I'm human

Intercursation template python

The technique of determining intermediate function values between two known points is called Linear Interpolation. This method estimates an unknown value that falls within two known values and is used in various fields such as statistics, economics, and price determination to fill gaps in data for continuity. The formula to linearly interpolate a given data point is: y = y1 + (x - x1) * ((y2 - y1) / (x2 - x1)), where (x1, y1) and (x2, y2) are the coordinates of two adjacent points, and x is the point at which interpolation is performed. For example, given a dataset of numbers and their square roots, we can find the square root of 5.5 using linear interpolation. By choosing the two adjacent points (5, 2.2360) and (6, 2.4494), we can calculate the interpolated value y = 2.3427 at x = 5.5. Another example involves finding the population of a city in a given year using a dataset of years and populations. By selecting two adjacent points, such as (2019, 12124) and (2021, 5700), we can predict the population for the year 2020 using linear interpolation. Linear interpolation can be achieved using a formula or by utilizing a library function like scipy.interpolate.interp1d in Python. This function allows for various kinds of interpolation, including 'linear', 'nearest', 'zero', and more, and offers options for specifying the axis, copying data, and handling bounds errors. Given article text here import scipy.interpolate as sp # One-dimensional linear interpolation for monotonically increasing sample points. x = [1, 2, 3, 4, 5] y = [11, 2.2, 3.5, -88, 1] # Create an interpolant object interpolate_x = 2.5 y_interp = sp.interp1d(x, y) print("Value of Y at x =", interpolate_x, "is", y_interp(interpolate_x)) # General facilities available in SciPy for interpolation and smoothing for data in 1, 2, and higher dimensions. # The choice of a specific interpolation routine depends on the data: whether it is one-dimensional, is given on a structured grid, or is unstructured. # One other factor is the desired smoothness of the interpolator. # Recommended routines for interpolation can be summarized as follows: # Further details are given in the links below # numpy.interp(x, xp, fp, left=None, right=None, period=None) [source] # One-dimensional linear interpolation for monotonically increasing sample points. # Returns the one-dimensional piecewise linear interpolant to a function with given discrete data points (xp, fp), evaluated at x. # Parameters: # xarray_like The x-coordinates at which to evaluate the interpolated values. # xp1-D sequence of floatsThe x-coordinates of the data points, must be increasing if argument period is not specified. Otherwise, xp is internally sorted after normalizing the periodic boundaries with xp = xp % period. # fp1-D sequence of float or complexThe y-coordinates of the data points, same length as xp. # leftoptional float or complex corresponding to fpValue to return for x < xp[0], default is fp[0]. # rightoptional float or complex corresponding to fpValue to return for x > xp[-1], default is fp[-1]. # periodNone or float, optionalA period for the x-coordinates. This parameter allows the proper interpolation of angular x-coordinates. # Parameters left and right are ignored if period is specified. # Returns: yfloat or complex (corresponding to fp) or ndarrayThe interpolated values, same shape as x. # Raises: ValueErrorIf xp and fp have different length If xp or fp are not 1-D sequences If period == 0 Warning The x-coordinate sequence is expected to be increasing, but this is not explicitly enforced. However, if the sequence xp is non-increasing, interpolation results are meaningless. Note that, since NaN is unsortable, xp also cannot contain NaNs. A simple check for xp being strictly increasing is: See also scipy.interpolate # Examples >>> import numpy as np >>> xp = [1, 2, 3] >>> fp = [3, 2, 0] >>> np.interp(2.5, xp, fp) 1.0 >>> np.interp([0, 1, 1.5, 2.72, 3.14], xp, fp) array([3. , 3. , 2.5 , 0.56, 0.]) # UNDEF = -99.0 >>> np.interp(3.14, xp, fp, right=UNDEF) -99.0 # Plot an interpolant to