tespackage

Release 0.2.0

Leonardo Assis

Sep 24, 2021

CONTENTS:

Check out the *Usage* section for further information, including how to *install* the project.

CHAPTER

ONE

CONTENTS

1.1 Usage

After downloading the files in this package to your computer, use the Jupyter notebooks on the folder 'Jupyter Notebooks' to use the package for: - Configure the multi-channel analyser (MCA); - Configure the measurements made by the hardware processor on the TES pulses; - Calibrate the TES detectors; - Transform area measurements in photon number information.

To use the package in your own programs/notebooks, import it using

```
import tes
```

1.1.1 Installation

We recommend that you create a virtual enviroment using conda to work and develop this package.

After downloading the contents of this repository to your machine, you can install the package using the terminal.

First open the folder where you installed the contents of this repository.

Then, install the package using:

```
python setup.py install
```

1.1.2 Requirements

The tes package requires:

- numpy
- scipy
- matplotlib
- numba
- yaml
- pyserial

1.2 tes

1.2.1 tes package

Submodules

tes.analysis module

Module written by Dr. Geoff Gillet for calibration and modelling.

This module was kept as it was in previous version for backward compatibility.

class tes.analysis.**Counts**(*count*, *uncorrelated*, *vacuum*, *has_noise_threshold*)

Bases: tuple

property count

Alias for field number 0

property has_noise_threshold

Alias for field number 3

property uncorrelated

Alias for field number 1

property vacuum

Alias for field number 2

class tes.analysis.**Guess**(*hist*, *f*, *bin_c*, *max_i*, *thresholds*)

Bases: tuple

property bin_c

Alias for field number 2

property f

Alias for field number 1

property hist

Alias for field number 0

property max_i

Alias for field number 3

property thresholds

Alias for field number 4

class tes.analysis.**MixtureModel**(*param_list*, *thresholds*, *zero_loc*, *log_likelihood*, *converged*, *dist*, *has_noise_threshold*)

Bases: *tes.analysis.MixtureModel*

Subclass of namedtuple used to represent a mixture model.

fields:

param_list list of parameters for each distribution in the mixture.

thresholds the intersection of neighbouring distributions in the mixture.

zero_loc Fixed location parameter used to fit the first distribution, or None if the location parameter was fitted.

log_likelihood the log of the likelihood that the model could produce the data used to construct it.

converged boolean indicating that the expectation maximisation algorithm terminated normally when fitting the model.

dist the type of distribution forming the components of the mixture, (see scipy.stats).

`cdf`(*x*, *d=None*, *scale=False*)
evaluate the pdf(s) of the distribution(s) selected by d at x.

Parameters

- **x** – value(s) where to the function is evaluated.
- **d** – selects element(s) of param_list that parameterise the distributions. If None all distributions in param_list are used.
- **scale** (*bool*) – when true return the scaled pdf, the scale is always param[-1].

Returns ndarray shape (len(d), len(x)) containing the pdf(s) or a single value if only 1 point in 1 distribution is selected.

`static load`(*filename*)
load model from .npz file.

Parameters **filename** – filename excluding extension"

Returns instance of MixtureModel.

`pdf`(*x*, *d=None*, *scale=False*)
evaluate the pdf(s) of the distribution(s) selected by d at x.

Parameters

- **x** – value(s) where to the function is evaluated.
- **d** – selects element(s) of param_list that parameterise the distributions. If None all distributions in param_list are used.
- **scale** (*bool*) – when true return the scaled pdf, the scale is always param[-1].

Returns ndarray shape (len(d), len(x)) containing the pdf(s) or a single value if only 1 point in 1 distribution is selected.

`save`(*filename*)
save model as .npz file.

Parameters **filename** – filename to save as, excluding extension"

Returns None

`tes.analysis.Povm`
alias of `tes.analysis.povm`

`tes.analysis.coherent_ml`(*counts*, *povm*, *max_photon_number*, *x0=None*)

`tes.analysis.coherent_neg_ll`(*x*, *counts*, *povm*, *max_photon_number*)

`tes.analysis.coincidence`(*abs_time*, *mask*, *low*, *high*)
Find coincidences between two channels returning indices of partner events.

Parameters

- **abs_time** (*ndarray*) – absolute times.
- **mask** (*ndarray*) – ndarray of bool that identifies the channels in abs_time. For abs_times where mask is False, time t is coincident if abs_time+low <= t <= abs_time+high. If the mask is True t is coincident if abs_time-high <= t <= abs_time-low.

- **low** (*float*) – the low side of the coincidence window.
- **high** (*float*) – the high side of the coincidence whindw.

Returns (coinc, coinc_mask) where coinc is a ndarray of indices of the coincident event in the other channel. When more than one event is found in the window the negated value of the first index is entered. coinc_mask is a ndarray of bool indicating where exactly one event was found in the window.

`tes.analysis.count`(*data*, *measurement_model*, *vacuum=False*, *has_noise_threshold=False*, *coinc_mask=None*, *herald_mask=None*)

Parameters

- **data** (*ndarray*) – measurement data.
- **measurement_model** (MixtureModel) – model created using expectation_maximisation.
- **vacuum** (*bool*) – estimate vacuum terms using coinc_mask and herald_mask.
- **coinc_mask** – ndarray of bool indicating coincidence.
- **herald_mask** – ndarray of bool indicating which events are in the heralding channel.
- **has_noise_threshold** – When True model.thresholds[1] represents the boundary between noise and 1 photon.

