kneed

Release 0.6.0

Jul 30, 2022

Contents

This is the documentation for the kneed Python package. Given *x* and *y* arrays, *kneed* attempts to identify the knee/elbow point of a line fit to the data. The knee/elbow is defined as the point of the line with maximum curvature. For more information about how each of the parameters affect identification of knee points, check out *Parameter Examples*. For a full reference of the API, head over to the *API Reference*.

CHAPTER 1

Parameter Examples

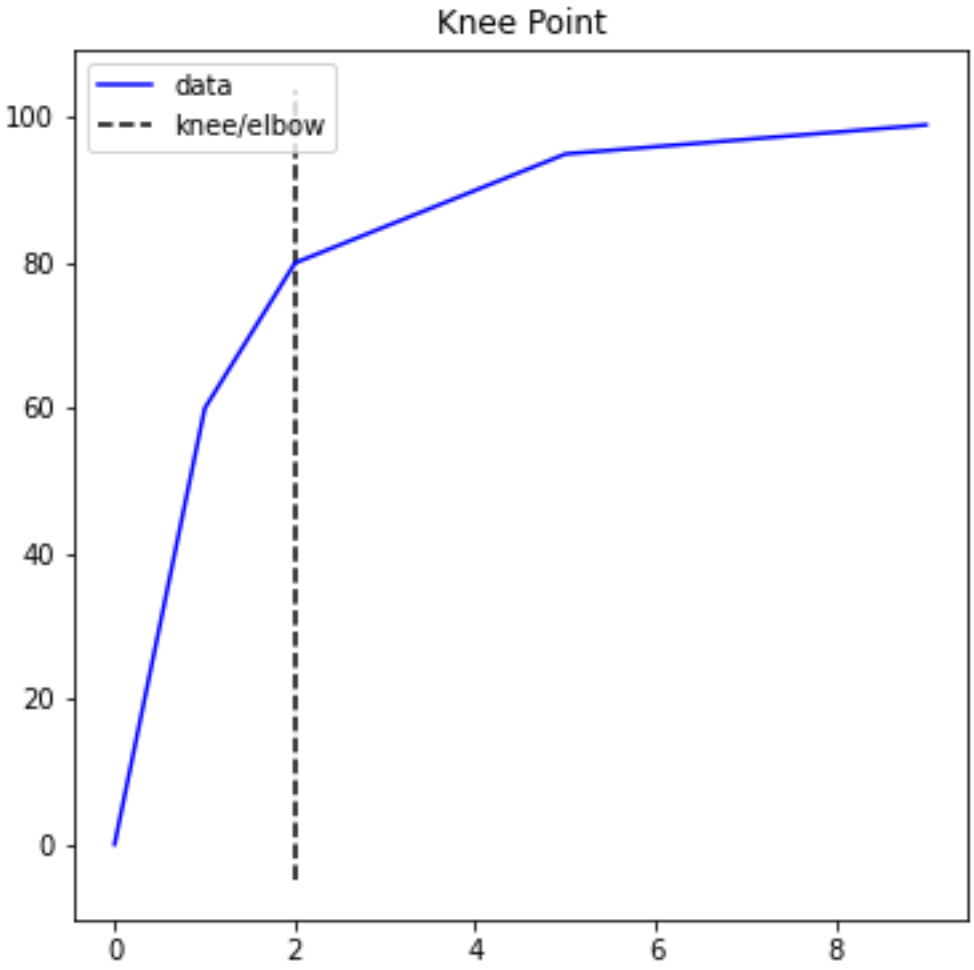
This page provides examples that outline the effect of tuning the parameters for *KneeLocator*.

1.1 curve

If *curve="concave"*, kneed will detect knees. If *curve="convex"*, it will detect elbows. Use the *DataGenerator* class to generate synthetic data:

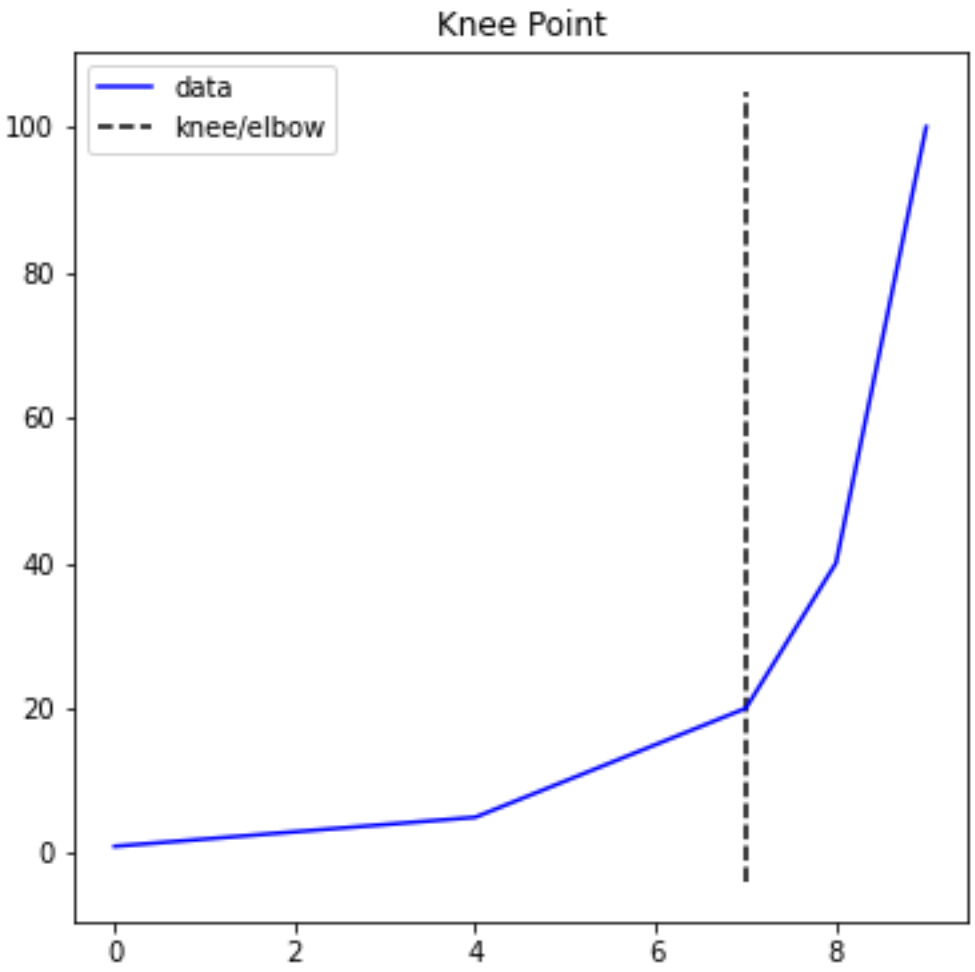
An concave curve example:

```
from kneed import KneeLocator, DataGenerator as dg
x, y = dg.concave_increasing()
kl = KneeLocator(x, y, curve="concave")
kl.plot_knee()
```



A convex curve example:

```
from kneed import KneeLocator, DataGenerator as dg
x, y = dg.convex_increasing()
kl = KneeLocator(x, y, curve="convex")
kl.plot_knee()
```

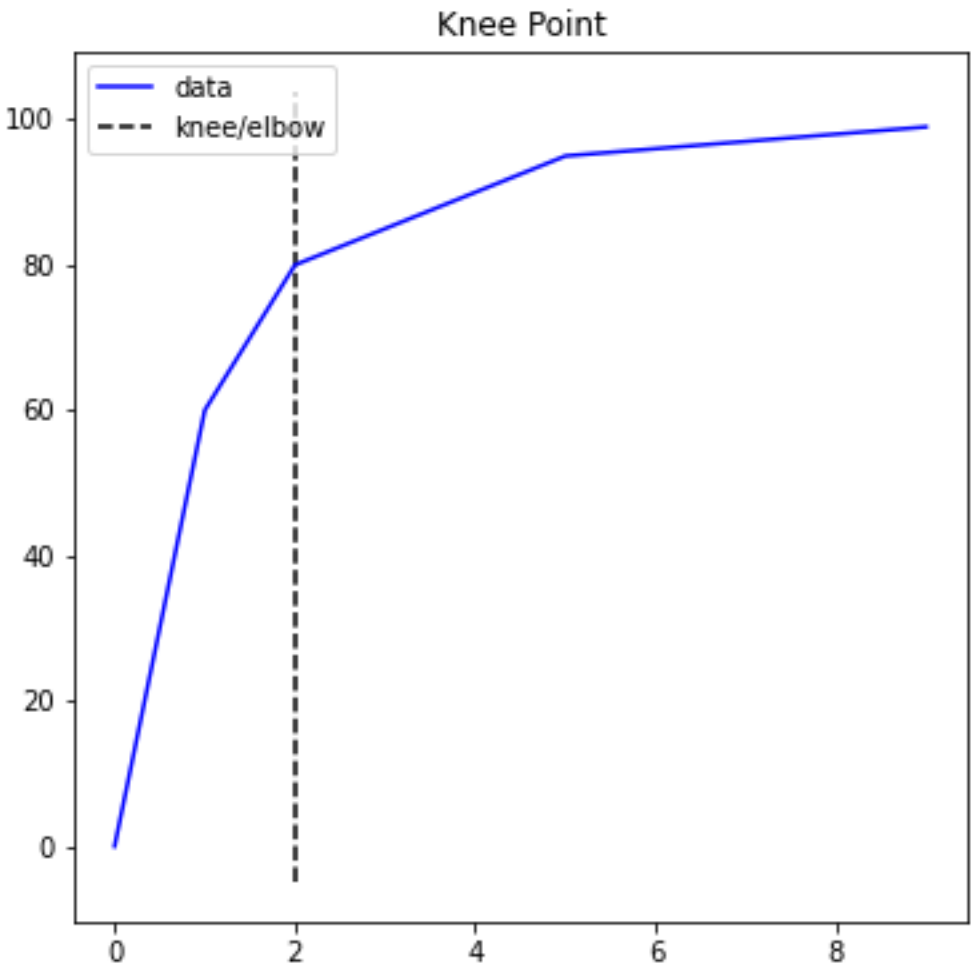


1.2 direction

The direction parameter describes the line from left to right on the x-axis. If the knee/elbow you are trying to identify is on a positive slope use *direction="increasing"*, if the knee/elbow you are trying to identify is on a negative slope, use *direction="decreasing"*. Use the *DataGenerator* class to generate synthetic data.

