Note

Click here to download the full example code

Interpolation

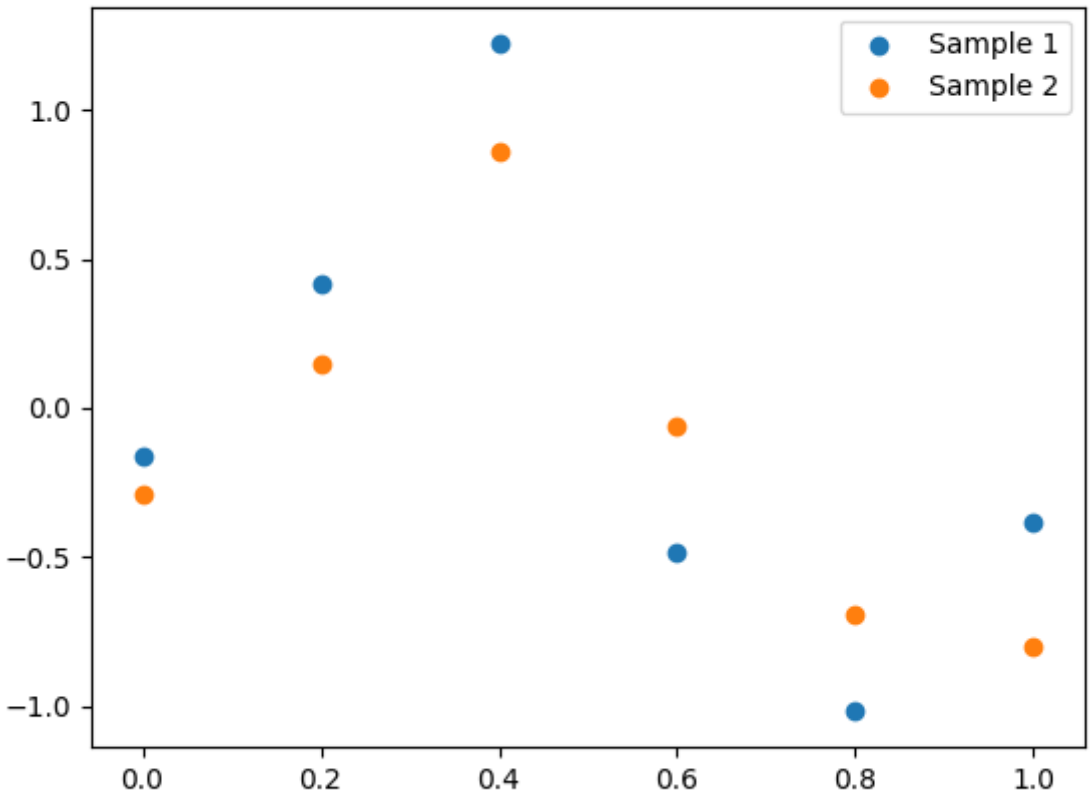
This example shows the types of interpolation used in the evaluation of FDataGrids.

```
# Author: Pablo Marcos Manchón
# License: MIT

# sphinx_gallery_thumbnail_number = 3

import skfda
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from skfda.representation.interpolation import SplineInterpolator
```

The `FDataGrid` class is used for datasets containing discretized functions. For the evaluation between the points of discretization, or sample points, is necessary to interpolate.