Returns Counts(count, uncorrelated, vacuum). Where count is an ndarray of counts for each photon number and len(count)=len(measurement_model.thresholds). uncorrelated is a ndarray the same shape as count and counts events not correlated with the herald. uncorrelated is only calculated when coinc_mask and herald_mask are supplied. vacuum replicates the parameter value and indicates that count[0] contains the vacuum count.

`tes.analysis.displaced_thermal`(*max_photon_number*, *nbar*, *alpha*, *N=200*)

`tes.analysis.displaced_vacuum`(*max_photon_number*, *alpha*, *N=200*)

`tes.analysis.drive_correlation`(*data*, *model*, *abs_time*, *detection_mask*, *r*, *last_threshold=- 1*, *masks=None*, *verbose=False*)

Calculate the temporal cross-correlation between a channel measuring a heralding signal, ie the laser drive pulse, and a channel detecting photons. The correlation is calculated for the different photon numbers determined by the thresholds parameter.

Parameters

- **abs_time** (*ndarray*) – absolute timestamp sequence.
- **detection_mask** (*ndarray*) – boolean mask that identifies the entries in abs_time belonging to the photon channel, other entries are assumed to be the heralding channel.
- **data** (*ndarray*) – energy measurement data for the photon channel.
- **masks** (*ndarray*) – boolean masks that are appied to abs_time[detection_mask] and indicate which detections are assigned which threshold. Shape(N, len(abs_time[detection_mask]))
- **r** – the range of heralding channel delays to cross correlate.
- **verbose** (*bool*) – Print progress message as each threshold is processed.

Returns list of ndarrays representing the cross correlation.

Note abs_time[mask] and data must be the same length. TODO speed up the algorithm.

tes.analysis.**expectation_maximisation**(*data*, *initial_thresholds*, *has_noise_threshold=False*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*, *tol=0.01*, *max_iter=30*, *verbose=True*)

Fit a mixture of distributions to data.

Parameters

- **data** (*ndarray*) – the measurement data.
- **initial_thresholds** – initial thresholds that divide data into individual distributions in the mixture.
- **fix_noise** (*bool*) – When True keep threshold[1] fixed.
- **dist** – the type of distribution to use (see scipy.stats)
- **tol** (*float*) – termination tolerance for change in log_likelihood.
- **max_iter** (*int*) – max iterations of expectation maximisation to use.
- **verbose** (*bool*) – print progress during optimisation.
- **normalise** (*bool*) – passed to maximise step, thresholds are calculated using the intersection of the normalised distributions.

Returns (param_list, zero_loc, log_likelihood, converged) as a named tuple. Where param_list is a list of parameter values for distribution in the mixture, zero_loc is the arg used to fit, log_likelihood is the log_likelihood of the fit, converged is a bool indicating normal termination.

Note Uses The Expectation maximisation algorithm with hard assignment of responsibilities.

tes.analysis.**fit_state_least_squares**(*counts*, *povm*, *max_outcome=None*, *vacuum=True*, *thermal=True*, *N=150*)

tes.analysis.**guess_thresholds**(*data*, *bins=16000*, *win_length=200*, *maxima_threshold=10*)

Guess the photon number thresholds using a smoothed histogram.

Parameters

- **data** (*ndarray*) – the measurement data.
- **bins** – argument to numpy.histogram.
- **win_length** (*int*) – Hanning window length used for smoothing.
- **maxima_threshold** (*int*) – passed to maxima function.

Returns (hist, f, bin_c, max_i, thresholds) as a named tuple. Where hist is the histogram generated by np.histogram(), f is the smoothed histogram, bin_c is a ndarray of bin centers, max_i are the indices of the maxima in f, and thresholds is a ndarray of threshold guesses.

Notes The smoothing is achieved by convolving the Hanning window with a histogram generated using numpy.histogram. The guess still needs some human sanity checks, especially multiple maxima per photon peak. Use plot_guess() and adjust window and maxima_threshold to remove them.

tes.analysis.**hard_expectation**(*param_list*, *fix_noise=None*, *normalise=False*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*)

Calculate the thresholds for a mixture model that define the data partitions.

Parameters

- **param_list** (*list*) – list of parameters for the distributions in the mixture model, returned by expectation_maximisation().

- **fix_noise** – Fix threshold[1] at this value.
- **normalise** (*bool*) – calculate thresholds based on a mixture of normalised pdf's.
- **dist** – type of distribution

Returns ndarray of threshold values.

`tes.analysis.`**hard_maximisation**(*data*, *thresholds*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*, *verbose=True*, *has_noise_threshold=False*)

Fit distributions to the data partitioned by thresholds.

Parameters

- **data** – the data to model.
- **thresholds** – thresholds that divide data into separate distributions.
- **dist** – type of distribution to fit (scipy.stats).
- **verbose** (*bool*) – Print fitted distribution parameters.

Returns list of parameters for each distribution.

`tes.analysis.`**maxima**(*f*, *thresh=10*)

Find local maxima of f using rising zero crossings of the gradient of f.

Parameters

- **f** – function to find maxima for
- **thresh** – Only return a maxima if f[maxima]>thresh

Returns array of maxima.

Notes Used to find the maxima of the smoothed measurement histogram.

`tes.analysis.`**mixture_model_ll**(*data*, *param_list*, *has_noise_threshold=False*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*)

Calculate the log likelihood of a mixture model.

Parameters

- **data** (*ndarray*) – the data used to construct the model.
- **param_list** – list of parameters for each distribution in the model.
- **dist** – the type of distribution to use (see scipy.stats).

Returns the log likelihood.

