welly Documentation

Release 0.3.5

Agile Geoscience

Oct 14, 2018

Table of Contents

Welly is a family of classes to facilitate the loading, processing, and analysis of subsurface wells and well data, such as striplogs, well log curves, and synthetic seismograms.

CHAPTER 1

Requirements

- *NumPy*, which handles the numerics.
- *matplotlib*, a plotting library.
- *SciPy*, which handles curve interpolation.
- *lasio*, for reading and writing LAS files.
- *striplog*, highly recommended for helping control plotting.

CHAPTER 2

Content

Welly is a family of classes to facilitate the loading, processing, and analysis of subsurface wells and well data, such as striplogs, well log curves, and synthetic seismograms.

2.1 Requirements

- *NumPy*, which handles the numerics.
- *matplotlib*, a plotting library.
- *SciPy*, which handles curve interpolation.
- *lasio*, for reading and writing LAS files.
- *striplog*, highly recommended for helping control plotting.

2.2 welly

2.2.1 welly package

Submodules

welly.canstrat module

Functions for importing Canstrat ASCII files.

copyright 2016 Agile Geoscience

license Apache 2.0

`welly.canstrat.`**`cols`**`(`*c*`)`

`welly.canstrat.`**`interval_to_card_7`**`(`*iv*, *lith_field*`)` welly.canstrat.**well_to_card_1**(*well*)

welly.canstrat.**well_to_card_2**(*well*, *key*)

Parameters

- **well** (Well) –
- **key** (*str*) – The key of the predicted Striplog in *well.data*.

Returns dict.

welly.canstrat.**write_row**(*dictionary*, *card*, *log*)
Processes a single row from the file.

welly.canstrat_codes module

Codes for Canstrat ASCII files; only used by canstrat.py.

copyright 2016 Agile Geoscience

license Apache 2.0

welly.crs module

CRS functions. Modeled on fiona by Sean Gillies. https://github.com/Toblerity/Fiona

This version… :copyright: 2016 Agile Geoscience :license: Apache 2.0

Original code… Copyright (c) 2007, Sean C. Gillies All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 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.
- Neither the name of Sean C. Gillies 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 OWNER 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.

class welly.crs.**CRS**(**args*, ***kwargs*)
Bases: collections.abc.MutableMapping

data

classmethod from_epsg(*code*)
Given an integer code, returns an EPSG-like mapping. Note: the input code is not validated against an EPSG database.

classmethod from_string(*prjs*)
Turn a PROJ.4 string into a mapping of parameters. Bare parameters like "+no_defs" are given a value of `True`. All keys are checked against the `all_proj_keys` list.

Parameters **prjs** (*str*) – A PROJ4 string.

to_string()
Turn a CRS dict into a PROJ.4 string. Mapping keys are tested against `all_proj_keys` list. Values of `True` are omitted, leaving the key bare: {'no_defs': True} -> "+no_defs" and items where the value is otherwise not a str, int, or float are omitted.

Parameters **crs** – A CRS dict as used in Location.

Returns str. The string representation.

welly.curve module

Defines log curves.

copyright 2016 Agile Geoscience

license Apache 2.0

class welly.curve.**Curve**
Bases: `numpy.ndarray`

A fancy ndarray. Gives some utility functions, plotting, etc, for curve data.

apply(*window_length*, *samples=True*, *func1d=None*)
Runs any kind of function over a window.

Parameters

- **window_length** (*int*) – the window length. Required.
- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **func1d** (*function*) – a function that takes a 1D array and returns a scalar. Default: `np.mean()`.

Returns Curve.

basis

block(*cutoffs=None*, *values=None*, *n_bins=0*, *right=False*, *function=None*)
Block a log based on number of bins, or on cutoffs.

Parameters

- **cutoffs** (*array*) –
- **values** (*array*) – the values to map to. Defaults to [0, 1, 2,...]
- **n_bins** (*int*) –
- **right** (*bool*) –
- **function** (*function*) – transform the log if you want.

Returns Curve.

despike(*window_length=33*, *samples=True*, *z=2*)

Parameters

- **window** (*int*) – window length in samples. Default 33 (or 5 m for most curves sampled at 0.1524 m intervals).
- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **z** (*float*) – Z score

Returns Curve.

extrapolate()

From `bruges`

Extrapolate up and down an array from the first and last non-NaN samples.

E.g. Continue the first and last non-NaN values of a log up and down.

classmethod from_lasio_curve(*curve*, *depth=None*, *basis=None*, *start=None*, *stop=None*, *step=0.1524*, *run=-1*, *null=-999.25*, *service_company=None*, *date=None*)

Makes a curve object from a lasio curve object and either a depth basis or start and step information.

Parameters

- **curve** (*ndarray*) –
- **depth** (*ndarray*) –
- **basis** (*ndarray*) –
- **start** (*float*) –
- **stop** (*float*) –
- **step** (*float*) – default: 0.1524
- **run** (*int*) – default: -1
- **null** (*float*) – default: -999.25
- **service_company** (*str*) – Optional.
- **data** (*str*) – Optional.

Returns Curve. An instance of the class.

get_alias(*alias*)

Given a mnemonic, get the alias name(s) it falls under. If there aren't any, you get an empty list.

get_stats()

interpolate()

Interpolate across any missing zones.

plot(*ax=None*, *legend=None*, *return_fig=False*, ***kwargs*)

Plot a curve.

Parameters

- **ax** (*ax*) – A matplotlib axis.
- **legend** (*striplog.legend*) – A legend. Optional.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.

- **kwargs** – Arguments for `ax.set()`

Returns ax. If you passed in an ax, otherwise None.

plot_2d(*ax=None*, *width=None*, *aspect=60*, *cmap=None*, *ticks=(1*, *10)*, *return_fig=False*)
Plot a 2D curve.

Parameters

- **ax** (*ax*) – A matplotlib axis.
- **width** (*int*) – The width of the image.
- **aspect** (*int*) – The aspect ratio (not quantitative at all).
- **cmap** (*str*) – The colourmap to use.
- **ticks** (*tuple*) – The tick interval on the y-axis.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.

Returns ax. If you passed in an ax, otherwise None.

plot_kde(*ax=None*, *amax=None*, *amin=None*, *label=None*, *return_fig=False*)
Plot a KDE for the curve. Very nice summary of KDEs: https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/

Parameters

- **ax** (*axis*) – Optional matplotlib (MPL) axis to plot into. Returned.
- **amax** (*float*) – Optional max value to permit.
- **amin** (*float*) – Optional min value to permit.
- **label** (*string*) – What to put on the y-axis. Defaults to curve name.
- **return_fig** (*bool*) – If you want to return the MPL figure object.

Returns depending on what you ask for.

Return type None, axis, figure

qflag(*tests*, *alias=None*)
Run a test and return the corresponding results on a sample-by-sample basis.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

qflags(*tests*, *alias=None*)
Run a series of tests and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

quality(*tests*, *alias=None*)
Run a series of tests and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

quality_score(*tests*, *alias=None*)

Run a series of tests and return the normalized score. 1.0: Passed all tests. (0-1): Passed a fraction of tests. 0.0: Passed no tests. -1.0: Took no tests.