the sine function: >>> x = np.linspace(0, 2*np.pi, 10) >>> y = np.sin(x) >>> xvals = np.linspace(0, 2*np.pi, 50) >>> yinterp = np.interp(xvals, x, y) >>> import matplotlib.pyplot as plt >>> plt.plot(x, y, 'o') [] >>> plt.plot(xvals, yinterp, '-x') [] >>> plt.show() # Interpolation with periodic x-coordinates: >>> x = [-180, -170, -185, 185, -10, -5, 0, 365] >>> xp = [190, -190, 350, -350] >>> fp = [] Given article text here 10, 3, 4] >>> interp_vals = np.interp(x, xp= [0, 5, 10], fp=[7.5, 5., 8.75], period=360) print(interp_vals) # array([7.5 , 5. , 8.75, 6.25, 3. , 3.25, 3.5 , 3.75]) Complex interpolation: >>> x = [1.5, 4.0] >>> xp = [2,3,5] >>> fp = [1.0j, 0, 2+3j] >>> interp_vals = np.interp(x, xp=xp, fp=fp) print(interp_vals) # array([0.+1.j , 1.+1.5j]) In this article, we'll delve into interpolation using the SciPy module in Python. We'll cover various interpolation techniques with implementations. Interpolation: A Technique for Constructing Data Points Between Given Points Interpolation is a technique used to construct data points between given data points. The scipy.interpolate module provides classes and functions for interpolation, including spline functions and univariate/multivariate interpolation classes. Types of Interpolation Include: 1-D Interpolation Spline Interpolation Univariate Spline Interpolation RBF Interpolation We'll discuss each method in detail and visualize the results. 1-D Interpolation To create a function based on fixed data points, scipy.interpolate.interp1d is used. It takes data points x and y and returns a function that can be called with new x to return the corresponding y point. Syntax: scipy.interpolate.interp1d(x , y , kind , axis , copy , bounds_error , fill_value , assume_sorted) import matplotlib.pyplot as plt from scipy import interpolate import numpy as np x = np.arange(0, 10) y = x**2 temp = interpolate.interp1d(x, y) xnew = np.arange(0, 9, 0.2) ynew = temp(xnew) plt.title("1-D Interpolation") plt.plot(x, y, '*', xnew, ynew, '-', color="green") plt.show() Spline Interpolation In spline interpolation, a spline representation of the curve is computed, and then the spline is computed at the desired points. The function splrep is used to find the spline representation of a curve in a two-dimensional plane. To find the B-spline representation of a 1-D curve, scipy.interpolate.splrep is used. Syntax: scipy.interpolate.splrep(x, y, w, xb, xe, k, task, s, t, full_output, per, quiet) To compute a B-spline or its derivatives, scipy.interpolate.splev is used. Syntax: scipy.interpolate.splev(x , tck , der , ext) import numpy as np import matplotlib.pyplot as plt from scipy import interpolate x = np.arange(0, 10) y = np.cos(x**3) temp = interpolate.splrep(x, y, s=0) xnew = np.arange(0, np.pi**2, np.pi/100) ynew = interpolate.splev(xnew, temp, der=0) plt.figure() plt.plot(x, y, '*', xnew, ynew, xnew, np.cos(xnew), x, y, 'b', color="green") plt.legend(['Linear', 'Cubic Spline', 'True']) plt.axis([-0.1, 6.5, -1.1, 1.1]) plt.title('Cubic-spline Interpolation in Python') plt.show() Univariate Spline It is a 1-D smoothing spline that fits a given group of data points. The scipy.interpolate.UnivariateSpline is used to fit a spline y = spl(x) of degree k to the provided x, y data. s specifies the number of knots by specifying a smoothing condition. The scipy.interpolate.UnivariateSpline. set_smoothing_factor: Spline computation with the given smoothing factor s and with the knots found at the last call. Syntax: scipy.interpolate.UnivariateSpline(x, y, w, bbox, k, s, ext) import matplotlib.pyplot as plt from scipy.interpolate import UnivariateSpline x = np.linspace(-3, 3, 50) y = np.exp(-x**2) + 0.1 * np.random.randn(50) plt.title("Univariate Spline") Scattered data can be interpolated in n-dimensions using the scipy.interpolate.Rbf class. This method defines a radial basis function centered on a fixed reference point. The syntax for Rbf is scipy.interpolate.Rbf(*args). An example usage involves interpolating a cosine curve with 9 points, and then plotting the original data and the interpolated result against a sine curve.