Numerical interpolation

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

Today's outline

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Interpolation problem

Definition

Given a set of points x_k, $k = 0, \dots, n$, $x_i \neq x_j$ with associated function values f_k, $k = 0, \dots, n$, or simply: $\{x_k, f_k\}_{k=0}^{n}$. The interpolation problem is defined as: find a polynomial p_n such that this interpolates the values of f_k on the points x_k:

$$p_n(x_k) = f_k, \quad k = 0, \dots, n$$

Theorem

The interpolation problem for $\{x_k, f_k\}_{k=0}^{n}$ has a unique solution when $x_i \neq x_j$ for $i \neq j$. Note that we cannot allow multiple function values f_k for the same value of x_k.

What is interpolation?

Interpolation means constructing additional data points within the range of, and using, a discrete set of known data points.

It is typically performed on a uniformly spread data set, but this is not strictly necessary for all methods

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Is interpolation the same as curve fitting?

NO

- Curve-fitting requires additionally some way of computing the error between function (curve) and data
- Curve-fitting does not strictly enforce the function to match the data exactly
- Curve-fitting may be done on multiple datapoints at one position
- Curve-fitting is much more expensive to do, requires optimisation

Why do chemical engineers need interpolation?

- Comparison of two data sets which are given at different positions
  - An experimental data set may have been recorded at a constant rate, but the numerical solution is computed at irregular intervals
- Reconstruction of field values distant of computing nodes
  - A CFD simulation on a regular grid containing structures that are not grid-conformant requires interpolation to the structures
- Calculation of a physical property at a condition between those of a lookup table
  - The viscosity of a substance may have been measured at 20°C and 30°C, but not at the desired 28.5°C

General

Several important numerical interpolation methods are discussed today:

- Piecewise constant interpolation
- Linear interpolation
  - Bilinear interpolation
- Polynomial interpolation (Newton's method)
- Spline interpolation

Today's outline

Today's data set

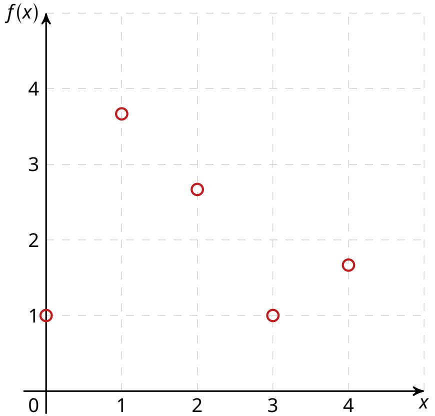
Generate the following data set:

```python
import numpy as np
xdata = np.arange(0,6)
fun = lambda x: x**3/2 - (10*x**2)/3 + 11*x
     /2 + 1
ydata = fun(xdata)
```

This yields some sample points on which we base our examples:

x_k	f_k
0	1.00
1	$\frac{11}{3} = 3.67$
2	$\frac{8}{3} = 2.67$
3	1.00
4	$\frac{5}{3} = 1.67$
5	$\frac{23}{3} = 7.67$

Data set $f_n(x_n)$ represented by ○ at discrete intervals $x_n \in \{0, 5\}$



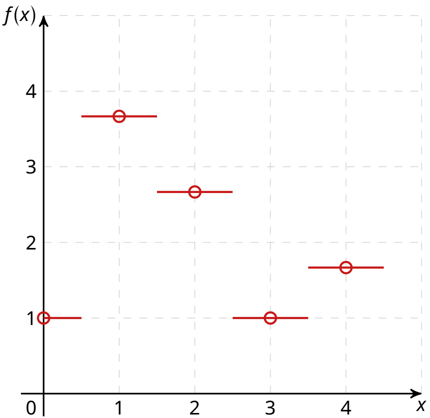
Piecewise constant interpolation

- Nearest-neighbor interpolation in the continuous range $x \in [0, 5]$
- How to treat the point halfway (e.g. at $x = 2.5$)?

$$
\begin{aligned}
x &\in [0, 0.5] && \rightarrow f(x) = f(0)\\
x &\in [0.5, 1.5] && \rightarrow f(x) = f(1)\\
x &\in [1.5, 2.5] && \rightarrow f(x) = f(2)\\
x &\in [2.5, 3.5] && \rightarrow f(x) = f(3)\\
x &\in [3.5, 4.5] && \rightarrow f(x) = f(4)
\end{aligned}
$$

- Not often used for simple problems, but e.g. for 2D (Voronoi)

