continues on next page

Table 1 – continued from previous page

analyzeSpectrum(channels, knots, detArr, ...)	Given the response function file and the DANTE measurement data file, run cubic spline analysis to reconstruct spectrum for a given time.
analyzeStreak(channels, responseFrame, ...)	Given the response function file and the DANTE measurement data file, run cubic spline analysis to reconstruct spectrum for a given time.
feelingLucky(dataFile, attenuatorsFile, ...)	Attempt processing dante signals given dante data file and calibration files using sensible defaults.

9.1.1 simulateSignal

fiducia.main.**simulateSignal**()

Takes the inferred spectrum and folds it with the instrument function to retrieve the forward propagated signal for each DANTE channel.

Notes

Examples

9.1.2 inferRadTemp

fiducia.main.**inferRadTemp**(*power*, *area*, *angle*, *powerUncertainty=None*)

Gets the inferred radiation temperature by calculating in from radiated power through the Stefan-Boltzmann Law.

Parameters

- **power** (*float, np.ndarray*) – Total radiated power as a function of time calculated from unfolded spectra. See `main.inferPower()`.
- **area** (*float*) – Area of emitting surface in units of mm^2. For hohlraums/halfraums, this is the area of the LEH.
- **angle** (*float*) – Angle between the surface area normal and the Dante line of sight in degrees. Usually 37.4 degrees for hohlraums/halfraums. Must be between 0 and 90 degrees.
- **powerUncertainty** (*float, np.ndarray, optional*) – Uncertainty in total radiated power as a function of time calculated from unfolded spectra. See `main.inferPower()`. The default is None.

Returns

- **tRad** (*numpy.ndarray*) – Radiation temperature of the blackbody emitter.
- **tRadVariance** (*numpy.ndarray*) – Variance σ^2 on the radiation temperature.

Notes

Total x-ray flux (power) from a black body emitter is given by:

$$P = \sigma_{SB} A \cos(\theta) T^4$$

Where P = power, σ_{SB} = Stefan-Boltzmann constant, A is the area of radiating surface, θ is the viewing angle between the surface area normal and the Dante line-of-sight, T is the radiation temperature of the black body emitter.

Notes

Examples

9.1.3 inferPower

fiducia.main.**inferPower**(*energies*, *spectra*, *spectraUncertainty=None*)

Gets the inferred total radiation power as a function of time.

Parameters

- **energies** (*numpy.ndarray*) – Photon energies corresponding to input spectrum
- **spectra** (*numpy.ndarray*) – Spectral Flux values as a function of photon energy in units of (GW/sr/eV)

Returns

- **power** (*numpy.ndarray*) – Total x-ray power (flux) as a function of time.
- **powerVariance** (*numpy.ndarray*) – Variance σ^2 on total x-ray power.

Notes

Examples

9.1.4 analyzeSpectrum

fiducia.main.**analyzeSpectrum**(*channels*, *knots*, *detArr*, *detArrBoundaryCol*, *detArrVarianceBoundaryCol*, *detArrInv*, *stdDetArrInv*, *measurementFrame*, *time*, *signalsUncertainty=None*, *yGuess=0*, *boundary='y0'*, *nPtsIntegral=100*, *nPtsSpectrum=100*, *plotSignal=False*, *plotKnots=False*, *plotSpectrum=True*)

Given the response function file and the DANTE measurement data file, run cubic spline analysis to reconstruct spectrum for a given time.

Parameters

- **channels** (*list, numpy.ndarray*) – List or array of relevant DANTE channel numbers.
- **responseFrame** (*pandas.core.frame.DataFrame*) – Pandas dataFrame containing response functions for each DANTE channel. See loadResponses().
- **knots** (*list, numpy.ndarray*) – List or array of knot point photon energy value. See knotFind().

- **detArr** (*xarray.DataArray*) – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. 2D array of channels and knot points of shape (n, n).
- **detArrBoundaryCol** (*xarray.DataArray*) – Column of cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrVarianceBoundaryCol** (*xarray.DataArray*) – Column of variances in the cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrInv** (*xarray.DataArray*) – Inversion of detArr, with the column corresponding to boundary removed so detArr is invertible.
- **stdDetArrInv** (*xarray.DataArray*) – Array of the standard deviation of each element in detArrInv based on variance using the *responseUncertaintyFrame* propagated with Monte Carlo.
- **measurementFrame** (*pandas.core.frame.DataFrame*) – Pandas dataframe containing DANTE measurement data. See readDanteData() and readDanProcessed().
- **time** (*float*) – Time for which we want DANTE signals (in ns).
- **signalsUncertainty** (*numpy.ndarray, optional*) – One dimensional array with each element corresponding to the uncertainty each signal. The default is None.
- **yGuess** (*float, optional*) – Guess for position of boundary knot point. Default 0.
- **boundary** (*str, optional*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are *y0* or *yn+1*. Default 'y0'.
- **nPtsIntegral** (*int, optional*) – Number of points used in computing the integral. Default is 100.
- **nPtsSpectrum** (*int, optional*) – Number of points to use in reconstructing the spectrum. Default is 100.
- **plotKnots** (*Bool, optional*) – Flag for plotting the Dante signal at the given time across all channels. Default is False.
- **plotKnots** – Flag for plotting just the solved knot points. Default is False.
- **plotSpectrum** (*Bool, optional*) – Flag for plotting the unfolded spectrum. Default is True.

Returns

- **knotsYAll** (*numpy.ndarray*) – Array of knot point intensity values with yGuess appended. See knotSolve() and analyzeSpectrum().
- **knotsYVariance** (*numpy.ndarray*) – Array of knot point intensity uncertainty values with yGuess appended. See knotSolve() and analyzeSpectrum().
- **photonEnergies** (*numpy.ndarray*) – Photon energy axis of unfolded spectrum.
- **intensities** (*numpy.ndarray*) – Spectral intensity axis of unfolded spectrum.
- **intensitiesVariance** (*numpy.ndarray*) – Uncertaitny (1 σ) on spectral intensity values.

Notes

Examples

9.1.5 analyzeStreak

fiducia.main.**analyzeStreak**(*channels*, *responseFrame*, *knots*, *detArr*, *detArrBoundaryCol*, *detArrVarianceBoundaryCol*, *detArrInv*, *stdDetArrInv*, *measurementFrame*, *timeStart*, *timeStop*, *timeStep*, *signalsUncertainty=None*, *yGuess=0*, *boundary='y0'*, *nPtsIntegral=100*, *nPtsSpectrum=100*)

Given the response function file and the DANTE measurement data file, run cubic spline analysis to reconstruct spectrum for a given time.

