# Table of Contents

1 Introduction .............................. 1
2 Requirements ............................... 3
3 Content .................................. 5
   3.1 Introduction ........................... 5
   3.2 Requirements .......................... 5
   3.3 welly ................................ 5
   3.4 Indices and tables .................... 39
4 Indices and tables ......................... 41
Python Module Index ......................... 43
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 2

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 3

3.1 Introduction

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.

3.2 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.

3.3 welly

3.3.1 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
classmethod `from_epsg`

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

classmethod `from_string`

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

```python
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\( (\text{window\_length}=33, \text{samples}=\text{True}, z=2) \)

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.

exfopulate()

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\( (\text{curve}, \text{depth}={\text{None}}, \text{basis}={\text{None}}, \text{start}={\text{None}}, \text{stop}={\text{None}}, \text{step}=0.1524, \text{run}=-1, \text{null}=-999.25, \text{service\_company}={\text{None}}, \text{date}={\text{None}}) \)

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\( (\text{alias}) \)

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\( (\text{ax}={\text{None}}, \text{legend}={\text{None}}, \text{return\_fig}={\text{False}}, **\text{kwargs}) \)

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.

• **alias**(dict) – a dictionary mapping mnemonics to lists of mnemonics.

**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.

• **alias**(dict) – a dictionary mapping mnemonics to lists of mnemonics.

**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.

• **alias** *(dict)* – a dictionary mapping mnemonics to lists of mnemonics.

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

```python
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.

• **alias** *(dict)* – a dictionary mapping mnemonics to lists of mnemonics.

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

```python
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.

```python
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.

```python
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().

  **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.

```python
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 deviation 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
```

```python
crs_from_epsg(epsg)
    Sets the CRS using an EPSG code.

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

    Returns None.
```

```python
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.
```

```python
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 mnemonics 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)

df()

Makes a pandas DataFrame containing Curve data for all the wells in the Project. The DataFrame has a dual index of well UWI and curve Depths.

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 indices 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 \((v_{\text{min}}, v_{\text{max}}, \text{minpos})\)

Returns the range \(v_{\text{min}}, v_{\text{max}}\), 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 \((v_{\text{min}}, v_{\text{max}}, \text{minpos})\)

Returns the range \(v_{\text{min}}, v_{\text{max}}\), 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 \((x_{\text{from}}, y_{\text{to}})\)

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.

\[ x = \text{self}.\text{inverted}().\text{transform}(\text{self}.\text{transform}(x)) \]

is_separable = True

output_dims = 1

transform_non_affine \((x)\)

Performs only the non-affine part of the transformation.

\[ \text{transform}(\text{values}) \text{ is always equivalent to } \text{transform_affine}(\text{transform_non_affine}(\text{values})). \]

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

Accepts a numpy array of shape \((N x \text{input_dims})\) and returns a numpy array of shape \((N x \text{output_dims})\).

Alternatively, accepts a numpy array of length \text{input_dims} and returns a numpy array of length \text{output_dims}. 
welly.synthetic module

Defines a synthetic seismogram.

```python
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.

```python
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.

```python
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)
two to return. :type d: float

Returns

float. The interpolated value.
welly.utils.\texttt{list\_and\_add}(a, b)
Concatenate anything into a list.

\begin{itemize}
\item \textbf{Parameters}
\begin{itemize}
\item \texttt{a} -- the first thing
\item \texttt{b} -- the second thing
\end{itemize}
\end{itemize}

\begin{itemize}
\item \textbf{Returns} list. All the things in a list.
\end{itemize}

welly.utils.\texttt{moving\_average}(a, length, mode='valid')
From bruges
Computes the mean in a moving window. Naive implementation.

\textbf{Example}

\begin{verbatim}
>>> test = np.array([1,9,9,9,9,9,2,3,9,2,2,3,1,1,1,1,3,4,9,9,9,8,3])
>>> 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,
 3.14285714, 3. , 2.14285714, 1.57142857, 1.71428571,
 2. , 2.85714286, 4. , 5.14285714, 6.14285714,
 6.42857143, 6.42857143, 6.28571429, 5.42857143]
\end{verbatim}

\begin{itemize}
\item \textbf{Todo}: Other types of average.
\end{itemize}

welly.utils.\texttt{moving\_avg\_conv}(a, length)
From bruges
Moving average via convolution. Seems slower than naive.

welly.utils.\texttt{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

\begin{itemize}
\item \textbf{Parameters} \texttt{y} (\texttt{ndarray}) -- 1D array with possible NaNs
\end{itemize}

\begin{itemize}
\item \textbf{Returns}
\begin{itemize}
\item \texttt{nans}, logical indices of NaNs index, a function, with signature indices= index(logical_indices),
\item to convert logical indices of NaNs to ‘equivalent’ indices
\end{itemize}
\end{itemize}

\textbf{Example}

\begin{verbatim}
>>> # linear interpolation of NaNs
>>> nans, x= nan_helper(y)
>>> y[nans]= np.interp(x[nans], x[-nans], y[-nans])
\end{verbatim}

welly.utils.\texttt{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.

\begin{itemize}
\item \textbf{Parameters}
\end{itemize}
• 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, alias=None, start=None, stop=None, step=None, window_length=None, window_step=1)
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.
- **alias**(dict) – A mapping of alias names to lists of mnemonics.
- **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.
- **step**(float) – Override the step in the basis from survey_basis.
- **window_length**(int) – The number of samples to return around each sample. This will provide one or more shifted versions of the features.
- **window_step**(int) – How much to step the offset versions.

Returns ndarray.

```python
def (keys=None, basis=None, uwi=False)
    Return current curve data as a pandas.DataFrame object.
    Everything has to have the same basis, because the depth is going to become the index of the DataFrame. If you don’t provide one, welly will make one using survey_basis().

Parameters

- **keys**(list) – List of strings: the keys of the data items to survey, if not all of them.
- **basis**(array) – A basis, if you want to enforce one, otherwise you’ll get the result of survey_basis().
- **uwi**(bool) – Whether to add a ‘UWI’ column.

Returns pandas.DataFrame.
```

```python
classmethod from_las (fname=None, 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.
```

```python
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, mapping mnemonics to lists of mnemonics.

**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, mapping mnemonics to lists of mnemonics.

**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(*sr0=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.
- **alias (dict)** – a dictionary mapping mnemonics to lists of mnemonics.
- **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 matplotilig 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 explicitly.

Returns None. The plot is a side-effect.

qc_curve_group (tests, alias=None)
Run tests on a cohort of curves.

Parameters **alias (dict)** – an alias dictionary, mapping mnemonics to lists of mnemonics.

Returns dict.

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().

Returns str. An HTML string.

survey_basis (keys=None, alias=None, step=None)
Look at the basis of all the curves in 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.
- **alias (dict)** – a dictionary mapping mnemonics to lists of mnemonics.
- **step (float)** – a new step, if you want to change it.

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, keys=None, basis=None)*
Write 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** *(keys=None, basis=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 mnemonics 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)

df()  

Makes a pandas DataFrame containing Curve data for all the wells in the Project. The DataFrame has a dual index of well UWI and curve Depths.

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, alias=None, start=None, stop=None, step=None, window_length=None, window_step=1)
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.
• alias (dict) – A mapping of alias names to lists of mnemonics.
• 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.
• step (float) – Override the step in the basis from survey_basis.
• window_length (int) – The number of samples to return around each sample. This will provide one or more shifted versions of the features.
• window_step (int) – How much to step the offset versions.

Returns ndarray.

df (keys=None, basis=None, uwi=False)
Return current curve data as a pandas.DataFrame object.
Everything has to have the same basis, because the depth is going to become the index of the DataFrame. If you don’t provide one, welly will make one using survey_basis().

Parameters
• keys (list) – List of strings: the keys of the data items to survey, if not all of them.
• basis (array) – A basis, if you want to enforce one, otherwise you’ll get the result of survey_basis().
• uwi (bool) – Whether to add a ‘UWI’ column.

Returns pandas.DataFrame.

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, mapping mnemonics to lists of mnemonics.

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, mapping mnemonics to lists of mnemonics.

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 granular 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.
- **alias**(dict) – a dictionary mapping mnemonics to lists of mnemonics.
- **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 matplotlib 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 explicitly.

Returns None. The plot is a side-effect.

cq_curve_group(tests, alias=None)
Run tests on a cohort of curves.

Parameters **alias**(dict) – an alias dictionary, mapping mnemonics to lists of mnemonics.

