Spline interpolation

Missing data

Contents

SPLINE INTERPOLATION

IMPLEMENTATION IN PANDAS

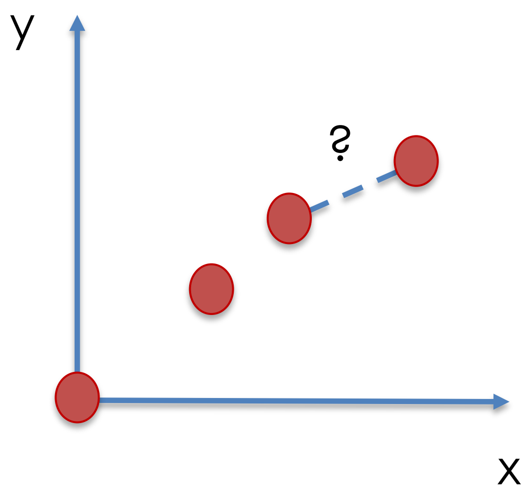
PRACTICAL CONSIDERATIONS

What is interpolation?

- Interpolation is the estimation of the value of a function, $y = f(x)$, between discrete known values
- Typically, we know the inputs and outputs but not the equation for $f(x)$ itself

x	y = f(x)
0	0
1.5	3
2	5
3	6

Known values

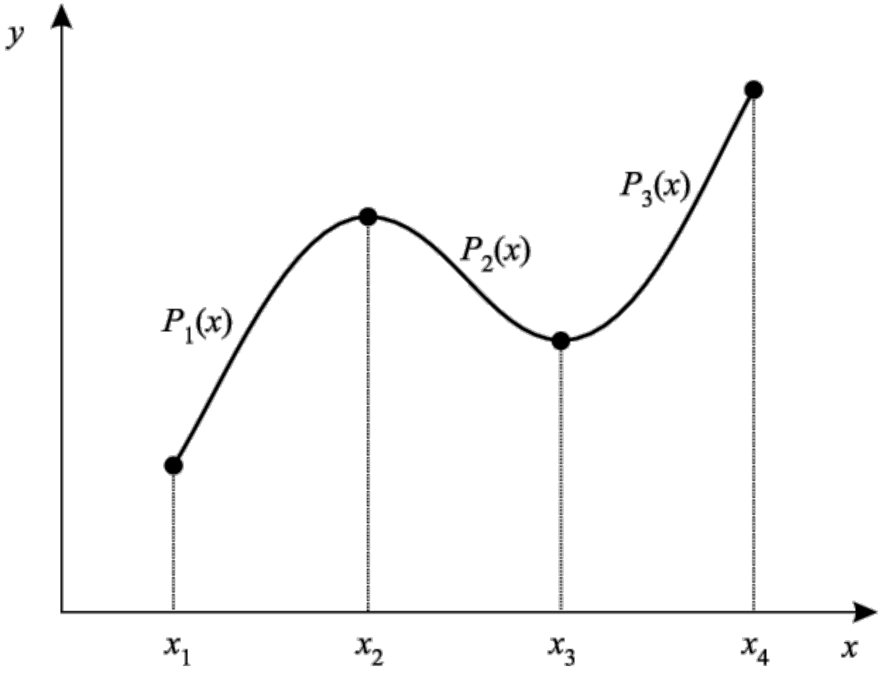


What is a good estimate for $f(x = 2.5)$?

Interpolation methods try to answer this

Spline interpolation

- Consider that we have samples at $[(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)]$
- In spline interpolation a low-degree polynomial is fit between each of the known data points with a constrain that the polynomials fit smoothly together



$$f(x) = \begin{cases} P_1(x), if\ x \in [x_1, x_2] \\ P_2(x), if\ x \in [x_2, x_3] \\ P_3(x), if\ x \in [x_3, x_4] \end{cases}$$

Polynomials of degree d. d is set by the user, typically to a low order such as 3

