lerp Documentation

Release

ER

Oct 30, 2017

Contents

1 Installation 3

2 License 5

3 Content 7

4 Indices and tables 31

Python Module Index 33

`lerp` aims to supply lookup table facility in python on top of numpy.

This project is in early alpha phase.

Documentation (and example of use): http://lerp.readthedocs.io/

Source code repository (and issue tracker): https://github.com/gwin-zegal/lerp

CHAPTER 1

Installation

To install the development version:

```
pip install git+https://github.com/gwin-zegal/lerp
```

Update with:

```
pip install --upgrade --no-deps git+https://github.com/gwin-zegal/lerp
```

CHAPTER 2

License

MIT – see the file `LICENSE` for details.

CHAPTER 3

Content

In mathematics, linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points.[1]

```
In [1]: %matplotlib inline
        %watermark -dtvmp numpy,matplotlib,pandas,cython

        import matplotlib.pylab as plt
        import lerp
        from lerp import *
```

```
2017-10-12 22:57:48

CPython 3.6.2
IPython 6.1.0

numpy 1.13.1
matplotlib 2.0.2
pandas 0.20.3
cython 0.26.1

compiler   : GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)
system     : Darwin
release    : 15.6.0
machine    : x86_64
processor  : i386
CPU cores  : 2
interpreter: 64bit
```

```
In [2]: lerp.options.display.max_rows = 15
```

3.1 Usage

3.1.1 BreakPoints

```
In [3]: A = BreakPoints(d=[1.040, 1.051, 1.057, 1.063, 1.064, 1.067, 1.068, 1.068, 1.068, 1.066, 1.06
                           1.060, 1.056, 1.050, 1.042, 1.032], label="Ballistic coefficient", unit="G1")
```

Display in the notebook, integer above the value are helps for indexing purpose

```
In [4]: A
```

```
Out[4]: BreakPoints(d=[ 1.04,  1.05,  1.06,  1.06,  1.06,  1.07,  1.07,  1.07,  1.07,
                        1.07,  1.06,  1.06,  1.06,  1.05,  1.04,  1.03], label="Ballistic coefficient",
```

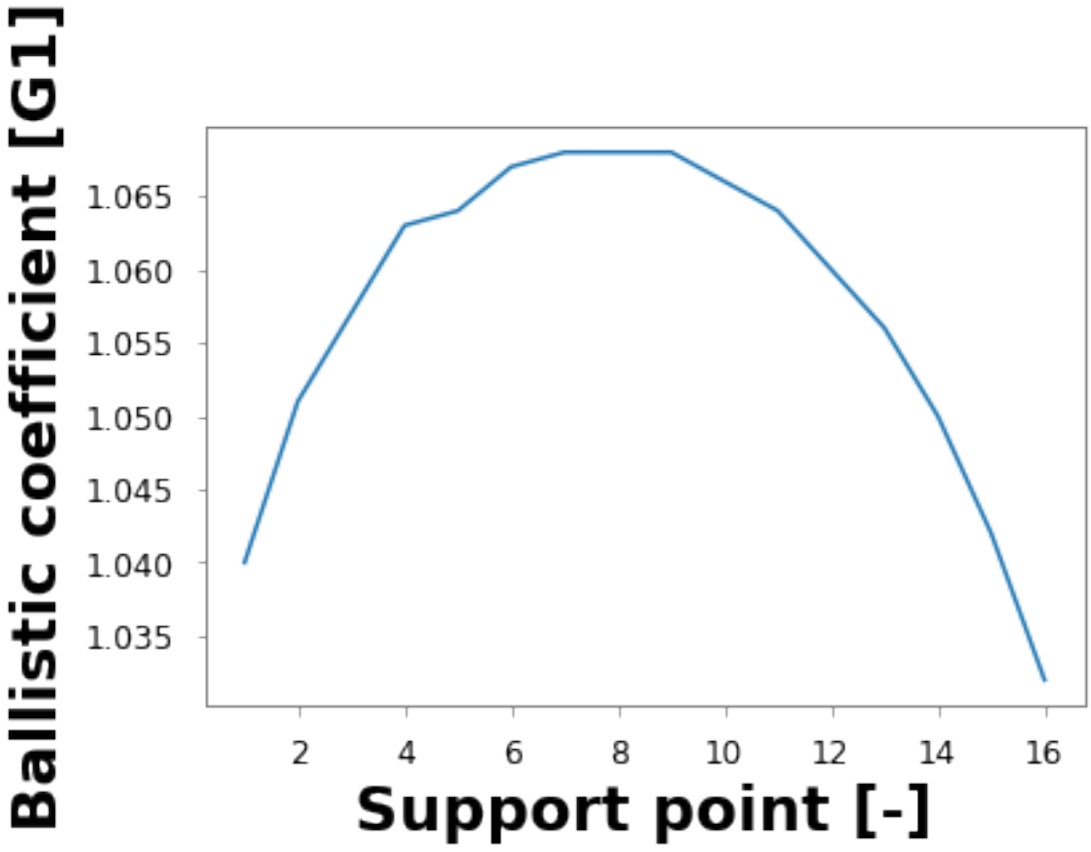
```
In [5]: A[6]
```

```
Out[5]: 1.0680000000000001
```

Get a plot with the `plot()` method

```
In [6]: A.plot()
```

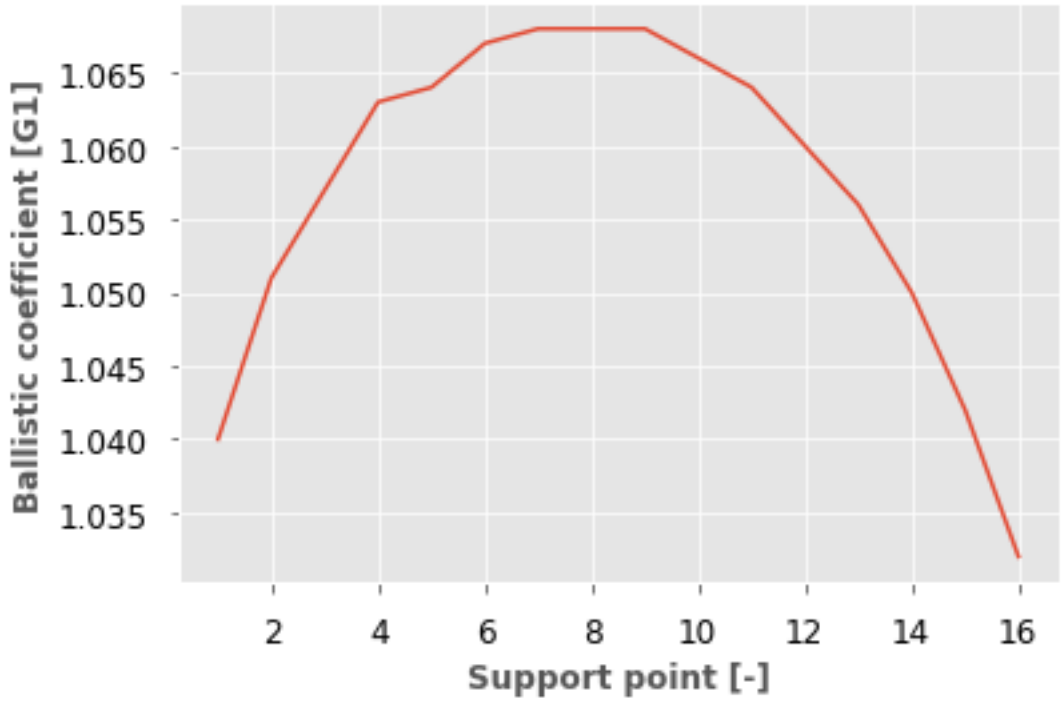
```
Out[6]: [<matplotlib.lines.Line2D at 0x181b8a0400>]
```