Returns dict.

cq_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.

cq_table_html(tests, alias=None)
Makes a nice table out of cq_data()

Returns str. An HTML string.

cq_table_html(tests, alias=None)

survey_basis(keys=None, alias=None, step=None)
Look at the basis of all the curves in well.data and return a basis with the minimum start, maximum depth, and minimum step.

Parameters
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, keys=None, basis=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 (keys=None, basis=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, well 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.

```python
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.
```

```python
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.
Retrun Curve.

despike\( \text{\(window\_length\)=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\( \text{\(curve\), \(depth\)=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\( \text{\(alias\)}\)

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\( \text{\(ax\)=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
```
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
```
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/](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
```
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.
- `alias` (dict) – a dictionary mapping mnemonics to lists of mnemonics.

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

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

**Parameters**

- `tests` (list) – a list of functions.
- `alias` (dict) – a dictionary mapping mnemonics to lists of mnemonics.

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

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

- **tests (list)** – a list of functions.
- **alias (dict)** – a dictionary mapping mnemonics to lists of mnemonics.

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.
- **alias (dict)** – a dictionary mapping mnemonics to lists of mnemonics.

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 alongside 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 deviation 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.

### 3.4 Indices and tables

- genindex
- modindex
- search
Indices and tables

- genindex
- modindex
- search
W
welly, 27
welly.canstrat, 5
welly.canstrat_codes, 6
welly.crs, 6
welly.curve, 7
welly.defaults, 11
welly.fields, 11
welly.header, 11
welly.location, 12
welly.project, 13
welly.quality, 14
welly.scales, 15
welly.synthetic, 17
welly.tools, 17
welly.utils, 17
welly.well, 22
Index

A
add_canstrat_striplogs()  (welly.Project method), 27
add_canstrat_striplogs()  (welly.project.Project method), 13
add_curves_from_las()  (welly.Well method), 28
add_curves_from_las()  (welly.well.Well method), 22
add_curves_from_lasio()  (welly.Well method), 28
add_curves_from_lasio()  (welly.well.Well method), 22
add_deviation()  (welly.Location method), 37
add_deviation()  (welly.location.Location method), 12
alias_has_multiple()  (welly.Well method), 28
alias_has_multiple()  (welly.well.Well method), 22
all_above()  (in module welly.quality), 15
all_below()  (in module welly.quality), 15
all_between()  (in module welly.quality), 15
all_positive()  (in module welly.quality), 15
apply()  (welly.Curve method), 33
apply()  (welly.curve.Curve method), 7
are_close()  (in module welly.utils), 18
as_curve()  (welly.synthetic.Synthetic method), 37
as_curve()  (welly.Synthetic method), 33

B
basis  (welly.Curve attribute), 33
basis  (welly.curve.Curve attribute), 7
basis  (welly.Synthetic attribute), 37
basis  (welly.synthetic.Synthetic attribute), 17
block()  (welly.Curve method), 33
block()  (welly.curve.Curve method), 7
BoundedScale  (class in welly.scales), 15

C
cols()  (in module welly.canstrat), 5
count_curves()  (welly.Well method), 28
count_curves()  (welly.well.Well method), 22
count_spikes()  (in module welly.quality), 15
crs_from_epsg()  (welly.Location method), 37
crs_from_epsg()  (welly.location.Location method), 12
crs_from_string()  (welly.Location method), 38
crs_from_string()  (welly.location.Location method), 12
curve_table_html()  (welly.Project method), 27
curve_table_html()  (welly.project.Project method), 13
CurveError, 11
data  (welly.CRS attribute), 38
data  (welly.crs.CRS attribute), 6
data_as_matrix()  (welly.Project method), 27
data_as_matrix()  (welly.project.Project method), 13
data_as_matrix()  (welly.Well method), 29
data_as_matrix()  (welly.well.Well method), 22
dd2dms()  (in module welly.utils), 18
despike()  (welly.Curve method), 34
despike()  (welly.curve.Curve method), 8
df()  (welly.Project method), 27
df()  (welly.project.Project method), 14
df()  (welly.Well method), 29
df() (welly.well.Well method), 23

dms2dd() (in module welly.utils), 18

E
extrapolate() (in module welly.utils), 18
extrapolate() (welly.Curve method), 34
extrapolate() (welly.curve.Curve method), 8

F
find_edges() (in module welly.utils), 18
find_file() (in module welly.utils), 18
find_nearest() (in module welly.utils), 18
find_previous() (in module welly.utils), 18
find_wells_with_curve() (welly.Project method), 27
find_wells_with_curve() (welly.project.Project method), 14
fix_ticks() (in module welly.utils), 19
flatten_list() (in module welly.utils), 19
fraction_not_nans() (in module welly.quality), 15
fraction_not_zeros() (in module welly.quality), 15
fraction_within_range() (in module welly.quality), 15
from_csv() (welly.Header class method), 33
from_csv() (welly.header.Header class method), 11
from_epsg() (welly.CRS class method), 38
from_epsg() (welly.crs.CRS class method), 7
from_las() (welly.Project class method), 27
from_las() (welly.project.Project method), 14
from_las() (welly.Well class method), 29
from_las() (welly.well.Well class method), 23
from_lasio() (welly.well.Well class method), 33
from_lasio() (welly.header.Header class method), 11
from_lasio() (welly.Location class method), 38
from_lasio() (welly.location.Location class method), 12
from_lasio() (welly.well.Well class method), 30
from_lasio() (welly.well.Well class method), 30
from_lasio_curve() (welly.Curve class method), 34
from_lasio_curve() (welly.curve.Curve class method), 8
from_string() (welly.CRS class method), 38
from_string() (welly.crs.CRS class method), 7

G
get_alias() (welly.Curve method), 34
get_alias() (welly.curve.Curve method), 8
get_curve() (welly.Well method), 30
get_curve() (welly.well.Well method), 24
get_lines() (in module welly.utils), 19
get_mnemonic() (welly.Well method), 30
get_mnemonic() (welly.well.Well method), 24
get_mnemonics() (welly.Project method), 28
get_mnemonics() (welly.project.Project method), 14
get_mnemonics() (welly.well.Well method), 30
get_mnemonics_from_regex() (welly.well.Well method), 24
get_stats() (welly.Curve method), 34
get_stats() (welly.curve.Curve method), 8
get_transform() (welly.scales.PiecewiseLinearScale method), 16
get_wells() (welly.Project method), 28
get_wells() (welly.project.Project method), 14

H
has_inverse (welly.scales.PiecewiseLinearTransform attribute), 16
Header (class in welly), 33
Header (class in welly.header), 11
hex_is_dark() (in module welly.utils), 19
hex_to_rgb() (in module welly.utils), 19

I
input_dims (welly.scales.PiecewiseLinearTransform attribute), 16
interpolate() (welly.Curve method), 34
interpolate() (welly.curve.Curve method), 8
interval_to_card_7() (in module welly.canstrat), 6
inverted() (welly.scales.PiecewiseLinearTransform method), 16
is_complete() (welly.Well method), 30
is_complete() (welly.well.Well method), 24
is_separable (welly.scales.PiecewiseLinearTransform attribute), 16

L
lasio_get() (in module welly.utils), 19
limit_range_for_scale() (welly.scales.BoundedScale method), 15
limit_range_for_scale() (welly.scales.PiecewiseLinearScale method), 16
linear() (in module welly.utils), 19
link() (welly.utils.Linker method), 17
Linker (class in welly.utils), 17
list_and_add() (in module welly.utils), 19
Location (class in welly), 37
Location (class in welly.location), 12

M
make_synthetic() (welly.Well method), 30
make_synthetic() (welly.well.Well method), 24
md (welly.Location attribute), 38
md (welly.location.Location attribute), 12
md2tvd (welly.Location attribute), 38
md2tvd (welly.location.Location attribute), 13
mean_above() (in module welly.quality), 15
mean_below() (in module welly.quality), 15
mean_between() (in module welly.quality), 15
moving_average() (in module welly.utils), 20
moving_avg_conv() (in module welly.utils), 20

N
name (welly.scales.BoundedScale attribute), 16
name (welly.scales.PiecewiseLinearScale attribute), 16
nan_idx() (in module welly.utils), 20
next() (welly.Project method), 28
next() (welly.project.Project method), 14
no_flat() (in module welly.quality), 15
no_gaps() (in module welly.quality), 15
no_monotonic() (in module welly.quality), 15
no_nans() (in module welly.quality), 15
no_similarities() (in module welly.quality), 15
no_spikes() (in module welly.quality), 15
normalize() (in module welly.utils), 20
not_empty() (in module welly.quality), 15
null() (in module welly.utils), 21
null_default() (in module welly.utils), 21

O
output_dims (welly.scales.PiecewiseLinearTransform attribute), 16

P
parabolic() (in module welly.utils), 21
PiecewiseLinearScale (class in welly.scales), 16
PiecewiseLinearTransform (class in welly.scales), 16
plot() (welly.Curve method), 34
plot() (welly.curve.Curve method), 8
plot() (welly.synthetic.Synthetic method), 37
plot() (welly.synthetic.Synthetic method), 17
plot() (welly.Well method), 31
plot() (welly.well.Well method), 24
plot_2d() (welly.Curve method), 35
plot_2d() (welly.curve.Curve method), 9
plot_kde() (welly.Curve method), 35
plot_kde() (welly.curve.Curve method), 9
plot_kdes() (welly.Project method), 28
plot_kdes() (welly.project.Project method), 14
pop() (welly.Project method), 28
pop() (welly.project.Project method), 14
Project (class in welly), 27
Project (class in welly.project), 13

Q
qc_curve_group() (welly.Well method), 31
qc_curve_group() (welly.well.Well method), 25
qc_data() (welly.Well method), 31
qc_data() (welly.well.Well method), 25
qc_table_html() (welly.Well method), 31
qc_table_html() (welly.well.Well method), 25
qflag() (welly.Curve method), 35
qflag() (welly.curve.Curve method), 9
qflags() (welly.Curve method), 35
qflags() (welly.curve.Curve method), 9
quality() (welly.Curve method), 35
quality() (welly.curve.Curve method), 9
quality_score() (welly.Curve method), 36
quality_score() (welly.curve.Curve method), 10

R
read_at() (welly.Curve method), 36
read_at() (welly.curve.Curve method), 10
rescale() (welly.utils.Linker method), 18
RGBLog (class in welly.tools), 17
ricker() (in module welly.utils), 21
rms() (in module welly.utils), 21
round_to_n() (in module welly.utils), 21

S
set_default_locators_and_formatters() (welly.scales.PiecewiseLinearScale method), 16
sharey() (in module welly.utils), 21
skip() (in module welly.utils), 21
smooth() (welly.Curve method), 36
smooth() (welly.curve.Curve method), 10
spike_locations() (in module welly.quality), 15
stop() (welly.Curve attribute), 36
stop() (welly.curve.Curve attribute), 10
stop() (welly.synthetic.Synthetic attribute), 37
stop() (welly.synthetic.Synthetic attribute), 17
survey_basis() (welly.Well method), 31
survey_basis() (welly.well.Well method), 25
Synthetic (class in welly), 37
Synthetic (class in welly.synthetic), 17

T
text_colour_for_hex() (in module welly.utils), 21
to_basis() (welly.Curve method), 36
to_basis() (welly.curve.Curve method), 10
to_basis() (welly.project.Project method), 14
to_basis_like() (welly.Curve method), 37
to_basis_like() (welly.curve.Curve method), 11
to_canstrat() (welly.Well method), 32
to_canstrat() (welly.well.Well method), 25
to_las() (welly.Well method), 32
to_las() (welly.well.Well method), 26

transform_non_affine()
  (welly.scales.PiecewiseLinearTransform method), 16

tvd (welly.Location attribute), 38
tvd (welly.location.Location attribute), 13
tvd2md (welly.Location attribute), 38
tvd2md (welly.location.Location attribute), 13

unify_basis() (welly.Well method), 32

unlink() (welly.utils.Linker method), 18

uwi (welly.well.Well attribute), 26

uwis (welly.well.Well attribute), 26

uwis (welly.project.Project attribute), 28

Well (class in welly), 28

Well (class in welly.well), 22

well_to_card_1() (in module welly.canstrat), 6

WellError, 26

welly (module), 27

welly.canstrat (module), 5

welly.canstrat_codes (module), 6

welly.crs (module), 6

welly.curve (module), 7

welly.defaults (module), 11

welly.fields (module), 11

welly.header (module), 11

welly.location (module), 12

welly.project (module), 13

welly.quality (module), 14

welly.scales (module), 15

welly.synthetic (module), 17

welly.tools (module), 17

welly.utils (module), 17

welly.well (module), 22

write_row() (in module welly.canstrat), 6