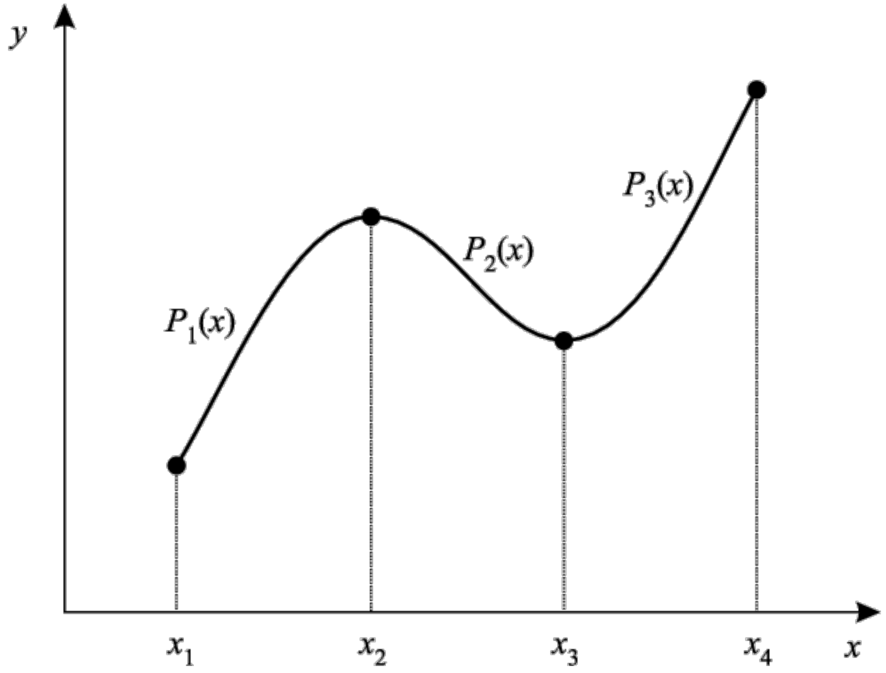
where

$$P_i'(x_{i+1}) = P_{i+1}'(x_{i+1})$$
$$P_i''(x_{i+1}) = P_{i+1}''(x_{i+1})$$

Smoothness constraints

Spline interpolation

- Multiple algorithms exist to determine the parameters for each polynomial (out of scope)



$$f(x) = \begin{cases} P_1(x), if\ x\ \epsilon\ [x_1, x_2] \\ P_2(x), if\ x\ \epsilon\ [x_2, x_3] \\ P_3(x), if\ x\ \epsilon\ [x_3, x_4] \end{cases}$$

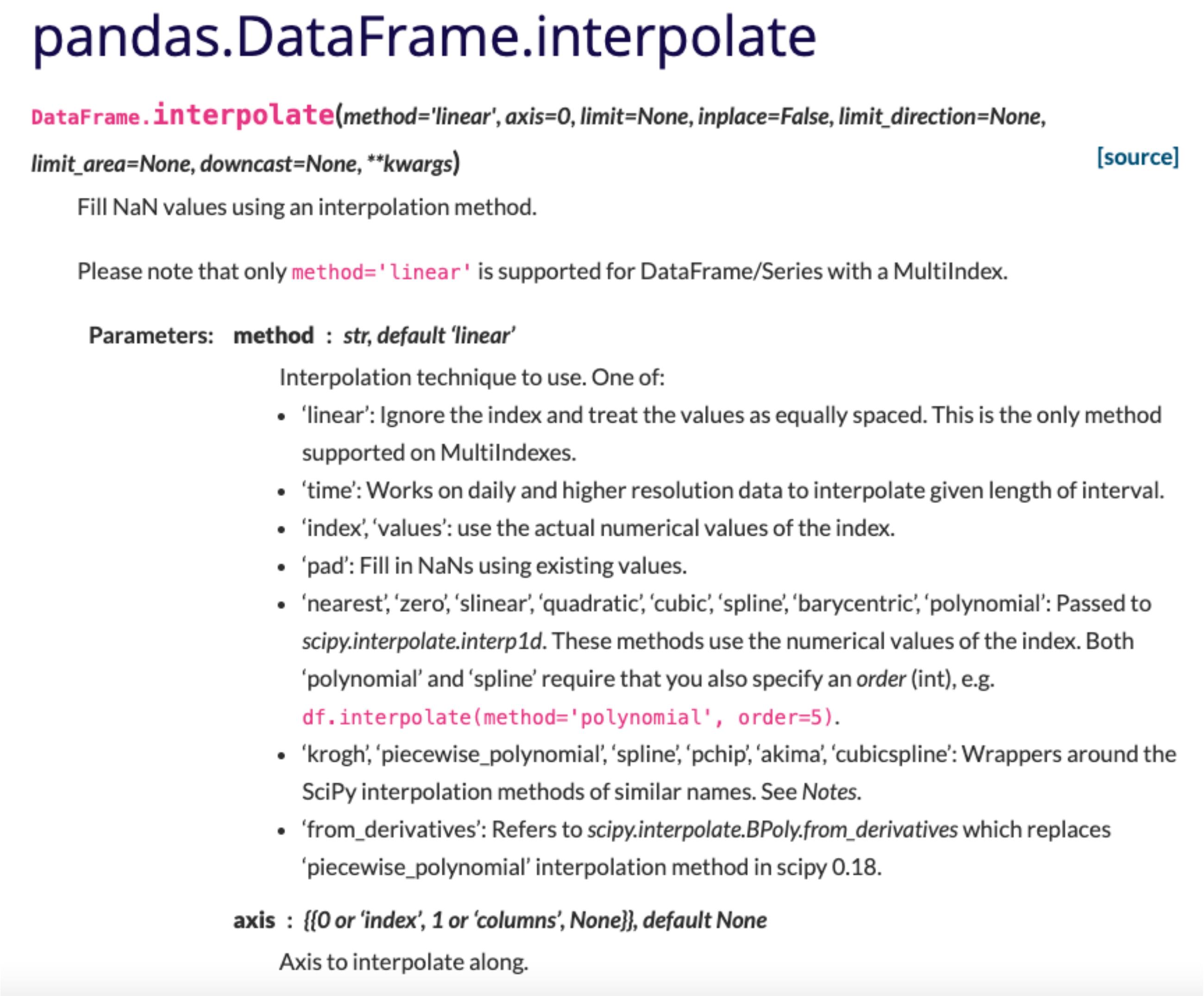
Polynomials of degree d. d is set by the user, typically to a low order such as 3