Parameters

- **channels** (*list, numpy.ndarray*) – List or array of relevant DANTE channel numbers.
- **responseFrame** (*pandas.core.frame.DataFrame*) – Pandas dataFrame containing response functions for each DANTE channel. See loadResponses().
- **knots** (*list, numpy.ndarray*) – List or array of knot point photon energy value. See knotFind().
- **detArr** (*xarray.DataArray*) – Matrix representing the spectrally integrated folding of the detector response with a cubic spline interpolation of the x-ray spectrum. 2D array of channels and knot points of shape (n, n).
- **detArrBoundaryCol** (*xarray.DataArray*) – Column of cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrVarianceBoundaryCol** (*xarray.DataArray*) – Column of variances in the cublic spline matrix corresponding to the knots at the boundary chosen with *boundary*.
- **detArrInv** (*xarray.DataArray*) – Inversion of detArr, with the column corresponding to boundary removed so detArr is invertible.
- **stdDetArrInv** (*xarray.DataArray*) – Array of the standard deviation of each element in detArrInv based on variance using the *responseUncertaintyFrame* propagated with Monte Carlo.
- **measurementFrame** (*pandas.core.frame.DataFrame*) – Pandas dataframe containing DANTE measurement data. See `loader.readDanteData()` and `readDanProcessed()`.
- **timeStart** (*float*) – Start time for producing temporally streaked DANTE spectra (in ns).
- **timeStop** (*float*) – End time for producing temporally streaked DANTE spectra (in ns).
- **timeStep** (*float*) – Time step size for producing temporally streaked DANTE spectra (in ns).
- **signalsUncertainty** (*numpy.ndarray, optional*) – One dimensional array with each element corresponding to the uncertainty each signal. The default is None.
- **yGuess** (*float, optional*) – Guess for position of boundary knot point. Default is 1e-77.
- **boundary** (*str, optional*) – Choose whether yGuess corresponds to y_0 (lowest photon energy) or y_{n+1} (highest photon energy) boundary condition. This should correspond to the photon energy value given in knots. Options are *y0* or *yn+1*. Default 'y0'.

- **nPtsIntegral** (*int, optional*) – Number of points used in computing the integral. Default is 100.
- **nPtsSpectrum** (*int, optional*) – Number of points to use in reconstructing the spectrum. Default is 100.

Returns

Return type times

energies

spectra

spectraVariance

Notes

Examples

9.1.6 feelingLucky

fiducia.main.**feelingLucky**(*dataFile*, *attenuatorsFile*, *offsetsFile*, *responseFile*, *csplineDatasetFile*, *channels*, *area*, *angle*, *signalsUncertainty=None*, *peaksNum=2*)

Attempt processing dante signals given dante data file and calibration files using sensible defaults.

Parameters

- **dataFile** (*str*) – Full path to the Dante .dat file containing dante signals from LLE site.
- **attenuatorsFile** (*str*) – Full path to file containing attenuator serial numbers and corresponding attenuation factors.
- **offsetsFile** (*str*) – Full path to file containing oscilloscope channel offsets in time and voltage.
- **responseFile** (*str*) – Full path and filename of .csv file containing DANTE respones functions corresponding to dataFile.
- **csplineDatasetFile** (*str*) – File pointing to the path of the saved dataset containing ''detArr", ''detArrBoundaryCol", ''detArrInv", and ''stdDetArrInv". See :func:'error.analyzeSpectrumUncertainty()'.
- **channels** (*list, numpy.ndarray*) – List or array of relevant channels for which to apply analysis.
- **area** (*float*) – Area of emitting surface in units of mm^2. For hohlraums/halfraums, this is the area of the LEH. Used in Trad calculation.
- **angle** (*float*) – Angle between the surface area normal and the Dante line of sight in degrees. Usually 37.4 degrees for hohlraums/halfraums. Used in Trad calculation.
- **signalsUncertainty** (*numpy.ndarray, optional*) – One dimensional array with each element corresponding to the uncertainty each signal. The default is None.

Notes

Examples

CHAPTER

TEN

MISCELLANEOUS FUNCTIONS (*FIDUCIA.MISC*)

Created on Fri Mar 8 09:25:05 2019

Miscellaneous utilities

@author: Pawel M. Kozlowski

10.1 Functions

find_nearest(array, value)	Find nearest value in array and return index, and value as a tuple.
areDataFramesCompatible(channels, *frames)	Check DataFrame compatibility for specified channels.

10.1.1 find_nearest

fiducia.misc.**find_nearest**(*array*, *value*)

Find nearest value in array and return index, and value as a tuple.

Parameters

- **array** (*list, numpy.ndarray*) – Array of values to be searched.
- **value** (*int, float*) – Value for which this function will find the nearest value in the array.

Returns

- **idx** (*int*) – Index at which nearest value to input value occurs in the array.
- **array[idx]** (*int, float*) – The nearest value to the input value.

Notes

Examples

10.1.2 areDataFramesCompatible

fiducia.misc.**areDataFramesCompatible**(*channels*, **frames*)

Check DataFrame compatibility for specified channels.

Checks if multiple pandas.core.frame.DataFrame objects are compatible and have the channels that are requested. Checks that the DataFrames span the same energy range. Returns true if the frames pass all checks, false otherwise.

Parameters

- **channels** (*list*) – List of relevant channels
- ***frames** (*pandas.core.frame.DataFrame*) – The DataFrames that you want to check for compatiblity with the relevant channels

Returns True if frames are compatible with the requested channels, and False otherwise.

Return type bool

Notes

Examples

CHAPTER

ELEVEN

PLOT DEFAULTS (*FIDUCIA.PLTDEFAULTS*)

Created on Fri Oct 27 02:37:12 2017

Default plotting parameters

@author: Pawel M. Kozlowski

11.1 Functions

plot_line_shaded(xData, yData, yErrsPos[, ...])	Generate a line plot with shaded region representing y-error bars.
plot_scatter_bars(xData, yData, yErrsPos[, ...])	Generate a scatter plot with y-error bars.

11.1.1 plot_line_shaded

fiducia.pltDefaults.**plot_line_shaded**(*xData*, *yData*, *yErrsPos*, *yErrsNeg=[]*, *label=''*, ***kwargs*)

Generate a line plot with shaded region representing y-error bars. Can be run multiple times before plt.show(), to plot multiple data sets on the same axes.

Parameters

- **xData** (*numpy.ndarray*) – X-axis data to be plotted.
- **yData** (*numpy.ndarray*) – Y-axis data to be plotted.
- **yErrsPos** (*numpy.ndarray*) – Errors on yData.
- **yErrsPos** – When errors on yData are asymmetric, these are the positive side errors.
- **yErrsNeg** (*numpy.ndarray*) – When errors on yData are asymmetric, these are the negative side errors.

Notes

Examples

11.1.2 plot_scatter_bars

fiducia.pltDefaults.**plot_scatter_bars**(*xData*, *yData*, *yErrsPos*, *yErrsNeg=[]*, *label=''*, ***kwargs*)

Generate a scatter plot with y-error bars. Can be run multiple times before plt.show(), to plot multiple data sets on the same axes.

Parameters

- **xData** (*numpy.ndarray*) – X-axis data to be plotted.
- **yData** (*numpy.ndarray*) – Y-axis data to be plotted.
- **yErrsPos** (*numpy.ndarray*) – Errors on yData.
- **yErrsPos** – When errors on yData are asymmetric, these are the positive side errors.
- **yErrsNeg** (*numpy.ndarray*) – When errors on yData are asymmetric, these are the negative side errors.