Parameters **tests** (*list*) – a list of functions.

Returns float. The fraction of tests passed, or -1 for 'took no tests'.

read_at (*d*, ***kwargs*)

Read the log at a specific depth or an array of depths.

Parameters

- **d** (*float or array-like*) –
- **interpolation** (*str*) –
- **index** (*bool*) –
- **return_basis** (*bool*) –

Returns float or ndarray.

smooth (*window_length*, *samples=True*, *func1d=None*)

Runs any kind of function over a window.

Parameters

- **window_length** (*int*) – the window length. Required.
- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **func1d** (*function*) – a function that takes a 1D array and returns a scalar. Default: `np.mean()`.

Returns Curve.

stop

to_basis (*basis=None*, *start=None*, *stop=None*, *step=None*, *undefined=None*)

Make a new curve in a new basis, given a basis, or a new start, step, and/or stop. You only need to set the parameters you want to change. If the new extents go beyond the current extents, the curve is padded with the `undefined` parameter.

Parameters

- **basis** (*ndarray*) –
- **start** (*float*) –
- **stop** (*float*) –
- **step** (*float*) –
- **undefined** (*float*) –

Returns Curve. The current instance in the new basis.

to_basis_like (*basis*)

Make a new curve in a new basis, given an existing one. Wraps `to_basis()`.

Pass in a curve or the basis of a curve.

Parameters **basis** (*ndarray*) – A basis, but can also be a Curve instance.

Returns Curve. The current instance in the new basis.

exception welly.curve.**CurveError**

Bases: `Exception`

Generic error class.

welly.defaults module

Defines some default values.

copyright 2016 Agile Geoscience

license Apache 2.0

welly.fields module

Field mapping from welly to LAS.

copyright 2016 Agile Geoscience

license Apache 2.0

welly.header module

Defines well headers.

copyright 2016 Agile Geoscience

license Apache 2.0

class `welly.header.`**`Header`**(*params*)

Bases: `object`

The well metadata or header information.

Not the same as an LAS header, but we might get info from there.

classmethod from_csv(*csv_file*)

Not implemented. Will provide a route from CSV file.

classmethod from_lasio(*l*, *remap=None*, *funcs=None*)

Assumes we're starting with a lasio object, l.

Parameters

- **l** (*lasio*) – A lasio instance.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

welly.location module

Defines well location.

copyright 2016 Agile Geoscience

license Apache 2.0

class `welly.location.`**`Location`**(*params*)

Bases: `object`

Contains all location and spatial information.

add_deviation(*dev*, *td=None*)

Add a deviation survey to this instance, and try to compute a position log from it.

compute_position_log (*td=None*, *method='mc'*, *update_deviation=True*)

Parameters

- **deviation** (*ndarray*) – A deviation survey with rows like MD, INC, AZI
- **td** (*Number*) – The TD of the well, if not the end of the deviation survey you're passing.
- **method** (*str*) – 'aa': average angle 'bt': balanced tangential 'mc': minimum curvature
- **update_deviation** – This function makes some adjustments to the dev- iation survey, to account for the surface and TD. If you do not want to change the stored deviation survey, set to False.

Returns ndarray. A position log with rows like X-offset, Y-offset, Z-offset

crs_from_epsg (*epsg*)

Sets the CRS using an EPSG code.

Parameters **epsg** (*int*) – The EPSG code.

Returns None.

crs_from_string (*string*)

Sets the CRS using a PROJ4 string.

Parameters **string** (*int*) – The PROJ4 string, eg '+init=epsg:4269 +no_defs'.

Returns None.

classmethod from_lasio (*l*, *remap=None*, *funcs=None*)

Make a Location object from a lasio object. Assumes we're starting with a lasio object, l.

Parameters

- **l** (*lasio*) –
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns Location. An instance of this class.

md

The measured depth of the deviation survey.

Returns ndarray.

md2tvd

Provides an transformation and interpolation function that converts MD to TVD.

Parameters **kind** (*str*) – The kind of interpolation to do, e.g. 'linear', 'cubic', 'nearest'.

Returns function.

tvd

The true vertical depth of the deviation survey.

Returns ndarray.

tvd2md

Provides an transformation and interpolation function that converts MD to TVD.

Parameters **kind** (*str*) – The kind of interpolation to do, e.g. 'linear', 'cubic', 'nearest'.

Returns function.

welly.project module

Defines a multi-well 'project'.

copyright 2016 Agile Geoscience

license Apache 2.0

class `welly.project.`**`Project`**(*list_of_Wells*, *source=''*)

Bases: `object`

Just a list of Well objects.

One day it might want its own CRS, but then we'd have to cast the CRSs of the contained data.

`add_canstrat_striplogs`(*path*, *uwi_transform=None*, *name='canstrat'*)

This may be too specific a method... just move it to the workflow.

Requires striplog.

`count_mnemonic`(*mnemonic*, *uwis=<property object>*, *alias=None*)

Counts the wells that have a given curve, given the mnemonic and an alias dict.

`curve_table_html`(*uwis=None*, *keys=None*, *alias=None*, *tests=None*, *exclude=None*, *limit=0*)

Another version of the curve table.

Parameters

- **uwis** (*`list`*) – Only these UWIs. List of `str`.
- **keys** (*`list`*) – Only these names. List of `str`.
- **alias** (*`dict`*) – Alias table, maps names to mnemomnics in order of preference.
- **tests** (*`dict`*) – Test table, maps names to lists of functions.
- **exclude** (*`list`*) – Except these names. List of `str`. Ignored if you pass `keys`.
- **limit** (*`int`*) – Curve must be present in at least this many wells.

Returns str. HTML representation of the table.

`data_as_matrix`(*X_keys*, *y_key=None*, *alias=None*, *legend=None*, *match_only=None*, *field=None*, *field_function=None*, *table=None*, *legend_field=None*, *basis=None*, *step=None*, *window_length=None*, *window_step=1*, *test=None*, *remove_zeros=False*, *include_basis=False*, *include_index=False*, *include=None*, *complete_only=False*)

`find_wells_with_curve`(*mnemonic*, *alias=None*)

Returns a new Project with only the wells which have the named curve.

classmethod `from_las`(*path=None*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *max=None*, *encoding=None*, *printfname=None*)

Constructor. Essentially just wraps `Well.from_las()`, but is more convenient for most purposes.

Parameters

- **path** (*`str`*) – The path of the LAS files, e.g. `./*.las` (the default). It will attempt to load everything it finds, so make sure it only leads to LAS files.
- **remap** (*`dict`*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*`dict`*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **data** (*`bool`*) – Whether to load curves or not.

- **req** (*list*) – A list of alias names, giving all required curves. If not all of the aliases are present, the well is not loaded.
- **alias** (*dict*) – The alias dict, e.g. `alias = {'gamma': ['GR', 'GR1'], 'density': ['RHOZ', 'RHOB'], 'pants': ['PANTS']}`

Returns project. The project object.

get_mnemonics (*mnemonics*, *uwis=None*, *alias=None*)

Looks at all the wells in turn and returns the highest thing in the alias table.