where

$$P_i'(x_{i+1}) = P_{i+1}'(x_{i+1})$$
$$P_i''(x_{i+1}) = P_{i+1}''(x_{i+1})$$

Smoothness constraints

Implementation



pandas.DataFrame.interpolate

`DataFrame.interpolate`(*method='linear', axis=0, limit=None, inplace=False, limit_direction=None, limit_area=None, downcast=None, **kwargs*) [source]

Fill NaN values using an interpolation method.

Please note that only `method='linear'` is supported for DataFrame/Series with a MultiIndex.

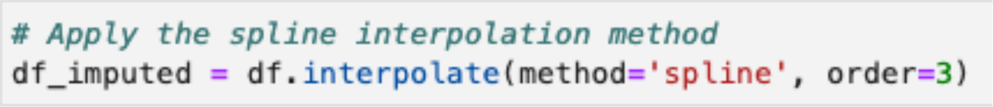
Parameters: **method** : *str, default 'linear'*

Interpolation technique to use. One of:

- 'linear': Ignore the index and treat the values as equally spaced. This is the only method supported on MultiIndexes.
- 'time': Works on daily and higher resolution data to interpolate given length of interval.
- 'index', 'values': use the actual numerical values of the index.
- 'pad': Fill in NaNs using existing values.
- 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'spline', 'barycentric', 'polynomial': Passed to *scipy.interpolate.interp1d*. These methods use the numerical values of the index. Both 'polynomial' and 'spline' require that you also specify an *order* (int), e.g. `df.interpolate(method='polynomial', order=5)`.
- 'krogh', 'piecewise_polynomial', 'spline', 'pchip', 'akima', 'cubicspline': Wrappers around the SciPy interpolation methods of similar names. See *Notes*.
- 'from_derivatives': Refers to *scipy.interpolate.BPoly.from_derivatives* which replaces 'piecewise_polynomial' interpolation method in scipy 0.18.