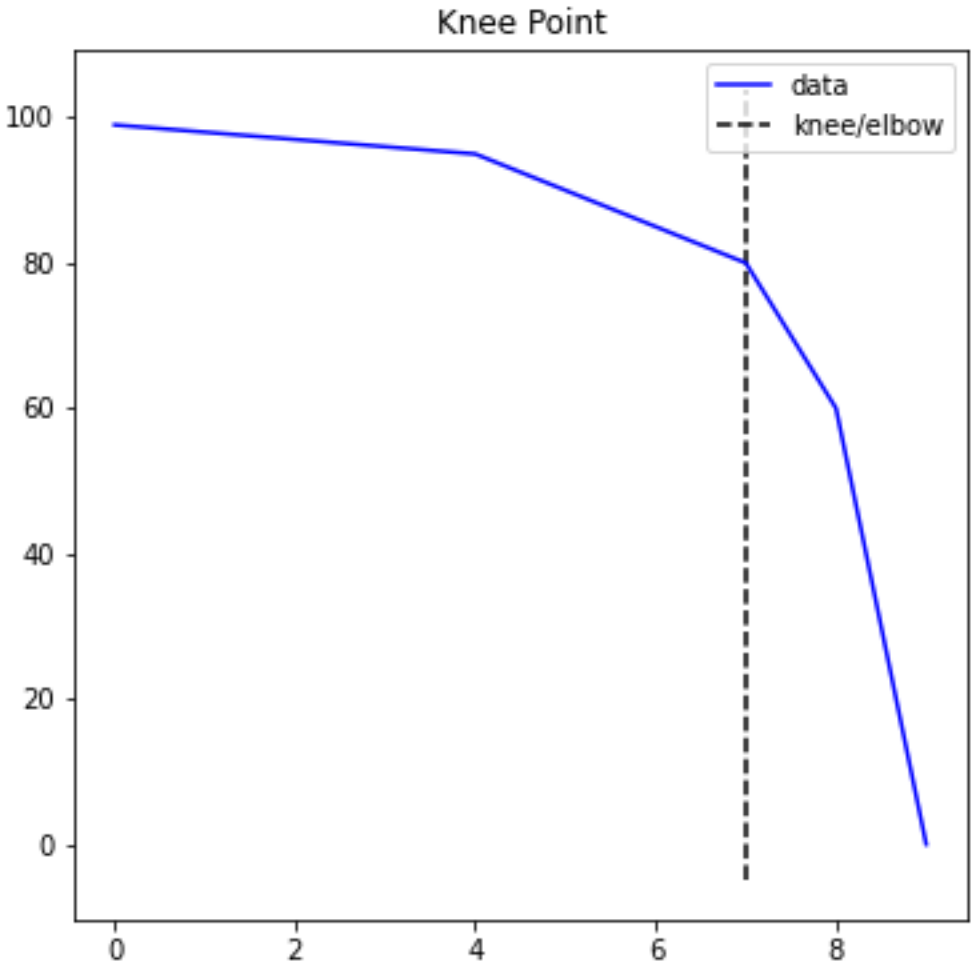
An example of an increasing curve:

```
from kneed import KneeLocator, DataGenerator as dg
x, y = dg.concave_increasing()
kl = KneeLocator(x, y, curve="concave", direction="increasing")
kl.plot_knee()
```



An example of a decreasing curve:

```
from kneed import KneeLocator, DataGenerator as dg
x, y = dg.concave_decreasing()
kl = KneeLocator(x, y, curve="concave", direction="decreasing")
kl.plot_knee()
```



1.3 S

The selected knee point is tunable by setting the sensitivity parameter *S*. From the kneedle manuscript:

> The sensitivity parameter allows us to adjust how aggressive we want Kneedle to be when detecting knees. Smaller values for S detect knees quicker, while larger values are more conservative. Put simply, S is a measure of how many "flat" points we expect to see in the unmodified data curve before declaring a knee.

```
import numpy as np

np.random.seed(23)

sensitivity = [1, 3, 5, 10, 100, 200, 400]
knees = []
norm_knees = []
```

(continues on next page)

(continued from previous page)