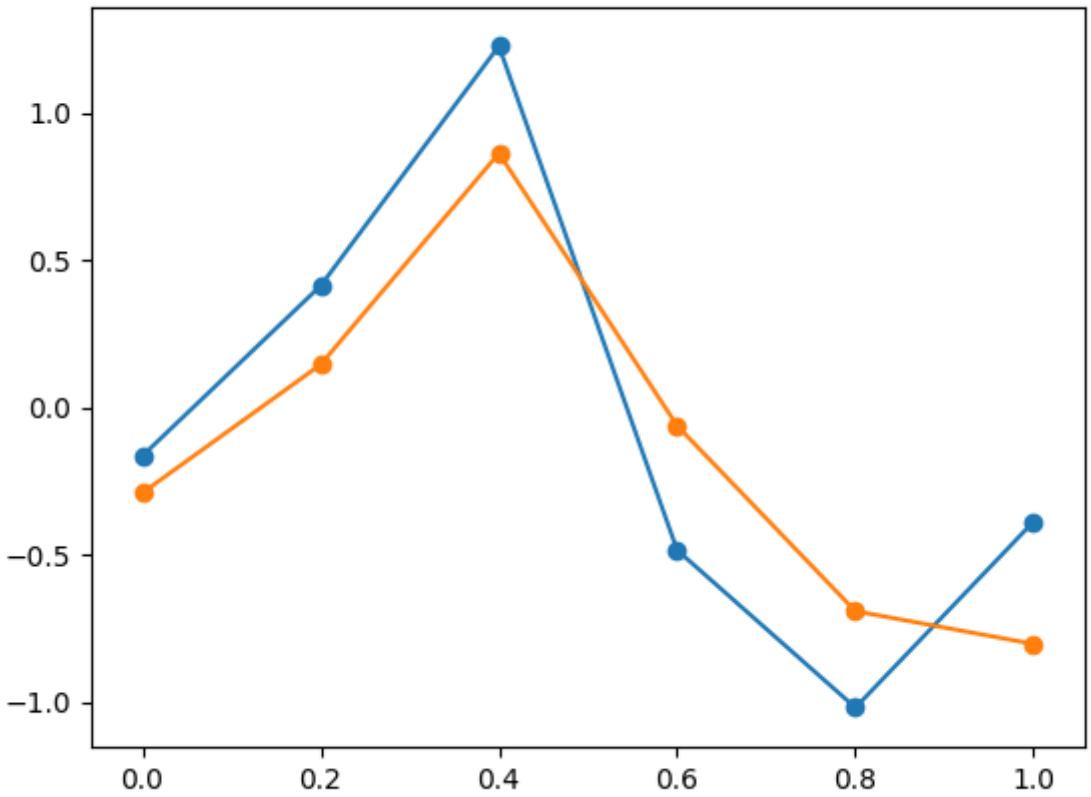
We will construct an example dataset with two curves with 6 points of discretization.

```
fd = skfda.datasets.make_sinusoidal_process(n_samples=2, n_features=6,
                                            random_state=1)
fd.scatter()
plt.legend(["Sample 1", "Sample 2"])
```



By default it is used linear interpolation, which is one of the simplest methods of interpolation and therefore one of the least computationally expensive, but has the disadvantage that the interpolant is not differentiable at the points of discretization.