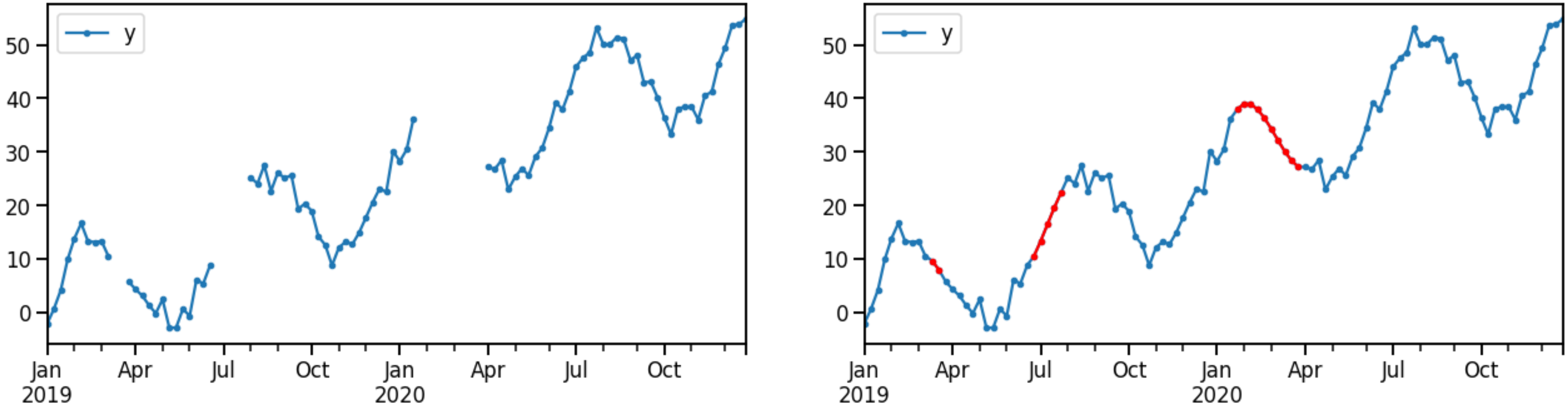
axis : *{{0 or 'index', 1 or 'columns', None}}, default None*

Axis to interpolate along.



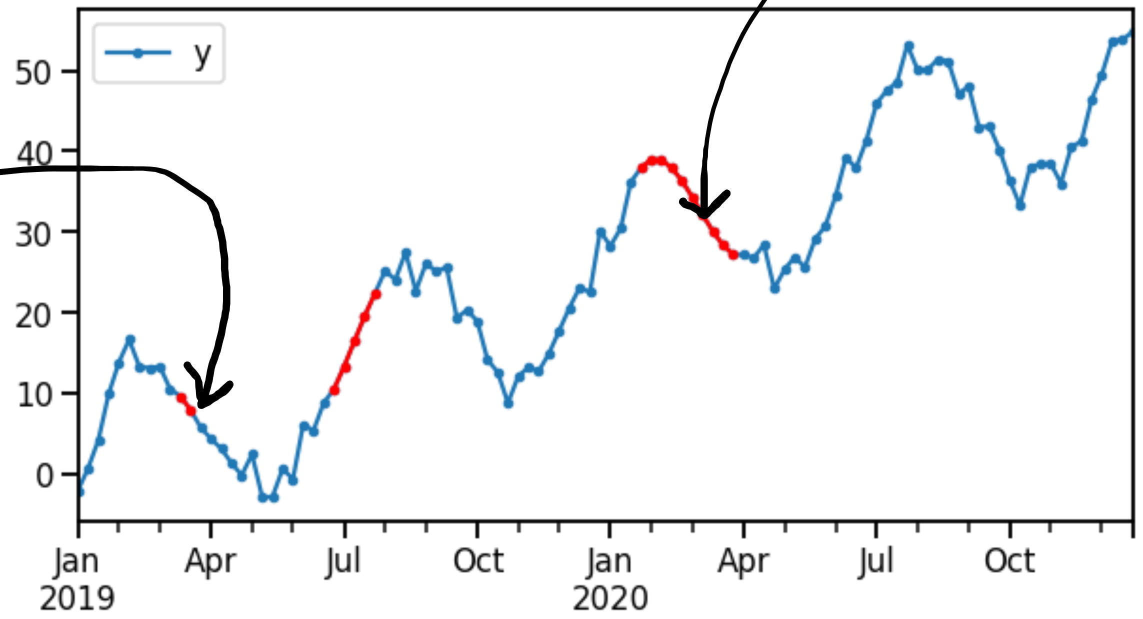
```
# Apply the spline interpolation method
df_imputed = df.interpolate(method='spline', order=3)
```