Parameters

- **mnemonics** (*list*) –
- **alias** (*dict*) –

Returns list. A list of lists.

get_wells (*uwis=None*)

next ()

Retains Python 2 compatibility.

plot_kdes (*mnemonic*, *alias=None*, *uwi_regex=None*, *return_fig=False*)

Plot KDEs for all curves with the given name.

pop (*index*)

uwis

welly.quality module

Quality functions for welly.

copyright 2016 Agile Geoscience

license Apache 2.0

welly.quality.**all_above** (*value*)

welly.quality.**all_below** (*value*)

welly.quality.**all_between** (*lower*, *upper*)

welly.quality.**all_positive** (*curve*)

Define it this way to avoid NaN problem.

welly.quality.**check_units** (*list_of_units*)

welly.quality.**count_spikes** (*curve*)

welly.quality.**fraction_not_nans** (*curve*)

Returns the fraction of the curve extents that are good (non-nan data).

welly.quality.**fraction_not_zeros** (*curve*)

Returns the fraction of the curve extents that are not zeros.

welly.quality.**fraction_within_range** (*xmin*, *xmax*)

welly.quality.**mean_above** (*value*)

welly.quality.**mean_below** (*value*)

welly.quality.**mean_between** (*lower*, *upper*) welly.quality.**no_flat**(*curve*)

welly.quality.**no_gaps**(*curve*)
Check for gaps, after ignoring any NaNs at the top and bottom.

welly.quality.**no_monotonic**(*curve*)

welly.quality.**no_nans**(*curve*)
Check for NaNs anywhere at all in the curve, even the top or bottom.

welly.quality.**no_similarities**(*well*, *keys*, *alias*)

welly.quality.**no_spikes**(*tolerance*)
Arg `tolerance` is the number of spiky samples allowed.

welly.quality.**not_empty**(*curve*)

welly.quality.**spike_locations**(*curve*)
Return the indicies of the spikes.

welly.scales module

Custom scales for matplotlib.

copyright 2016 Joe Kington

Note: For the two scales, I've set the bounds such that you can never go beyond a set range. This gives "stretchy" panning when you reach the ends of a well. Sometimes you'll want it, sometimes you won't. In a lot of cases (e.g. multiple wells or flattening on a marker, etc) you'll want to be able to go beyond the limits of the well. In that case, remove the "limit_range_for_scale" methods below (and BoundedScale entirely) and use an interpolation function that allows extrapolation beyond the limits of the input data.

class welly.scales.**BoundedScale**(*axis*, *vmin=None*, *vmax=None*)
Bases: `matplotlib.scale.LinearScale`

Linear scale with set bounds that can't be exceeded. Gives a "stretchy" panning effect.

limit_range_for_scale(*vmin*, *vmax*, *minpos*)
Returns the range *vmin*, *vmax*, possibly limited to the domain supported by this scale.

minpos **should be the minimum positive value in the data.** This is used by log scales to determine a minimum value.

name = 'bounded'

class welly.scales.**PiecewiseLinearScale**(*axis*, *x=None*, *y=None*)
Bases: `matplotlib.scale.ScaleBase`

Scale based on a piecewise-linear transformation. For example, this might be used to show ticks in two-way time alongside a well log plotted in measured depth using a time-depth curve.

get_transform()
Return the `Transform` object associated with this scale.

limit_range_for_scale(*vmin*, *vmax*, *minpos*)
Returns the range *vmin*, *vmax*, possibly limited to the domain supported by this scale.

minpos **should be the minimum positive value in the data.** This is used by log scales to determine a minimum value.

name = 'piecewise'

set_default_locators_and_formatters(*axis*)

Set the `Locator` and `Formatter` objects on the given axis to match this scale.

class welly.scales.**PiecewiseLinearTransform**(*x_from*, *y_to*)

Bases: `matplotlib.transforms.Transform`

Transform between two coordinate systems by interpolating between a pre-calculated set of points. For example, transform between time and depth using an average velocity curve.

has_inverse = True

input_dims = 1

inverted()

Return the corresponding inverse transformation.

The return value of this method should be treated as temporary. An update to *self* does not cause a corresponding update to its inverted copy.

`x === self.inverted().transform(self.transform(x))`

is_separable = True

output_dims = 1

transform_non_affine(*x*)

Performs only the non-affine part of the transformation.

`transform(values)` is always equivalent to `transform_affine(transform_non_affine(values))`.

In non-affine transformations, this is generally equivalent to `transform(values)`. In affine transformations, this is always a no-op.

Accepts a numpy array of shape (N x `input_dims`) and returns a numpy array of shape (N x `output_dims`).

Alternatively, accepts a numpy array of length `input_dims` and returns a numpy array of length `output_dims`.

welly.synthetic module

Defines a synthetic seismogram.

copyright 2016 Agile Geoscience

license Apache 2.0

class welly.synthetic.**Synthetic**

Bases: `numpy.ndarray`

Synthetic seismograms.

as_curve(*start=None*, *stop=None*)

Get the synthetic as a Curve, in depth. Facilitates plotting along- side other curve data.

basis

Compute basis rather than storing it.

plot(*ax=None*, *return_fig=False*, ***kwargs*)

Plot a synthetic.

Parameters

- **ax** (*ax*) – A matplotlib axis.

- **legend** (*Legend*) – For now, only here to match API for other plot methods.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.

Returns ax. If you passed in an ax, otherwise None.

stop
Compute stop rather than storing it.

welly.tools module

Some extra bits.

copyright 2016 Agile Geoscience

license Apache 2.0

class welly.tools.**RGBLog**(*curves*)
Bases: object

Attempt at RGB. Incomplete.

welly.utils module

Utility functions for welly.

copyright 2016 Agile Geoscience

license Apache 2.0

class welly.utils.**Linker**(*axes*)
Bases: object

Keeps y-limits of a sequence of axes in sync when panning/zooming.

By Joe Kington

link(*ax*)

rescale(*axes*)

unlink(*ax*)

welly.utils.**are_close**(*x, y*)

welly.utils.**dd2dms**(*dd*)
Decimal degrees to DMS.

Parameters **dd** (*float*) –

Returns tuple. Degrees, minutes, and seconds.

welly.utils.**dms2dd**(*dms*)
DMS to decimal degrees.

Parameters **dms** (*list*) –

Returns float.

welly.utils.**extrapolate**(*a*)
From bruges

Extrapolate up and down an array from the first and last non-NaN samples.

E.g. Continue the first and last non-NaN values of a log up and down.

`welly.utils.`**`find_edges`**`(`*a*`)`

Return two arrays: one of the changes, and one of the values.

Returns Two ndarrays, tops and values.

Return type tuple

`welly.utils.`**`find_file`**`(`*pattern*, *path*`)`

A bit like grep. Finds a pattern, looking in path. Returns the filename.

`welly.utils.`**`find_nearest`**`(`*a*, *value*, *index=False*`)`

Find the array value, or index of the array value, closest to some given value.

Parameters

- **a** (*ndarray*) –
- **value** (*float*) –
- **index** (*bool*) – whether to return the index instead of the array value.