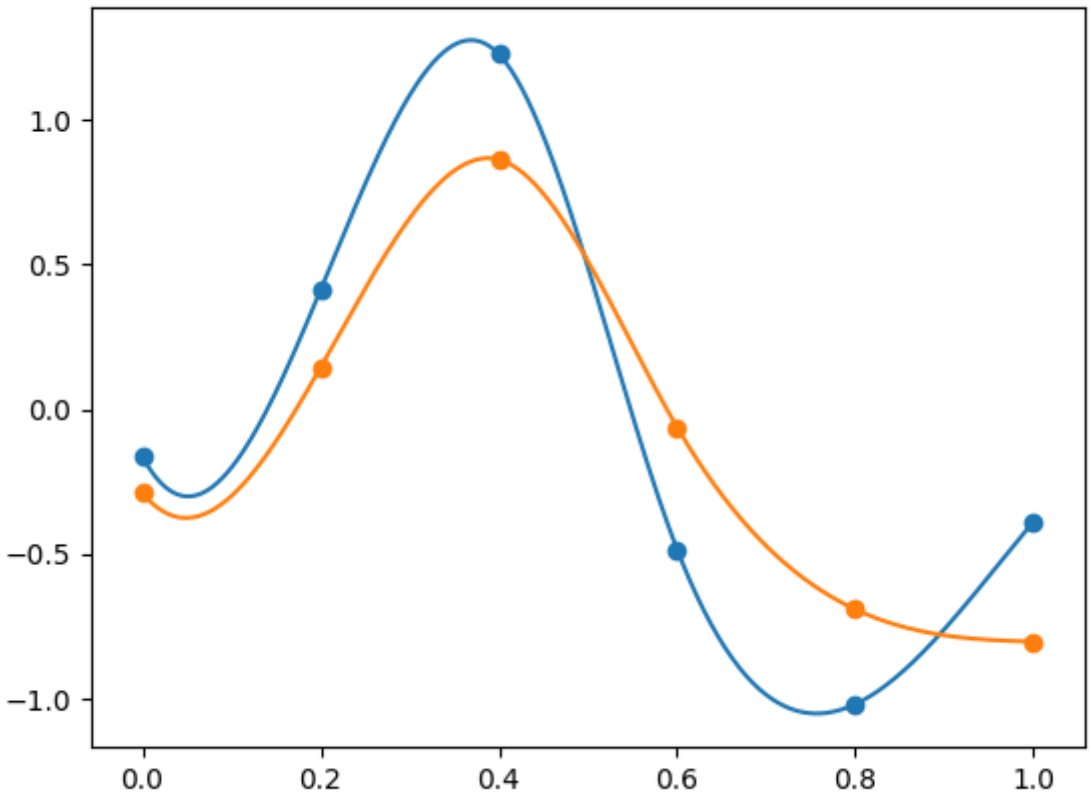
```
fd.plot()
fd.scatter()
```



The interpolation method of the FDataGrid could be changed setting the attribute *interpolator*. Once we have set an interpolator it is used for the evaluation of the object.

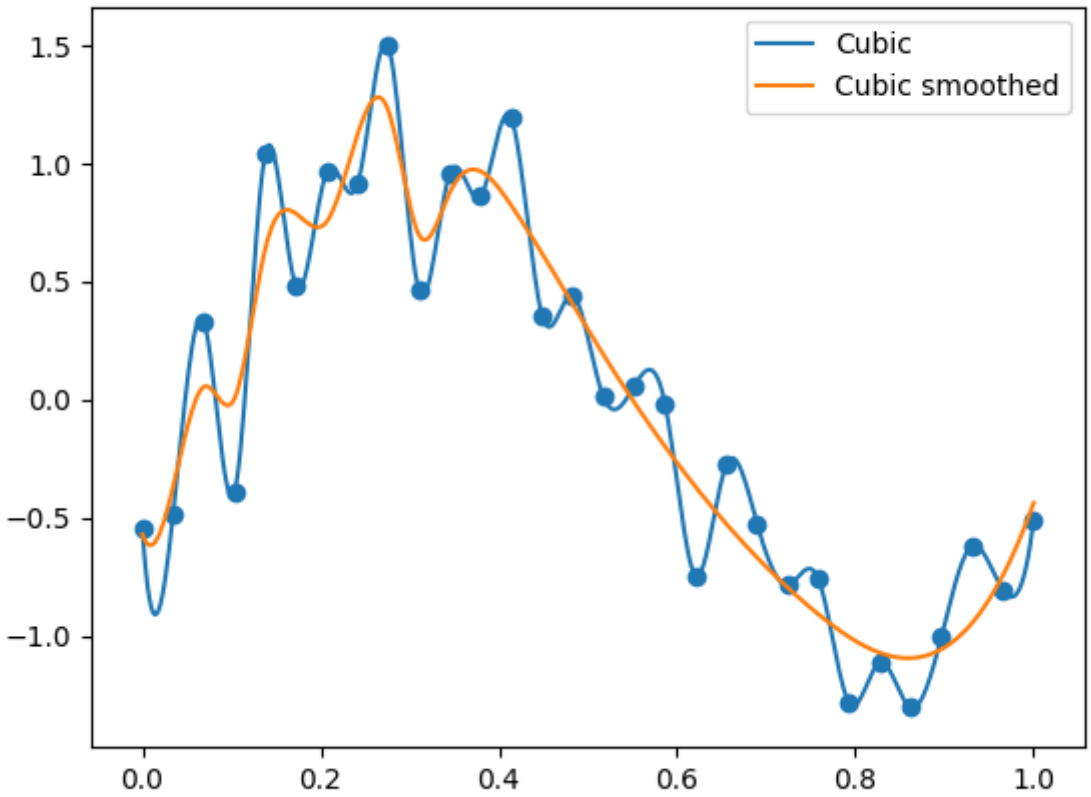
Polynomial spline interpolation could be performed using the interpolator `SplineInterpolator`. In the following example a cubic interpolator is set.

```
fd.interpolator = SplineInterpolator(interpolation_order=3)

fd.plot()
fd.scatter()
```