Notes

Examples

CHAPTER

TWELVE

RAW DANTE DATA PROCESSING (*FIDUCIA.RAWPROCESS*)

Created on Wed Mar 13 16:43:39 2019

Utilities for processing raw DANTE data. Typical steps include:

- attenuator correction
- background shot subtraction
- channel alignment (via e.g. peak finding)
- temporal axis calibration

@author: Pawel M. Kozlowski

12.1 Functions

noScope(hf)	Given a header frame, return a list of channels with no scope.
noXRD(hf)	Given a header frame, return a list of channels with no XRD.
onChannels(hf)	Given a header frame, return a list of which dante channels were on for the shot.
timesScope(hf)	Given a headerFrame, returns a timesFrame containing an array of oscilloscope times for each channel and background shot in the headerFrame.
voltageScale(hf, df)	Scales voltage (vertical) axis of dante signals based on information contained in the header.
bkgCorrect(df, timesFrame)	Give a Dante data frame containing measurement data and background shot data, remove the background from the data and return the corrected data as a dataframe.
offsetCorrect(df, timesFrame, offsetsFile)	Reads given offset correction file (.xls) and applies offsets to dante measurement data given in dataframe.
attenuationFactors(hf, channels, attenuatorsPath)	Given a header frame, return the attenuation factors applied to each channel.
attenuationCorrect(attenuatorsFile, hf, df, ...)	Given a Dante data frame and header frame, return a data frame with attenuation corrections applied to each channel.
timeAvgBkg(times, signals, timeStart, timeEnd)	Calculates time averaged background for given data.

continues on next page

Table 1 – continued from previous page

avgBkgCorrect(timesFrame, df, channels[, ...])	Applies background correction to bring the signal down to zero, based on averaging the signal background over a section of time from earliest time contained in timesFrame to earliest time plus timeLength.
polyBkg(time, signal, lowerEdge, upperEdge)	Fit polynomial function to ends of the signal as an estimate of the background signal + hyesteresis.
signalEdges(timesFrame, df, channels[, ...])	Determines locations and widths of peaks above the mean of the signal for each dante channel.
polyBkgFrame(timesFrame, df, edgesFrame, ...)	**param timesFrame** A dataframe containing time axis values corresponding to signals in
highestPeak(signal, peakIdxs)	Find the highest peak, and return list of peaks with the highest peak removed from the list.
highestN(signal, peakIdxs[, peaksNum])	Select the N tallest peaks.
getPeaks(timesFrame, df, channels[, ...])	**param timesFrame** A dataframe containing time axis values corresponding to signals in
alignPeaks(timesFrame, df, peaksFrame, channels)	**param timesFrame** A dataframe containing time axis values corresponding to signals in
constructMeasurementFrame(timesFrame, df, ...)	Takes out put timesFrame and dataFrame from rawProcess.py functions and generates a measurementFrame that can be passed to analyzeStreak() and other main.py functions.
loadCorrected(danteFile, attenuatorsFile, ...)	Given a dante data file, an attenuators file, and an offsets file, reads the file and applies background correction, attenuation correction, and channel offset correction.
hysteresisCorrect(timesFrame, df, channels)	Corrects for hysteresis by detecting edges of signal containing region and fitting a polynomial background to regions that do not belong to signal.
align(timesFrame, df, channels[, peaksNum, ...])	Aligns dante signals based on peak finding.

12.1.1 noScope

fiducia.rawProcess.**noScope**(*hf*)

Given a header frame, return a list of channels with no scope. These are the channels that are off.

Parameters **hf** (*pandas.core.frame.DataFrame*) – Header frame from DANTE measurement data. See readDanteData().

Returns Set of channels corresponding to oscilloscopes marked as "off" in the Dante data file header.

Return type set

Notes

Examples

12.1.2 noXRD

fiducia.rawProcess.**noXRD**(*hf*)

Given a header frame, return a list of channels with no XRD. If there is a scope, then these channels may still register a signal!

Parameters **hf** (*pandas.core.frame.DataFrame*) – Header frame from DANTE measurement data. See readDanteData().

Returns Set of channels corresponding to no XRDs marked in the Dante data file header.

Return type set

Notes

Examples

12.1.3 onChannels

fiducia.rawProcess.**onChannels**(*hf*)

Given a header frame, return a list of which dante channels were on for the shot.

Parameters **hf** (*pandas.core.frame.DataFrame*) – Header frame from DANTE measurement data. See readDanteData().

Returns Set of channels corresponding oscilloscopes and XRDs marked as on within the Dante data file header.

Return type set

Notes

Examples

12.1.4 timesScope

fiducia.rawProcess.**timesScope**(*hf*)

Given a headerFrame, returns a timesFrame containing an array of oscilloscope times for each channel and background shot in the headerFrame.

Parameters **hf** (*pandas.core.frame.DataFrame*) – Header frame from DANTE measurement data. See readDanteData().

Returns **timesFrame** – Returns a timesFrame containing the corresponding times for each oscilloscope trace contained in the header frame.

Return type pandas.core.frame.DataFrame

Notes

Examples

12.1.5 voltageScale

fiducia.rawProcess.**voltageScale**(*hf*, *df*)

Scales voltage (vertical) axis of dante signals based on information contained in the header. Returns a dataframe with the dante signals in units of volts. Also returns an errors/uncertainties frame in units of volts, where the uncertainty due to the 11-bit ADC converter has been calculated.

Parameters

- **hf** (*pandas.core.frame.DataFrame*) – Header dataframe from dante .dat file. See readDanteData().
- **df** (*pandas.core.frame.DataFrame*) – Dante dataframe. See readDanteData().

Returns

- **dfScaled** (*pandas.core.frame.DataFrame*) – Dante dataframe with signals in units of volts.
- **errFrame** (*pandas.core.frame.DataFrame*) – Correspoding errors for dfScaled. Also in units of volts.

Notes

Examples

12.1.6 bkgCorrect

fiducia.rawProcess.**bkgCorrect**(*df*, *timesFrame*)

Give a Dante data frame containing measurement data and background shot data, remove the background from the data and return the corrected data as a dataframe. Note that the returned dataframe is different in a few ways from the input dataframe. First, the returned dataframe is assumed to have strings as column headers, whereas the returned dataframe will have integers (corresponding to dante channel number) as the column headers. In addition, the input dataframe will start indexing at some number above 0 (usually 18, due to the header length), whereas the returned dataframe is re-indexed to begin at 0.

A dataframe with corresponding time scales to df is also passed to this function for reindexing from strings to integers. This also acts as a placeholder in case it is necessary to interpolate values if the background shot and measurement shot timescales are not the same. Though this type of interpolation is not currently implemented.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – Dataframe of raw dante data. This should contain both the shot measurement and the shot background as columns. See readDanteData(). The columns in this dataframe are assumed to be strings.
- **timesFrame** (*pandas.core.frame.DataFrame*) – Dataframe containing time axis corresponding to dante signals in df dataframe. See timesScope().