Not that fancy... you can set the `ggplot` style

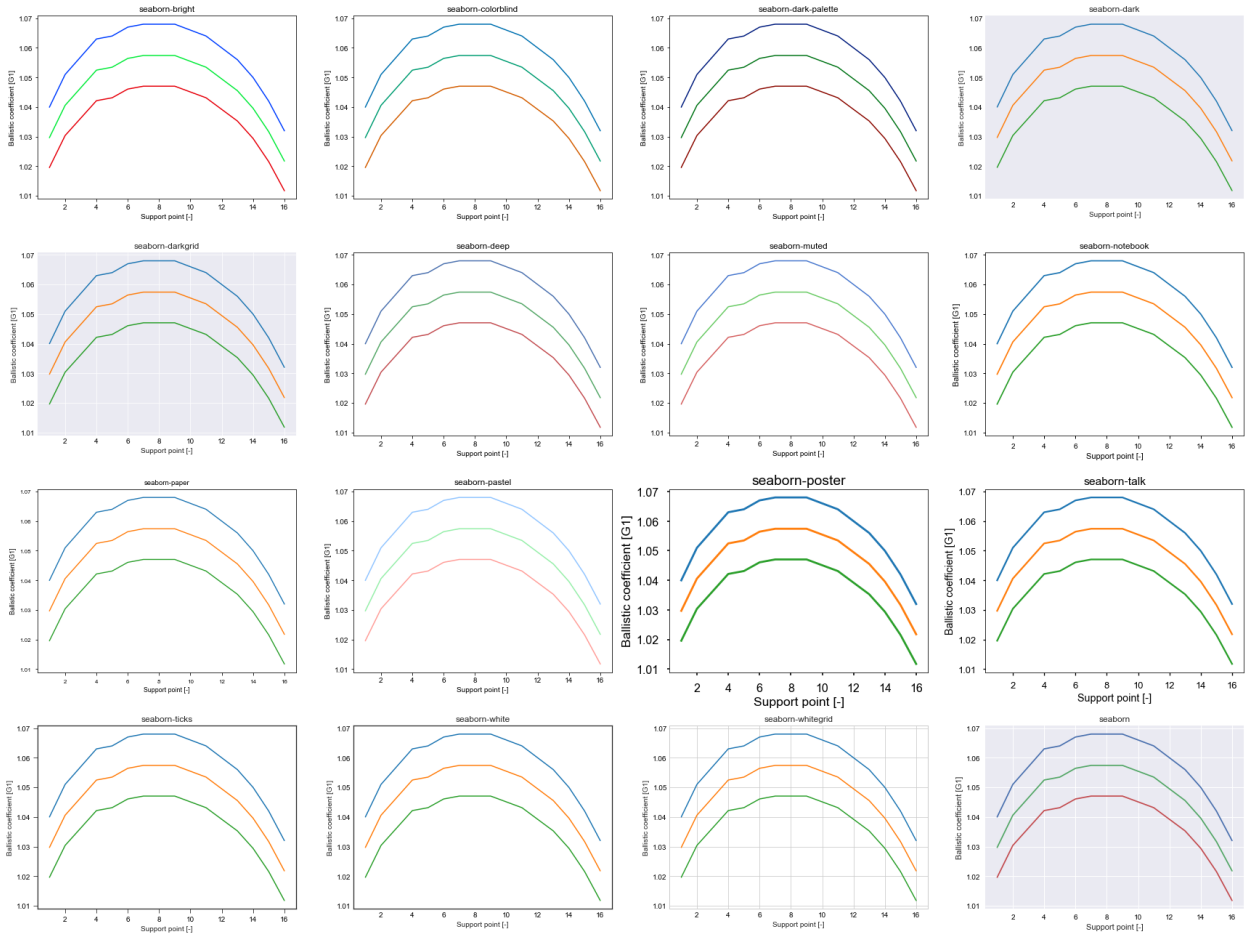
```
In [7]: plt.style.use('ggplot')
        A.plot()
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x181ba223c8>]
```



If you don't like ggplot, choose one of the seaborn styles

```
In [8]: import matplotlib.gridspec as gridspec
        import matplotlib as mpl
        styles = [_s for _s in plt.style.available if 'seaborn' in _s]
        n = np.ceil(np.sqrt(len(styles))).astype(np.int)

        gs = gridspec.GridSpec(n, n)

        plt.figure(figsize=(24,18))

        for i, s in enumerate(styles):
            mpl.rcParams.update(mpl.rcParamsDefault)
            plt.style.use(s)
            plt.subplot(gs[i])
            A.plot()
            (A / 1.01).plot()
            (A / 1.02).plot()
            plt.title(s)

        plt.tight_layout()
```



3.1.2 mesh2d

From Ballistic coefficient article in Wikipedia.

Doppler radar measurement results for a lathe turned monolithic solid .50 BMG very-low-drag bullet (Lost River J40 13.0 millimetres (0.510 in), 50.1 grams (773 gr) monolithic solid bullet / twist rate 1:380 millimetres (15 in)) look like this:

```
In [9]: BC = mesh2d(x=np.arange(500,2100,100), x_label="Range", x_unit="m",
                    d=[1.040, 1.051, 1.057, 1.063, 1.064, 1.067, 1.068, 1.068, 1.068, 1.066, 1.064,
                       1.060, 1.056, 1.050, 1.042, 1.032], label="Ballistic coefficient", unit="G1")
```

Display in the jupyter notebooks / ipython

```
In [10]: BC
```

```
Out[10]: x = BreakPoints(d=[ 500,  600,  700,  800,  900, 1000, 1100, 1200, 1300, 1400,
                      1500, 1600, 1700, 1800, 1900, 2000], label="Range", unit="m")
         d = array([ 1.04 ,  1.051,  1.057,  1.063,  1.064,  1.067,  1.068,  1.068,
                1.068,  1.066,  1.064,  1.06 ,  1.056,  1.05 ,  1.042,  1.032])
```

Interpolation

```
In [11]: BC(501)
```

```
Out[11]: 1.04011
```

```
In [12]: BC([501, 609, 2500])
```

```
Out[12]: array([ 1.04011,  1.05154,  0.982  ])
```

Default : values are extrapolated

```
In [13]: BC.interpolate([501, 609, 2500])
```

```
Out[13]: array([ 1.04011,  1.05154,  1.032  ])
```

Interpolation is performed and boundaries values are kept

```
In [14]: BC.options
```

```
Out[14]: {'extrapolate': True, 'step': False}
```

```
In [15]: BC.options['extrapolate']= False
```

This can be controled though the dict key extrapolate in options or interpolate method.

```
In [16]: BC([501, 609, 2500])
```

```
Out[16]: array([ 1.04011,  1.05154,  1.032  ])
```

```
In [17]: BC.max()
```

```
Out[17]: 1.0680000000000001
```

```
In [18]: BC.max(argwhere=True)
```

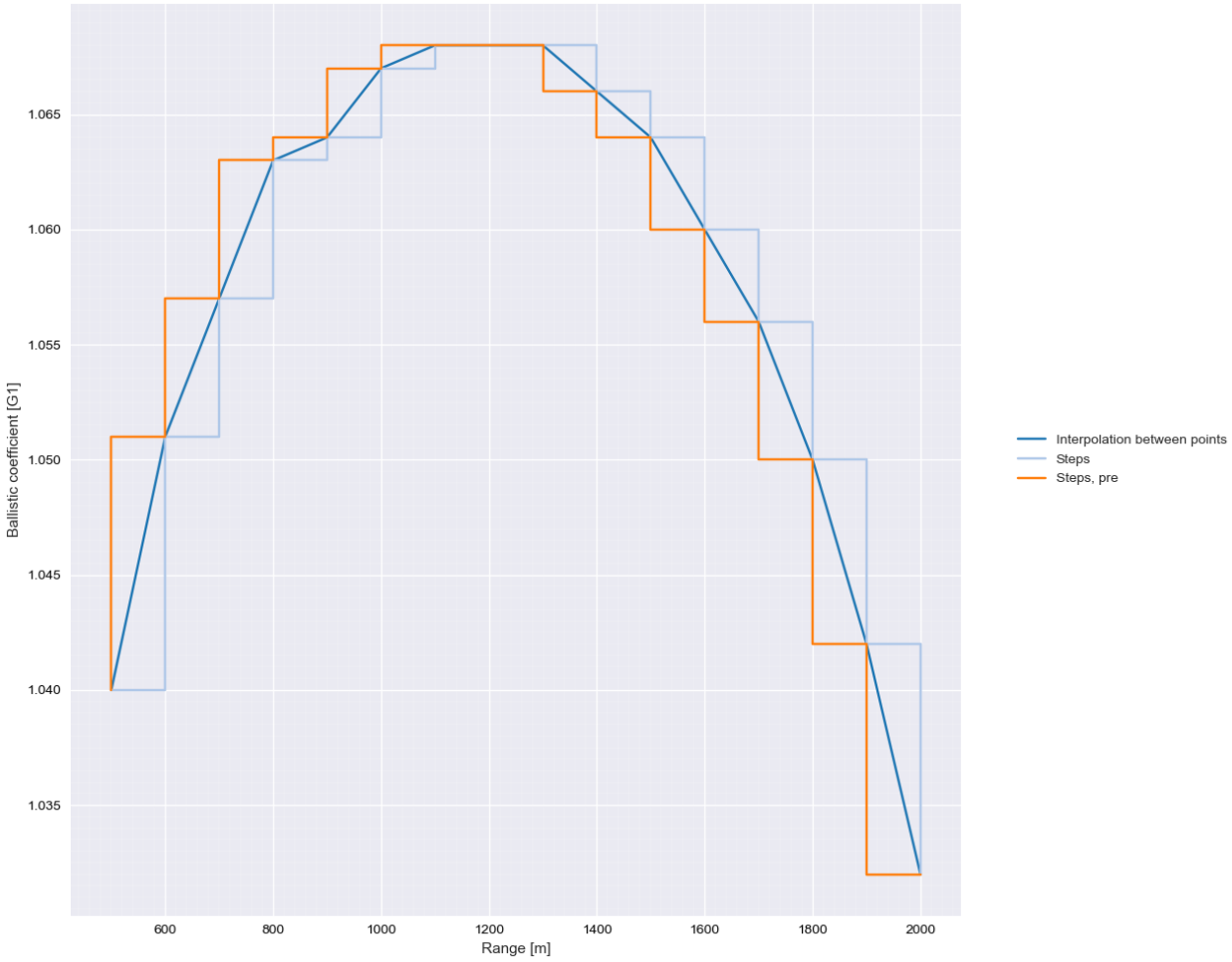
```
Out[18]: (1100, 1.0680000000000001)
```

Plot as steps

I like the color from vega

```
In [19]: from cycler import cycler
         category20 = cycler('color', ['#1f77b4', '#aec7e8', '#ff7f0e',
                                       '#ffbb78', '#2ca02c', '#98df8a',
                                       '#d62728', '#ff9896', '#9467bd',
                                       '#c5b0d5', '#8c564b', '#c49c94',
                                       '#e377c2', '#f7b6d2', '#7f7f7f',
                                       '#c7c7c7', '#bcbd22', '#dbdb8d',
                                       '#17becf', '#9edae5'])

         plt.style.use('seaborn-darkgrid')
         plt.rc('axes', prop_cycle=category20)
```

```
In [20]: plt.figure(figsize=(10,10))

         BC.plot(label="Interpolation between points")
         BC.steps.plot(label="Steps")
         BC.steps.plot(where="pre", label="Steps, pre")

         plt.graphpaper(dx=200, dy=0.005)
         plt.legend(bbox_to_anchor=(1.05, 0.5), loc='center left', facecolor="white", frameon=False)
         plt.tight_layout()
```