`tes.analysis.`**normalised_pdf**(*x*, *params*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*)

Convenience function, calculate normalised pdf at each x for dist.

Parameters

- **x** (*union[numpy.ndarray, int]*) – calculate the pdf a each x
- **params** (*iterable*) – fitted distribution parameters
- **dist** – type of distribution (see scipy.stats)

Returns ndarray containing pdf(x)

`tes.analysis.`**outcome_probabilities**(*state*, *povm*, *max_photon_number*)

`tes.analysis.`**partition**(*data*, *thresholds*, *i*)

Return the ith partition of the data as defined by thresholds.

Parameters

- **data** (*ndarray*) – the data to partition.
- **thresholds** (*iterable*) – the threshold values that partition data.
- **i** (*int*) – the partition to return, the ith partition is defined as threshold[i] < data <= threshold[i+1]. The last partition, when i = len(thresholds-1), is defined as

Returns ndarray of the data in the partition.

Note This is used to assign a hard responsibility in the expectation maximisation algorithm used to fit data to a mixture model.

`tes.analysis.`**`plot_guess`**(*hist*, *f*, *x*, *max_i*, *init_t*, *figsize=None*, *measurement_name='\\texttt{area}'*)

`tes.analysis.`**`plot_mixture_model`**(*model*, *vacuum=False*, *data=None*, *counts=None*, *xrange=None*, *normalised=False*, *bins=500*, *bar=False*, *measurement_name='\\texttt{area}'*, *last_threshold=None*, *figsize=None*)

Plot a mixture model optionally including a histogram of the modeled data. if no data is None x must not be None.

Parameters

- **model** (MixtureModel) – the mixture model.
- **data** (*ndarray*) – the data to histogram.
- **xrange** (*tuple*) – the x range to plot.
- **normalised** (*bool*) – normalise the model distributions.
- **bins** – passed to numpy.histogram().
- **bar** (*bool*) – Plot bars instead of markers.
- **figsize** – passed to matplotlib.pyplot.figure().

Returns the figure handle.

`tes.analysis.`**`plot_state_fit`**(*nbar*, *alpha*, *counts*, *povm=None*, *figsize=None*)

`tes.analysis.`**`povm_elements`**(*measurement_model*, *counts*)

estimate the elements of the system povm from the measurement model and the counting data.

Parameters

- **data** (*array like*) – measurement data.
- **measurement_model** (*mixturemodel*) – model created using expectation_maximisation.
- **counts** (Counts) – the count data returned by count().

Returns (elements count vacuum). where counts is a ndarray with shape (len(n)) containing the count for each measurement outcome in n. elements is a ndarray with shape(len(n), len(measurement_model.thresholds)) the first index is the fock state, the second is the outcome. The contents of elements is the probability of the outcome when measuring fock state.

Notes POVM elements is indexed by (fock state, measurement outcome). The last threshold in the measurement model marks the boundary of the data that was used to create the model, it is *not* altered by the expectation maximisation algorithm and is essentially a guess. The second last threshold is the boundary of the overflow bin, this last photon number bin is counts detections of >= len(measurement_model.param_list)-2 photons. TODO expand and clarify description.

`tes.analysis.`**`resize_vector`**(*a*, *max_index*, *copy=True*)

`tes.analysis.`**`retime`**(*events*, *tdat*, *fidx*, *tidx*) `tes.analysis.`**`scaled_pdf`**(*x*, *params*, *dist=<scipy.stats._continuous_distns.gamma_gen object>*)

Convenience function, calculate pdf at each x for dist, normalisation is given by params[-1].

Parameters

- **x** (*union[numpy.ndarray, int]*) – calculate the pdf a each x
- **params** (*iterable*) – fitted distribution parameters
- **dist** – type of distribution (see scipy.stats)

Returns ndarray containing pdf(x)

`tes.analysis.`**`thermal_ml`**(*counts*, *povm*, *max_photon_number*, *x0=None*)

`tes.analysis.`**`thermal_neg_ll`**(*x*, *counts*, *povm*, *max_photon_number*)

`tes.analysis.`**`threshold_masks`**(*data*, *model*)

Parameters

- **data** –
- **model** –

Returns

`tes.analysis.`**`window`**(*i*, *abs_time*, *low*, *high*)

get offsets from the current index i in abs_time that are in the relative time window defined by low and high.

Parameters

- **i** (*int*) – current abs_time index.
- **abs_time** (*ndarray*) – absolute times.
- **low** – low end of relative window.
- **high** – high end of relative window.

Returns (low_o, high_o) coincident times are abs_time[low_o:i+high_o]

`tes.analysis.`**`x_correlation`**(*s1*, *s2*, *r*)

Cross correlation of timestamp sequences s1 and s2 over the delay range r.

Parameters

- **s1** (*ndarray*) – Monotonically increasing timestamp sequence.
- **s2** (*ndarray*) – Monotonically increasing timestamp sequence.
- **r** – range of delays added to s1

Returns ndarray representing the cross correlation function.

`tes.analysis.`**`xcor`**(*s1*, *s2*)

Count correlations between timestamp sequences.

Parameters

- **s1** (*ndarray*) – Monotonically increasing timestamp sequence.
- **s2** (*ndarray*) – Monotonically increasing timestamp sequence.

Returns correlation count.

Notes iterates over s1 and performs a linear search of s2 for the s1 timestamp.

tes.base module

Classes:

1) VhdlEnum
2) TraceType
3) Signal
4) Height
5) Timing
6) detection
7) Payload
8) lookup

Function: 1) pulse_fmt

class tes.base.**Detection**(*value*)

Bases: *tes.base.VhdlEnum*

Value of the event.packet register.

area = 1

pulse = 2

rise = 0

trace = 3

class tes.base.**Height**(*value*)

Bases: *tes.base.VhdlEnum*

Value of the event.height register.

cfd_height = 2

cfd_high = 1

max_slope = 3

peak = 0

class tes.base.**Payload**(*value*)

Bases: *tes.base.VhdlEnum*

Payload type in capture files.

area = 1

average_trace = 4

bad_frame = 9

dot_product = 5

dot_product_trace = 6

mca = 8

pulse = 2

rise = 0

single_trace = 3

tick = 7

class tes.base.**Signal**(*value*)
Bases: *tes.base.VhdlEnum*

The signal recorded in a trace.

f = 2

none = 0

raw = 1

s = 3

class tes.base.**Timing**(*value*)
Bases: *tes.base.VhdlEnum*

Value of the event.timing register.

cfd_low = 2

max_slope = 3

pulse_threshold = 0

slope_threshold = 1

class tes.base.**TraceType**(*value*)
Bases: *tes.base.VhdlEnum*

Value of the trace_type event.trace_type register.

average = 1

dot_product = 2

dot_product_trace = 3

single = 0

class tes.base.**VhdlEnum**(*value*)
Bases: int, enum.Enum

An enumeration.

latex()

select()

tes.base.**lookup**(*value*, *enum*)

tes.base.**pulse_fmt**(*n*)

tes.base.**rise2_fmt**(*n*)

tes.calibration module

Functions to calibrate the TES detector.

Author: Leonardo Assis Morais

`tes.calibration.`**`area_histogram`**(*max_area*, *bin_number*, *areas*)

Create an area histogram.

Parameters

- **`max_area`** (*`float`*) –

  Maximum area allowed (remove extremely large areas not due to photon detections).

- **`bin_number`** (*`int`*) – Number of histogram bins.
- **`areas`** (*`np.ndarray`*) – Data with areas of TES pulses.

Returns

- **bin_centre** (*np.darray*) – Array with the positions of the histograms bin centres. Used for plotting the histogram with correct values for areas in the x-axis.
- **counts** (*np.ndarray*) – Histogram counts.
- **error** (*np.ndarray*) – Standard deviation for counts (poissonian distribution assumed: std. = sqrt(counts).
- **bin_width** (*float*) – Bin width for plotting.

`tes.calibration.`**`find_thresholds`**(*gauss_fit*, *maxima_list*, *bin_centre*, *counts*, *const*)

Find the counting thresholds given a gaussian fit.

Parameters

- **`gauss_fit`** (*`lmfit fit`*) – Result of a least square minimisation using lmfit.
- **`maxima_list`** (*`np.ndarray`*) – Array with the positions of maxima points.
- **`bin_centre`** (*`np.ndarray`*) – Array with centre bin positions for histogram.
- **`counts`** (*`np.ndarray`*) – List with counts for histogram.
- **`const`** (*`float`*) – Value used for scipy.optimise.brentq. See 'Notes' below for more information.

Returns

- **dist** (*list*) – List with the normalised distributions.
- *new_thresh_list* – List of counting thresholds.
- *Requires*
- ——— – scipy.optimize.brentq

Notes

Sometimes this function breaks if brentq function cannot find roots. If that is the case, slowly increase the value of the variable 'const'.

tes.calibration.**gaussian_model**(*fitting*, *counts*, *bin_centre*, *max_idx*)

Implement mixture model comprised of gaussians.

Parameters

- **fitting** (*class lmfit.minimizer.MinimizerResult*) – Result from fitting using lmfit.
- **counts** (*np.ndarray*) – Histogram counts.
- **bin_centre** (*np.darray*) – Array with the positions of the histograms bin centres. Used for plotting the histogram with correct values for areas in the x-axis.
- **max_idx** (*int*) – Number of gaussian distributions used in the model.

Returns **model** – Model of composed of sum of gaussians using the results obtained from the fitting with lmfit.

Return type np.ndarray

tes.calibration.**guess_histogram**(*areas*, *bin_number*, *win_length*, *minimum*, *max_area*)

Create first guess for histogram fitting.

Using a Hann function to create an approximate fit for the data, estimates the position for the centre of the different peaks in the area histogram.

WARNING: The Hann window fitting must be checked to see if it is reasonable using the function plot_guess.

maxima

Parameters

- **bin_number** (*int*) – Number of bins used in histogram.
- **win_length** (*int*) – Length of the Hann window used in the fit.
- **minimum** (*int*) – Minimum value for the function maxima to consider a given number of counts as a peak.
- **max_area** (*float*) – Areas > max_area are considered invalid counts and removed from the analysis. Used to remove event with anomalous areas.

Returns

- **counts** (*np.darray*) – Array with counts for each histogram bin.
- **smooth_hist** (*np.darray*) – Array with the continuous fit used to determine the positions of the maxima in the data.
- **bin_centre** (*np.darray*) – Array with the positions of the histograms bin centres. Used for plotting the histogram with correct values for areas in the x-axis.
- **max_i** (*list*) – List with the position of the maxima points.

tes.calibration.**maxima**(*function*, *thresh=10*)

Find local maxima of a function f.

Uses rising zero crossings of the gradient of f to find function local maxima.

Parameters

- **function** (*ndarray*) – Function f to be analysed.
- **thresh** (*int*) – Only return a maxima if f[maxima] > thresh.