Returns

- **timesBkg** (*pandas.core.frame.DataFrame*) – Returns a dataframe of times corresponding to dfCorrected signals. The columns in this dataframe are integers corresponding to Dante channel number.

- **dfCorrected** (*pandas.core.frame.DataFrame*) – Returns a dataframe of background subtracted dante signals. The columns in this dataframe are integers corresponding to Dante channel number.

Notes

Examples

12.1.7 offsetCorrect

fiducia.rawProcess.**offsetCorrect**(*df*, *timesFrame*, *offsetsFile*)

Reads given offset correction file (.xls) and applies offsets to dante measurement data given in dataframe. The input dataframe should already be background corrected and scaled to units of volts, see bkgCorrect() and voltageScale(). Note that although timing offsets are also applied, they are not as relevant since timing should be realigned to a fiducial peak anyway. Additional attenuation is not implemented and an error will be raised if the offsets file contains attenuation values other than 1. Returns a dataframe with applied offsets.

Parameters

- **df** (*pandas.core.frame.DataFrame*) – Dante dataframe with background corrected values and scaled to units of volts. See readDanteData(), bkgCorrect() and voltageScale().
- **timesFrame** (*pandas.core.frame.DataFrame*) – Dataframe containing time axis corresponding to dante signals in df dataframe. See timesScope() and bkgCorrect().

Returns

- **offsetsFile** (*str*) – Full path to .xls file containing dante channel offsets.
- **dfOffset** (*pandas.core.frame.DataFrame*) – Dante dataframe with applied offset corrections

Notes

Examples

12.1.8 attenuationFactors

fiducia.rawProcess.**attenuationFactors**(*hf*, *channels*, *attenuatorsPath*)

Given a header frame, return the attenuation factors applied to each channel.

Parameters

- **hf** (*pandas.core.frame.DataFrame*) – Pandas dataframe containing dante header information. See readDanteData().
- **channels** (*set*) – Set of channels to be analyzed.
- **attenuatorsPath** (*str*) – Full path to excel file containing attenuator serial numbers and corresponding attenuation factors.

Returns

Return type chFactors

Notes

Examples

12.1.9 attenuationCorrect

fiducia.rawProcess.**attenuationCorrect**(*attenuatorsFile*, *hf*, *df*, *channels*)

Given a Dante data frame and header frame, return a data frame with attenuation corrections applied to each channel.

Parameters

- **attenuatorsFile** (*str*) – Full path to .xls file containing attenuator serial numbers and corresponding attenuation factors. See attenuationFactors().
- **hf** (*pandas.core.frame.DataFrame*) – Header dataframe from dante .dat file. See readDanteData().
- **df** (*pandas.core.frame.DataFrame*) – Dante dataframe. This frame should already be voltage scaled, background corrected, and offset corrected. See readDanteData().
- **channels** (*list*) – List of dante channels in df to be analyzed.

Returns **dfAtten** – Returns dataframe with attenuation corrected signal values for the given channels.

Return type pandas.core.frame.DataFrame

Notes

Examples

12.1.10 timeAvgBkg

fiducia.rawProcess.**timeAvgBkg**(*times*, *signals*, *timeStart*, *timeEnd*)

Calculates time averaged background for given data.

Parameters **times** –

signals:

timeStart:

timeEnd:

Returns

Return type avg

Notes

Examples

12.1.11 avgBkgCorrect

fiducia.rawProcess.**avgBkgCorrect**(*timesFrame*, *df*, *channels*, *timeLength=1e-09*)

Applies background correction to bring the signal down to zero, based on averaging the signal background over a section of time from earliest time contained in timesFrame to earliest time plus timeLength.

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – Dataframe of time axis values corresponding to signals in df. These should be in units of seconds.
- **df** (*pandas.core.frame.DataFrame*) – Dataframe of dante signals. These should already be attenuation corrected and in units of volts.
- **channels** (*set*) – Set of channels to be analyzed.
- **timeLength** (*float*) – Duration of time from initial time over which to take the average. In units of seconds.
- **dfAvg** (*pandas.core.frame.DataFrame*) – Returns a dataframe containing average background corrected signals.

Returns

Return type dfAvg

Notes

Examples

12.1.12 polyBkg

fiducia.rawProcess.**polyBkg**(*time*, *signal*, *lowerEdge*, *upperEdge*, *order=3*, *lowerLength=None*, *upperLength=None*, *plot=False*)

Fit polynomial function to ends of the signal as an estimate of the background signal + hyesteresis. Default is cubic fit.

Parameters

- **time** (*numpy.ndarray*) – array of times corresponding to signal
- **signal** (*numpy.ndarray*) – array of signal values for a single dante channel
- **lowerEdge** (*int*) – Index of time array corresponding to lower edge of detected signal. See signalEdges().
- **upperEdge** (*int*) – Index of time array corresponding to upper edge of detected signal. See signalEdges().
- **order** (*int*) – Order of polynomial to be fitted to estimated background/hysteresis.
- **lowerLength** (*int*) – Length over which to take the polynomial background fit on the lower end (earlier in time) segment of the signal, with respect to lowerEdge. Default is None, which just takes the first point in the signal.

- **upperLength** (*int*) – Length over which to take the polynomial background fit on the upper end (later in time) segment of the signal, with respect to upperEdge. Defualt is None, which then just picks the second to last point in the signal.
- **plot** (*bool*) – Flag for plotting polynomial fitted background signal. Default is False.

Returns

Return type time

fitSignal:

Notes

Examples

12.1.13 signalEdges

fiducia.rawProcess.**signalEdges**(*timesFrame*, *df*, *channels*, *sigmaMult=3*, *plot=False*, *prominence=0.1*, *width=10*, *avgMult=1*)

Determines locations and widths of peaks above the mean of the signal for each dante channel. Edges of the signal containing region are then obtained by moving sigmaMult peak widths away from the earliest and latest peaks. Returns these lower and upper bound edges of the signal containing region as a dataframe. These edges are useful for fitting and removing the background/hysteresis.

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – A dataframe containing time axis values corresponding to signals in df.
- **df** (*pandas.core.frame.DataFrame*) – A dataframe of corrected/calibrated dante signal measurements.
- **channels** (*list*) – List of channels in df for which edges will be determined.
- **sigmaMult** (*float*) – Multiplier factor by which the lower and upper bounds of the signal containing region are determined. The lower bound is determined by sigmaMult times the width of the earliest peak away from the earliest peak. The upper bound is determined by sigmaMult times the width of the latest peak away from the latest peak. Default is 3 for approximately 3*sigma away from each peak.
- **plot** (*bool*) – Flag for plotting peak locations and widths. Default is False.
- **edgesFrame** (*pandas.core.frame.DataFrame*) – Lower and upper bound edges of the signal containing region for each dante channel. The lower bound is in 0 index and the upper bound is in 1 index. The bounds are given in index coordinates and they have been rounded to the nearest point.
- **prominence** (*float*) – Prominence threshold for identifying peaks in scipy's find_peaks().
- **width** (*int*) – Width in index units for identifying peaks in scipy's find_peaks().
- **avgMult** (*float*) – Multiplicative factor for setting minimum intensity threshold for indentifying peaks in scipy's find_peaks(). This is a multiple of the signal average.