Slicing

```
In [21]: BC[2:4]

Out[21]: x = BreakPoints(d=[700, 800], label="Range", unit="m")
         d = array([ 1.057,  1.063])
```

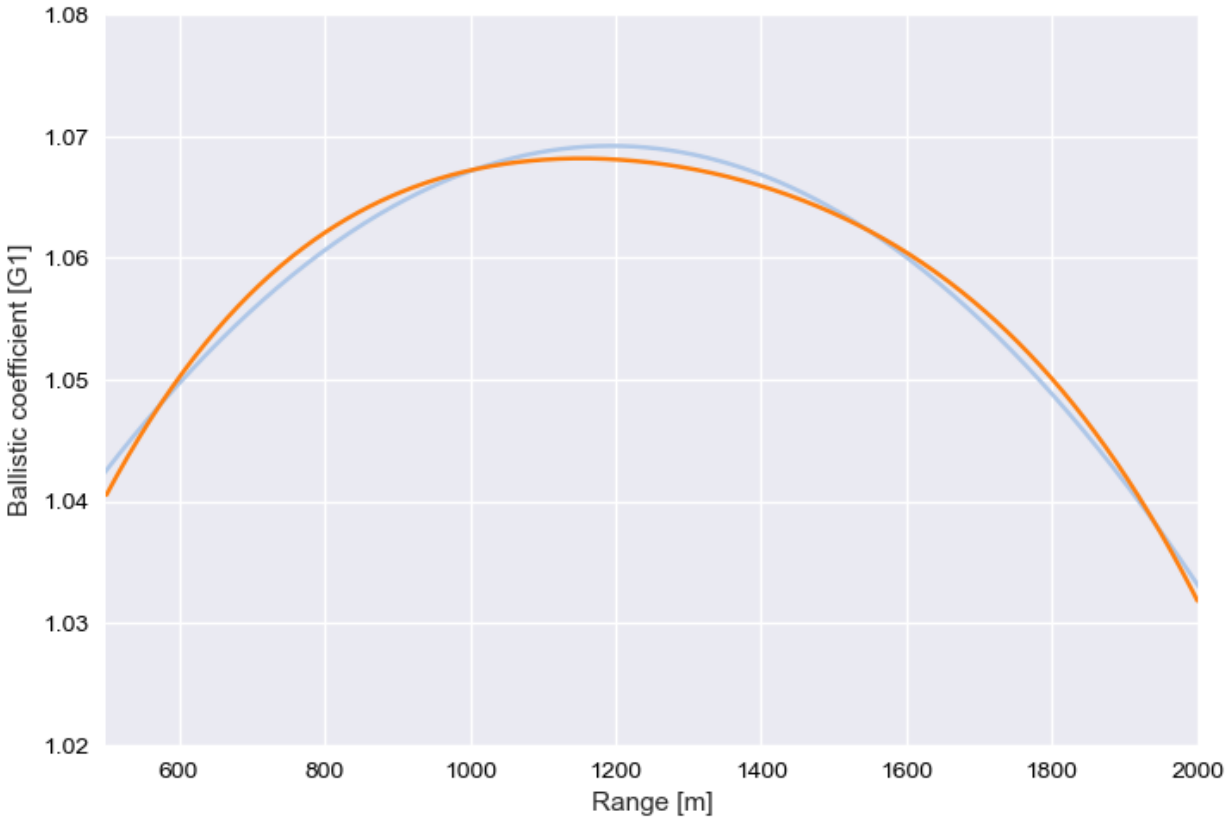
Breakpoints strictly monotone, reverse order has no effect

```
In [22]: BC[::-1]

Out[22]: x = BreakPoints(d=[ 500,  600,  700,  800,  900, 1000, 1100, 1200, 1300, 1400,
                     1500, 1600, 1700, 1800, 1900, 2000], label="Range", unit="m")
         d = array([ 1.04 ,  1.051,  1.057,  1.063,  1.064,  1.067,  1.068,  1.068,
                1.068,  1.066,  1.064,  1.06 ,  1.056,  1.05 ,  1.042,  1.032])

In [23]: BC[6]

Out[23]: (1100, 1.0680000000000001)

In [24]: BC.x

Out[24]: BreakPoints(d=[ 500,  600,  700,  800,  900, 1000, 1100, 1200, 1300, 1400,
                     1500, 1600, 1700, 1800, 1900, 2000], label="Range", unit="m")

In [25]: BC(np.arange(500, 550, 10))

Out[25]: array([ 1.04  ,  1.0411,  1.0422,  1.0433,  1.0444])

In [26]: BC.resample(np.arange(500, 2000, 200))
```

```
Out[26]: x = BreakPoints(d=[ 500,  700,  900, 1100, 1300, 1500, 1700, 1900], label="Range", unit="m")
         d = array([ 1.04 ,  1.057,  1.064,  1.068,  1.068,  1.064,  1.056,  1.042])

In [27]: BC.steps(BC.x)

Out[27]: array([ 1.04 ,  1.051,  1.057,  1.063,  1.064,  1.067,  1.068,  1.068,
                 1.068,  1.066,  1.064,  1.06 ,  1.056,  1.05 ,  1.042,  1.032])

In [28]: BC.__dict__

Out[28]: {'_d': array([ 1.04 ,  1.051,  1.057,  1.063,  1.064,  1.067,  1.068,  1.068,
                        1.068,  1.066,  1.064,  1.06 ,  1.056,  1.05 ,  1.042,  1.032]),
          '_options': {'extrapolate': False, 'step': False},
          '_steps': x = BreakPoints(d=[ 500,  600,  700,  800,  900, 1000, 1100, 1200, 1300, 1400,
                                        1500, 1600, 1700, 1800, 1900, 2000], label="Range", unit="m")
                    d = array([ 1.04 ,  1.051,  1.057,  1.063,  1.064,  1.067,  1.068,  1.068,
                                1.068,  1.066,  1.064,  1.06 ,  1.056,  1.05 ,  1.042,  1.032]),
          '_x': BreakPoints(d=[ 500,  600,  700,  800,  900, 1000, 1100, 1200, 1300, 1400,
                                1500, 1600, 1700, 1800, 1900, 2000], label="Range", unit="m"),
          'label': 'Ballistic coefficient',
          'unit': 'G1'}
```

.polyfit(): how to get a polymesh object from discrete values

```
In [29]: BC.plot("+")
         BC.polyfit(degree=2).plot(xlim=[500,2000])
         BC.polyfit(degree=4).plot(xlim=[500,2000], ylim=[1.02, 1.08])
```



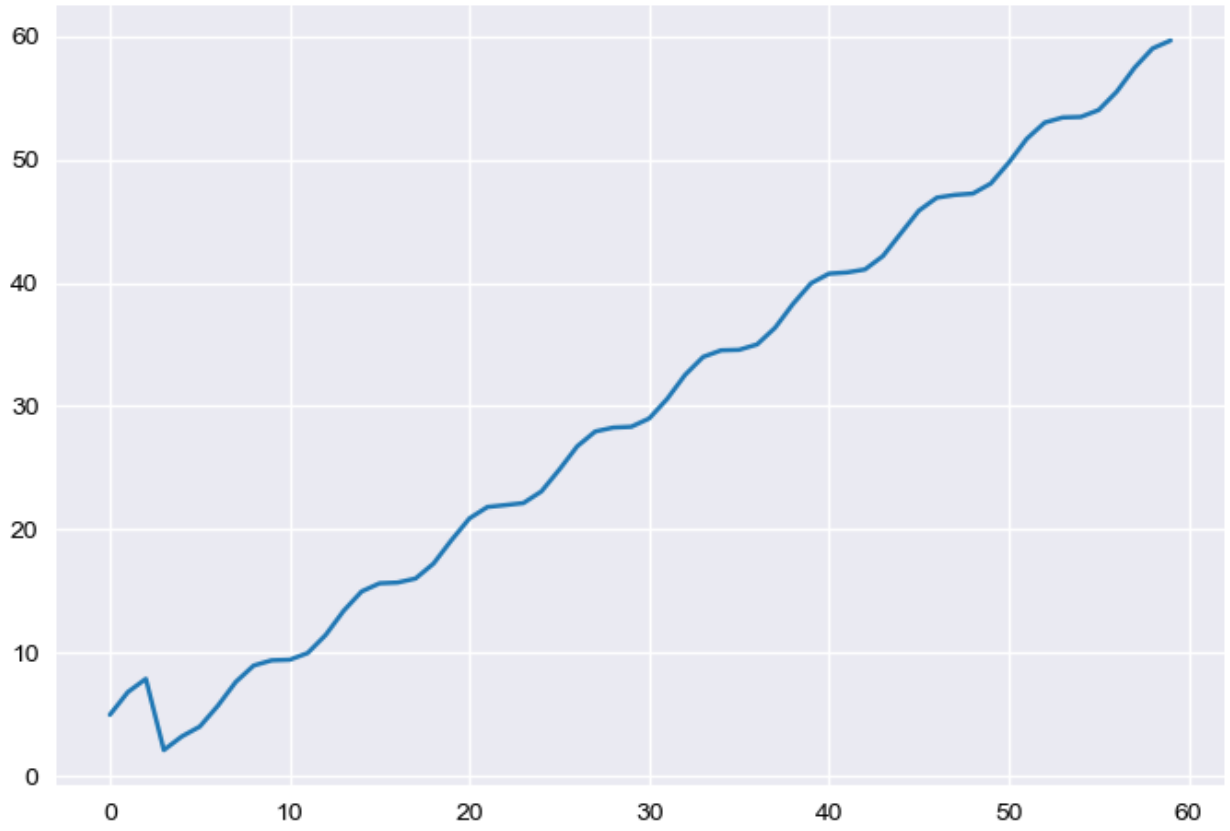

```
In [30]: BC.polyfit(degree=4)

Out[30]: -3.5896443597682554e-14·x^4 + 1.820225053584262e-10·x^3 - 3.821239680082758e-07·x^2 + 0.0003

In [31]: from sklearn.metrics import r2_score
```

```
        for i in range(10):
            print(i, r2_score(BC.d, BC.polyfit(degree=i)(BC.x).d))
```

```
0 0.0
1 0.0698263920953
2 0.988101760851
3 0.988408030976
4 0.997727098729
5 0.998238919776
6 0.998565636158
7 0.998569159428
8 0.998610724807
9 0.998652377691
```