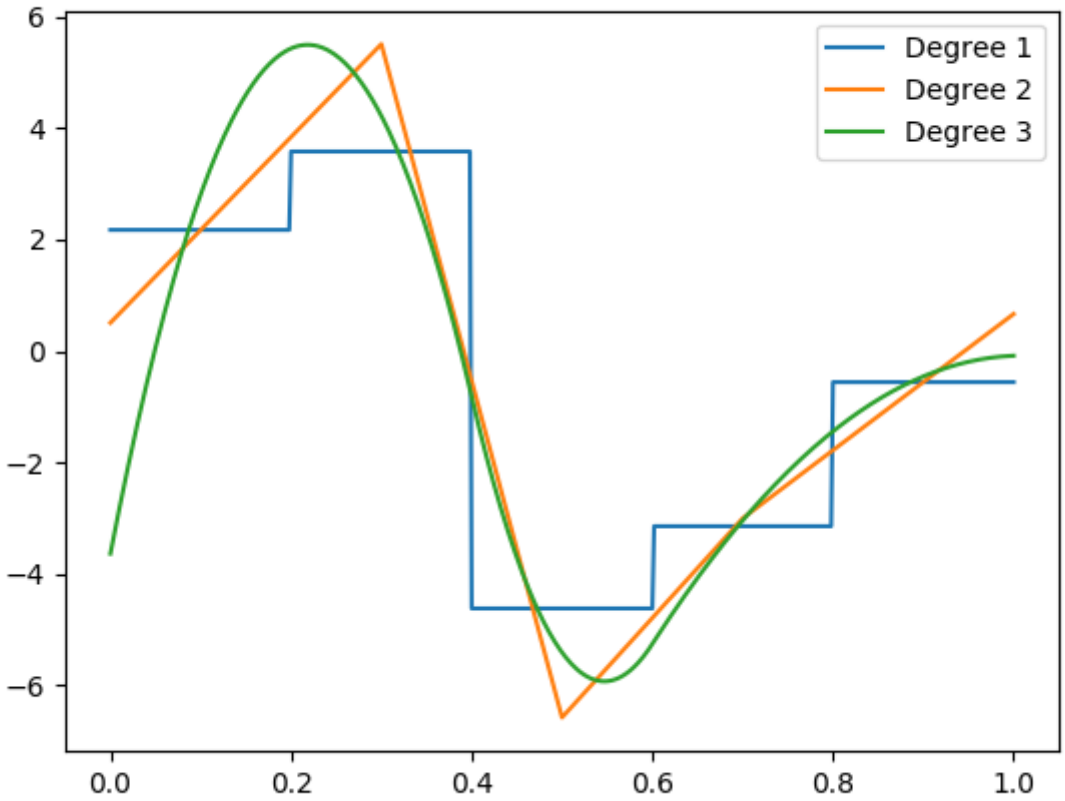
Smooth interpolation could be performed with the attribute *smoothness_parameter* of the spline interpolator.

```
# Sample with noise
fd_smooth = skfda.datasets.make_sinusoidal_process(n_samples=1, n_features=30,
                                                   random_state=1, error_std=.3)

# Cubic interpolator
fd_smooth.interpolator = SplineInterpolator(interpolation_order=3)

fd_smooth.plot(label="Cubic")

# Smooth interpolation
fd_smooth.interpolator = SplineInterpolator(interpolation_order=3,
                                            smoothness_parameter=1.5)

fd_smooth.plot(label="Cubic smoothed")

fd_smooth.scatter()
plt.legend()
```