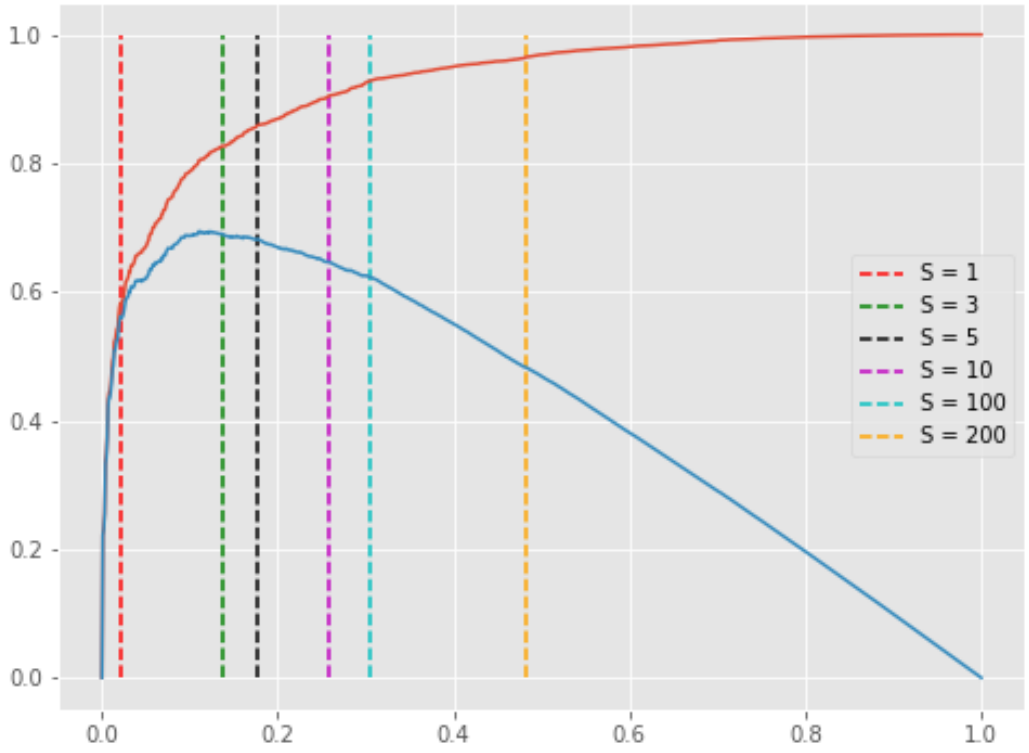
```
n = 1000
x = range(1, n + 1)
y = sorted(np.random.gamma(0.5, 1.0, n), reverse=True)
for s in sensitivity:
    kl = KneeLocator(x, y, curve="convex", direction="decreasing", S=s)
    knees.append(kl.knee)
    norm_knees.append(kl.norm_knee)

print(knees)
[43, 137, 178, 258, 305, 482, 482]

print([nk.round(2) for nk in norm_knees])
[0.04, 0.14, 0.18, 0.26, 0.3, 0.48, 0.48]

import matplotlib.pyplot as plt

plt.style.use("ggplot")
plt.figure(figsize=(8, 6))
plt.plot(kl.x_normalized, kl.y_normalized)
plt.plot(kl.x_difference, kl.y_difference)
colors = ["r", "g", "k", "m", "c", "orange"]
for k, c, s in zip(norm_knees, colors, sensitivity):
    plt.vlines(k, 0, 1, linestyles="--", colors=c, label=f"S = {s}")
plt.legend()
```



Any *S*>200 will result in a knee at 482 (0.48, normalized) in the plot above.

1.4 online

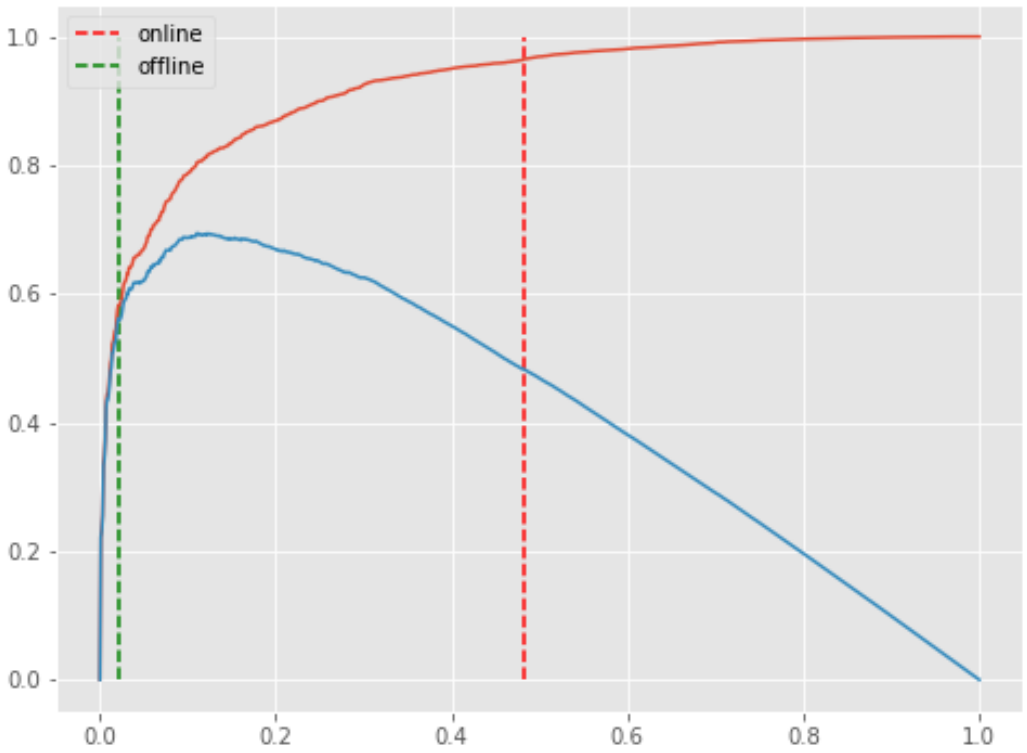
The knee point can be corrected if the parameter online is *True* (default). This mode will step through each element in x. In contrast, if online is False, kneed will run in offline mode and return the first knee point identified. When *online=False* the first knee point identified is returned regardless of whether it's the local maxima on the difference curve or the global maxima. So the algorithm stops early. When *online=True*, kneed runs in online mode and "corrects" itself by continuing to overwrite any previously identified knees.