Returns

Return type edgesFrame

Notes

Examples

12.1.14 polyBkgFrame

fiducia.rawProcess.**polyBkgFrame**(*timesFrame*, *df*, *edgesFrame*, *channels*, *order=3*, *plot=False*)

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – A dataframe containing time axis values corresponding to signals in df.
- **df** (*pandas.core.frame.DataFrame*) – A dataframe of corrected/calibrated dante signal measurements.
- **edgesFrame** (*pandas.core.frame.DataFrame*) – Dataframe describing edges of the signal containing region, outside of which should be just background. This function will fit to these two early time and late time background containing regions. See signalEdges().
- **channels** (*list*) – A list of channels for which to apply analyis.
- **order** (*int*) – Order of polynomial to be fitted to estimated background/hysteresis.
- **plot** (*bool*) – Flag for plotting fitted background and background subtracted signal. Default is False.

Returns

Return type dfPoly

Notes

Examples

12.1.15 highestPeak

fiducia.rawProcess.**highestPeak**(*signal*, *peakIdxs*)

Find the highest peak, and return list of peaks with the highest peak removed from the list.

Parameters

- **signal** (*pandas.core.series.Series*) – A data series consisting of signals from a single dante channel.
- **peakIdxs** (*list*) – A list of indices corresponding to peaks identified in the signal by using scipy's find_peaks() function.

Returns

- **peakHighestIdx** (*int*) – Returns the index corresponding to the highest peak.
- **peakIdxs2** (*list*) – Returns a list of peak index locations with the highest peak removed from the list. This makes it easier for highestN() to apply highestPeak() iteratively to find the N highest peaks.

Notes

Examples

12.1.16 highestN

fiducia.rawProcess.**highestN**(*signal*, *peakIdxs*, *peaksNum=2*)

Select the N tallest peaks.

Parameters

- **signal** (*pandas.core.series.Series*) – A data series consisting of signals from a single dante channel.
- **peakIdxs** (*list*) – A list of indices corresponding to peaks identified in the signal by using scipy's find_peaks() function.
- **peaksNum** (*int*) – Number of peaks to grab from peakIdxs. This function will grab just the N tallest peaks where N=peaksNum.

Returns **highestPeaks** – Array of indices corresponding to the highest peaks. Peaks are ordered from highest to lowest.

Return type numpy.ndarray

Notes

Examples

12.1.17 getPeaks

fiducia.rawProcess.**getPeaks**(*timesFrame*, *df*, *channels*, *peaksNum=2*, *plot=False*, *prominence=0.1*, *width=10*, *avgMult=1*)

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – A dataframe containing time axis values corresponding to signals in df.
- **df** (*pandas.core.frame.DataFrame*) – A dataframe of corrected/calibrated dante signal measurements.
- **channels** (*list*) – A list of channels for which to apply analyis.
- **peaksNum** (*int*) – Number of peaks to grab from peakIdxs. This function will grab just the N tallest peaks where N=peaksNum.
- **peaksNum** – Number of peaks to grab from peakIdxs. This function will grab just the N tallest peaks where N=peaksNum.
- **plot** (*bool*) – Flag for plotting identified peaks, with prominences, and widths, overlaid with the corresponding dante signal, and the average dante signal.
- **peaksFrame** (*pandas.core.frame.DataFrame*) – Returns a dataframe containing indices of the identified peaks sorted from the peak that occurs earliest in time to the latest in time.
- **prominence** (*float*) – Prominence threshold for identifying peaks in scipy's find_peaks().
- **width** (*int*) – Width in index units for identifying peaks in scipy's find_peaks().

- **avgMult** (*float*) – Multiplicative factor for setting minimum intensity threshold for indentifying peaks in scipy's find_peaks(). This is a multiple of the signal average.

Returns

Return type peaksFrame

Notes

Examples

12.1.18 alignPeaks

fiducia.rawProcess.**alignPeaks**(*timesFrame*, *df*, *peaksFrame*, *channels*, *peakAlignIdx=0*, *referenceTime=1e-09*, *plot=False*)

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – A dataframe containing time axis values corresponding to signals in df.
- **df** (*pandas.core.frame.DataFrame*) – A dataframe of corrected/calibrated dante signal measurements.
- **peaksFrame** (*pandas.core.frame.DataFrame*) – Dataframe containing positions of N highest peaks and sorted from earliest in time to latest in time. See getPeaks().
- **peakAlignIdx** (*int*) – Picks which peak to align to. 0 is first peak, 1 is second peak in peaksFrame, etc.
- **referenceTime** (*float*) – Time in s to which align peaks. Default is 1e-9 s or 1 ns.
- **plot** (*bool*) – Flag for plotting aligned dante signals. Default is False.

Returns **timesAligned** – Returns a dataframe identical in shape to timesFrame, but with the times for each dante channel offset such that the selected peaks are temporally aligned.

Return type pandas.core.frame.DataFrame

Notes

Examples

12.1.19 constructMeasurementFrame

fiducia.rawProcess.**constructMeasurementFrame**(*timesFrame*, *df*, *channels*)

Takes out put timesFrame and dataFrame from rawProcess.py functions and generates a measurementFrame that can be passed to analyzeStreak() and other main.py functions.

Converts units from seconds to nanoseconds.

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – A dataframe containing time axis values corresponding to signals in df.
- **df** (*pandas.core.frame.DataFrame*) – A dataframe of corrected/calibrated dante signal measurements.
- **channels** (*list*) – A list of channels for which to apply analyis.

Returns measurementFrame – Returns a measurementFrame which can be passed to main.py functions such as analyzeSpectrum() and analyzeStreak().

Return type pandas.core.frame.DataFrame

Notes

Examples

12.1.20 loadCorrected

fiducia.rawProcess.**loadCorrected**(*danteFile*, *attenuatorsFile*, *offsetsFile*, *cut=None*, *plot=False*, *addCh=[]*)

Given a dante data file, an attenuators file, and an offsets file, reads the file and applies background correction, attenuation correction, and channel offset correction. Returns the corrected data traces as a pandas dataframe. The row indices of this dataframe also contain the correct time scaling given the oscilloscope settings, but note that the channels are not aligned. User must apply alignment correction using some measured signal as a temporal fiducial.

Parameters

- **danteFile** (*str*) – Full path to .dat file containing raw dante traces.
- **attenuatorsFile** (*str*) – Full path to the .xls file containing attenuator serial numbers and corresponding attenuation factors.
- **offsetsFile** (*str*) – Full path to .xls file containing dante channel offsets.
- **cut** (*int*) – Number of points to cut from leading and trailing end of each Dante channel trace. This is used to remove noise that occurs at the edges of the signal. Default is None, which means no cut is applied.
- **plot** (*bool*) – Flag for plotting data after each calibration/correction step. Default is False.
- **addCh** (*list*) – Add channels to analyze. This is used to override which channels are listed as on in the header of the data dante data file.