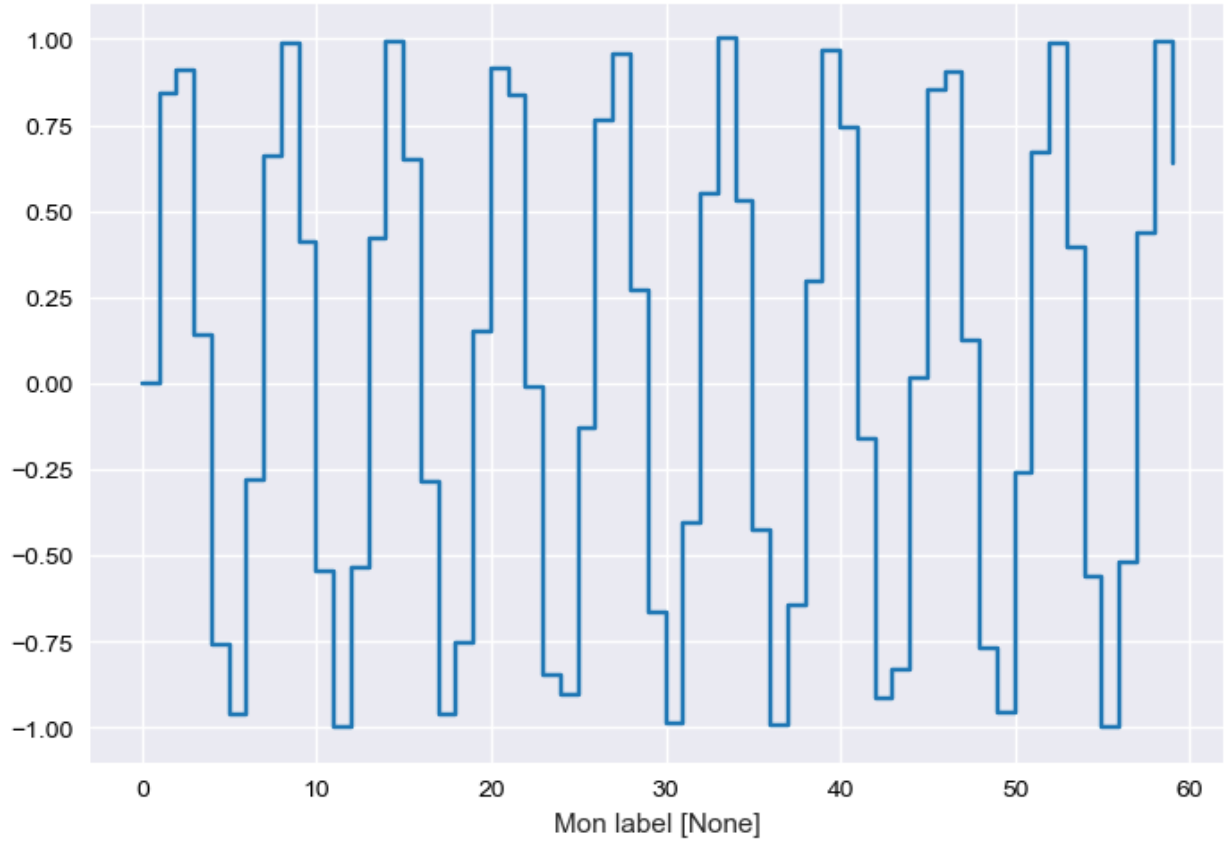
.add()

```
In [32]: # prepare some data
         x = [1, 2, 3, 4, 5]
         y = [6, 7, 2, 4, 5]
         test2 = mesh2d(x, y)
         x2 = np.arange(0,60)
         test = mesh2d(x2, np.sin(x2), x_label="Mon label", unit="%")

In [33]: (test + test2)

Out[33]: x = BreakPoints(d=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                     15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                     30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                     45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], label="None", unit
         d = array([  5.        ,   6.84147098,   7.90929743,   2.14112001,
                  3.2431975 ,   4.04107573,   5.7205845 ,   7.6569866 ,
                  8.98935825,   9.41211849,   9.45597889,  10.00000979,
                 11.46342708,  13.42016704,  14.99060736,  15.65028784,
                 15.71209668,  16.03860251,  17.24901275,  19.14987721,
                 20.91294525,  21.83665564,  21.99114869,  22.1537796 ,
                 23.09442164,  24.86764825,  26.76255845,  27.95637593,
                 28.27090579,  28.33636612,  29.01196838,  30.59596235,
                 32.55142668,  33.99991186,  34.52908269,  34.57181733,
                 35.00822115,  36.35646187,  38.29636858,  39.96379539,
                 40.74511316,  40.84137733,  41.08347845,  42.16822526,
                 44.01770193,  45.85090352,  46.90178835,  47.12357312,
                 47.23174534,  48.04624735,  49.73762515,  51.67022918,
                 52.98662759,  53.39592515,  53.44121095,  54.00024483,
                 55.478449  ,  57.43616476,  58.99287265,  59.63673801])

In [34]: test[:4]

Out[34]: x = BreakPoints(d=[0, 1, 2, 3], label="Mon label", unit="None")
         d = array([ 0.        ,  0.84147098,  0.90929743,  0.14112001])

In [35]: test[-3:] + test[:4]

Out[35]: x = BreakPoints(d=[ 0,  1,  2,  3, 57, 58, 59], label="None", unit="None")
         d = array([-31.29618514, -29.89800626, -29.27347192, -29.48494145,
                -40.90429585, -41.11576538, -42.24007744])

In [36]: (test + test2).plot()

Out[36]: [<matplotlib.lines.Line2D at 0x181d938240>]
```




```
In [37]: test.steps.plot()
```

```
Out[37]: [<matplotlib.lines.Line2D at 0x181c046630>]
```




```
In [38]: test
```

```
Out[38]: x = BreakPoints(d=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                      15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                      30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                      45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], label="Mon label",
         d = array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
                -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
                -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
                 0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.14987721,
                 0.91294525,  0.83665564, -0.00885131, -0.8462204 , -0.90557836,
                -0.13235175,  0.76255845,  0.95637593,  0.27090579, -0.66363388,
                -0.98803162, -0.40403765,  0.55142668,  0.99991186,  0.52908269,
                -0.42818267, -0.99177885, -0.64353813,  0.29636858,  0.96379539,
                 0.74511316, -0.15862267, -0.91652155, -0.83177474,  0.01770193,
                 0.85090352,  0.90178835,  0.12357312, -0.76825466, -0.95375265,
                -0.26237485,  0.67022918,  0.98662759,  0.39592515, -0.55878905,
                -0.99975517, -0.521551  ,  0.43616476,  0.99287265,  0.63673801])
```