Returns float. The array value (or index, as int) nearest the specified value.

`welly.utils.`**`find_previous`**`(`*a*, *value*, *index=False*, *return_distance=False*`)`

Find the nearest array value, or index of the array value, before some given value. Optionally also return the fractional distance of the given value from that previous value.

Parameters

- **a** (*ndarray*) –
- **value** (*float*) –
- **index** (*bool*) – whether to return the index instead of the array value. Default: False.
- **return_distance** (*bool*) – whether to return the fractional distance from the nearest value to the specified value. Default: False.

Returns

float. The array value (or index, as int) before the specified value. If `return_distance==True` then a tuple is returned, where the second value is the distance.

`welly.utils.`**`fix_ticks`**`(`*ax*`)`

Center ticklabels and hide any outside axes limits.

By Joe Kington

`welly.utils.`**`flatten_list`**`(`*l*`)`

Unpacks lists in a list:

[1, 2, [3, 4], [5, [6, 7]]]

becomes

[1, 2, 3, 4, 5, 6, 7]

http://stackoverflow.com/a/12472564/3381305

`welly.utils.`**`get_lines`**`(`*handle*, *line*`)`

Get zero-indexed line from an open file-like.

`welly.utils.`**`hex_is_dark`**`(`*hexx*, *percent=50*`)`

Function to decide if a hex colour is dark.

Parameters **hexx** (*str*) – A hexadecimal colour, starting with '#'.

Returns The colour's brightness is less than the given percent.

Return type bool

`welly.utils.`**`hex_to_rgb`**(*hexx*)

Utility function to convert hex to (r,g,b) triples. http://ageo.co/1CFxXpO

Parameters **hexx** (*str*) – A hexadecimal colour, starting with '#'.

Returns The equivalent RGB triple, in the range 0 to 255.

Return type tuple

`welly.utils.`**`lasio_get`**(*l*, *section*, *item*, *attrib='value'*, *default=None*, *remap=None*, *funcs=None*)

Grabs, renames and transforms stuff from a lasio object.

Parameters

- **l** (*lasio*) – a lasio instance.
- **section** (*str*) – The LAS section to grab from, eg `well`
- **item** (*str*) – The item in the LAS section to grab from, eg `name`
- **attrib** (*str*) – The attribute of the item to grab, eg `value`
- **default** (*str*) – What to return instead.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns The transformed item.

`welly.utils.`**`linear`**(*u*, *v*, *d*)

Linear interpolation. :param u: :type u: float :param v: :type v: float :param d: the relative distance between the two to return. :type d: float

Returns float. The interpolated value.

`welly.utils.`**`list_and_add`**(*a*, *b*)

Concatenate anything into a list.

Parameters

- **a** – the first thing
- **b** – the second thing

Returns list. All the things in a list.

`welly.utils.`**`moving_average`**(*a*, *length*, *mode='valid'*)

From `bruges`

Computes the mean in a moving window. Naive implementation.

Example

```
>>> test = np.array([1,9,9,9,9,9,9,2,3,9,2,2,3,1,1,1,1,3,4,9,9,9,8,3])
>>> moving_average(test, 7, mode='same')
[ 4.42857143,  5.57142857,  6.71428571,  7.85714286,  8.         ,
7.14285714,  7.14285714,  6.14285714,  5.14285714,  4.28571429,
```

(continues on next page)

(continued from previous page)

```
3.14285714,  3.        ,  2.71428571,  1.57142857,  1.71428571,
2.        ,  2.85714286,  4.        ,  5.14285714,  6.14285714,
6.42857143,  6.42857143,  6.28571429,  5.42857143]
```

Todo: Other types of average.

`welly.utils.`**`moving_avg_conv`**(*a*, *length*)

From `bruges`

Moving average via convolution. Seems slower than naive.

`welly.utils.`**`nan_idx`**(*y*)

Helper to handle indices and logical indices of NaNs.

From https://stackoverflow.com/questions/6518811/interpolate-nan-values-in-a-numpy-array

Parameters **y** (*ndarray*) – 1D array with possible NaNs

Returns

nans, logical indices of NaNs index, a function, with signature indices= index(logical_indices),

to convert logical indices of NaNs to 'equivalent' indices

Example

```
>>> # linear interpolation of NaNs
>>> nans, x= nan_helper(y)
>>> y[nans]= np.interp(x(nans), x(~nans), y[~nans])
```

`welly.utils.`**`normalize`**(*a*, *new_min=0.0*, *new_max=1.0*)

From `bruges`

Normalize an array to [0,1] or to arbitrary new min and max.

Parameters

- **a** (*ndarray*) –
- **new_min** (*float*) – the new min, default 0.
- **new_max** (*float*) – the new max, default 1.

Returns ndarray. The normalized array.

`welly.utils.`**`null`**(*x*)

Null function. Used for default in functions that can apply a user- supplied function to data before returning.

`welly.utils.`**`null_default`**(*x*)

Null function. Used for default in functions that can apply a user- supplied function to data before returning.

`welly.utils.`**`parabolic`**(*f*, *x*)

Interpolation. From ageobot, from somewhere else.

`welly.utils.`**`ricker`**(*f*, *length*, *dt*)

A Ricker wavelet.

Parameters

- **f** (*float*) – frequency in Haz, e.g. 25 Hz.

- **length** (*float*) – Length in s, e.g. 0.128.
- **dt** (*float*) – sample interval in s, e.g. 0.001.

Returns tuple. time basis, amplitude values.

`welly.utils.`**`rms`**(*a*)

From `bruges`

Calculates the RMS of an array.

Parameters **a** – An array.

Returns The RMS of the array.

`welly.utils.`**`round_to_n`**(*x*, *n*)

Round to sig figs

`welly.utils.`**`sharey`**(*axes*)

Shared axes limits without shared locators, ticks, etc.

By Joe Kington

`welly.utils.`**`skip`**(*x*)

Always returns None.

`welly.utils.`**`text_colour_for_hex`**(*hexx*, *percent=50*, *dark='#000000'*, *light='#ffffff'*)

Function to decide what colour to use for a given hex colour.

Parameters **hexx** (*str*) – A hexadecimal colour, starting with '#'.

Returns The colour's brightness is less than the given percent.

Return type bool

`welly.utils.`**`top_and_tail`**(**arrays*)

From `bruges`

Top and tail all arrays to the non-NaN extent of the first array.

E.g. crop the NaNs from the top and tail of a well log.

`welly.utils.`**`unsharey`**(*ax*)

Remove sharing from an axes.

By Joe Kington

welly.well module

Defines wells.

copyright 2016 Agile Geoscience

license Apache 2.0

class `welly.well.`**`Well`**(*params*)

Bases: `object`

Well contains everything about the well.

`add_curves_from_las`(*fname*, *remap=None*, *funcs=None*)

Given a LAS file, add curves from it to the current well instance. Essentially just wraps `add_curves_from_lasio()`.