Example: Spline interpolation d=3



Practical considerations

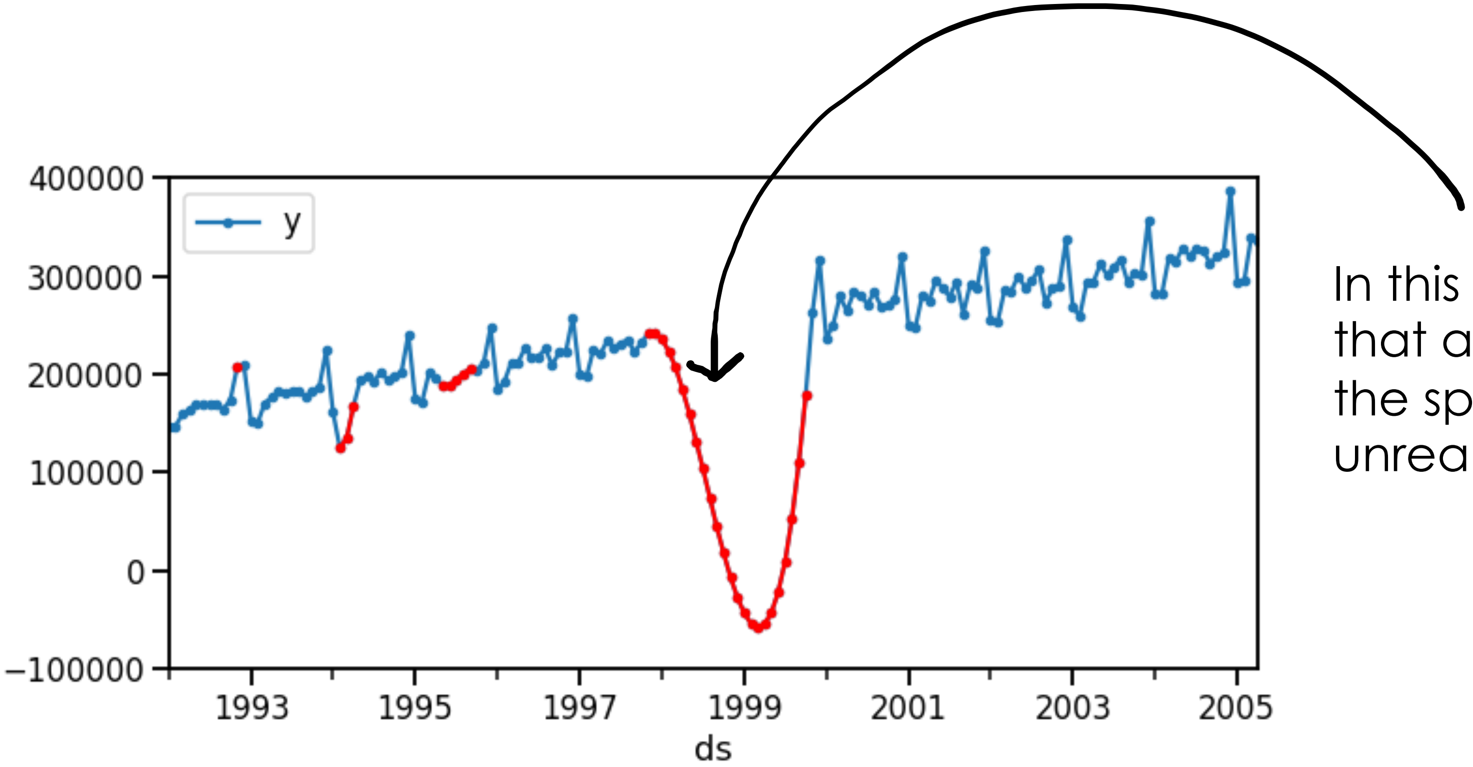
For short gaps there is minimal distortion to the time series



We can see that a cubic spline provides a less distorted imputation to the data than linear or the filling methods shown earlier

Spline interpolation is computationally more costly than forward or backward filling

Practical considerations



In this example we see that a large gap allows the spline to deviate to unrealistic values

Imputation methods for time series

1. Forward filling (aka last observation carried forward)
2. Backward filling (aka next observation carried backwards)
3. Linear interpolation
4. Spline interpolation
5. Seasonal decomposition and interpolation