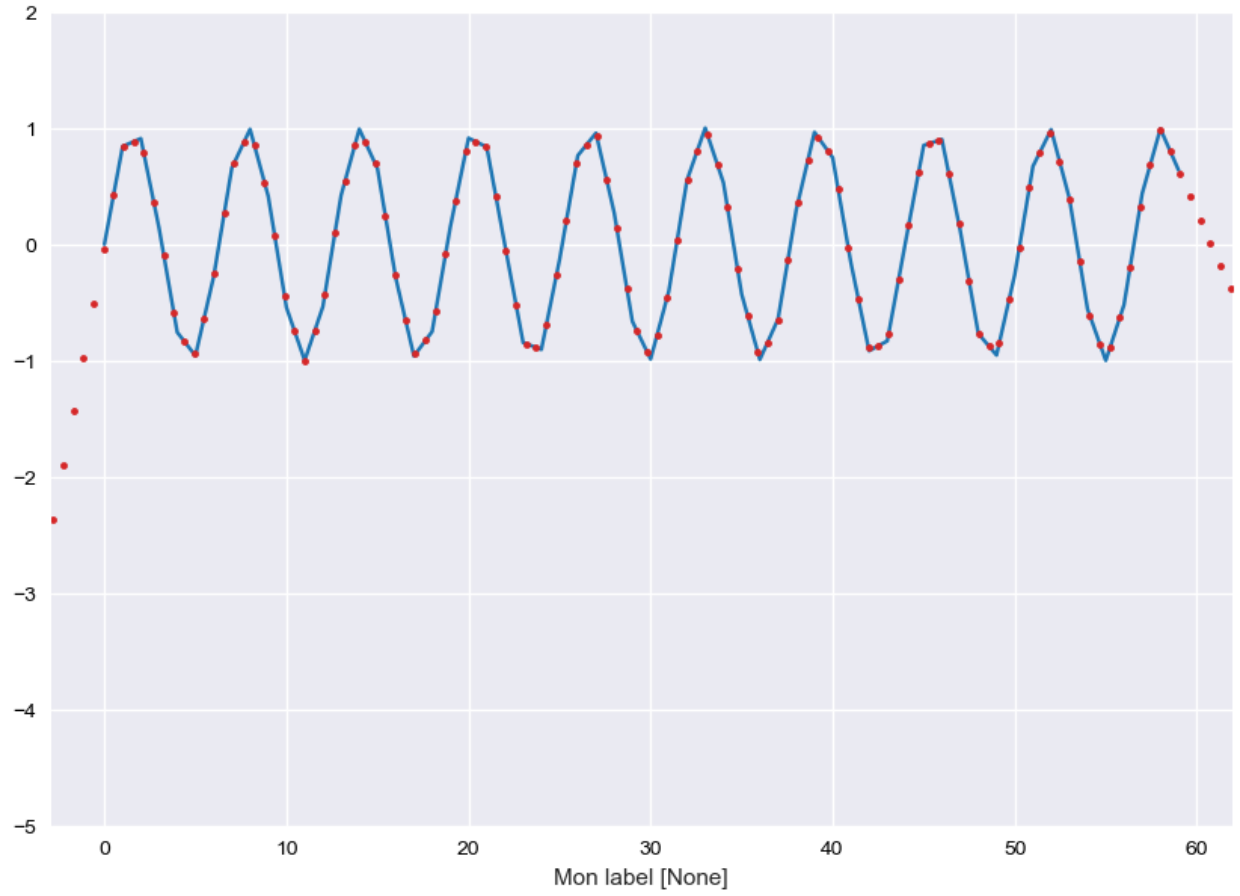
```
In [39]: test.resample(np.linspace(0,60,100))
```

```
Out[39]: x = BreakPoints(d=[  0.  ,   0.61,   1.21,   1.82,   2.42,   3.03,   3.64,
                       4.24,   4.85,   5.45,   6.06,   6.67,   7.27,   7.88,
                       8.48,   9.09,   9.7 ,  10.3 ,  10.91,  11.52,  12.12,
                      12.73,  13.33,  13.94,  14.55,  15.15,  15.76,  16.36,
                      16.97,  17.58,  18.18,  18.79,  19.39,  20.  ,  20.61,
                      21.21,  21.82,  22.42,  23.03,  23.64,  24.24,  24.85,
                      25.45,  26.06,  26.67,  27.27,  27.88,  28.48,  29.09,
                      29.7 ,  30.3 ,  30.91,  31.52,  32.12,  32.73,  33.33,
                      33.94,  34.55,  35.15,  35.76,  36.36,  36.97,  37.58,
```

```
                    38.18,  38.79,  39.39,  40.  ,  40.61,  41.21,  41.82,
                    42.42,  43.03,  43.64,  44.24,  44.85,  45.45,  46.06,
                    46.67,  47.27,  47.88,  48.48,  49.09,  49.7 ,  50.3 ,
                    50.91,  51.52,  52.12,  52.73,  53.33,  53.94,  54.55,
                    55.15,  55.76,  56.36,  56.97,  57.58,  58.18,  58.79,
                    59.39,  60.  ], label="Mon label", unit="None")
       d = array([ 0.        ,  0.50998242,  0.85585841,  0.89696535,  0.58340398,
               0.11391024, -0.43028522, -0.80580171, -0.92829976, -0.65005665,
              -0.22266386,  0.34485257,  0.74763341,  0.94907077,  0.70948442,
               0.3251967 , -0.25428184, -0.68219356, -0.95853847, -0.76126009,
              -0.42060444,  0.15923796,  0.61031381,  0.95603522,  0.80497853,
               0.50813766, -0.06046304, -0.53281029, -0.94098858, -0.8402522 ,
              -0.58719371, -0.04121525,  0.45047977,  0.91294525,  0.86670912,
               0.65730568,  0.14487723, -0.3640988 , -0.84801913, -0.88399365,
              -0.71812949, -0.2495073 ,  0.27442561,  0.77430496,  0.8917701 ,
               0.76942953,  0.35399308, -0.18220436, -0.69312459, -0.88972928,
              -0.81106375, -0.45712801,  0.08817125,  0.60578852,  0.87759772,
               0.8429688 ,  0.55761779,  0.00693795, -0.51357603, -0.85514948,
              -0.86514586, -0.65409088, -0.10237972,  0.41771891,  0.82222   ,
               0.87764784,  0.74511316,  0.19739448, -0.3193891 , -0.77872175,
              -0.88056836, -0.80603303, -0.29119868,  0.21969019,  0.72466086,
               0.87403299,  0.85462379,  0.3829782 , -0.11965264, -0.66015432,
              -0.85819308, -0.89090013, -0.47188328,  0.02023243,  0.58544699,
               0.8332223 ,  0.9150273 ,  0.55702582,  0.07768708, -0.50092758,
              -0.79931603, -0.9273    , -0.63747929, -0.17329073,  0.40714307,
               0.75669354,  0.9281209 ,  0.71228172,  0.49644254,  0.28060337])
```

```
In [40]: plt.figure(figsize=(12,7))
         test.plot()
         newX = np.linspace(-10,100,200)
         test.resample(newX).plot('.', c=category20.by_key()['color'][6], lw=0.1)
         plt.ylim(-5, 2)
```

```
Out[40]: (-5, 2)
```



Not implemented

```
In [41]: test + test2.steps
```

```
Out[41]: x = BreakPoints(d=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                      15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                      30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                      45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], label="None", unit
         d = array([  5.        ,   6.84147098,   7.90929743,   2.14112001,
                  3.2431975 ,   4.04107573,   5.7205845 ,   7.6569866 ,
                  8.98935825,   9.41211849,   9.45597889,  10.00000979,
                 11.46342708,  13.42016704,  14.99060736,  15.65028784,
                 15.71209668,  16.03860251,  17.24901275,  19.14987721,
                 20.91294525,  21.83665564,  21.99114869,  22.1537796 ,
                 23.09442164,  24.86764825,  26.76255845,  27.95637593,
                 28.27090579,  28.33636612,  29.01196838,  30.59596235,
                 32.55142668,  33.99991186,  34.52908269,  34.57181733,
                 35.00822115,  36.35646187,  38.29636858,  39.96379539,
                 40.74511316,  40.84137733,  41.08347845,  42.16822526,
                 44.01770193,  45.85090352,  46.90178835,  47.12357312,
                 47.23174534,  48.04624735,  49.73762515,  51.67022918,
                 52.98662759,  53.39592515,  53.44121095,  54.00024483,
                 55.478449  ,  57.43616476,  58.99287265,  59.63673801])
```

```
In [42]: test.steps([-10, 2.3, 3, 3.1, 59, 58.9, 90])
```

```
Out[42]: array([ -8.41470985,   0.6788442 ,   0.14112001,   0.05132776,
                  0.63673801,   0.67235147, -10.40343586])
```

```
In [43]: test.steps
```

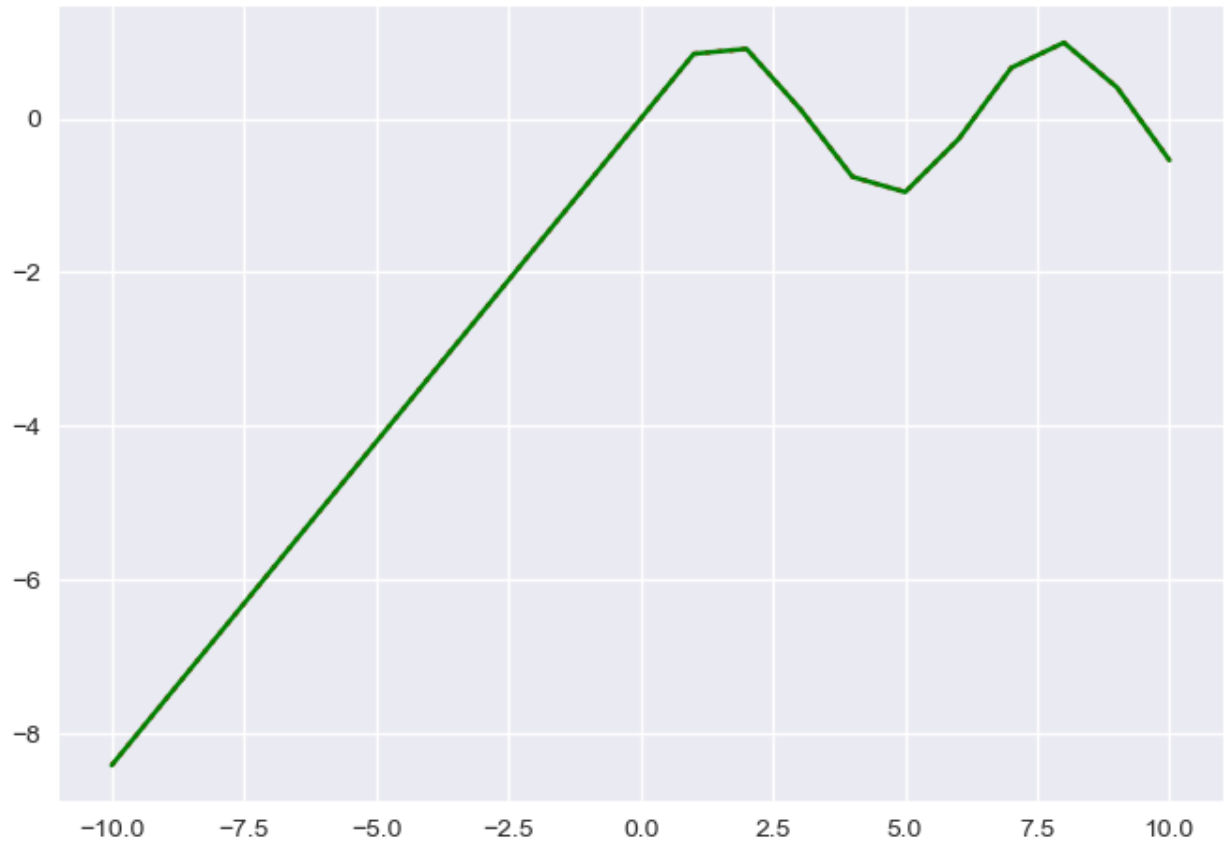
```
Out[43]: x = BreakPoints(d=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                      15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                      30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                      45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], label="Mon label",
         d = array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
                -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
                -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
                 0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.14987721,
                 0.91294525,  0.83665564, -0.00885131, -0.8462204 , -0.90557836,
                -0.13235175,  0.76255845,  0.95637593,  0.27090579, -0.66363388,
                -0.98803162, -0.40403765,  0.55142668,  0.99991186,  0.52908269,
                -0.42818267, -0.99177885, -0.64353813,  0.29636858,  0.96379539,
                 0.74511316, -0.15862267, -0.91652155, -0.83177474,  0.01770193,
                 0.85090352,  0.90178835,  0.12357312, -0.76825466, -0.95375265,
                -0.26237485,  0.67022918,  0.98662759,  0.39592515, -0.55878905,
                -0.99975517, -0.521551  ,  0.43616476,  0.99287265,  0.63673801])
```

```
In [44]: test.diff(n=1)
```

```
Out[44]: x = BreakPoints(d=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
                      15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                      30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                      45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58], label="Mon label", uni
         d = array([ 0.84147098,  0.06782644, -0.76817742, -0.8979225 , -0.20212178,
                 0.67950878,  0.9364021 ,  0.33237165, -0.57723976, -0.9561396 ,
                -0.4559691 ,  0.46341729,  0.95673995,  0.57044032, -0.34031952,
                -0.93819116, -0.67349418,  0.21041025,  0.90086446,  0.76306804,
                -0.07628961, -0.84550695, -0.83736909, -0.05935796,  0.77322661,
                 0.8949102 ,  0.19381748, -0.68547014, -0.93453967, -0.32439774,
                 0.58399398,  0.95546433,  0.44848518, -0.47082917, -0.95726536,
                -0.56359618,  0.34824072,  0.93990671,  0.66742681, -0.21868223,
                -0.90373583, -0.75789888,  0.08474681,  0.84947667,  0.8332016 ,
                 0.05088482, -0.77821522, -0.89182778, -0.18549799,  0.6913778 ,
                 0.93260403,  0.31639842, -0.59070244, -0.9547142 , -0.44096612,
                 0.47820417,  0.95771576,  0.55670789, -0.35613464])
```

```
In [45]: #plt.plot(test.X, test.Y, c="b")
         newX = np.arange(-10, 10, 0.001)
         plt.plot(newX, test(newX), ":r")
         plt.plot(newX, test.steps(newX), "-g")
```

```
Out[45]: [<matplotlib.lines.Line2D at 0x181ba962e8>]
```