Returns

Return type timeOffset

dfAvg:

onChList:

hf:

dfVolt:

Notes

Examples

12.1.21 hysteresisCorrect

fiducia.rawProcess.**hysteresisCorrect**(*timesFrame*, *df*, *channels*, *order=5*, *prominence=0.2*, *width=10*, *avgMult=1*)

Corrects for hysteresis by detecting edges of signal containing region and fitting a polynomial background to regions that do not belong to signal. This background is then subtracted.

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – Time corresponding to df
- **df** (*pandas.core.frame.DataFrame*) – Dataframe of dante signals. See loadCorrected().
- **channels** (*list*) – A list of channels for which to apply analyis.
- **order** (*int*) – Polynomial order to be fitted to hysteresis/background.
- **prominence** (*float*) – Prominence threshold for identifying peaks in scipy's find_peaks().
- **width** (*int*) – Width in index units for identifying peaks in scipy's find_peaks().
- **avgMult** (*float*) – Multiplicative factor for setting minimum intensity threshold for indentifying peaks in scipy's find_peaks(). This is a multiple of the signal average.

Returns **dfPoly** – Returns a dataframe of hysteresis corrected dante signals.

Return type pandas.core.frame.DataFrame

Notes

Examples

12.1.22 align

fiducia.rawProcess.**align**(*timesFrame*, *df*, *channels*, *peaksNum=1*, *peakAlignIdx=0*, *referenceTime=1e-09*, *prominence=0.01*, *width=10*, *avgMult=1.5*)

Aligns dante signals based on peak finding.

Parameters

- **timesFrame** (*pandas.core.frame.DataFrame*) – Time corresponding to df
- **df** (*pandas.core.frame.DataFrame*) – Dataframe of dante signals. See loadCorrected().
- **channels** (*list*) – A list of channels for which to apply analyis.
- **peaksNum** (*int*) – Number of peaks to grab from peakIdxs. This function will grab just the N tallest peaks where N=peaksNum.
- **peakAlignIdx** (*int*) – Picks which peak to align to. 0 is first peak, 1 is second peak in peaksFrame, etc.
- **referenceTime** (*float*) – Time in s to which align peaks. Default is 1e-9 s or 1 ns.
- **prominence** (*float*) – Prominence threshold for identifying peaks in scipy's find_peaks().
- **width** (*int*) – Width in index units for identifying peaks in scipy's find_peaks().
- **avgMult** (*float*) – Multiplicative factor for setting minimum intensity threshold for indentifying peaks in scipy's find_peaks(). This is a multiple of the signal average.

Returns **timesAligned** – Returns a dataframe of times corresponding to signals in df, such that the signals are now aligned to the given peak.

Return type pandas.core.frame.DataFrame

Notes

Examples

CHAPTER

THIRTEEN

DANTE RESPONSE FUNCTIONS (*FIDUCIA.RESPONSE*)

Created on Fri Mar 8 09:23:02 2019

Utilities for working with DANTE response functions (e.g. plotting, locating edges).

@author: Pawel M. Kozlowski

13.1 Functions

knotFind(channels, responseFrame[, ...])	Find knot points.

13.1.1 knotFind

fiducia.response.**knotFind**(*channels*, *responseFrame*, *forceKnot=array([], dtype=float64)*, *knotBoundary=0*, *boundary='y0'*)

Find knot points.

Find knot points for cubic splines based on positions of K-edges of each DANTE channel filter.

Parameters

- **channels** (*list, numpy.ndarray*) – List or array of relevant channels
- **responseFrame** (*pandas.core.frame.DataFrame*) – Pandas dataFrame containing response functions for each DANTE channel. See loadResponses().
- **forceKnot** (*numpy.ndarray*) – Numpy array where first column is channelNumber and second column is the corresponding photonEnergy we wish to force. Use this for channels that do not have a distinct K-edge.
- **knotBoundary** (*float*) – Photon energy value for either y_0 or y_{n+1} boundary condition. This value gets appended to the array of photon energies otherwise found by knotFind().

Returns **knotsAppend** – An array of knot points, with each element corresponding to a channel or boundary condition.

Return type numpy.ndarray

Notes

Examples

CHAPTER

FOURTEEN

UNCERTAINTY PROPAGATION FOR COMMON OPERATIONS (*FIDUCIA.STATS*)

Created on Tue Jul 21 12:11:12 2020

Common statistic operations

@author: Myles T. Brophy

14.1 Functions

simpsVariance(yUnc[, x, dx])	Propagates variance (σ^2) through Simpson's rule numerical integration.
trapzVariance(yUnc[, x, dx])	Error propogation for Trapezoidal rule integration using uniform or non-uniform grids.
gradientVariance(yUnc[, x, dx])	Propogates uncertainty for the gradient operator of an array of a given step size.
dotVariance(a, b[, aUncertainty, bUncertainty])	Propogate uncertainty for the dot product of matrix a and 1D vector b.
interpVariance(x, xp, fpUnc[, leftVar, ...])	Propagate uncertainty for linear interpolation.

14.1.1 simpsVariance

fiducia.stats.**simpsVariance**(*yUnc*, *x=None*, *dx=1.0*)

Propagates variance (σ^2) through Simpson's rule numerical integration.

NOTE: THIS FUNCTION IS INCOMPLETE AND HAS NOT BEEN VERIFIED 2020-08-21 PMK.

Parameters

- **yUnc** (*list, numpy.ndarray*) – Uncertainties in the vertical axis.
- **x** (*list, numpy.ndarray, optional*) – Horizontal coordinates corresponding to yUnc. Default is None, which generates a uniformly spaced linear array of horizontal coordinates based on the length of yUnc and the value of dx.
- **dx** (*float, optional*) – Uniform spacing between horizontal coordinates corresponding to yUnc. Default is 1.0.

Returns **variance** – Variance (σ^2) on value of integral from Simpson's rule numerical integration.

Return type float

Notes

Based on a modified verison of: https://en.wikipedia.org/wiki/Simpson's_rule#Composite_Simpson's_rule_for_irregularly_spa

Examples

14.1.2 trapzVariance

`fiducia.stats.`**`trapzVariance`**(*yUnc*, *x=None*, *dx=1.0*)

Error propogation for Trapezoidal rule integration using uniform or non-uniform grids.

Parameters

- **yUnc** (*list, numpy.ndarray*) – The list of uncertainities, referenced as σ_i.
- **x** (*list, numpy.ndarray, optional*) – The sampling points for which the uncertainites ''y" were found. Must be the same length as ''y". If none are provided, then the step size will be uniform and set with ''dx". The default is None.
- **dx** (*int, float, optional*) – Step size. Only applies if sampling points aren't specified with ''x". The default is 1.0.

Returns **variance** – The total variance (σ^2) found by propagating ''y".

Return type float

Notes

Trap rule integration with non uniform spacing takes the form

$$\sum_{k=1}^{N} \frac{\Delta x_i}{2} \left(f(x)_{i-1} + f(y)_i\right)$$

Propogating the uncertainties through this integration results in

$$\sigma^2 = \frac{1}{4}\left(\sum_{k=1}^{N} \Delta x_i \sigma_{i-1}^2 + \sigma_i^2 + 2\sum_{k=1}^{N-1} \Delta x_i \Delta x_i + 1\sigma_i^2\right)$$

The equation is generalized and applies to uniform and non-uniform step sizes.