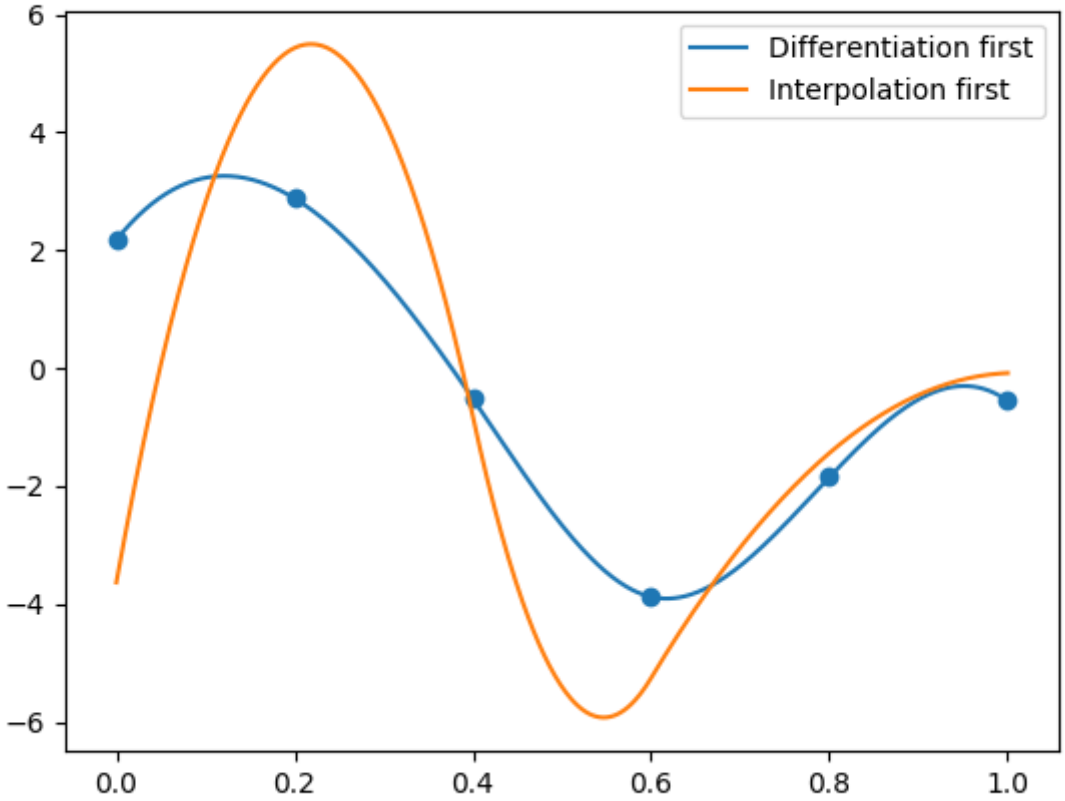
It is possible to evaluate derivatives of the FDatagrid, but due to the fact that interpolation is performed first, the interpolation loses one degree for each order of derivation. In the next example, it is shown the first derivative of a sample using interpolation with different degrees.

```
fd = fd[1]

for i in range(1, 4):
    fd.interpolator = SplineInterpolator(interpolation_order=i)
    fd.plot(derivative=1, label=f"Degree {i}")

plt.legend()
```



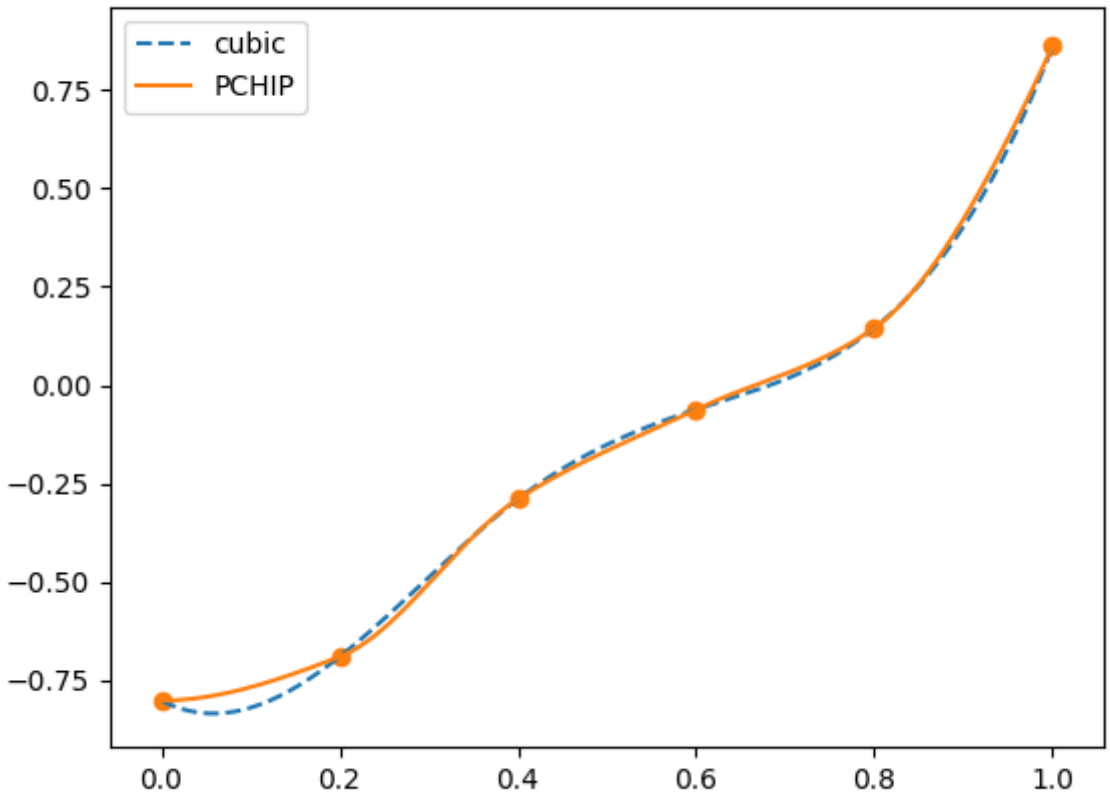
FDataGrids can be differentiate using lagged differences with the method `derivative()`, creating another FDataGrid which could be interpolated in order to avoid interpolating before differentiating.

```
fd_derivative = fd.derivative()

fd_derivative.plot(label="Differentiation first")
fd_derivative.scatter()

fd.plot(derivative=1, label="Interpolation first")

plt.legend()
```



Sometimes our samples are required to be monotone, in these cases it is possible to use monotone cubic interpolation with the attribute *monotone*. A piecewise cubic hermite interpolating polynomial (PCHIP) will be used.

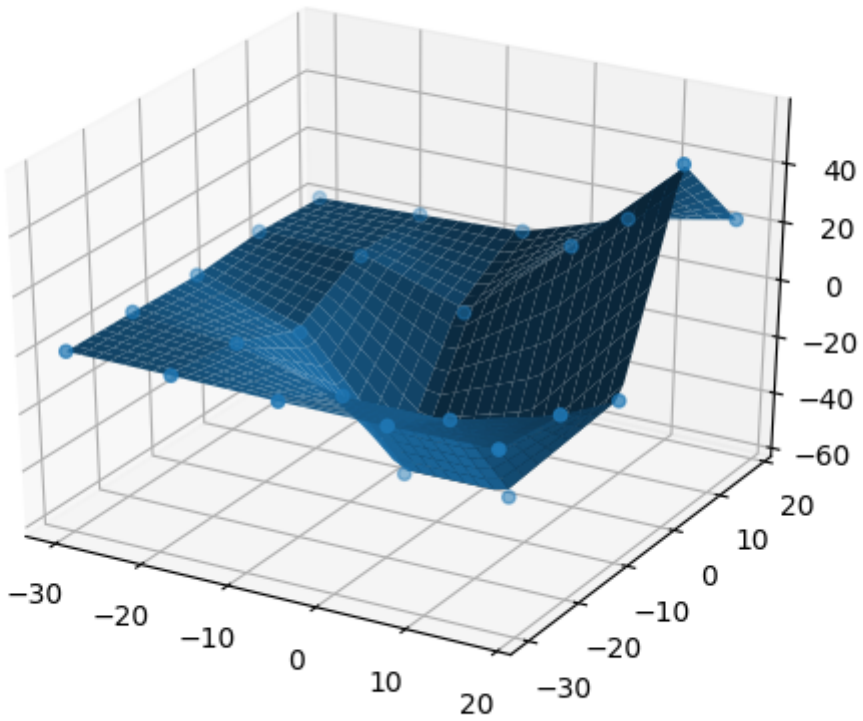
```
fd_monotone = fd.copy(data_matrix=np.sort(fd.data_matrix, axis=1))


fd_monotone.plot(linestyle='--', label="cubic")



fd_monotone.interpolator = SplineInterpolator(interpolation_order=3,
                                              monotone=True)
fd_monotone.plot(label="PCHIP")

fd_monotone.scatter(c='C1')
plt.legend()
```



All the interpolators will work regardless of the dimension of the image, but depending on the domain dimension some methods will not be available.

For the next examples it is constructed a surface, $x_i : \mathbb{R}^2 \longmapsto \mathbb{R}$. By default, as in unidimensional samples, it is used linear interpolation.

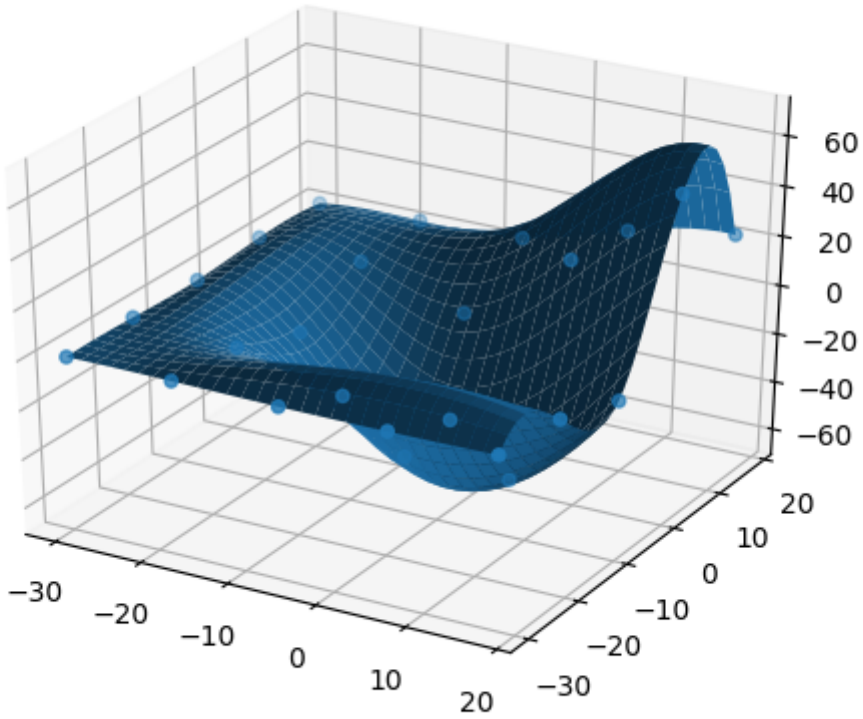
```
X, Y, Z = axes3d.get_test_data(1.2)
data_matrix = [Z.T]
sample_points = [X[0,:], Y[:, 0]]


fd = skfda.FDataGrid(data_matrix, sample_points)

fig, ax = fd.plot()
fd.scatter(ax=ax)
```



In the following figure it is shown the result of the cubic interpolation applied to the surface.

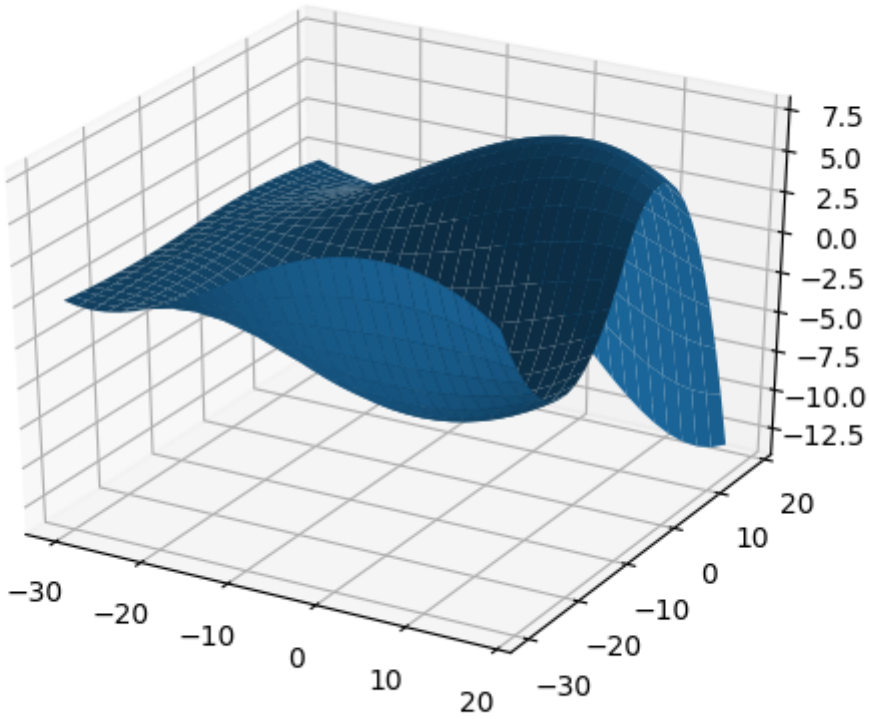
The degree of the interpolator polynomial does not have to coincide in both directions, for example, cubic interpolation in the first component and quadratic in the second one could be defined using a tuple with the values (3,2).

```
fd.interpolator = SplineInterpolator(interpolation_order=3)

fig, ax = fd.plot()
fd.scatter(ax=ax)

plt.show()
```



In case of surface derivatives could be taked in two directions, for this reason a tuple with the order of derivates in each direction could be passed. Let $x(t, s)$ be the surface, in the following example it is shown the derivative with respect to the second coordinate, $\frac{\partial}{\partial s}x(t, s)$.

```
fd.plot(derivative=(0, 1))

plt.show()
```



The following table shows the interpolation methods available by the class `SplineInterpolator` depending on the domain dimension.

Domain dimension	Linear	Up to degree 5	Monotone	Derivatives	Smoothing
1	✔	✔	✔	✔	✔
2	✔	✔	✖	✔	✔
3 or more	✔	✖	✖	✖	✖

Total running time of the script: (0 minutes 2.082 seconds)

Download Python source code: plot_interpolation.py

Download Jupyter notebook: plot_interpolation.ipynb