In [46]:
```
test.steps(newX)
```

Out[46]:
```
array([-8.41470985, -8.41386838, -8.41302691, ..., -0.54115269,
       -0.54210883, -0.54306497])
```

In [47]:
```
import random
from numba import jit

N = 200
myMesh = mesh2d(np.arange(N)*5, [random.uniform(2.5, 10.0) for i in range(N)], x_label="MON
```

In [48]:
```
@jit
def _extrapolate(self, X):
    """
    """
    if X <= self.x[0]:
        res = self.d[0] + (X - self.x[0]) *\
            (self.d[1] - self.d[0]) / (self.x[1] - self.x[0])
    elif X >= self.x[-1]:
        res = self.d[-1] + (x - self.x[-1]) *\
            (self.d[-1] - self.d[-2]) / (self.x[-1] - self.x[-2])
    else:
        res = np.interp(X, self.x, self.d)
    return res
```

In [49]:
```
%timeit myMesh.extrapolate(255)
```

```
21.7 µs ± 915 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [50]:
```
%timeit _extrapolate(myMesh, 255)
```

```
The slowest run took 5.50 times longer than the fastest. This could mean that an intermediate result
94.3 µs ± 87.9 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [51]: np.diff(test2.d) / np.diff(test2.x)
```

```
Out[51]: BreakPoints(d=[ 1., -5.,  2.,  1.], label="None", unit="None")
```

3.1.3 mesh3d

```
In [52]: m3d = mesh3d(x=np.arange(10), x_label="X", x_unit="X unit",
                      y=np.arange(10), y_label="Y", y_unit="Y unit",
                      d=np.random.random((10,10)), label="data", unit="data unit")
```

```
In [53]: lerp.options.display.max_rows = 15
```

```
In [54]: m3d
```

```
Out[54]: <lerp.mesh.mesh3d at 0x1a20a6f198>
```

```
In [55]: m3d(4.5, 5.7)
```

```
Out[55]: 0.22891672933562421
```

Interpolation

```
In [56]: m3d(3.5, 5.6)
```

```
Out[56]: 0.24804759269057369
```

Call method default : extrapolation above boundaries

```
In [57]: m3d(9.1,9.4)
```

```
Out[57]: 1.2617807437078306
```

As far mesh2d, can be set with options attributes or method call .interpolate

```
In [58]: m3d.options
```

```
Out[58]: {'extrapolate': True}
```

```
In [59]: m3d.interpolate(9.1,9.4)
```

```
Out[59]: 0.9606969801252196
```

```
In [60]: m3d(x=3.5)
```

```
Out[60]: x = BreakPoints(d=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], label="Y", unit="Y unit")
         d = array([ 0.69944545,  0.45196195,  0.62338401,  0.22360536,  0.56315821,
                 0.30124409,  0.21258326,  0.25238123,  0.70839876,  0.35742004])
```

Slicing

```
In [61]: m3d[3:6]
```

```
Out[61]: <lerp.mesh.mesh3d at 0x1a20a6fba8>
```

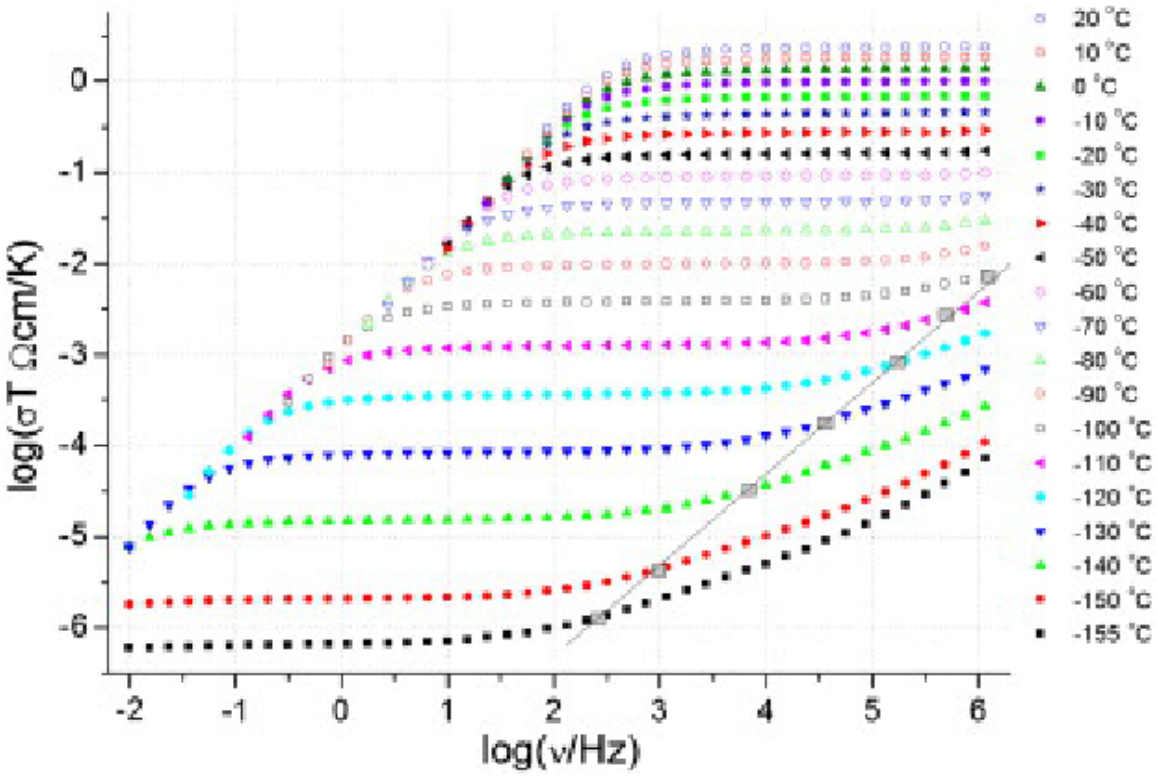
Garbage after that point

```
In [62]: from scipy import misc
         from scipy import ndimage
         # http://www.ndt.net/article/wcndt00/papers/idn360/idn360.htm

         from PIL import Image
         from urllib.request import urlopen
         import io

         URL = 'https://www.researchgate.net/profile/Robert_Dinnebier/publication/268693474/figure/f
```

```
        with urlopen(URL) as url:
            im = misc.imread(io.BytesIO(url.read()))