Using the *x* and *y* from the sensitivity example above, this time, keep *S=1* but modify *online*.

```
kl_online = KneeLocator(x, y, curve="convex", direction="decreasing", online=True)
kl_offline = KneeLocator(x, y, curve="convex", direction="decreasing", online=False)

import matplotlib.pyplot as plt

plt.style.use("ggplot")
plt.figure(figsize=(8, 6))
plt.plot(kl_online.x_normalized, kl_online.y_normalized)
plt.plot(kl_online.x_difference, kl_online.y_difference)
colors = ["r", "g"]
for k, c, o in zip(
    [kl_online.norm_knee, kl_offline.norm_knee], ["r", "g"], ["online", "offline"]
):
    plt.vlines(k, 0, 1, linestyles="--", colors=c, label=o)
plt.legend()
```



1.5 interp_method

This parameter controls the interpolation method for fitting a spline to the input *x* and *y* data points. Valid arguments are *"interp1d"* and *"polynomial"*.

If *interp_method="interp1d"*, then *x* and *y* will be fit using scipy.interpolate.interp1d.

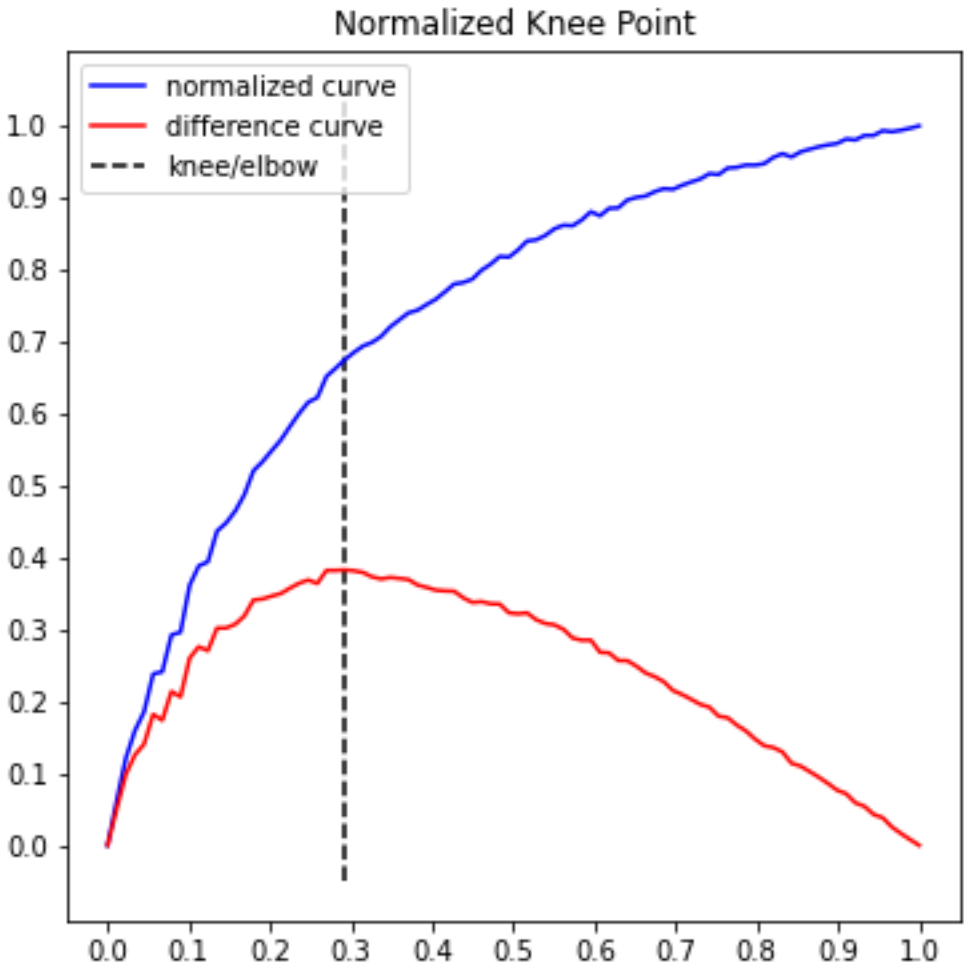
```
x = list(range(90))
y = [
    7304, 6978, 6666, 6463, 6326, 6048, 6032, 5762, 5742,
    5398, 5256, 5226, 5001, 4941, 4854, 4734, 4558, 4491,
    4411, 4333, 4234, 4139, 4056, 4022, 3867, 3808, 3745,
    3692, 3645, 3618, 3574, 3504, 3452, 3401, 3382, 3340,
    3301, 3247, 3190, 3179, 3154, 3089, 3045, 2988, 2993,
    2941, 2875, 2866, 2834, 2785, 2759, 2763, 2720, 2660,
    2690, 2635, 2632, 2574, 2555, 2545, 2513, 2491, 2496,
    2466, 2442, 2420, 2381, 2388, 2340, 2335, 2318, 2319,
    2308, 2262, 2235, 2259, 2221, 2202, 2184, 2170, 2160,
    2127, 2134, 2101, 2101, 2066, 2074, 2063, 2048, 2031
]

kneedle = KneeLocator(
    x, y, S=1.0, curve="convex", direction="decreasing", interp_method="interp1d"
```