Examples

14.1.3 gradientVariance

`fiducia.stats.`**`gradientVariance`**(*yUnc*, *x=None*, *dx=1.0*)

Propogates uncertainty for the gradient operator of an array of a given step size.

Parameters

- **yUnc** (*list, numpy.ndarray*) – The list of uncertainities, referenced as :math:'sigma_i'.
- **x** (*list, numpy.ndarray, optional*) – The sampling points for which the uncertainites ''y" were found. Must be the same length as ''y". If none are provided, then the step size will be uniform and set with ''dx". The default is None.

- **dx** (*list, numpy.ndarray, optional*) – Step size. Only applies if sampling points aren't specified with ''x". The default is 1.0.

Returns **variance** – The total variance (σ^2) found by propagating ''y".

Return type float

Notes

$$\mathrm{Var}(\nabla y_i) = \frac{h_{i-1}^2\sigma_{i+1}^2 + (h_i^2 + h_{i-1}^2)^2\sigma_i^2 - h_i^4\sigma_{i-1}^2}{(h_i h_{i-1}(h_i + h_{i-1}))^2}$$

At the boundaries

$$Var(\nabla y_0) = \frac{\sigma_1^2 - \sigma_0^2}{h_0^2}, Var(\nabla y_{N-1}) = \frac{\sigma_{N-1}^2 - \sigma_{N-2}^2}{h_{N-2}^2}$$

Examples

14.1.4 dotVariance

`fiducia.stats.`**`dotVariance`**(*a*, *b*, *aUncertainty=None*, *bUncertainty=None*)

Propogate uncertainty for the dot product of matrix a and 1D vector b.

Propogate uncertainty for the dot product of a matrix and a 1D vector. Assumes no covariance between *a* and *b*. Methodology is similar to `numpy.dot()` where:

- If both *a* and *b* are 1D, the uncertainty of the inner product of vectors is returned.
- If 'a' is N dimensional (Where :math: *N>=2*) and *b* is 1D, the uncertainty of the sum product of the last axis of a with b is returned.

0-D (scalar) arrays are not supported. *b* arrays that have more than one axis are not supported. *a* and *b* must have the same shape as *aUncertainty* and *bUncertainty*, respectively.

Parameters

- **a** (*numpy.ndarray, list*) – Matrix or vector to dot with 'b'.
- **b** (*numpy.ndarray, list*) – Vector that 'a' will be dotted with. Must be the same size as the last axis of a.
- **aUncertainty** (*numpy.ndarray, optional*) – Uncertainty of each element in 'a'. The default is None.
- **bUncertainty** (*numpy.ndarray, optional*) – Uncertainty of each element in 'b'. The default is None.

Returns **variance**

Return type float

Notes

$$\mathrm{Var}(A \cdot B) = \sum_{i=1}^{N} \mathrm{Var}(a_i b_i)$$

Assuming covariance between independent variables

$$\sum_{i=1}^{N} (a_i \sigma_{b_i})^2 + (b_i \sigma_{a_i})^2$$

Examples

14.1.5 interpVariance

`fiducia.stats.`**`interpVariance`**(*x*, *xp*, *fpUnc*, *leftVar=None*, *rightVar=None*, *period=None*)

Propagate uncertainty for linear interpolation.

Parameters

- **x** (*numpy.ndarray, list*) – The x-coordinates at which to evaluate the interpolated values.
- **xp** (*numpy.ndarray, list*) – The 1D x-coordinates of the data points, must be increasing order
- **fpUnc** (*numpy.ndarray, list*) – The uncertainty in the y-coordinates of the data points, same length as *xp*.
- **leftVar** (*float, optional*) – Variance to return for *x* < *xp[0]*. If not given, the first *yUnc* element will be used. Default is *None*
- **rightVar** (*float, optional*) – Variance to return for *x* > *xp[-1]*. If not given, the last *yUnc* element will be used. Default is *None*

Returns **yVar** – 1D array containing the variance for each interpolated *x*.

Return type numpy.ndarray

Notes

Variance of interpolated point, assuming no uncertainty in *x* and *xp*, and no covariance between y-coordinates, is given by

$$\mathrm{Var}(y) = \frac{1}{(x_1 - x_0)^2}((x_1 - x)^2 \sigma_{y_0}^2 + (x - x_0)^2 \sigma_{y_1}^2)$$

Examples

Derivations for propagating uncertainty for some common operations.

14.1.6 Weighted Summation

Gien a vector X_i with *N* elements

$$\mathrm{Var}\left(\sum_{k=1}^{N} a_i X_i\right) = \sum_{k=1}^{N} a_i^2 \,\mathrm{Var}(X_i) + 2 \sum_{1 \le i} \sum_{< j < n} a_i a_j \,\mathrm{Cov}(X_i, X_j)$$

A simplified version of this can be written as

$$\mathrm{Var}(ax+by) = \sigma^2_{\text{summing independent variables}} = a^2\sigma_x^2 + b^2\sigma_y^2 + 2ab\sigma_{xy}^2$$

14.1.7 Trap Rule Variance

For N steps of $\Delta x_k = x_{k+1} - x_k$ where $f(x_k) = y_k$, where each y_k is independent, an integral can be approximated as

$$\int_a^b f(x)dx \approx \sum_{k=1}^{N} \frac{\Delta x_{k-1}}{2}(y_{k-1} + y_k)$$

To find the variance in the general case, use the last 3 equations.

$$\mathrm{Var}\left(\sum_{k=1}^{N} \frac{\Delta x_{k-1}}{2}(y_{k-1}+y_k)\right) = \mathrm{Var}\left(\sum_{k=1}^{N} \frac{\Delta x_{k-1}}{2} y_{k-1} + \sum_{k=1}^{N} \frac{\Delta x_{k-1}}{2} y_k\right)$$

$$= \mathrm{Var}\left(\frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_{k-1}\right) + \mathrm{Var}\left(\frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_k\right) + 2\,\mathrm{Cov}\left(\frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_{k-1}, \frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_k\right)$$

The two first Var terms are simple

$$\frac{1}{4}\left(\sum_{k=1}^{N} \mathrm{Var}(\Delta x_{k-1} y_{k-1})\right) + \frac{1}{4}\left(\sum_{k=1}^{N} \mathrm{Var}(\Delta x_{k-1} y_k)\right)$$

$$= \frac{1}{4}\left(\sum_{k=1}^{N} \Delta x_{k-1}^2\,\mathrm{Var}(y_{k-1}) + \sum_{k=1}^{N} \Delta x_{k-1}^2\,\mathrm{Var}(y_k)\right)$$

$$= \frac{1}{4}\left(\sum_{k=1}^{N} \Delta x_{k-1}^2 \sigma_{k-1}^2 + \sum_{k=1}^{N} \Delta x_{k-1}^2 \sigma_k^2\right) = \frac{1}{4}\sum_{k=1}^{N} \Delta x_{k-1}^2(\sigma_{k-1}^2 + \sigma_k^2)$$

where σ_k is the uncertainty of y_k. Now for the Covariance of the two summations

$$\mathrm{Cov}\left(\frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_{k-1}, \frac{1}{2}\sum_{k=1}^{N} \Delta x_{k-1} y_k\right) = \frac{1}{4}\sum_{k=1}^{N-1} \Delta x_{k-1}\Delta x_k \sigma_k^2$$