Parameters

- **fname** (*str*) – The path of the LAS file to read curves from.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns None. Works in place.

add_curves_from_lasio(*l*, *remap=None*, *funcs=None*)

Given a LAS file, add curves from it to the current well instance. Essentially just wraps `add_curves_from_lasio()`.

Parameters

- **fname** (*str*) – The path of the LAS file to read curves from.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns None. Works in place.

alias_has_multiple(*mnemonic*, *alias*)

count_curves(*keys=None*, *alias=None*)

Counts the number of curves in the well that will be selected with the given key list and the given alias dict. Used by Project's curve table.

data_as_matrix(*keys=None*, *return_basis=False*, *basis=None*, *start=None*, *stop=None*, *step=None*, *window_length=None*, *window_step=1*, *alias=None*)

Provide a feature matrix, given a list of data items.

I think this will probably fail if there are striplogs in the data dictionary for this well.

Todo: Deal with striplogs and other data, if present.

Parameters

- **keys** (*list*) – List of the logs to export from the data dictionary.
- **return_basis** (*bool*) – Whether or not to return the basis that was used.
- **basis** (*ndarray*) – The basis to use. Default is to survey all curves to find a common basis.
- **start** (*float*) – Optionally override the start of whatever basis you find or (more likely) is surveyed.
- **stop** (*float*) – Optionally override the stop of whatever basis you find or (more likely) is surveyed.
- **window** (*int*) – The number of samples to return around each sample.
- **step** (*float*) – Override the step in the basis from survey_basis.

df()

Just use lasio's df().

classmethod from_las(*fname*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *encoding=None*, *printfname=False*)

Constructor. Essentially just wraps `from_lasio()`, but is more convenient for most purposes.

Parameters

- **fname** (*str*) – The path of the LAS file.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **printfname** (*bool*) – prints filename before trying to load it, for debugging

Returns well. The well object.

classmethod from_lasio(*l*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *fname=None*)

Constructor. If you already have the lasio object, then this makes a well object from it.

Parameters

- **l** (*lasio object*) – a lasio object.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **data** (*bool*) – Whether to load curves or not.
- **req** (*dict*) – An alias list, giving all required curves. If not all of the aliases are present, the well is empty.

Returns well. The well object.

get_curve(*mnemonic*, *alias=None*)

Wraps get_mnemonic.

Instead of picking curves by name directly from the data dict, you can pick them up with this method, which takes account of the alias dict you pass it. If you do not pass an alias dict, then you get the curve you asked for, if it exists, or None. NB Wells do not have alias dicts, but Projects do.

Parameters

- **mnemonic** (*str*) – the name of the curve you want.
- **alias** (*dict*) – an alias dictionary, like welly.

Returns Curve.

get_mnemonic(*mnemonic*, *alias=None*)

Instead of picking curves by name directly from the data dict, you can pick them up with this method, which takes account of the alias dict you pass it. If you do not pass an alias dict, then you get the curve you asked for, if it exists, or None. NB Wells do not have alias dicts, but Projects do.

Parameters

- **mnemonic** (*str*) – the name of the curve you want.
- **alias** (*dict*) – an alias dictionary, like welly.

Returns Curve.

get_mnemonics_from_regex(*pattern*)

Should probably integrate getting curves with regex, vs getting with aliases, even though mixing them is probably confusing. For now I can't think of another use case for these wildcards, so I'll just implement for the curve table and we can worry about a nice solution later if we ever come back to it.

is_complete (*keys=None*, *alias=None*)

Returns False if the well does not have one or more of the keys in its data dictionary. Used by project.data_to_matrix().

make_synthetic (*srd=0*, *v_repl_seismic=2000*, *v_repl_log=2000*, *f=50*, *dt=0.001*)

Early hack. Use with extreme caution.

Hands-free. There'll be a more granualr version in synthetic.py.

Assumes DT is in µs/m and RHOB is kg/m3.

There is no handling yet for TVD.

The datum handling is probably sketchy.

Todo: A lot.

plot (*legend=None*, *tracks=None*, *track_titles=None*, *alias=None*, *basis=None*, *return_fig=False*, *extents='td'*, ***kwargs*)

Plot multiple tracks.

Parameters

- **legend** (*striplog.legend*) – A legend instance.
- **tracks** (*list*) – A list of strings and/or lists of strings. The tracks you want to plot from `data`. Optional, but you will usually want to give it.
- **track_titles** (*list*) – Optional. A list of strings and/or lists of strings. The names to give the tracks, if you don't want welly to guess.
- **basis** (*ndarray*) – Optional. The basis of the plot, if you don't want welly to guess (probably the best idea).
- **return_fig** (*bool*) – Whether to return the matplotlig figure. Default False.
- **extents** (*str*) – What to use for the y limits: 'td' — plot 0 to TD. 'curves' — use a basis that accommodates all the curves. 'all' — use a basis that accommodates everything. (tuple) — give the upper and lower explictly.

Returns None. The plot is a side-effect.

qc_curve_group (*tests*, *alias=None*)

Run tests on a cohort of curves.

qc_data (*tests*, *alias=None*)

Run a series of tests against the data and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

qc_table_html (*tests*, *alias=None*)

Makes a nice table out of `qc_data()`

survey_basis (*keys=None*, *alias=None*, *step=None*)

Look at the basis of all the curves in the `well.data` and return a basis with the minimum start, maximum depth, and minimum step.

Parameters **keys** (*list*) – List of strings: the keys of the data items to survey, if not all of them.

Returns ndarray. The 'most complete common basis'.

to_canstrat (*key*, *log*, *lith_field*, *filename=None*, *as_text=False*)

Make a Canstrat DAT (aka ASCII) file.

Todo: The data part should probably belong to striplog, and only the header should be written by the well.

Parameters

- **filename** (*str*) –
- **key** (*str*) –
- **log** (*str*) – the log name, should be 6 characters.
- **lith_field** (*str*) – Primary component. Must match the Canstrat definitions.
- **filename** –
- **as_text** (*bool*) – if you don't want to write a file.

to_las (*fname*, *basis=None*, *keys=None*)

Writes the current well instance as a LAS file. Essentially just wraps `to_lasio()`, but is more convenient for most purposes.

Parameters

- **fname** (*str*) – The path of the LAS file to create.
- **basis** (*ndarray*) – Optional. The basis to export the curves in. If you don't specify one, it will survey all the curves with `survey_basis()`.
- **keys** (*list*) – List of strings: the keys of the data items to include, if not all of them. You can have nested lists, such as you might use for `tracks` in `well.plot()`.

Returns None. Writes the file as a side-effect.

to_lasio (*basis=None*, *keys=None*)

Makes a lasio object from the current well.

Parameters

- **basis** (*ndarray*) – Optional. The basis to export the curves in. If you don't specify one, it will survey all the curves with `survey_basis()`.
- **keys** (*list*) – List of strings: the keys of the data items to include, if not all of them. You can have nested lists, such as you might use for `tracks` in `well.plot()`.

Returns lasio. The lasio object.

unify_basis (*keys=None*, *basis=None*)

Give everything, or everything in the list of keys, the same basis. If you don't provide a basis, welly will try to get one using *survey_basis()*.

Parameters

- **basis** (*ndarray*) – A basis: the regularly sampled depths at which you want the samples.
- **keys** (*list*) – List of strings: the keys of the data items to unify, if not all of them.

Returns None. Works in place.

uwi
Property. Simply a shortcut to the UWI from the header, or the empty string if there isn't one.

exception welly.well.**WellError**
Bases: `Exception`

Generic error class.

Module contents

welly

class welly.**Project**(*list_of_Wells*, *source=''*)
Bases: `object`

Just a list of Well objects.

One day it might want its own CRS, but then we'd have to cast the CRSs of the contained data.

add_canstrat_striplogs(*path*, *uwi_transform=None*, *name='canstrat'*)
This may be too specific a method… just move it to the workflow.

Requires striplog.

count_mnemonic(*mnemonic*, *uwis=<property object>*, *alias=None*)
Counts the wells that have a given curve, given the mnemonic and an alias dict.

curve_table_html(*uwis=None*, *keys=None*, *alias=None*, *tests=None*, *exclude=None*, *limit=0*)
Another version of the curve table.

Parameters

- **uwis** (*list*) – Only these UWIs. List of `str`.
- **keys** (*list*) – Only these names. List of `str`.
- **alias** (*dict*) – Alias table, maps names to mnemomnics in order of preference.
- **tests** (*dict*) – Test table, maps names to lists of functions.
- **exclude** (*list*) – Except these names. List of `str`. Ignored if you pass `keys`.
- **limit** (*int*) – Curve must be present in at least this many wells.

Returns str. HTML representation of the table.

data_as_matrix(*X_keys*, *y_key=None*, *alias=None*, *legend=None*, *match_only=None*, *field=None*, *field_function=None*, *table=None*, *legend_field=None*, *basis=None*, *step=None*, *window_length=None*, *window_step=1*, *test=None*, *remove_zeros=False*, *include_basis=False*, *include_index=False*, *include=None*, *complete_only=False*)

find_wells_with_curve(*mnemonic*, *alias=None*)
Returns a new Project with only the wells which have the named curve.

classmethod from_las(*path=None*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *max=None*, *encoding=None*, *printfname=None*)
Constructor. Essentially just wraps `Well.from_las()`, but is more convenient for most purposes.

Parameters

- **path** (*str*) – The path of the LAS files, e.g. `./*.las` (the default). It will attempt to load everything it finds, so make sure it only leads to LAS files.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.

- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **data** (*bool*) – Whether to load curves or not.
- **req** (*list*) – A list of alias names, giving all required curves. If not all of the aliases are present, the well is not loaded.
- **alias** (*dict*) – The alias dict, e.g. `alias = {'gamma': ['GR', 'GR1'], 'density': ['RHOZ', 'RHOB'], 'pants': ['PANTS']}`

Returns project. The project object.

get_mnemonics (*mnemonics*, *uwis=None*, *alias=None*)
Looks at all the wells in turn and returns the highest thing in the alias table.

Parameters

- **mnemonics** (*list*) –
- **alias** (*dict*) –

Returns list. A list of lists.

get_wells (*uwis=None*)

next ()
Retains Python 2 compatibility.

plot_kdes (*mnemonic*, *alias=None*, *uwi_regex=None*, *return_fig=False*)
Plot KDEs for all curves with the given name.

pop (*index*)

uwis

class welly.**Well** (*params*)
Bases: `object`

Well contains everything about the well.

add_curves_from_las (*fname*, *remap=None*, *funcs=None*)
Given a LAS file, add curves from it to the current well instance. Essentially just wraps `add_curves_from_lasio()`.

Parameters

- **fname** (*str*) – The path of the LAS file to read curves from.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns None. Works in place.

add_curves_from_lasio (*l*, *remap=None*, *funcs=None*)
Given a LAS file, add curves from it to the current well instance. Essentially just wraps `add_curves_from_lasio()`.

Parameters

- **fname** (*str*) – The path of the LAS file to read curves from.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.

- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns None. Works in place.

alias_has_multiple(*mnemonic*, *alias*)

count_curves(*keys=None*, *alias=None*)

Counts the number of curves in the well that will be selected with the given key list and the given alias dict. Used by Project's curve table.

data_as_matrix(*keys=None*, *return_basis=False*, *basis=None*, *start=None*, *stop=None*, *step=None*, *window_length=None*, *window_step=1*, *alias=None*)

Provide a feature matrix, given a list of data items.

I think this will probably fail if there are striplogs in the data dictionary for this well.

Todo: Deal with striplogs and other data, if present.

Parameters

- **keys** (*list*) – List of the logs to export from the data dictionary.
- **return_basis** (*bool*) – Whether or not to return the basis that was used.
- **basis** (*ndarray*) – The basis to use. Default is to survey all curves to find a common basis.
- **start** (*float*) – Optionally override the start of whatever basis you find or (more likely) is surveyed.
- **stop** (*float*) – Optionally override the stop of whatever basis you find or (more likely) is surveyed.
- **window** (*int*) – The number of samples to return around each sample.
- **step** (*float*) – Override the step in the basis from survey_basis.

df()

Just use lasio's df().

classmethod from_las(*fname*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *encoding=None*, *printfname=False*)

Constructor. Essentially just wraps `from_lasio()`, but is more convenient for most purposes.

Parameters

- **fname** (*str*) – The path of the LAS file.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **printfname** (*bool*) – prints filename before trying to load it, for debugging

Returns well. The well object.

classmethod from_lasio(*l*, *remap=None*, *funcs=None*, *data=True*, *req=None*, *alias=None*, *fname=None*)

Constructor. If you already have the lasio object, then this makes a well object from it.

Parameters

- **l** (*lasio object*) – a lasio object.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.
- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.
- **data** (*bool*) – Whether to load curves or not.
- **req** (*dict*) – An alias list, giving all required curves. If not all of the aliases are present, the well is empty.

Returns well. The well object.

get_curve (*mnemonic*, *alias=None*)

Wraps get_mnemonic.

Instead of picking curves by name directly from the data dict, you can pick them up with this method, which takes account of the alias dict you pass it. If you do not pass an alias dict, then you get the curve you asked for, if it exists, or None. NB Wells do not have alias dicts, but Projects do.

Parameters

- **mnemonic** (*str*) – the name of the curve you want.
- **alias** (*dict*) – an alias dictionary, like welly.

Returns Curve.

get_mnemonic (*mnemonic*, *alias=None*)

Instead of picking curves by name directly from the data dict, you can pick them up with this method, which takes account of the alias dict you pass it. If you do not pass an alias dict, then you get the curve you asked for, if it exists, or None. NB Wells do not have alias dicts, but Projects do.

Parameters

- **mnemonic** (*str*) – the name of the curve you want.
- **alias** (*dict*) – an alias dictionary, like welly.

Returns Curve.

get_mnemonics_from_regex (*pattern*)

Should probably integrate getting curves with regex, vs getting with aliases, even though mixing them is probably confusing. For now I can't think of another use case for these wildcards, so I'll just implement for the curve table and we can worry about a nice solution later if we ever come back to it.

is_complete (*keys=None*, *alias=None*)

Returns False if the well does not have one or more of the keys in its data dictionary. Used by project.data_to_matrix().

make_synthetic (*srd=0*, *v_repl_seismic=2000*, *v_repl_log=2000*, *f=50*, *dt=0.001*)

Early hack. Use with extreme caution.

Hands-free. There'll be a more granualr version in synthetic.py.

Assumes DT is in µs/m and RHOB is kg/m3.

There is no handling yet for TVD.

The datum handling is probably sketchy.

Todo: A lot.

plot (*legend=None*, *tracks=None*, *track_titles=None*, *alias=None*, *basis=None*, *return_fig=False*, *extents='td'*, ***kwargs*)

Plot multiple tracks.

Parameters

- **legend** (*striplog.legend*) – A legend instance.
- **tracks** (*list*) – A list of strings and/or lists of strings. The tracks you want to plot from `data`. Optional, but you will usually want to give it.
- **track_titles** (*list*) – Optional. A list of strings and/or lists of strings. The names to give the tracks, if you don't want welly to guess.
- **basis** (*ndarray*) – Optional. The basis of the plot, if you don't want welly to guess (probably the best idea).
- **return_fig** (*bool*) – Whether to return the matplotlig figure. Default False.
- **extents** (*str*) – What to use for the y limits: 'td' — plot 0 to TD. 'curves' — use a basis that accommodates all the curves. 'all' — use a basis that accommodates everything. (tuple) — give the upper and lower explictly.

Returns None. The plot is a side-effect.

qc_curve_group (*tests*, *alias=None*)

Run tests on a cohort of curves.

qc_data (*tests*, *alias=None*)

Run a series of tests against the data and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

qc_table_html (*tests*, *alias=None*)

Makes a nice table out of `qc_data()`

survey_basis (*keys=None*, *alias=None*, *step=None*)

Look at the basis of all the curves in the `well.data` and return a basis with the minimum start, maximum depth, and minimum step.

Parameters **keys** (*list*) – List of strings: the keys of the data items to survey, if not all of them.

Returns ndarray. The 'most complete common basis'.

to_canstrat (*key*, *log*, *lith_field*, *filename=None*, *as_text=False*)

Make a Canstrat DAT (aka ASCII) file.

Todo: The data part should probably belong to striplog, and only the header should be written by the well.

Parameters

- **filename** (*str*) –
- **key** (*str*) –
- **log** (*str*) – the log name, should be 6 characters.
- **lith_field** (*str*) – Primary component. Must match the Canstrat definitions.
- **filename** –

- **as_text** (*bool*) – if you don't want to write a file.

to_las (*fname*, *basis=None*, *keys=None*)
Writes the current well instance as a LAS file. Essentially just wraps `to_lasio()`, but is more convenient for most purposes.

Parameters

- **fname** (*str*) – The path of the LAS file to create.
- **basis** (*ndarray*) – Optional. The basis to export the curves in. If you don't specify one, it will survey all the curves with `survey_basis()`.
- **keys** (*list*) – List of strings: the keys of the data items to include, if not all of them. You can have nested lists, such as you might use for `tracks` in `well.plot()`.

Returns None. Writes the file as a side-effect.

to_lasio (*basis=None*, *keys=None*)
Makes a lasio object from the current well.

Parameters

- **basis** (*ndarray*) – Optional. The basis to export the curves in. If you don't specify one, it will survey all the curves with `survey_basis()`.
- **keys** (*list*) – List of strings: the keys of the data items to include, if not all of them. You can have nested lists, such as you might use for `tracks` in `well.plot()`.

Returns lasio. The lasio object.

unify_basis (*keys=None*, *basis=None*)
Give everything, or everything in the list of keys, the same basis. If you don't provide a basis, welly will try to get one using *survey_basis()*.

Parameters

- **basis** (*ndarray*) – A basis: the regularly sampled depths at which you want the samples.
- **keys** (*list*) – List of strings: the keys of the data items to unify, if not all of them.

Returns None. Works in place.

uwi
Property. Simply a shortcut to the UWI from the header, or the empty string if there isn't one.

class welly.**Header** (*params*)
Bases: `object`

The well metadata or header information.

Not the same as an LAS header, but we might get info from there.

classmethod from_csv (*csv_file*)
Not implemented. Will provide a route from CSV file.

classmethod from_lasio (*l*, *remap=None*, *funcs=None*)
Assumes we're starting with a lasio object, l.

Parameters

- **l** (*lasio*) – A lasio instance.
- **remap** (*dict*) – Optional. A dict of 'old': 'new' LAS field names.

- **funcs** (*dict*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

class `welly.`**`Curve`**

Bases: `numpy.ndarray`

A fancy ndarray. Gives some utility functions, plotting, etc, for curve data.

`apply`(*window_length*, *samples=True*, *func1d=None*)

Runs any kind of function over a window.

Parameters

- **window_length** (*int*) – the window length. Required.
- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **func1d** (*function*) – a function that takes a 1D array and returns a scalar. Default: `np.mean()`.

Returns Curve.

`basis`

`block`(*cutoffs=None*, *values=None*, *n_bins=0*, *right=False*, *function=None*)

Block a log based on number of bins, or on cutoffs.

Parameters

- **cutoffs** (*array*) –
- **values** (*array*) – the values to map to. Defaults to [0, 1, 2,…]
- **n_bins** (*int*) –
- **right** (*bool*) –
- **function** (*function*) – transform the log if you want.

Returns Curve.

`despike`(*window_length=33*, *samples=True*, *z=2*)

Parameters

- **window** (*int*) – window length in samples. Default 33 (or 5 m for most curves sampled at 0.1524 m intervals).
- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **z** (*float*) – Z score

Returns Curve.

`extrapolate`()

From `bruges`

Extrapolate up and down an array from the first and last non-NaN samples.

E.g. Continue the first and last non-NaN values of a log up and down.

classmethod **`from_lasio_curve`**(*curve*, *depth=None*, *basis=None*, *start=None*, *stop=None*, *step=0.1524*, *run=-1*, *null=-999.25*, *service_company=None*, *date=None*)

Makes a curve object from a lasio curve object and either a depth basis or start and step information.

Parameters

- **curve** (*ndarray*) –
- **depth** (*ndarray*) –
- **basis** (*ndarray*) –
- **start** (*float*) –
- **stop** (*float*) –
- **step** (*float*) – default: 0.1524
- **run** (*int*) – default: -1
- **null** (*float*) – default: -999.25
- **service_company** (*str*) – Optional.
- **data** (*str*) – Optional.

Returns Curve. An instance of the class.

get_alias(*alias*)
Given a mnemonic, get the alias name(s) it falls under. If there aren't any, you get an empty list.

get_stats()

interpolate()
Interpolate across any missing zones.

plot(*ax=None*, *legend=None*, *return_fig=False*, ***kwargs*)
Plot a curve.

Parameters

- **ax** (*ax*) – A matplotlib axis.
- **legend** (*striplog.legend*) – A legend. Optional.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.
- **kwargs** – Arguments for `ax.set()`

Returns ax. If you passed in an ax, otherwise None.

plot_2d(*ax=None*, *width=None*, *aspect=60*, *cmap=None*, *ticks=(1, 10)*, *return_fig=False*)
Plot a 2D curve.