(continues on next page)

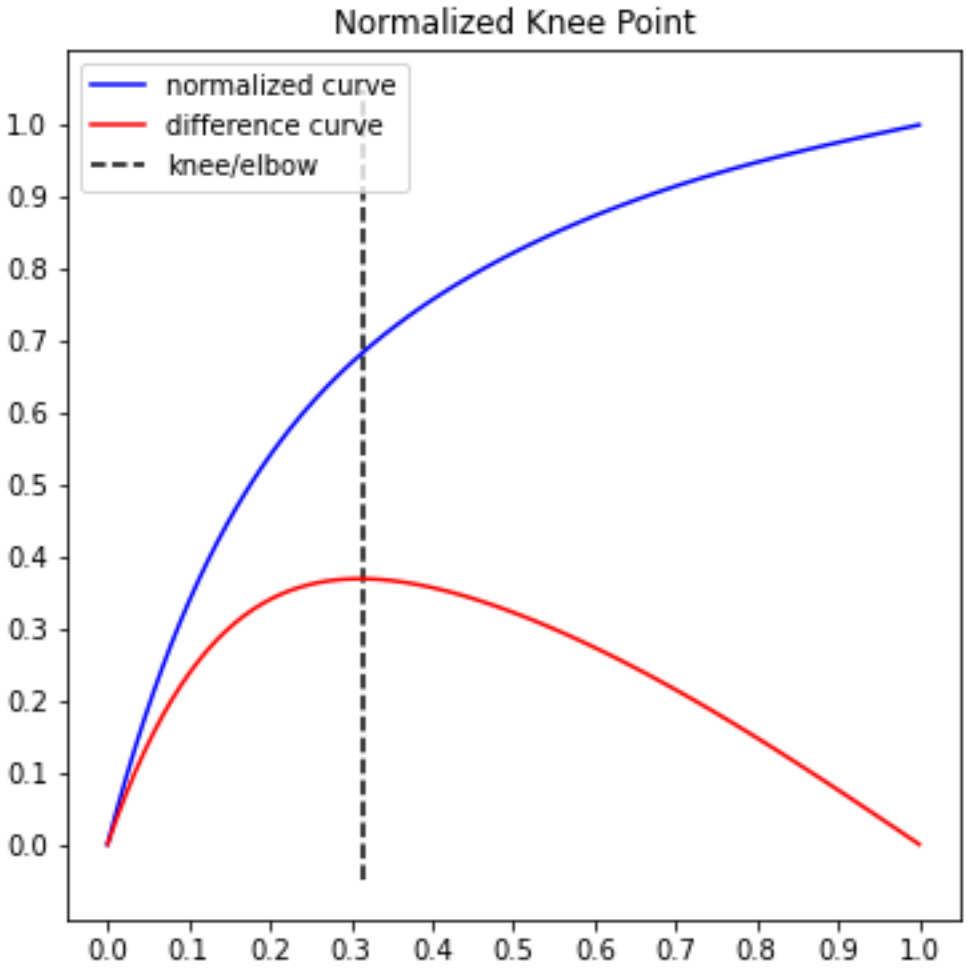
(continued from previous page)

```
)
kneedle.plot_knee_normalized()
```



If *interp_method="polynomial"*, then *x* and *y* will be fit using numpy.polyfit. Using the same data, change *interp_method* and note that the line is smoother.

```
kneedle = KneeLocator(
    x, y, S=1.0, curve="convex", direction="decreasing", interp_method="polynomial",
)
kneedle.plot_knee_normalized()
```