Combine all this and the fourth equation to get

$$\frac{1}{4}\left(\sum_{k=1}^{N} \Delta x_{k-1}^2(\sigma_{k-1}^2 + \sigma_k^2) + 2\sum_{k=1}^{N-1} \Delta x_{k-1}\Delta x_k \sigma_k^2\right)$$

In the simple case of a uniform grid, where all $\Delta x_k = \Delta x$ this can be expanded to

$$\frac{\Delta x^2}{4}\left(\sigma_0^2 + 4\sigma_1^2 + 4\sigma_2^2 + ... + 4\sigma_{N-1}^2 + \sigma_N^2\right)$$

14.1.8 Gradient Variance

From the numpy.grad documentation, the gradient for discrete step sizes is approximated by

$$\nabla f(x_i) = \frac{h_{i-1}^2 f(x_i + h_i) + (h_i^2 - h_{i-1}^2) f(x_i) - h_i^2 f(x_i - h_{i-1})}{h_i h_{i-1}(h_i + h_{i-1})}$$

with the gradient at the first and the last data point being

$$\nabla f(x_1) = \frac{y_2 - y_1}{h_1}, \nabla f(x_N) = \frac{y_N - y_{N-1}}{h_{N-1}}$$

for a list of x_i data points and $h_d = x_{i+1} - x_i = h_i$ and $h_s = x_i - x_{i-1} = h_{i-1}$. From this we can say that $f(x_i) = y_i$ and $f(x_i + h_i) = y_{i+1}$ and $f(x_i - h_s) = y_{i-1}$. Using the variance of weighted sums where the weights are the h terms we get

$$\mathrm{Var}(\nabla y_i) = \sigma^2_{\nabla y_i} = \frac{h^4_{i-1}\sigma^2_{i+1} + (h^2_i - h^2_{i-1})^2\sigma^2_i - h^4_i\sigma^2_{i-1}}{(h_i h_{i-1}(h_i + h_{i-1}))^2}$$

where σ_i corresponds to the uncertainty in y_i. Keep in mind that there are N y values and $N-1$ h values because h_i is the difference between the N data points. Note that unlike in the Trap rule integration, the covariant term is 0 because no repeated uncertainty terms appear in the sum. At the borders $i = 0, N$ the variance is

$$\mathrm{Var}(\nabla y_1) = \frac{\sigma^2_2 - \sigma^2_1}{h^2_1}, \mathrm{Var}(\nabla y_N) = \frac{\sigma^2_N - \sigma^2_{N-1}}{h^2_{N-1}}$$

14.1.9 Dot Product Variance

The dot product of vectors X, Y is defined as

$$X \cdot Y = \sum_{i=1}^{N} x_i y_i$$

Uncertainty propagation when multiplying two variables, $f(u, v) = auv$, with a as a constant, is given by

$$\mathrm{f(u, v)} = (au\sigma_v)^2 + (av\sigma_u)^2 + 2a^2uv\sigma^2_{uv}$$

For the dot product we assume there is no covariance between X and Y because they are independent. This gets us operatorname{Var}(X cdot Y) = sum_{i=1}^N (x_i sigma_{y_i})^2 + (y_i sigma_{x_i})^2

14.1.10 Linear interpolation

Linear interpolation of a point $x_0 < x < x_1$ is given by

$$y = \frac{y_0(x_1 - x) + y_1(x - x_0)}{x_1 - x_0}$$

First thing we can do, to make this calculation easier, is assume that there is no uncertainty in the x_i terms. This is a short cut, but it`s all that's required for the application in which this linear interpolation is being implemented. Starting with the numerator, we can use the weighted summation rules to say

$$\mathrm{Var}(y_0(x_1 - x) + y_1(x - x_0)) = (x_1 - x)^2\sigma^2_{y_0} + (x - x_0)^2\sigma^2_{y_1}$$

where there is no covariant term, because we assume y_0 and y_1 and independent. Then, including the denominator as a constant (because we assume all x values have no uncertainty, we get a variance of

$$\mathrm{Var}(y) = \frac{1}{(x_1 - x_0)^2}((x_1 - x)^2\sigma^2_{y_0} + (x - x_0)^2\sigma^2_{y_1})$$

CHAPTER

FIFTEEN

VISUALIZATION UTILITIES (*FIDUCIA.VISUALIZATION*)

Created on Fri Mar 8 10:49:17 2019

Utilities for visualizing DANTE data.

@author: Pawel M. Kozlowski

15.1 Functions

plotResponse(channels, responseFrame, knots)	Plots response function curves with knot locations identified as vertical dashed lines.
plotTraces(channels, measurementFrame[, scale])	Given a dataframe of Dante channel data, plot all the signal traces onto a single plot.
plotStreak(times, energies, spectra)	Plot streak of unfolded Dante spectra.
signalImg(signalsArr)	Visualize dante signals as an image.

15.1.1 plotResponse

fiducia.visualization.**plotResponse**(*channels*, *responseFrame*, *knots*, *solid=True*, *title='Dante Response Functions'*)

Plots response function curves with knot locations identified as vertical dashed lines.

channels: list, numpy.ndarray List or array of relevant channels

responseFrame: pandas.core.frame.DataFrame Pandas dataFrame containing response functions for each DANTE channel. See loadResponses().

knots: list, numpy.ndarray List or array of knot point photon energy value. See knotFind().

solid: Bool Includes solid angle in response function value if true. Necessary for plotting responses with correct units.

Notes

Examples

15.1.2 plotTraces

fiducia.visualization.**plotTraces**(*channels*, *measurementFrame*, *scale='regular'*)

Given a dataframe of Dante channel data, plot all the signal traces onto a single plot.

channels: list, numpy.ndarray List or array of relevant channels

measurementFrame: pandas.core.frame.DataFrame Pandas dataframe containing DANTE measurement data. See readDanteData() and readDanProcessed().

Notes

Examples

15.1.3 plotStreak

fiducia.visualization.**plotStreak**(*times*, *energies*, *spectra*)

Plot streak of unfolded Dante spectra. See analyzeStreak().

Parameters

- **times** (*numpy.ndarray*) – Array of times for which the unfold was analyzed.
- **energies** (*numpy.ndarray*) – Array of photon energies corresponding to the unfolded spectra.
- **spectra** (*numpy.ndarray*) – The unfolded spectral intensities as a 2D array. See analyzeStreak().

Notes

Examples

15.1.4 signalImg

fiducia.visualization.**signalImg**(*signalsArr*)

Visualize dante signals as an image.

Parameters **signalsArr** (*numpy.ndarray*) –

Notes

Examples

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX