M-Squared Automation

Yudong Sun

Feb 09, 2022

CONTENTS:

1 Indices and tables 13

Python Module Index 15

`fitting.fit_functions.convertODRtoOCF(`*func*`)`

Generates a function for scipy.optimize.curve_fit based on a scipy.odr function

Parameters

func [function] fcn(beta, x) –> y

Returns

func [function] fcn(x, *beta) –> y

where beta = paramaters

`fitting.fit_functions.iso_omega_z(`*params*, *z*`)`

Beam Radii Function to be fitted according to ISO 11146-1:2021

Version Referenced: BS EN ISO 11146-1:2021, equation (24) in Section 9 Beam Diameter Definition: BS EN ISO 11145:2018, Section 3.5.2 (d4sigma)

The original function uses the diameter, we change it to use the radii by dividing it by 2. We assume a stigmatic/simple astigmatic beam.

The ISO Norm says that normally, the errors in the x,y-axis are optimized (i.e. using scipy.optimize.curve_fit)

d_sigma(z) = sqrt(a + bz + cz^2) z_0 = - b / 2c d_0 = sqrt((4ac - b^2)/(4c)) w_0 = 0.5 * sqrt((4ac - b^2)/(4c)) z_R = sqrt(4ac - b^2)/(2c) M_sq = (pi/(8*lmbda)) * sqrt(4ac - b^2)

Parameters

params [array_like] rank-1 array of length 3 where `beta = array([a, b, c])`

z [array_like] rank-1 array of positions along an axis

Returns

y [array_like] Rank-1, calculated beam-radii of a single axis based on given parameters

Notes

1. **From Wikipedia: If the beam does not fill more than a third of the beam profiler's sensor area,** then there will be a significant number of pixels at the edges of the sensor that register a small baseline value (the background value). If the baseline value is large or if it is not subtracted out of the image, then the computed D4σ value will be larger than the actual value because the baseline value near the edges of the sensor are weighted in the D4σ integral by x2. Therefore, baseline subtraction is necessary for accurate D4σ measurements

`fitting.fit_functions.omega_z(`*params*, *z*`)`

Beam Radii Function to be fitted, according to https://docs.scipy.org/doc/scipy/reference/odr.html

Beam Radii to be the HALF of the D4Sigma definition of beam width.

Note that this function is normalized if: - Everything is in SI-Units, or - w, w_0: [um], z, z_0: [mm], lmbda: [nm]

Parameters

params [array_like] rank-1 array of length 3 where `beta = array([w_0, z_0, M_sq_lmbda])`

z [array_like] rank-1 array of positions along an axis

Returns

y [array_like] Rank-1, calculated beam-radii of a single axis based on given parameters `fitting.fit_functions.omega_z_lambda(`*wavelength*`)`

Returns a w_0 Function to be fitted, according to https://docs.scipy.org/doc/scipy/reference/odr.html that has wavelength already included

Refer to fit_functions.omega_z for documentation

Parameters

wavelength [float] Wavelength to be used for the M^2 Fit

Returns

func [f(params, z) -> y] omega_z function that has lambda included

`class fitting.fitter.Fitter`

Fitter provides the superclass for ODRFitter and OCFFitter

`getPlotOfFit(`*numpoints=4096*`)`

Plots the fitted function with the original data. Opens a *matplotlib* figure to achieve this.

Returns the *matplotlib* figures and axes.

Parameters

numpoints [int, optional] Number of data points along the x-axis, by default 4096

:rtype: [py:data:~*typing.Tuple*[`Figure`, `Axes`]]

`class fitting.fitter.MsqFitter(`*wavelength*, *wavelength_err=0*, *mode=3*`)`

Superclass of all Msq Fitters

Parameters

mode: int 0: Fit using Msq*lambda as one term in the beam width equation 1: Fit using Msq as one term in the beam width equation, lambda directly included 2: Fit using the ISO Method. Refer to fitting.fit_functions.iso_omega_z() for more information (Default)

If using *mode = 0*, fits using M_sq_lambda instead of just M_sq. This allows the error of the wavelength to be taken into account. The ISO Fitting method also takes into account the error of the wavelength. If using *mode = 1*, the error of the wavelength is disregarded.

`estimateInitialGuesses()`

Estimates the initial parameters w_0, z_0 from the data given using the minimum y-value and save it into self.initial_guesses. Only for mode = 0 or 1

`setInitialGuesses(`*w_0=1*, *z_0=1*, *M_sq=1*`)`

Sets the initial guesses, only for mode = 0 or 1

Parameters

w_0 [float, optional] Guess for beam waist radius, by default 1

z_0 [float, optional] Guess for focal point position, by default 1

`class fitting.fitter.MsqOCFFitter(`*x*, *y*, *yerror*, *wavelength*, *wavelength_err=0*, *mode=3*`)`

Class to fit for an M_Squared using fit_functions.omega_z (Guassian Beam Profile function) using scipy.optimize.curve_fit

By default, initial guesses for w_0 and z_0 are 1. Use self.estimateInitialGuesses() to estimate w_0, z_0

Note that the fit function is normalized if:

- Everything is in SI-Units, or
- w, w_0: [um], z, z_0: [mm], lmbda: [nm]

Using the second case seem to be more numerically stable.

Parameters

x [array_like] Rank-1, Independent variable

y [array_like] Rank-1, Dependent variable, should be of the same shape as `x`

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror

wavelength [float_like] Wavelength of the laser, to be given manually for fitting

wavelength_err [float_like, optional] Error of the wavelength of the laser, to be used in error propagation to find the m_squared By default: 0

mode: int 0: Fit using Msq*lambda as one term in the beam width equation 1: Fit using Msq as one term in the beam width equation, lambda directly included 2: Fit using the ISO Method. Refer to fitting.fit_functions.iso_omega_z() for more information (Default)

If using *mode = 0*, fits using M_sq_lambda instead of just M_sq. This allows the error of the wavelength to be taken into account. The ISO Fitting method also takes into account the error of the wavelength. If using *mode = 1*, the error of the wavelength is disregarded.

Attributes

wavelength [array_like of rank 2] [wv, wv_err] - wavelength of the data and its corresponding error

initial_guesses [array_like] initial_guesses for the fit

m_squared [array_like] `np.array([m_squared, m_squared_err])` of floats; calculated m_squared based on self.wavelength and the fit

msq_lambda [bool] Flag to fit to M_sq_lambda or M_sq

`estimateAndFit()`

Equivalent to running `estimateInitialGuesses()` then `fit()`

Returns

self.output [Output instance] See OCFFitter.fit() for more information

`fit()`

Fits using self.initial_guesses and OCFFitter.fit()

Returns

self.output [namedtuple] See OCFFitter.fit() for more information

`class fitting.fitter.MsqODRFitter`(*x*, *y*, *xerror*, *yerror*, *wavelength*, *wavelength_err=0*, *mode=3*)

Class to fit for an M_Squared using fit_functions.omega_z (Guassian Beam Profile function) using ODR,

By default, initial guesses for w_0 and z_0 are 1. Use self.estimateInitialGuesses() to estimate w_0, z_0

Note that the fit function is normalized if:

- Everything is in SI-Units, or
- w, w_0: [um], z, z_0: [mm], lmbda: [nm]

Using the second case seem to be more numerically stable.

Parameters

x [array_like] Rank-1, Independent variable

y [array_like] Rank-1, Dependent variable, should be of the same shape as `x`

xerror [array_like or function] Rank 1, Error in x, should be of the same shape as `x` or func(x) –> xerror

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror

wavelength [float_like] Wavelength of the laser, to be given manually for fitting

wavelength_err [float_like, optional] Error of the wavelength of the laser, to be used in error propagation to find the m_squared By default: 0

mode: int 0: Fit using Msq*lambda as one term in the beam width equation 1: Fit using Msq as one term in the beam width equation, lambda directly included 2: Fit using the ISO Method. Refer to fitting.fit_functions.iso_omega_z() for more information (Default)

If using *mode = 0*, fits using M_sq_lambda instead of just M_sq. This allows the error of the wavelength to be taken into account. The ISO Fitting method also takes into account the error of the wavelength. If using *mode = 1*, the error of the wavelength is disregarded.

Attributes

model [scipy.odr.Model Instance]

data [scipy.odr.RealData Instance]

odr [scipy.odr.ODR Instance]

output [scipy.odr.Output instance]

wavelength [array_like of rank 2] [wv, wv_err] - wavelength of the data and its corresponding error

initial_guesses [array_like] initial_guesses for the fit

m_squared [array_like] `np.array([m_squared, m_squared_err])` of floats; calculated m_squared based on self.wavelength and the fit

msq_lambda [bool] Flag to fit to M_sq_lambda or M_sq

`estimateAndFit()`

Equivalent to running `estimateInitialGuesses()` then `fit()`

Returns

self.output [Output instance] See ODRFitter.fit() for more information

`fit()`

Fits using self.initial_guesses and ODRFitter.fit()

Returns

self.output [Output instance] See ODRFitter.fit() for more information

`class fitting.fitter.OCFFitter(`*x*, *y*, *yerror*, *func*`)`

The OCFFitter class fits the given data using scipy.optimize.curve_fit (OCF) and the least-squares method

Parameters

x [array_like] Rank-1, Independent variable

y [array_like] Rank-1, Dependent variable, should be of the same shape as `x`

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror or scalar

func [function] fcn(beta, x) –> y

This is based on scipy.odr. It will be converted to a function suitable for scipy.optimize.curve_fit where necessary.

Attributes

data [namedtuple] .x = xdata .y = ydata .sy = yerror .sx = None

output: namedtuple .beta = params .sd_beta = one standard deviation errors on the parameters

`fit`(*initial_params*)
Fit the data using `scipy.optimize.curve_fit()` and saves the output to `self.output`

Parameters

initial_params [array_like] Represents the initial guesses. Rank 1 Array with length equal to the number of parameters defined for self.model. For w(z): Rank 1 of length 4 with `initial_params = array([w_0, z_0, M_sq, lmbda])`

Returns

self.output [array_like] Returns [optimalparams, sd_params], where sd_params = one standard deviation errors on the parameters

Raises

RuntimeError If the fit does not converge

`loadData`(*x*, *y*, *yerror*)
Load the data into a data object

Parameters

x [array_like] Rank 1, Independent variable

y [array_like] Rank 1, Dependent variable, should be of the same shape as `x`

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror or scalar

`predict`(*x*)
Predicts the *y* values based on the fitted result.

Parameters

x [array_like] Values to predict

Returns

y [array_like] Predicted Values

`printOutput`()
Prints the output of .fit(), otherwise raises a warning

Raises

RuntimeWarning Raised if .fit() has not been run.

`class fitting.fitter.ODRFitter`(*x*, *y*, *xerror*, *yerror*, *func*)
The ODRFitter class fits the given data using scipy.odr

Parameters

x [array_like] Rank-1, Independent variable

y [array_like] Rank-1, Dependent variable, should be of the same shape as `x`

xerror [array_like or function] Rank 1, Error in x, should be of the same shape as `x` or func(x) –> xerror or scalar

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror or scalar

func [function] fcn(beta, x) –> y

Attributes

model [scipy.odr.Model Instance]

data [scipy.odr.RealData Instance]

odr [scipy.odr.ODR Instance]

output [scipy.odr.Output instance]

`fit(`*initial_params*`)`
Fit the data using the odr Model and saves the output to `self.output`

Parameters

initial_params [array_like] Represents the initial guesses. Rank 1 Array with length equal to the number of parameters defined for self.model. For w(z): Rank 1 of length 3 with `initial_params = array([w_0, z_0, M_sq_lmbda])`

Returns

self.output [scipy.odr.Output instance] This object is also assigned to the attribute .output of Fitter https://docs.scipy.org/doc/scipy/reference/generated/scipy.odr.Output.html

In particular: self.output.res_var = chi_sq_red // https://arxiv.org/abs/1012.3754 self.output.beta = Estimated parameter values self.output.sd_beta = Standard deviations of the estimated parameters self.output.info = Reason for returning, as output by ODRPACK (cf. ODRPACK UG p. 38).

`loadData(`*x*, *y*, *xerror*, *yerror*`)`
Load the data into a data object

Parameters

x [array_like] Rank 1, Independent variable

y [array_like] Rank 1, Dependent variable, should be of the same shape as `x`

xerror [array_like or function] Rank 1, Error in x, should be of the same shape as `x` or func(x) –> xerror or scalar

yerror [array_like or function] Rank 1, Error in y, should be of the same shape as `y` or func(y) –> yerror or scalar

`predict(`*x*`)`
Predicts the y values based on the fitted result.

Parameters

x [array_like] Values to predict

Returns

y [array_like] Predicted Values

`printOutput()`
Prints the output of .fit(), otherwise raises a warning

Raises

RuntimeWarning Raised if .fit() has not been run.

`class stage.controller.Controller(`*devMode=True*, *implementation=False*`)`
Abstract Base Class for a controller

`KeyboardInterruptHandler(`*signal*, *frame*`)`
Abort and close the serial port if interrupted. Handles a SIGINT according to https://docs.python.org/3/library/signal.html#signal.signal.

Parameters

signal [int] signal number

frame [signal Frame object] Frame objects represent execution frames. They may occur in traceback objects (see below), and are also passed to registered trace functions.

`abstract move(`*pos*`)`
Relative Move, to be implemented

Parameters

pos [number] Position to move to

`abstract rmove(`*delta*`)`

Relative Move, to be implemented

Parameters

delta [number] Number of steps to move

`startSignalHandlers()`

Starts appropriate signal handlers to handle e.g. keyboard interrupts. Ensures safe exit and disconnecting of controller.

`class stage.controller.GSC01(`*stage=<stage._stage.SGSP26_200 object>*, **args*, ***kwargs*`)`

Class for the GSC-01 Controller Microcontroller Model: OptoSigma GSC-01

Currently the device is to CENTRAL HOME, i.e. the origin is the center of the stage.

`abort()`

Implementation of abort as specified in the parent class

`closeDevice()`

Closes the serial device connection

`findRange()`

Find the range of the stage in number of pulses. Updates *self.stage.pulseRange* directly and returns the pulseRange.

The *self.stage.um_per_pulse* is also recalculated.

Returns

self.stage.pulseRange [int] The obtained pulse range.

`getPositionReadOut()`

Gets the position from the controller. Only for the first run, defer others to using self.stage.position

Returns

position: integer Position in integer

`getStatus1(`**args*, ***kwargs*`)`

Checks Status1

Returns

ret: array of strings Coordinate, ACK1, ACK2, ACK3 - Coordinate: Fixed Length of 10 digits including symbols. Symbols are left-aligned, coord are right aligned, the extra spaces are removed by read - ACK1: X = Command Error, K = Command Accepted normally - ACK2: L = LS Stop, K = Normal Stop - ACK3: B = Busy Status, R = Ready Status

`homeStage()`

Home the stage

Speeds: - minSpeed = 500 PPS - maxSpeed = 5000 PPS - acdcTime = 200 ms The above cannot be changed.

`isBusy(`**args*, ***kwargs*`)`

Gets operating status, labelled as status2 (B = Busy Status, R = Ready Status)

Returns

ret: bool True if Busy, False if Ready, None if output is self.read returns None

`jog(`*positive=True*, *secs=None*`)`

Starts the stage jogging.

The stage moves continousely at a preset jog speed without acceleration/deceleration until stopped Use *self.setspeed(speed, jog = True)* to set the speed. Use *self.stop(emergency = False)* to stop.

Parameters

positive [bool, optional] Whether to move in the positive direction, by default True

secs [float, optional] If given, the amount of time in seconds to jog, by default None Uses the system time, so not very accurate, use at own risk.

Returns

ret [Status] See GSC01.safesend()

`move(`*pos*`)`

Absolution move to coordinate *pos*

Parameters

pos [int] Absolute coordinate to move to (in units of pulses). Positive for moving in the positive direction, and viceversa.

Returns

ret [Status] See GSC01.safesend()

Raises

stage.errors.PositionOutOfBoundsError If proposed move moves stage out of range

`rmove(`*delta*`)`

Relative move by *delta* pulses

Parameters

delta [int] Number of pulses to move. Positive for moving in the positive direction, and viceversa.

Returns

ret [Status] See GSC01.safesend()

Raises

stage.errors.PositionOutOfBoundsError If proposed relative move moves stage out of range

`send(`*cmd*, *waitClear=False*, *raw=False*, *waitTime=0*`)`

Sends a command to the GSC-01 Controller

Parameters

cmd [Union[bytearray, str]] If `` `raw = True` `` then cmd is a `` `bytearray` `` that is directly sent to the controller. Otherwise, cmd is a string command that is encoded into ASCII before being sent to the controller.

waitClear [bool, optional] [description], by default False

raw [bool, optional] Flag for whether the input command is a bytearray or string, by default False

waitTime [float, optional] Waiting time in seconds before writing to the device, by default 0. Can be used to cool down.

Returns

output [Union[bytearray,int]] Returns 0 if `` `self.devMode = True` `` else returns the results from `` `self.read()` ``

`setSpeed(`*jogSpeed=None*, *minSpeed=None*, *maxSpeed=None*, *acdcTime=None*, *init=False*`)`

Sets the driving speed of the stage.

Set speed in units of 100 PPS. Values less than 100 PPS are rounded down. If negative values are given, the absolute values will be taken. If any illegal values are given, the original values are taken. If this results in maxspeed < minspeed, then they will be switched.

Initial Values: - jogSpeed = 500 PPS (Restart initializes this) - minSpeed = 500 PPS - maxSpeed = 5000 PPS - acdcTime = 200 ms

Parameters

jogSpeed [Optional[int], optional] The jogging speed of the stage in Pulse Per Seconds, by default None If set to None, current speed is used. Acceptables values are 100 - 20000 PPS.

minSpeed [Optional[int], optional] The minimum speed of the stage in Pulse Per Seconds, by default None If set to None, current speed is used. Acceptables values are 100 - 20000 PPS.

maxSpeed [Optional[int], optional] The maximum speed of the stage in Pulse Per Seconds, by default None If set to None, current speed is used. Acceptables values are 100 - 20000 PPS.

acdcTime [Optional[int], optional] The acceleration and deceleration time of the stage in milliseconds, by default None Acceptables values are 0 to 1000 ms. If set to None, current acceleration and deceleration time is used.

init [Optional[bool], optional] Resets the speeds to the initial values. If set to True, other parameters are ignored. By default False.

Returns

(retSpeed, retJog) [Statuses] See GSC01.safesend()

Raises

AssertionError If *minSpeed* is more than *maxSpeed*, or if *maxSpeed* is 0

TypeError If any of the values are not integers.

`stop(`*emergency=False*`)`

Decelerates the stage and stops it

Parameters

emergency [bool, optional] Set to True to use immediate stop instead of decelerate and stop, by default False

`syncPosition()`

Gets the position from the controller and syncs it to *stage.position*. To calibrate in the other direction (using the software as the source), use *self.move*.

If the stage is powered, also clears the dirty state of the stage.

Returns

pos: int Current Position of the stage

`waitClear()`

Waits for the device to be ready.

Returns

True Returns True once the controller is ready

Raises

RuntimeError If the controller does not respond

`class stage.controller.SerialController(`*devConfig=None*, **args*, ***kwargs*`)`

Abstract Base Class for a serial controller

`closeDevice()`
Closes the serial device connection

`initializeDevice()`
Initializes the serial devices and saves it into self.dev

Raises

RuntimeError Raised if unable to establish serial communication

`loadConfig(`*devConfig=None*`)`
Load the config for device communication from either a json file or a dictionary into self.cfg

Parameters

devConfig [Union[dict,str,None], optional] json file or dictionary of configuation details, by default None

Raises

RuntimeError Raised if an invalid config file is found but self.devMode = False

`class stage._stage.GSC01_Stage(`*pos=0*`)`
`positionSetter(`*x*`)`
Checks if the position to be set is within range and sets it, Should be run before executing any moves

Parameters

x [int] Position; Position must be a number between -16,777,215 and 16,777,215

Raises

stage.errors.PositionOutOfBoundsError Raised when the given x is not within the bounds set by the controller

TypeError Raised when the given x is not an integer

`recalculateUmPerPulse()`
Recalculates the um_per_pulse after setting *self.pulseRange*

self.pulseRange needs to be set beforehand.

`resetStage()`
Meant to set the upper and lower limit based on pulseRange after homing

`setLimits(`*upper*, *lower*`)`
We should not need this method, but I implement it just so that this will instantiate

Parameters

upper [int] Upper limit

lower [int] Lower limit

`class stage._stage.SGSP26_200(`**args*, ***kwargs*`)`
`resetStage()`
Meant to set the upper and lower limit based on pulseRange after homing

`class stage._stage.Stage(`*pos=0*`)`
`positionSetter(`*x*`)`
Checks if the position to be set is within range and sets it, Should be run before executing any moves

!Unsafe state: if position is dirty but the dirty flag is not set/unset.

Parameters

x [number] Position; Position must be a number between self.LIMIT_LOWER and self.LIMIT_UPPER

Raises

stage.errors.PositionOutOfBoundsError Raised when the given x is not within the bounds set by the controller

`abstract setLimits(`*upper*, *lower*`)`

Sets the lower and upper limit. Also here just so that this class may not be instantiated.

Parameters

upper [number] Upper limit

lower [number] Lower limit

Raises

ValueError Raised when upper limit is lower than lower limit

`exception stage.errors.ControllerError`

Raised when there is an error when sending commands to the controller

`exception stage.errors.PositionDirtyError`

Raised when the position that is requested is dirty

`exception stage.errors.PositionOutOfBoundsError`

Raised when the position given is out of bounds of what the controller supports

`class cameras.nanoscan.NanoScan(`*devMode=False*, **args*, ***kwargs*`)`

Provides interface to the NanoScan 2s Pyro/9/5. Naive implementation following the example codes.

`AXES`

alias of `cameras.nanoscan_constants.NsAxes`

`SetDAQ(`*state*`)`

Sets the DAQ state. Use this instead of directly using *self.NS.SetDataAcquisition*. This helps to keep track of the DAQ State.

Do not use in conjunction with Sync1Rev, it will be useless.

Parameters

state [bool] Sets the Data Acquisition to *state*

:rtype: [py:obj:*None*]

`getAxis_avg_D4Sigma(`*axis*, *numsamples=20*`)`

Get the d4sigma in one *axis* and averages it over *numsamples* using the Sync1Rev implementation.

Using NsAxes somewhat changes the signature of this function in a strict sense, but at this point I think would make easier for me to check.

Parameters

axis [NsAxes] Either *NsAxes.X* or *NsAxes.Y*, or *NsAxes.BOTH*.

Arguably using *NsAxes.BOTH* is more efficient but leads to spaghetti code in that the return type is no longer consistent.

This is a compromise I am willing to take.

numsamples [int, optional] Number of samples to average over, by default 20

Returns

ret [(float, float) or array_like of form [[float, float], [float, float]]] Returns the d4sigma of the given axis in micrometer in the form of (average, stddev) or (x, y) where each axis is given in the form of (average, stddev) If the given *axis* is not *NsAxes.X* or *NsAxes.Y* or *NsAxes.XY*, then (*None*, *None*)

:rtype: [py:data:~*typing.Tuple*[`float`, `float`]] `waitForData()`

A valid method of determining whether data has been processed yet is to evaluate whether any Results (Parameters per NS1) have yet been computed. In this example the Centroid position result is used due to its benign nature, i.e. usually enabled and not affected by other settings or results.

Reference: Program.cs from Automation examples folder from NanoScan

Returns

success [bool] Returns true when data is available

:rtype: [py:class:*bool*]

`class cameras.nanoscan.NanoScanDLL(`**args*, ***kwargs*`)`

Provides interface to the 32-bit NanoScan C# DLL using msl-loadlib.

Provides reference lookup for WinCamD

`class cameras.wincamd_constants.StrEnum(`*value*`)`

Enum where members are also (and must be) strings

Copied from https://github.com/python/cpython/blob/817a6bc9f7b802511c4d42273a621c556a48870b/Lib/enum.py#L1114

`class cameras.wincamd_constants.WinCamAxes(`*value*`)`

An enumeration.

File provides the backend for the GUI. It is meant to combine all the modules together

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

cameras.nanoscan, 11
cameras.wincamd, 12
cameras.wincamd_constants, 12

f

fitting.fit_functions, 1
fitting.fitter, 2

m

measurement.measure, 12

s

stage._stage, 10
stage.controller, 6
stage.errors, 11

INDEX

\spxentryabort()\spxextrastage.controller.GSC01 method, 7
\spxentryAXES\spxextracameras.nanoscan.NanoScan attribute, 11

\spxentrycameras.nanoscan
\spxentrymodule, 11
\spxentrycameras.wincamd
\spxentrymodule, 12
\spxentrycameras.wincamd_constants
\spxentrymodule, 12
\spxentrycloseDevice()\spxextrastage.controller.GSC01 method, 7
\spxentrycloseDevice()\spxextrastage.controller.SerialController method, 9
\spxentryController\spxextraclass in stage.controller, 6
\spxentryControllerError, 11
\spxentryconvertODRtoOCF()\spxextrain module fitting.fit_functions, 1

\spxentryestimateAndFit()\spxextrafitting.fitter.MsqOCFFitter method, 3
\spxentryestimateAndFit()\spxextrafitting.fitter.MsqODRFitter method, 4
\spxentryestimateInitialGuesses()\spxextrafitting.fitter.MsqFitter method, 2

\spxentryfindRange()\spxextrastage.controller.GSC01 method, 7
\spxentryfit()\spxextrafitting.fitter.MsqOCFFitter method, 3
\spxentryfit()\spxextrafitting.fitter.MsqODRFitter method, 4
\spxentryfit()\spxextrafitting.fitter.OCFFitter method, 4
\spxentryfit()\spxextrafitting.fitter.ODRFitter method, 6
\spxentryFitter\spxextraclass in fitting.fitter, 2
\spxentryfitting.fit_functions
\spxentrymodule, 1
\spxentryfitting.fitter
\spxentrymodule, 2

\spxentrygetAxis_avg_D4Sigma()\spxextracameras.nanoscan.NanoScan method, 11
\spxentrygetPlotOfFit()\spxextrafitting.fitter.Fitter method, 2
\spxentrygetPositionReadOut()\spxextrastage.controller.GSC01 method, 7
\spxentrygetStatus1()\spxextrastage.controller.GSC01 method, 7
\spxentryGSC01\spxextraclass in stage.controller, 7
\spxentryGSC01_Stage\spxextraclass in stage._stage, 10

\spxentryhomeStage()\spxextrastage.controller.GSC01 method, 7

\spxentryinitializeDevice()\spxextrastage.controller.SerialController method, 10
\spxentryisBusy()\spxextrastage.controller.GSC01 method, 7
\spxentryiso_omega_z()\spxextrain module fitting.fit_functions, 1

\spxentryjog()\spxextrastage.controller.GSC01 method, 7

\spxentryKeyboardInterruptHandler()\spxextrastage.controller.Controlle method, 6

\spxentryloadConfig()\spxextrastage.controller.SerialController method, 10
\spxentryloadData()\spxextrafitting.fitter.OCFFitter method, 5
\spxentryloadData()\spxextrafitting.fitter.ODRFitter method, 6

\spxentrymeasurement.measure
\spxentrymodule, 12
\spxentrymodule
\spxentrycameras.nanoscan, 11
\spxentrycameras.wincamd, 12
\spxentrycameras.wincamd_constants, 12
\spxentryfitting.fit_functions, 1
\spxentryfitting.fitter, 2
\spxentrymeasurement.measure, 12
\spxentrystage._stage, 10
\spxentrystage.controller, 6
\spxentrystage.errors, 11
\spxentrymove()\spxextrastage.controller.Controller method, 6
\spxentrymove()\spxextrastage.controller.GSC01 method, 8
\spxentryMsqFitter\spxextraclass in fitting.fitter, 2

\spxentryMsqOCFFitter\spxextraclass in fitting.fitter, 2
\spxentryMsqODRFitter\spxextraclass in fitting.fitter, 3

\spxentryNanoScan\spxextraclass in cameras.nanoscan, 11
\spxentryNanoScanDLL\spxextraclass in cameras.nanoscan, 12

\spxentryOCFFitter\spxextraclass in fitting.fitter, 4
\spxentryODRFitter\spxextraclass in fitting.fitter, 5
\spxentryomega_z()\spxextrain module fitting.fit_functions, 1
\spxentryomega_z_lambda()\spxextrain module fitting.fit_functions, 1

\spxentryPositionDirtyError, 11
\spxentryPositionOutOfBoundsError, 11
\spxentrypositionSetter()\spxextrastage._stage.GSC01_Stage method, 10
\spxentrypositionSetter()\spxextrastage._stage.Stage method, 10
\spxentrypredict()\spxextrafitting.fitter.OCFFitter method, 5
\spxentrypredict()\spxextrafitting.fitter.ODRFitter method, 6
\spxentryprintOutput()\spxextrafitting.fitter.OCFFitter method, 5
\spxentryprintOutput()\spxextrafitting.fitter.ODRFitter method, 6

\spxentryrecalculateUmPerPulse()\spxextrastage._stage.GSC01_Stage method, 10
\spxentryresetStage()\spxextrastage._stage.GSC01_Stage method, 10
\spxentryresetStage()\spxextrastage._stage.SGSP26_200 method, 10
\spxentryrmove()\spxextrastage.controller.Controller method, 7
\spxentryrmove()\spxextrastage.controller.GSC01 method, 8

\spxentrysend()\spxextrastage.controller.GSC01 method, 8
\spxentrySerialController\spxextraclass in stage.controller, 9
\spxentrySetDAQ()\spxextracameras.nanoscan.NanoScan method, 11
\spxentrysetInitialGuesses()\spxextrafitting.fitter.MsqFitter method, 2
\spxentrysetLimits()\spxextrastage._stage.GSC01_Stage method, 10
\spxentrysetLimits()\spxextrastage._stage.Stage method, 11
\spxentrysetSpeed()\spxextrastage.controller.GSC01 method, 8
\spxentrySGSP26_200\spxextraclass in stage._stage, 10
\spxentryStage\spxextraclass in stage._stage, 10
\spxentrystage._stage
\spxentrymodule, 10
\spxentrystage.controller
\spxentrymodule, 6
\spxentrystage.errors
\spxentrymodule, 11
\spxentrystartSignalHandlers()\spxextrastage.controller.Controller method, 7
\spxentrystop()\spxextrastage.controller.GSC01 method, 9
\spxentryStrEnum\spxextraclass in cameras.wincamd_constants, 12
\spxentrysyncPosition()\spxextrastage.controller.GSC01 method, 9

\spxentrywaitClear()\spxextrastage.controller.GSC01 method, 9
\spxentrywaitForData()\spxextracameras.nanoscan.NanoScan method, 11
\spxentryWinCamAxes\spxextraclass in cameras.wincamd_constants, 12