        # face = misc.imread(f)
```

In [63]:
```
plt.imshow(im, cmap=plt.cm.gray, vmin=30, vmax=200)
plt.axis('off')
# plt.contour(im, [50, 200])
```

Out[63]: (-0.5, 388.5, 259.5, -0.5)



In [64]:
```
sx = ndimage.sobel(im, axis=0, mode='constant')
sy = ndimage.sobel(im, axis=1, mode='constant')
sob = np.hypot(sx, sy)
```

In [65]:
```
sx = ndimage.sobel(im, axis=0, mode='constant')
```

import numpy as np import matplotlib.pyplot as plt from scipy import interpolate

x = np.array([[0.12, 0.11, 0.1, 0.09, 0.08], [0.13, 0.12, 0.11, 0.1, 0.09], [0.15, 0.14, 0.12, 0.11, 0.1], [0.17, 0.15, 0.14, 0.12, 0.11], [0.19, 0.17, 0.16, 0.14, 0.12], [0.22, 0.19, 0.17, 0.15, 0.13], [0.24, 0.22, 0.19, 0.16, 0.14], [0.27, 0.24, 0.21, 0.18, 0.15], [0.29, 0.26, 0.22, 0.19, 0.16]])

y = np.array([[71.64, 78.52, 84.91, 89.35, 97.58], [66.28, 73.67, 79.87, 85.36, 93.24], [61.48, 69.31, 75.36, 81.87, 89.35], [57.61, 65.75, 71.7, 79.1, 86.13], [55.12, 63.34, 69.32, 77.29, 83.88], [54.58, 62.54, 68.7, 76.72, 82.92], [56.58, 63.87, 70.3, 77.69, 83.53], [61.67, 67.79, 74.41, 80.43, 85.86], [70.08, 74.62, 80.93, 85.06, 89.84]])

plt.figure(figsize = (9, 9)) plt.subplot(111) for i in range(5): $x_val = np.linspace(x[0, i], x[-1, i], 100) x_int = np.interp(x_val, x[:, i], y[:, i]) tck = interpolate.splrep(x[:, i], y[:, i], k = 2, s = 4) y_int = interpolate.splev(x_val, tck, der = 0) plt.plot(x[:, i], y[:, i], linestyle = ''$

$, marker =' o')plt.plot(x_val, y_int, linestyle =':', linewidth = 0.25, color ='$
$black')plt.xlabel('X')plt.ylabel('Y')plt.show()$

from scipy.interpolate import dfitpack, fitpack

def extrapolate(self, X, Y): if X <= self.X[0]: iX = 0 elif X >= self.X[-1]: iX = -2 else: iX = np.searchsorted(self.X, X) - 1

if Y <= self.Y[0]: iY = 0 elif Y >= self.Y[-1]: iY = -2 else: iY = np.searchsorted(self.Y, Y) - 1

Z1 = self.W[iX, iY] + (self.W[iX, iY+1] - self.W[iX, iY]) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY]) Z2 = self.W[iX+1, iY] + (self.W[iX+1, iY+1] - self.W[iX+1, iY]) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY])

return Z1 + (Z2 - Z1) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX])

def interpolate(self, X, Y): if X <= self.X[0]: return np.interp(Y, self.Y, self.W[0]) elif X >= self.X[-1]: return np.interp(Y, self.Y, self.W[-1]) else: iX = np.searchsorted(self.X, X) - 1

if Y <= self.Y[0]: return np.interp(X, self.X, self.W[:,0]) elif Y >= self.Y[-1]: return np.interp(X, self.X, self.W[:,-1]) else: iY = np.searchsorted(self.Y, Y) - 1

Z1 = self.W[iX, iY] + (self.W[iX, iY+1] - self.W[iX, iY]) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY]) Z2 = self.W[iX+1, iY] + (self.W[iX+1, iY+1] - self.W[iX+1, iY]) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY])

return Z1 + (Z2 - Z1) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX])

np.diff(m3d.W[1:3,2:4], axis=1)

np.diff(m3d.W[1:3,2:4], axis=0)

m3d = mesh3d(np.arange(100),np.arange(50),np.random.random((50,100)))

def extrapolate(self, X, Y): iX = 0 iY = 0

if X <= self.X[0]: iX = 0 elif X >= self.X[-1]: iX = -2 else: iX = np.where(self.X < X)[0][-1]

if Y <= self.Y[0]: iY = 0 elif Y >= self.Y[-1]: iY = -2 else: iY = np.where(self.Y < Y)[0][-1]

Z1 = self.W[iY, iX] + (self.W[iY, iX+1] - self.W[iY, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX]) Z2 = self.W[iY+1, iX] + (self.W[iY+1, iX+1] - self.W[iY+1, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX])

return Z1 + (Z2 - Z1) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY])

def extrapolate1(self, $_{X,Y}$) : $xNew = np.searchsorted(self.X,_X).clip(1, len(self.X)-1).astype(int)yNew =$
$np.searchsorted(self.Y,_Y).clip(1, len(self.Y) - 1).astype(int)$

4. Calculate the slope of regions that each X value falls in. xLo, xHi = xNew - 1, xNew yLo, yHi = yNew - 1, yNew

x11 = self.X[xLo] x12 = self.X[xHi] y11 = self.Y[yLo] y21 = self.Y[yHi] w11 = self.W[:,xLo] w12 = self.W[:,xHi]

Note that the following two expressions rely on the specifics of the broadcasting semantics. xSlope = (w12 - w11) / (x12 - x11)

5. Calculate the actual value for each entry in X. yNew = xSlope * $(_X - x11) + w11$

w11 = yNew[yLo] w21 = yNew[yHi]

ySlope = (w21 - w11) / (y21 - y11)

yNew = ySlope * $(_Y - y11) + w11$

return np.array(yNew)

def extrapolate2(self, X, Y):

iX = -2 if X >= self.X[-1] else np.searchsorted(self.X, X).clip(1, len(self.X)-1).astype(int) - 1

iY = -2 if Y >= self.Y[-1] else np.searchsorted(self.Y, Y).clip(1, len(self.Y)-1).astype(int) - 1

Z1 = self.W[iY, iX] + (self.W[iY, iX+1] - self.W[iY, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX]) Z2 = self.W[iY+1, iX] + (self.W[iY+1, iX+1] - self.W[iY+1, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX])

return Z1 + (Z2 - Z1) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY])

def extrapolate3(self, X, Y): gc.disable() st = time.time()

if X <= self.X[0]: iX = 0 elif X >= self.X[-1]: iX = -2 else: iX = np.searchsorted(self.X, X) - 1

if Y <= self.Y[0]: iY = 0 elif Y >= self.Y[-1]: iY = -2 else: iY = np.searchsorted(self.Y, Y) - 1

Z1 = self.W[iY, iX] + (self.W[iY, iX+1] - self.W[iY, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX]) Z2 = self.W[iY+1, iX] + (self.W[iY+1, iX+1] - self.W[iY+1, iX]) * (X - self.X[iX]) / (self.X[iX+1] - self.X[iX])

return Z1 + (Z2 - Z1) * (Y - self.Y[iY]) / (self.Y[iY+1] - self.Y[iY])

New mesh

from functools import singledispatch

class newmesh(np.ndarray):

def $_{new}{}_{(cls,data=None,label=None,unit=None):Wefirstcasttobeourclasstypenp.asfarray([],dtype='float64')@singledispatchdefmyArray(o):ifoisNone:o=[]Willcall}$

@myArray.register(mesh1d) def $_{(o)}$: $iflabelisnotNone$: $o.label = labelifunitisnotNone$: $o.unit = unitreturno$

return myArray(data)

see InfoArray.$_{arrayfinalize}{}_{forcommentsdef}{}_{arrayfinalize}{}_{(self,obj):"""}$

:type obj: object """ self.unit = getattr(obj, 'unit', None) self.label = getattr(obj, 'label', None)

@property def d(self): return self[0]

@d.setter def d(self, obj): self[0] = obj

test = newmesh([1, 3, 4])

test.d = 5

test.flags

Source

[1] lerp article on Wikipedia

3.2 mesh

3.2.1 mesh1d

lerp.**mesh1d**
alias of BreakPoints

3.2.2 mesh2d

class lerp.**mesh2d**(*x=[]*, *d=None*, *x_label=None*, *x_unit=None*, *label=None*, *unit=None*, *clipboard=False*, *extrapolate=True*, *contiguous=False*, *step=False*, ***kwargs*)
Fundamental 2D object, strict monotonic

Instantiation by giving (x, d) parameters or by loading a csv-file.

Parameters

- **x** (*numpy.array or mesh1d*) – 1D array of x-coordinates of the mesh on which to interpolate
- **d** (*numpy.array*) – 1D array of d-coordinates of the mesh on result to be interpolated
- **options** (*dict [optiona]*) –
- **clipboard** (*boolean [optionall]*) – when set, override any instantiation with x and d
- **fileName** (*string*) – Complete address to csv-file, further

Notes

Currently supported features:

- calling the object *cur(x)* return the interpolated value at x.
- common operations: +, -, , /
- standard functions:func:*len()*, `print()`

T

apply(*f*, *axis='d'*, *inplace=False*)
Apply a function along axis

Parameters

- **f** (*function*) –
- **axis** (*string*) – "x" or "d"
- **inplace** (*boolean*) – True if you want the mesh1d to be modified inplace

Returns Depends if inplace is set to False or True

Return type Nothing or *mesh1d*

d
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

diff(*n=1*)
Checked

difff

dropnan()
Drop NaN values and return new mesh2d.

extrapolate(*x*, **args*, ***kwargs*)
np.interp function with linear extrapolation np.polyfit np.poly1d

gradient(*x=None*)

interpolate(*x*, *assume_sorted=False*, **args*, ***kwargs*)
Purpose of this method is to return a linear interpolation of a d vector for an unknown value x. If the targeted value is out of the x range, the returned d-value is the first, resp. the last d-value.

No interpolation is made out of the x definition range. For such a functionality, use:py:meth:*extrapolate* instead.

:param x:: iterable or single element,: :param kind: :type kind: str or int, optional :param Specifies the kind of interpolation as a string (‘linear’, ‘nearest’,: :param ‘zero’, ‘slinear’, ‘quadratic’, ‘cubic’ where ‘slinear’, ‘quadratic’: :param and ‘cubic’ refer to a spline interpolation of first, second or third: :param order) or as an integer specifying the order of the spline: :param interpolator to use. Default is ‘linear’.:

Returns

- A single element or a `numpy.array` if the x parameter was
- a `numpy.array` or a list

plot(*data*, **args*, ***kwargs*)

polyfit(*degree=2*)

push(*x=None*, *d=None*)
Pushes an element/array to the array

Notes

The element or the array is added and sorted inplace

Parameters

- **x** (*single numeric, array, numpy.array*) –
- **d** (*single numeric, array, numpy.array*) –

read_clipboard()

resample(*x*)

steps

to_clipboard(*transpose=False*, *decimal=’, ’*)

to_csv(*fileName=None*, *nbreDecimales=2*)
Export CUR data into csv

Parameters

- **fileName** (*String*) – Complete path + filename where csv data will be wrote. Default to ‘C:/temp/aze.csv’
- **nbreDecimales** (*integer*) –

x
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

3.2.3 mesh3d

class lerp.**mesh3d**(*x=[]*, *y=[]*, *d=None*, *x_label=None*, *x_unit=None*, *y_label=None*, *y_unit=None*, *label=None*, *unit=None*, *extrapolate=True*, *clipboard=False*, *sort=True*, **pargs*, ***kwargs*)
Interpolate over a 2-D grid.

x, *y* and *d* are arrays of values used to approximate some function f: `d = f(x, y)`. This class returns a function whose call method uses spline interpolation to find the value of new points.

Parameters

- **x** (*array_like*) –

- **Y** (*array_like*) – Arrays defining the data point coordinates.

  If the points lie on a regular grid, *x* can specify the column coordinates and *Y* the row coordinates

Examples

Construct a 2-D grid and interpolate on it:

```
from scipy import interpolate
x = np.arange(-5.01, 5.01, 0.25)
y = np.arange(-5.01, 5.01, 0.25)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**2+yy**2)
```

T

apply(*f*, *inplace=False*)

d

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

diff(*axis=0*, *n=1*)

extrapolate(*x*, *y*)

from_pandas(*obj*)

interpolate(*x=None*, *y=None*)

plot(*xy=False*, *filename=None*, ***kwargs*)

pop(*axis=0*)

push(*s=None*, *d=None*, *axis=0*, *inplace=False*)

read_clipboard()

reshape(*sort=True*)

sort()

to_gpt(*fileName=None*)

x

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

y

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

3.2.4 mesh4d

class lerp.**mesh4d**(*x=[]*, *y=[]*, *z=[]*, *d=None*, *x_label=None*, *x_unit=None*, *y_label=None*, *y_unit=None*, *z_label=None*, *z_unit=None*, *label=None*, *unit=None*, *extrapolate=False*, *dtype='float64'*)

d
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

interpolate(*x=None*, *y=None*, *z=None*)
a

push(*s=None*, *d=None*, *axis=0*)

read_pickle(*fileName=None*)

reshape()

shape

sort()

to_pickle(*fileName=None*)

x
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

y
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

z
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

3.2.5 mesh5d

class lerp.**mesh5d**(*x=[]*, *y=[]*, *z=[]*, *v=[]*, *d=None*, *x_label=None*, *x_unit=None*, *y_label=None*, *y_unit=None*, *z_label=None*, *z_unit=None*, *v_label=None*, *v_unit=None*, *label=None*, *unit=None*, *extrapolate=True*, *dtype='float64'*)

d
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

read_pickle(*fileName=None*)

reshape()

shape

sort()

to_pickle(*fileName=None*)

v
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

x
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

y
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

z
partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

3.3 polymesh

3.3.1 polymesh2d

class lerp.**polymesh2d**(*p=[]*, *x_label=None*, *x_unit=None*, *label=None*, *unit=None*)
Polynom based mash support.

plot(**pargs*, ***kwargs*)

resample(*x*)

3.3.2 polymesh3d

class lerp.**polymesh3d**(*x_label=None*, *x_unit=None*, *y_label=None*, *y_unit=None*, *label=None*, *unit=None*)

plot(**pargs*, ***kwargs*)

push(*y*, *p*)

resample(*y*)

x

y
Describe the highest coefficent

3.4 API

Apply to all mesh objects:

mesh.max
mesh.mean
mesh.median
mesh.min
mesh.shape
mesh.read_pickle
mesh.to_pickle

3.4.1 mesh2d

mesh2d.apply(f[, axis, inplace])	Apply a function along axis
mesh2d.diff([n])	Checked
mesh2d.dropnan()	Drop NaN values and return new mesh2d.
mesh2d.extrapolate(x, *args, **kwargs)	np.interp function with linear extrapolation

Continued on next page

Table 3.2 – continued from previous page

mesh2d.gradient([x])	
mesh2d.interpolate(x[, assume_sorted])	Purpose of this method is to return a linear interpolation of a d vector for an unknown value x.
mesh2d.plot(data, *args, **kwargs)	
mesh2d.polyfit([degree])	
mesh2d.push([x, d])	Pushes an element/array to the array
mesh2d.read_clipboard()	
mesh2d.resample(x)	
mesh2d.step	
mesh2d.steps	
mesh2d.to_clipboard([transpose, decimal])	
mesh2d.to_csv([fileName, nbreDecimales])	Export CUR data into csv

3.4.2 mesh3d

mesh3d.apply(f[, inplace])	
mesh3d.diff([axis, n])	
mesh3d.extrapolate(x, y)	
mesh3d.from_pandas(obj)	
mesh3d.interpolate([x, y])	
mesh3d.plot([xy, filename])	
mesh3d.pop([axis])	
mesh3d.push([s, d, axis, inplace])	
mesh3d.read_clipboard()	
mesh3d.reshape([sort])	
mesh3d.sort()	

3.4.3 mesh4d

mesh4d.interpolate([x, y, z])	a
mesh4d.push([s, d, axis])	
mesh4d.reshape()	
mesh4d.sort()	

3.4.4 polymesh2d

polymesh2d.resample(x)	

3.4.5 polymesh3d

polymesh3d.resample(y)	
polymesh3d.plot(*pargs, **kwargs)	
polymesh3d.push(y, p)	

CHAPTER 4

Indices and tables

- genindex

Python Module Index

l

lerp, 29

Index

A

apply() (lerp.mesh2d method), 25
apply() (lerp.mesh3d method), 27

D

d (lerp.mesh2d attribute), 25
d (lerp.mesh3d attribute), 27
d (lerp.mesh4d attribute), 27
d (lerp.mesh5d attribute), 28
diff() (lerp.mesh2d method), 25
diff() (lerp.mesh3d method), 27
difff (lerp.mesh2d attribute), 25
dropnan() (lerp.mesh2d method), 25

E

extrapolate() (lerp.mesh2d method), 25
extrapolate() (lerp.mesh3d method), 27

F

from_pandas() (lerp.mesh3d method), 27

G

gradient() (lerp.mesh2d method), 25

I

interpolate() (lerp.mesh2d method), 25
interpolate() (lerp.mesh3d method), 27
interpolate() (lerp.mesh4d method), 28

L

lerp (module), 24, 29

M

mesh1d (in module lerp), 24
mesh2d (class in lerp), 24
mesh3d (class in lerp), 26
mesh4d (class in lerp), 27
mesh5d (class in lerp), 28

P

plot() (lerp.mesh2d method), 26
plot() (lerp.mesh3d method), 27
plot() (lerp.polymesh2d method), 29
plot() (lerp.polymesh3d method), 29
polyfit() (lerp.mesh2d method), 26
polymesh2d (class in lerp), 29
polymesh3d (class in lerp), 29
pop() (lerp.mesh3d method), 27
push() (lerp.mesh2d method), 26
push() (lerp.mesh3d method), 27
push() (lerp.mesh4d method), 28
push() (lerp.polymesh3d method), 29

R

read_clipboard() (lerp.mesh2d method), 26
read_clipboard() (lerp.mesh3d method), 27
read_pickle() (lerp.mesh4d method), 28
read_pickle() (lerp.mesh5d method), 28
resample() (lerp.mesh2d method), 26
resample() (lerp.polymesh2d method), 29
resample() (lerp.polymesh3d method), 29
reshape() (lerp.mesh3d method), 27
reshape() (lerp.mesh4d method), 28
reshape() (lerp.mesh5d method), 28

S

shape (lerp.mesh4d attribute), 28
shape (lerp.mesh5d attribute), 28
sort() (lerp.mesh3d method), 27
sort() (lerp.mesh4d method), 28
sort() (lerp.mesh5d method), 28
steps (lerp.mesh2d attribute), 26

T

T (lerp.mesh2d attribute), 25
T (lerp.mesh3d attribute), 27
to_clipboard() (lerp.mesh2d method), 26
to_csv() (lerp.mesh2d method), 26

to_gpt() (lerp.mesh3d method), 27
to_pickle() (lerp.mesh4d method), 28
to_pickle() (lerp.mesh5d method), 28

V

v (lerp.mesh5d attribute), 28

X

x (lerp.mesh2d attribute), 26
x (lerp.mesh3d attribute), 27
x (lerp.mesh4d attribute), 28
x (lerp.mesh5d attribute), 28
x (lerp.polymesh3d attribute), 29

Y

y (lerp.mesh3d attribute), 27
y (lerp.mesh4d attribute), 28
y (lerp.mesh5d attribute), 28
y (lerp.polymesh3d attribute), 29

Z

z (lerp.mesh4d attribute), 28
z (lerp.mesh5d attribute), 28