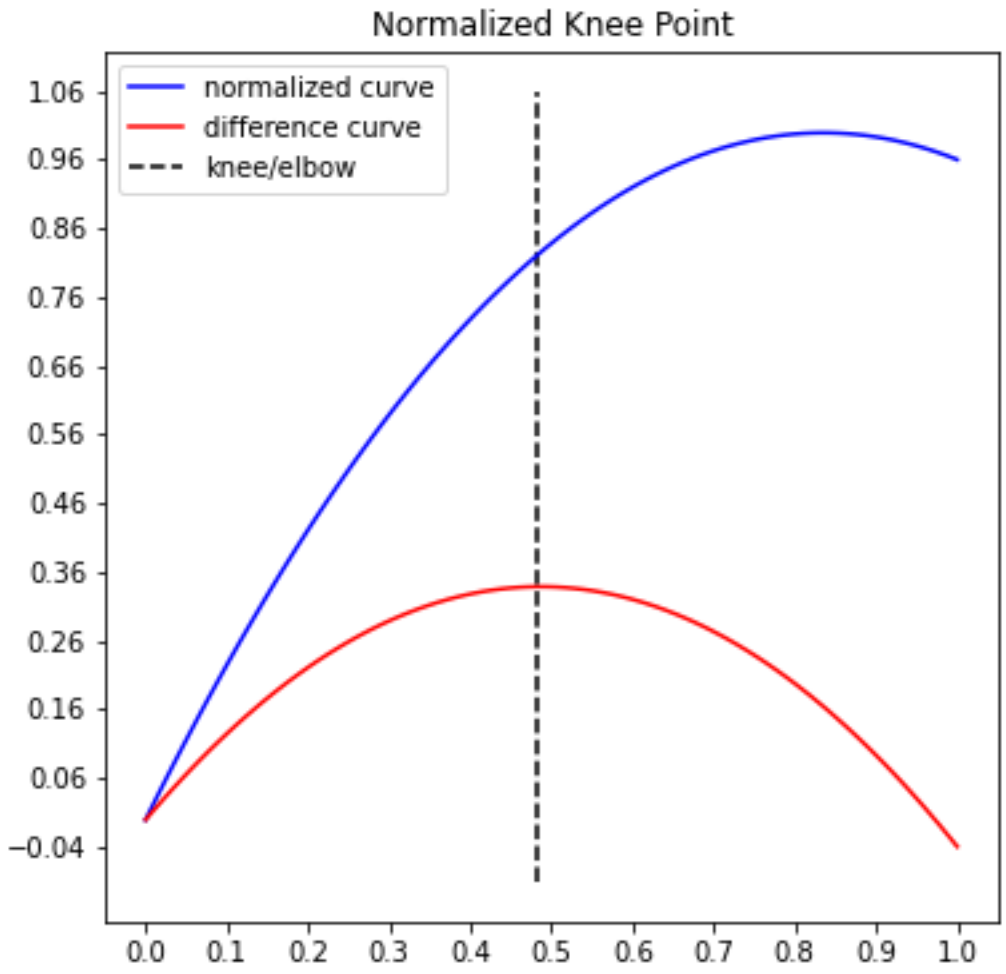


1.6 polynomial_degree

This parameter controls the degree of the polynomial fit. This parameter is passed as the argument to the *deg* parameter in numpy.polyfit.

Using the same data from the *interp_method* example, note how the line (and knee point) change when *polynomial_degree=2* instead of the default value, *7*:

```
kneedle = KneeLocator(
    x, y, S=1.0, curve="convex", direction="decreasing", interp_method="polynomial",
    polynomial_degree=2
)
kneedle.plot_knee_normalized()
```


Normalized Knee Point
normalized curve
difference curve
knee/elbow
1.06
0.96
0.86
0.76
0.66
0.56
0.46
0.36
0.26
0.16
0.06
−0.04
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

CHAPTER 2

API Reference

There are two classes in *kneed*: *KneeLocator* identifies the knee/elbow point(s) and and *DataGenerator* creates synthetic *x* and *y* numpy arrays to explore *kneed*.

2.1 KneeLocator

class kneed.knee_locator.**KneeLocator**(*x: Iterable[float], y: Iterable[float], S: float = 1.0, curve: str = 'concave', direction: str = 'increasing', interp_method: str = 'interp1d', online: bool = False, polynomial_degree: int = 7*)

Once instantiated, this class attempts to find the point of maximum curvature on a line. The knee is accessible via the *.knee* attribute.

Parameters

- **x** (*array-like*) – x values.
- **y** (*array-like*) – y values.
- **S** (*float*) – Sensitivity, original paper suggests default of 1.0
- **curve** (*str*) – If 'concave', algorithm will detect knees. If 'convex', it will detect elbows.
- **direction** (*str*) – one of {"increasing", "decreasing"}
- **interp_method** (*str*) – one of {"interp1d", "polynomial"}
- **online** (*bool*) – kneed will correct old knee points if True, will return first knee if False
- **polynomial_degree** (*int*) – The degree of the fitting polynomial. Only used when interp_method="polynomial". This argument is passed to numpy polyfit *deg* parameter.

Variables

- **x** (*array-like*) – x values.
- **y** (*array-like*) – y values.