Returns

- *array*
- *Array with maxima points of f.*

`tes.calibration.`**`plot_area`**(*ax*, *bin_centre*, *counts*, *error*, *model*, *plot_steps*)

Require area_histogram to be run before.

Parameters

- **ax** (*figure axis*) – Figure axis of the figure to be plotted.
- **bin_centre** (*np.darray*) – Array with the positions of the histograms bin centres. Used for plotting the histogram with correct values for areas in the x-axis.
- **counts** (*np.ndarray*) – Histogram counts.
- **error** (*np.ndarray*) – Standard deviation for counts
- **model** (*np.ndarray*) – Model generated with gaussian_model
- **plot_steps** (*int*) – Number of histogram points to be skipped when plotting.

Returns

- *None.*
- *Requires*
- *——*
- *area_histogram*

`tes.calibration.`**`plot_guess`**(*ax*, *hist*, *smooth_hist*, *bin_centres*, *max_i*)

Plot the histogram with an educated guess for its fitting.

Visual check if the maxima positions determined by the guess_thresholds are reasonable.

Requires guess_thresholds.

Parameters

- **ax** (*figure axis*) – Figure axis of the figure to be plotted.
- **hist** (*np.darray*) – Histogram obtained from the guess_thresholds function.
- **smooth_hist** – Smoothed histogram obtained from the guess_thresholds function.
- **bin_centres** – Centre of the bins for the histogram. Obtained from the guess_thresholds function.
- **max_i** (*list*) – List with the position of the maxima points for the peaks in histogram. Obtained from the guess_thresholds function.

Returns **ax** – Axis for the figure with the histogram, smoothed histogram and maxima points plotted.

Return type figure axis

`tes.calibration.`**`plot_histogram`**(*ax*, *data*, *bin_number*, *measurement*)

Plot histogram of TES characteristics.

Parameters

- **ax** (*figure axis*) – Figure axis of the figure to be plotted.

- **data** (*np.ndarray*) – Data extracted from TES (height, area, length, maximum slope).
- **bin_number** (*int*) – Number of histogram bins.
- **measurement** (*str*) – Type of measurement plotted.

Returns

Return type None.

`tes.calibration.plot_normalised(ax, max_i, bin_centre, dist, thresholds)`

Plot the graph with the normalised distributions.

Parameters

- **ax** (*figure axis*) – Figure axis of the figure to be plotted.
- **max_i** (*np.ndarray*) – Number of distributions to be plotted.
- **bin_centre** (*np.ndarray*) – x-axis positions for histogram data.
- **dist** (*np.ndarray*) – Distributions to be plotted.
- **thresholds** (*np.ndarray*) – Counting thresholds positions to be plotted.

Returns

Return type None.

`tes.calibration.residual_gauss(params, i_var, data, eps_data, max_idx)`

Gaussian model to fit histogram from TES detections.

Model composed of a sum of N gaussian distributions, to be used with the lmfit package to fit histograms from TES detections. Returns the residual between data and model, divided by the error in the data.

Parameters

- **params** (*lmfit object*) – List of parameters to be used in the fit.
- **i_var** – Independent variable to be used in the fit.
- **data** (*np.ndarray*) – Data to be fitted.
- **eps_data** (*np.ndarray*) – Standard deviation of each data point.
- **max_idx** (*int*) – Number of gaussian distributions used in the model.

Returns **residual** – Array with the different between data and model divided by the error in data.

Return type np.ndarray

tes.counts module

Module to assist in counting photons routine.

1) coincidence 3) window

`class tes.counts.Counts(count, uncorrelated, vacuum)`

Bases: `tuple`

`property count`

Alias for field number 0

`property uncorrelated`

Alias for field number 1

`property vacuum`
Alias for field number 2

`tes.counts.coincidence`(*abs_time*, *mask*, *low*, *high*)
Find coincidences between two channels.

After finding a coincidence, return indices of coincident events.

Parameters

- `abs_time` (*ndarray*) – Array where each entry corresponds to the time which the detection was performed with respect to an initial time t0. This array can be constructed using np.cumsum(CaptureData(datapath).times).
- `mask` (*ndarray bool*) – Identifies the channels in abs_time. For abs_times where mask is False, time t is coincident if abs_time+low <= t <= abs_time+high. If the mask is True t is coincident if abs_time-high <= t <= abs_time-low.
- `low` (*float*) – The starting of the coincidence window.
- `high` (*float*) – The ending side of the coincidence window.

Returns

- **coincidences(coinc, coinc_mask)** (*tuple*) –

  coinc [ndarray] Indexes of the coincident event in the other channel. When more than one event is found in the window the negated value of the first index is entered.

  coinc_mask [ndarray bool] Indicates where exactly one event was found in the window.
- *Requires*
- ——— – tes.counts.window

`tes.counts.counting_photons`(*data*, *thresh_list*, *vacuum=False*, *coinc_mask=None*, *herald_mask=None*)
Convert TES pulse area in photon-number.

Given an TES pulse area data set and the calibration (list with the counting thresholds, passed through thresh_list), returns the number of counts for each Fock state.

If the mask with the heralding source is given, also includes the number of vacuum counts.

Parameters

- `data` (*np.ndarray*) – Measured data.
- `thresh_list` (*np.ndarray*) – List with the counting thresholds obtained after detector calibration.
- `vacuum` (*bool*) – If true, it will use the herald_mask to calculate vacuum counts
- `coinc_mask` (*ndarray, bool*) – Mask indicating coincidence counts.
- `herald_mask` (*ndarray, bool*) – Mask indicating which events are in the heralding channel.

Returns

Counts(count, uncorrelated, vacuum) –

count [ndarray] Counts for each photon number

uncorrelated : Counts events not correlated with herald. Only calculated when coinc_mask and herald_mask are provided.

vacuum : indicates if count[0] contains vacuum counts

Return type named tuple

`tes.counts.`**`get_thresholds`**`(`*calibration_file*`)`

Load the threshold from a calibration file.

Calibration file must be generated with "TES_Calibration.ipynb".

Parameters **`calibration_file`** (*`str`*) – Calibration file path.

Returns **thresholds** – Array with the threshold positions.

Return type np.array

`tes.counts.`**`window`**`(`*ind*, *abs_time*, *low*, *high*`)`

Find coincident events in the given window.

Get offsets from the current index ind in abs_time that are in the relative time window defined by low and high.

Parameters

- **`ind`** (*`int`*) – Current index for abs_time.
- **`abs_time`** (*`ndarray`*) – Absolute times.
- **`low`** (*`float`*) – Starting point of the relative window.
- **`high`** (*`float`*) – Ending point of the relative window.

Returns **(low_index, high_index)** – coincident times are abs_time[low_index:ind+high_index]

Return type tuple

tes.data module

Functions:

1) FpgaStats
2) capture
3) read_mca
4) av_trace
5) _memmap_data

Classes:

1) FpgaStats
2) CaptureResult
3) CaptureData

`class` `tes.data.`**`CaptureData`**`(`*path*`)`

Bases: `object`

`property homogeneous`

`mask``(`*channel*`)`

`class` `tes.data.`**`CaptureResult`**`(`*ticks*, *events*, *traces*, *mca*, *frames*, *dropped*, *invalid*`)`

Bases: *`tes.data.CaptureResult`*

`class` `tes.data.`**`FpgaStats`**`(`*frames*, *dropped*, *bad*, *tick*, *mca*, *trace*, *event*, *types*`)`

Bases: `tes.data.FpgaOutputStats`

Class holding FPGA ethernet output statistics

`tes.data.av_trace(timeout=30)`
Capture an average trace.

Parameters `timeout` – Timeout value in seconds.

Returns

`tes.data.capture(filename, measurement, use_existing=1, access_mode=1, ticks=10, events=0, conversion_mode=0)`
Capture FPGA output as a collection of data and index files.

Parameters

- **filename** –
- **measurement** –
- **ticks** –
- **events** –
- **write_mode** –
- **conversion_mode** –
- **capture_mode** –

Returns

`tes.data.fpga_stats(time=1.0)`
Diagnostic statistics for the FPGA ethernet output.

Parameters **time** (*float*) – time to capture statistics for

Returns FpgaStats object that subclasses namedtuple.

`tes.data.read_mca(n)`
Capture MCA histograms

Parameters **n** (*int*) – number of histograms to capture

Returns List of tes.mca.Distributions.

tes.data_acquisition module

Module used to perform measurements with the TES.

Functions:

1) trace_drive
2) pulse_drive

`tes.data_acquisition.pulse_drive(time, channel, p_thres, s_thres, baseline_sub, datapath, filename)`
Perform measurements over TES traces.

Measure the following characteristics of TES traces:

1) Length
2) Area
3) Maximum slope or height
4) Rise Time

Parameters

- **time** (*int*) – Time in seconds to take measurements.
- **channel** (*int*) – Processing channel chosen to take measurements.
- **p_thres** (*int*) – Pulse threshold chosen using the MCA.
- **s_thres** (*int*) – Slope threshold chosen using the MCA.
- **base_sub** (*bool*) – If True, the baseline correction will be activated. It can automatically update the baseline level in the case where it changes.
- **datapath** (*str*) – Folder where the registers will be saved.
- **filename** (*str*) – Name of the file where the registers will be saved.

Returns

Return type None

tes.data_acquisition.**trace_drive**(*time*, *channel*, *p_thres*, *s_thres*, *baseline_sub*, *datapath*, *filename*)

Record TES traces.

Parameters

- **time** (*int*) – Time in seconds to take measurements.
- **channel** (*int*) – Processing channel chosen to take measurements.
- **p_thres** (*int*) – Pulse threshold chosen using the MCA.
- **s_thres** (*int*) – Slope threshold chosen using the MCA.
- **base_sub** (*bool*) – If True, the baseline correction will be activated. It can automatically update the baseline level in the case where it changes.
- **datapath** (*str*) – Folder where the registers will be saved.
- **filename** (*str*) – Name of the file where the registers will be saved.

Returns

Return type None

Notes

Keep your trace measurements up to 1 minute.

tes.folder_management module

Module with convenient functions for folder management.

Contains: - find_folders - manage_folders

tes.folder_management.**find_folders**(*master_folder*)

Search for folders inside the given address.

Parameters **master_folder** (*string*) – address of the master folder on local computer.

Raises **AttributeError:** – Raises error when no folder can be found.

Returns **measurements** – list with the address of folders inside master_folder.

Return type list

`tes.folder_management.`**`manage_folders`**(*index*, *measurement_folders*)

Create a folder structure for a given measurement set.

Parameters

- **`index`** (*`integer`*) – Select the measurement folder to be analysed
- **`measurements`** (*`list`*) – contains all addresses of measurement folders

Raises `AttributeError:` – Raises error when no folder can be found.

Returns

datapath [path] address of the measurement folder

folder_analysis [path] address of the analysis folder

folder_figures [path] address of the figures folder

Return type tuple with

tes.maps module

Classes:

1) RegisterError
2) Borg
3) Transport_base
4) _ZmqTransport
5) _DirectSerial
6) RegInfo

Functions:

1) _map_str
2) def _field_str
3) _cpu_version
4) _from_onehot
5) _to_onehot
6) _to_cf
7) _from_cf
8) _to_gain(g):
9) _from_gain

`class` `tes.maps.`**`RegInfo`**(*address*, *field*, *strobe*, *output_transform*, *input_transform*, *loadable*, *name=None*, *doc=None*)

Bases: `object`

Class describing A FPGA register""

`get`(*transform=None*, *indices=None*)

`set`(*values*, *transform=None*, *indices=None*)

exception tes.maps.**RegisterError**(*non_hex=False*, *bad_length=False*, *axi='OKAY'*)
Bases: AttributeError

tes.mca module

MCA low level control.

Classes:

1) Value
2) Trigger
3) Qualifier
4) Distribution

class tes.mca.**Distribution**(*data*, *buffer=True*)
Bases: object

wrapper for transmitted zmq frame representing a MCA distribution

property bin_width

property bins

property channel

property highest_value

property last_bin

property lowest_value

property most_frequent

property overflow

property qualifier

property start_time

property stop_time

property total

property trigger

property underflow

property value

class tes.mca.**Qualifier**(*value*)
Bases: *tes.base.VhdlEnum*

An enumeration.

above = 4

above_area = 3

all = 1

armed = 6

disabled = 0

rise = 8

`valid_peak1 = 9`

`valid_peak2 = 10`

`valid_peak3 = 11`

`valid_rise = 2`

`will_arm = 7`

`will_cross = 5`

class tes.mca.**Trigger**(*value*)

Bases: *tes.base.VhdlEnum*

An enumeration.

`cfd_high = 9`

`cfd_low = 10`

`clock = 1`

`disabled = 0`

`f_0xing = 5`

`max_slope = 11`

`pulse_t_neg = 3`

`pulse_t_pos = 2`

`s_0xing = 6`

`s_0xing_neg = 8`

`s_0xing_pos = 7`

`slope_t_pos = 4`

class tes.mca.**Value**(*value*)

Bases: *tes.base.VhdlEnum*

An enumeration.

`cfd_high = 9`

`disabled = 0`

`f = 1`

`f_area = 2`

`f_extrema = 3`

`pulse_area = 7`

`pulse_timer = 10`

`raw = 8`

`rise_timer = 11`

`s = 4`

`s_area = 5`

`s_extrema = 6`

tes.mca_control module

Module with functions to control the MCA.

Should be used together with the Jupyter Notebook "MCA_and_Measurements.ipynb".

Functions defined here:

1) configure_channels
2) baseline_offset
3) pulse_threshold
4) slope_threshold
5) area_histogram

`tes.mca_control.`**`area_histogram`**(*registers*, *time*, *channel*, *bin_width*, *p_thres*, *s_thres*, *xrange*, *yrange*)

Plot area histogram given the thresholds chosen.

Always run this function to realise a sanity test for the selected values for baseline offset, pulse threshold and slope threshold. You should be able to see clear distinct peaks indicating that the current configuration of the TES can discriminate number of photons.

Parameters

- **registers** (*dict*) – Dictionary that stores the values of all registers that control the measurements performed by the FPGA.
- **time** (*int*) – Time (in seconds) the histogram should be accumulated over.
- **channel** (*int*) – Channel to be analysed.
- **bin_width** (*int*) – 2^(bin_width) will be the width of the histogram bin.
- **s_thres** (*int*) – Determine the pulse threshold.

Returns **(fig,)** – Figure with the plotted histogram.

Return type tuple

`tes.mca_control.`**`baseline_offset`**(*registers*, *time*, *channel*, *bin_width*, *baseline_offset*)

Determine the detection baseline for TES detections.

Change the values of baseline_offset until you see a big peak in the histogram. When the peak appears, put its centre at x = 0 by choosing an appropriate baseline_offset.

Parameters

- **registers** (*dict*) – Dictionary that stores the values of all registers that control the measurements performed by the FPGA.
- **time** (*int*) – Time (in seconds) the histogram should be accumulated over.
- **channel** (*int*) – Channel to be analysed.
- **bin_width** (*int*) – 2^(bin_width) will be the width of the histogram bin
- **baseline_offset** (*int*) – Determine the baseline offset.

Returns **(fig,)** – Figure with the plotted histogram.

Return type tuple

`tes.mca_control.configure_channels`(*registers*, *adc_channel*, *proc_channel*, *invert*)

Configure connections to ADC channels.

Connects the selected ADC channel (check in the lab to which ADC channel you connected the TES cables) to the selected processing channel (digital channel to which you will refer in the next steps of the notebook).

Parameters

- **registers** (*dict*) – Dictionary that stores the values of all registers that control the measurements performed by the FPGA.
- **adc_channel** (*int*) – Determine which adc channel is being used. May assume values 0-7.
- **proc_channel** (*int*) – Determine which processing channel is being used. May assume values 0-1.
- **invert** (*bool*) – Determines if the signal polarisation should be inverted. True inverts, False does not invert.

Returns

Return type None.

`tes.mca_control.pulse_threshold`(*registers*, *time*, *channel*, *bin_width*, *p_thres*)

Determine the pulse threshold.

Parameters

- **registers** (*dict*) – Dictionary that stores the values of all registers that control the measurements performed by the FPGA.
- **time** (*int*) – Time (in seconds) the histogram should be accumulated over.
- **channel** (*int*) – Channel to be analysed.
- **bin_width** (*int*) – 2^(bin_width) will be the width of the histogram bin.
- **pulse_threshold** (*int*) – Determine the pulse threshold.

Returns **(fig,)** – Figure with the plotted histogram.

Return type tuple

`tes.mca_control.slope_threshold`(*registers*, *time*, *channel*, *bin_width*, *s_thres*)

Determine the slope threshold.

Parameters

- **registers** (*dict*) – Dictionary that stores the values of all registers that control the measurements performed by the FPGA.
- **time** (*int*) – Time (in seconds) the histogram should be accumulated over.
- **channel** (*in*) – Channel to be analysed.
- **bin_width** (*int*) – 2^(bin_width) will be the width of the histogram bin.
- **s_thres** (*int*) – Determine the pulse threshold.

Returns **(fig,)** – Figure with the plotted histogram.

Return type tuple

tes.protocol module

Classes:

1) PayloadType

`class tes.protocol.PayloadType(value)`
Bases: `enum.IntEnum`

An enumeration.

`area = 1`

`mca = 5`

`peak = 0`

`pulse = 2`

`tick = 4`

`trace = 3`

tes.registers module

Functions:

1) _adc_spi_transform
2) _channel_transform
3) save
4) load

Classes:

1) _RegisterMap
2) _RegisterGroup
3) _GroupIterator
4) Registers

`class tes.registers.Registers(port=None)`
Bases: `tes.registers._RegisterMap`

Client for reading and writing the internal FPGA control registers.

Registers are arranged in functional groups and accessed through an instance of the Registers class. Let r be an instance of Registers then r.regname references the regname register while r.groupname.regname the regname register of the groupname group. Some groups support indexing to reference a register for a particular channel. Slicing and fancy indexing are supported while ommiting indexing is equvalent to referencing ALL channels.

For example: r.groupname[0].regname refers to the regname register of the groupname group for channel 0. While r.groupname.regname refers to the same register for all channels. Therefore, r.groupname.regname will return a list containing the value of the register for each channel, r.groupname.regname = value will set the for all channels to the same value and r.groupname.regname = [value0, value1, . . . , valuen] will broadcast the list of values to the appropriate channel.

Groups without indexing: No groupnme - accesses a general register. See help(Registers). mca - accesses the registers controlling the MCA. See help(McaRegisters).

Groups supporting indexing: channel controls input to the processing channels. See help(ChannelRegisters event controls event output. See help(EventRegisters). baseline controls the baseline process. See help(BaselineRegisters). cfd controls the constant fraction process. See help(CfdRegisters). adc controls the ADC chips. See help(AdcRegisters).

The the number of channels in the adc group is twice the value of the general register adc_chips while the number of channels in all other groups is the value of the general register channel_count.

TODO add dict indexing

`adc`
Registers controlling the ADC chips on the FMC108 digitiser card.

`baseline`
Registers controlling baseline correction.

`cfd`
Registers controlling the cfd process.

`channel`
Registers controlling channel input.

`event`
Registers controlling event output.

`mca`
Registers controlling the MCA

`tes.registers.load`(*filename*)

`tes.registers.save`(*d*, *filename*)

tes.traces module

Module to be used with jupyter notebook TES Trace Generator.

Contains:

1) extract_data
2) create_statistics_file
3) plot_traces
4) correct_xticks
5) correct_yticks

`tes.traces.correct_xticks`(*ax*)
Properly edit the xticks for a matplotlib plot.

Parameters **ax** (*axis object*) – Axis with x-axis to be edited.

Returns

- **ax** (*axis object*) – Edited axis object.
- **expx** (*int*) – Integer to be added to the x-axis label.

`tes.traces.correct_yticks`(*ax*)
Properly edit the yticks for a matplotlib plot.

Parameters **ax** (*axis object*) – Axis with x-axis to be edited.

Returns

- **ax** (*axis object*) – Edited axis object.
- **expy** (*int*) – Integer to be added to the x-axis label.

tes.traces.**create_statistics_file**(*measurement_folders*)

Create file with measurements statistics in measurement folder.

Parameters **measurement_folder** (*list of paths*) – list containing the paths of the folders where the measurements are stored and where the stats files will be created.

Returns

Return type None

tes.traces.**extract_data**(*datapath*, *measurements*)

Extract data and registers for a given address.

Parameters

- **datapath** (*str*) – String containing the measurement folder address.
- **measurements** (*list of paths*) – List with the paths of all measurement folders.

Returns

- **data** (*tes.data.CaptureData*) – Contains the data to be plotted.
- **registers** (*TYPE*) – Contains the information required to plot timing information properly.

tes.traces.**plot_traces**(*data*, *ax*, *number_traces*, *trace_length*, *time_register*, *choose_trace*)

Plot traces using collected using the FPGA.

Can plot:

- **a single trace (number_traces = 1, slope = False,** details = False)
- many traces (number_traces > 1)

Parameters

- **data** (tes.data.CaptureData) – Contains the data to be plotted.
- **number_traces** (*int*) – The number of different TES pulses to be plotted.
- **trace_length** (*int*) – The number of points in each TES pulse.
- **time_register** (*dict*) – Contains the time information for the pulses.
- **choose_trace** (*int*) – Choose a single TES pulse to plot. (if number_traces == 1)
- **ax** (*matplotlib.axis*) – Figure axis where traces will be plotted.

Returns

Return type None.

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

Note: This project is under active development.

PYTHON MODULE INDEX

t

INDEX

G

H

L

M

N

O

P

Q

R