Parameters

- **ax** (*ax*) – A matplotlib axis.
- **width** (*int*) – The width of the image.
- **aspect** (*int*) – The aspect ratio (not quantitative at all).
- **cmap** (*str*) – The colourmap to use.
- **ticks** (*tuple*) – The tick interval on the y-axis.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.

Returns ax. If you passed in an ax, otherwise None.

plot_kde(*ax=None*, *amax=None*, *amin=None*, *label=None*, *return_fig=False*)
Plot a KDE for the curve. Very nice summary of KDEs: https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/

Parameters

- **ax** (*axis*) – Optional matplotlib (MPL) axis to plot into. Returned.
- **amax** (*float*) – Optional max value to permit.
- **amin** (*float*) – Optional min value to permit.
- **label** (*string*) – What to put on the y-axis. Defaults to curve name.
- **return_fig** (*bool*) – If you want to return the MPL figure object.

Returns depending on what you ask for.

Return type None, axis, figure

qflag (*tests*, *alias=None*)
Run a test and return the corresponding results on a sample-by-sample basis.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

qflags (*tests*, *alias=None*)
Run a series of tests and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

quality (*tests*, *alias=None*)
Run a series of tests and return the corresponding results.

Parameters **tests** (*list*) – a list of functions.

Returns list. The results. Stick to booleans (True = pass) or ints.

quality_score (*tests*, *alias=None*)

Run a series of tests and return the normalized score. 1.0: Passed all tests. (0-1): Passed a fraction of tests. 0.0: Passed no tests. -1.0: Took no tests.

Parameters **tests** (*list*) – a list of functions.

Returns float. The fraction of tests passed, or -1 for ‘took no tests’.

read_at (*d*, ***kwargs*)
Read the log at a specific depth or an array of depths.

Parameters

- **d** (*float or array-like*) –
- **interpolation** (*str*) –
- **index** (*bool*) –
- **return_basis** (*bool*) –

Returns float or ndarray.

smooth (*window_length*, *samples=True*, *func1d=None*)
Runs any kind of function over a window.

Parameters

- **window_length** (*int*) – the window length. Required.

- **samples** (*bool*) – window length is in samples. Use False for a window length given in metres.
- **func1d** (*function*) – a function that takes a 1D array and returns a scalar. Default: `np.mean()`.

Returns Curve.

stop

to_basis (*basis=None*, *start=None*, *stop=None*, *step=None*, *undefined=None*)

Make a new curve in a new basis, given a basis, or a new start, step, and/or stop. You only need to set the parameters you want to change. If the new extents go beyond the current extents, the curve is padded with the `undefined` parameter.

Parameters

- **basis** (*ndarray*) –
- **start** (*float*) –
- **stop** (*float*) –
- **step** (*float*) –
- **undefined** (*float*) –

Returns Curve. The current instance in the new basis.

to_basis_like (*basis*)

Make a new curve in a new basis, given an existing one. Wraps `to_basis()`.

Pass in a curve or the basis of a curve.

Parameters basis (*ndarray*) – A basis, but can also be a Curve instance.

Returns Curve. The current instance in the new basis.

class welly.**Synthetic**

Bases: `numpy.ndarray`

Synthetic seismograms.

as_curve (*start=None*, *stop=None*)

Get the synthetic as a Curve, in depth. Facilitates plotting along- side other curve data.

basis

Compute basis rather than storing it.

plot (*ax=None*, *return_fig=False*, ***kwargs*)

Plot a synthetic.

Parameters

- **ax** (*ax*) – A matplotlib axis.
- **legend** (*Legend*) – For now, only here to match API for other plot methods.
- **return_fig** (*bool*) – whether to return the matplotlib figure. Default False.

Returns ax. If you passed in an ax, otherwise None.

stop

Compute stop rather than storing it.

class `welly.`**`Location`**(*params*)

Bases: `object`

Contains all location and spatial information.

`add_deviation`(*dev*, *td=None*)

Add a deviation survey to this instance, and try to compute a position log from it.

`compute_position_log`(*td=None*, *method='mc'*, *update_deviation=True*)

Parameters

- **`deviation`** (*`ndarray`*) – A deviation survey with rows like MD, INC, AZI
- **`td`** (*`Number`*) – The TD of the well, if not the end of the deviation survey you're passing.
- **`method`** (*`str`*) – 'aa': average angle 'bt': balanced tangential 'mc': minimum curvature
- **`update_deviation`** – This function makes some adjustments to the dev- iation survey, to account for the surface and TD. If you do not want to change the stored deviation survey, set to False.

Returns ndarray. A position log with rows like X-offset, Y-offset, Z-offset

`crs_from_epsg`(*epsg*)

Sets the CRS using an EPSG code.

Parameters **`epsg`** (*`int`*) – The EPSG code.

Returns None.

`crs_from_string`(*string*)

Sets the CRS using a PROJ4 string.

Parameters **`string`** (*`int`*) – The PROJ4 string, eg '+init=epsg:4269 +no_defs'.

Returns None.

`classmethod from_lasio`(*l*, *remap=None*, *funcs=None*)

Make a Location object from a lasio object. Assumes we're starting with a lasio object, l.

Parameters

- **`l`** (*`lasio`*) –
- **`remap`** (*`dict`*) – Optional. A dict of 'old': 'new' LAS field names.
- **`funcs`** (*`dict`*) – Optional. A dict of 'las field': function() for implementing a transform before loading. Can be a lambda.

Returns Location. An instance of this class.

`md`

The measured depth of the deviation survey.

Returns ndarray.

`md2tvd`

Provides an transformation and interpolation function that converts MD to TVD.

Parameters **`kind`** (*`str`*) – The kind of interpolation to do, e.g. 'linear', 'cubic', 'nearest'.

Returns function.

`tvd`

The true vertical depth of the deviation survey.

Returns ndarray.

tvd2md

Provides an transformation and interpolation function that converts MD to TVD.

Parameters **kind** (*str*) – The kind of interpolation to do, e.g. 'linear', 'cubic', 'nearest'.

Returns function.

class welly.**CRS**(**args*, ***kwargs*)

Bases: `collections.abc.MutableMapping`

data

classmethod from_epsg(*code*)

Given an integer code, returns an EPSG-like mapping. Note: the input code is not validated against an EPSG database.

classmethod from_string(*prjs*)

Turn a PROJ.4 string into a mapping of parameters. Bare parameters like "+no_defs" are given a value of `True`. All keys are checked against the `all_proj_keys` list.

Parameters **prjs** (*str*) – A PROJ4 string.

to_string()

Turn a CRS dict into a PROJ.4 string. Mapping keys are tested against `all_proj_keys` list. Values of `True` are omitted, leaving the key bare: {'no_defs': True} -> "+no_defs" and items where the value is otherwise not a str, int, or float are omitted.

Parameters **crs** – A CRS dict as used in Location.

Returns str. The string representation.

2.3 Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index