- **S** (*integer*) – Sensitivity, original paper suggests default of 1.0
- **curve** (*str*) – If 'concave', algorithm will detect knees. If 'convex', it will detect elbows.
- **direction** (*str*) – one of {"increasing", "decreasing"}
- **interp_method** (*str*) – one of {"interp1d", "polynomial"}
- **online** (*str*) – kneed will correct old knee points if True, will return first knee if False
- **polynomial_degree** (*int*) – The degree of the fitting polynomial. Only used when interp_method="polynomial". This argument is passed to numpy polyfit *deg* parameter.
- **N** (*integer*) – The number of *x* values in the
- **all_knees** (*set*) – A set containing all the x values of the identified knee points.
- **all_norm_knees** (*set*) – A set containing all the normalized x values of the identified knee points.
- **all_knees_y** (*list*) – A list containing all the y values of the identified knee points.
- **all_norm_knees_y** (*list*) – A list containing all the normalized y values of the identified knee points.
- **Ds_y** (*numpy array*) – The y values from the fitted spline.
- **x_normalized** (*numpy array*) – The normalized x values.
- **y_normalized** (*numpy array*) – The normalized y values.
- **x_difference** (*numpy array*) – The x values of the difference curve.
- **y_difference** (*numpy array*) – The y values of the difference curve.
- **maxima_indices** (*numpy array*) – The indices of each of the maxima on the difference curve.
- **maxima_indices** – The indices of each of the maxima on the difference curve.
- **x_difference_maxima** (*numpy array*) – The x values from the difference curve where the local maxima are located.
- **y_difference_maxima** (*numpy array*) – The y values from the difference curve where the local maxima are located.
- **minima_indices** (*numpy array*) – The indices of each of the minima on the difference curve.
- **minima_indices** – The indices of each of the minima on the difference curve.
- **x_difference_minima** (*numpy array*) – The x values from the difference curve where the local minima are located.
- **y_difference_minima** (*numpy array*) – The y values from the difference curve where the local minima are located.
- **Tmx** (*numpy array*) – The y values that correspond to the thresholds on the difference curve for determining the knee point.
- **knee** (*float*) – The x value of the knee point.
- **knee_y** (*float*) – The y value of the knee point.
- **norm_knee** (*float*) – The normalized x value of the knee point.
- **norm_knee_y** (*float*) – The normalized y value of the knee point.

- **all_knees** – The x values of all the identified knee points.
- **all_knees_y** – The y values of all the identified knee points.
- **all_norm_knees** – The normalized x values of all the identified knee points.
- **all_norm_knees_y** – The normalized y values of all the identified knee points.
- **elbow** (*float*) – The x value of the elbow point (elbow and knee are interchangeable).
- **elbow_y** (*float*) – The y value of the knee point (elbow and knee are interchangeable).
- **norm_elbow** – The normalized x value of the knee point (elbow and knee are interchangeable).
- **norm_elbow_y** (*float*) – The normalized y value of the knee point (elbow and knee are interchangeable).
- **all_elbows** (*set*) – The x values of all the identified knee points (elbow and knee are interchangeable).
- **all_elbows_y** – The y values of all the identified knee points (elbow and knee are interchangeable).
- **all_norm_elbows** (*set*) – The normalized x values of all the identified knee points (elbow and knee are interchangeable).
- **all_norm_elbowss_y** – The normalized y values of all the identified knee points (elbow and knee are interchangeable).

2.1.1 Plotting methods

There are two methods for basic visualizations of the knee/elbow point(s).

KneeLocator.**plot_knee**(*figsize: Optional[Tuple[int, int]] = None*)
Plot the curve and the knee, if it exists

Parameters **figsize** – Optional[Tuple[int, int] The figure size of the plot. Example (12, 8)

Returns NoReturn

KneeLocator.**plot_knee_normalized**(*figsize: Optional[Tuple[int, int]] = None*)
Plot the normalized curve, the difference curve (x_difference, y_normalized) and the knee, if it exists.

Parameters **figsize** – Optional[Tuple[int, int] The figure size of the plot. Example (12, 8)

Returns NoReturn

2.2 DataGenerator

class kneed.data_generator.**DataGenerator**
Generate synthetic data to work with kneed.

static bumpy() → Tuple[Iterable[float], Iterable[float]]
Generate a sample function with local minima/maxima.

Returns tuple(x, y)

static concave_decreasing() → Tuple[Iterable[float], Iterable[float]]
Generate a sample decreasing concave function.

Returns tuple(x, y)

static concave_increasing() → Tuple[Iterable[float], Iterable[float]]
Generate a sample increasing concave function.

Returns tuple(x, y)

static convex_decreasing() → Tuple[Iterable[float], Iterable[float]]
Generate a sample decreasing convex function.

Returns tuple(x, y)

static convex_increasing() → Tuple[Iterable[float], Iterable[float]]
Generate a sample increasing convex function.

Returns tuple(x, y)

static figure2() → Tuple[Iterable[float], Iterable[float]]
Recreate the values in figure 2 from the original kneedle paper.

Returns tuple(x, y)

static noisy_gaussian(*mu: float = 50*, *sigma: float = 10*, *N: int = 100*, *seed=42*) → Tuple[Iterable[float], Iterable[float]]
Recreate NoisyGaussian from the orignial kneedle paper.

Parameters

- **mu** – The mean value to build a normal distribution around
- **sigma** – The standard deviation of the distribution.
- **N** – The number of samples to draw from to build the normal distribution.
- **seed** – An integer to set the random seed.

Returns tuple(x, y)

CHAPTER 3

Interactive Streamlit App

An interactive streamlit app was developed to help users explore the effect of tuning the parameters. There are two sites where you can test out kneed by copy-pasting your own data:

1. https://share.streamlit.io/arvkevi/ikneed/main/ikneed.py
2. https://ikneed.herokuapp.com/

You can also run your own version – head over to the source code for ikneed.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index