Data set $f_n(x_n)$ represented by ○ at discrete intervals $x_n \in \{0, 5\}$



Today's outline

Linear interpolation

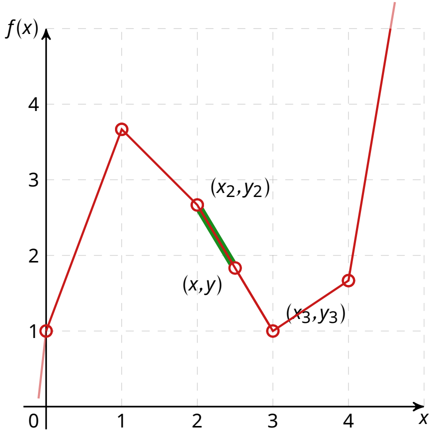
- Linear interpolation to (x,y) between 2 data points (x_2,y_2) and (x_3,y_3):

$$\frac{y-y_2}{x-x_2}=\frac{y_3-y_2}{x_3-x_2}$$

- Reordered, and more formally:

$$y=y_n+(y_{n+1}-y_n)\frac{x-x_n}{x_{n+1}-x_n}$$

Data set $f_n(x_n)$ represented by ○ at discrete intervals $x_n \in \{0,5\}$



Linear interpolation

- While linear interpolation is fast, and relatively easy to program, it is not very accurate
- At the nodes, the derivatives are discontinuous i.e. not differentiable
- Error is proportional to the square of the distance between nodes

Interpolation in Python

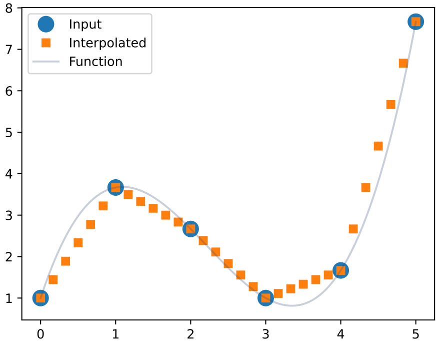
Interpolation can be done using the SciPy interpolation submodule, e.g.:

```
from scipy.interpolate import interp1d
f = interp1d(xdata, ydata, kind='linear')
```

This creates a function object `f` based on the given data.

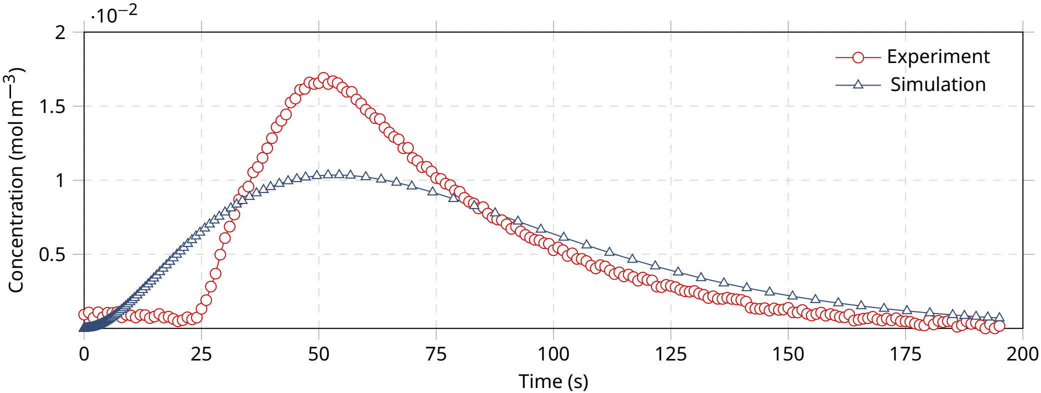
```
from scipy.interpolate import interp1d
import numpy as np

fun = lambda x: x**3/2 - (10*x**2)/3 + 11*x/2 + 1
xdata = np.arange(0,6)
ydata = fun(xdata)

f = interp1d(xdata,ydata)
xint = np.linspace(0,5,31)
yint = f(xint)
```

```
import matplotlib.pyplot as plt
plt.plot(xdata,ydata,'o',markersize=12,label='Input')
plt.plot(xint,yint,'s',label='Interpolated')
```



Example: Linear interpolation in Python

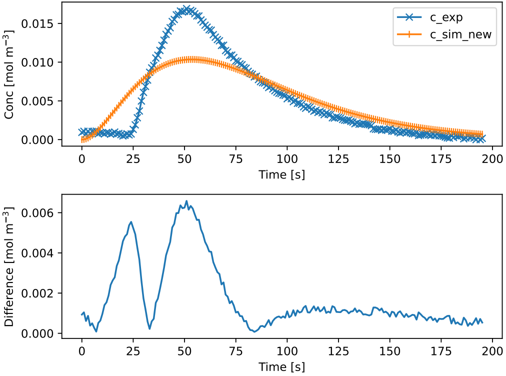
Consider the data sets in `exp_data.txt` and `sim_data.txt`, containing a normalized concentration and time vector for an experiment and a simulation. The simulation was performed with adaptive node distance to save computation time, thus the concentration is not known at the same times. We are not able to compare yet.



Example: Linear interpolation in Python

Consider the data sets in **`exp_data.txt`** and **`sim_data.txt`**, containing a normalized concentration and time vector for an experiment and a simulation. The simulation was performed with adaptive node distance to save computation time, thus the concentration is not known at the same times. We are not able to compare yet.

```python
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

t_sim, c_sim = np.loadtxt("scripts/interpolation/sim_data.txt").T
t_exp, c_exp = np.loadtxt("scripts/interpolation/exp_data.txt").T

# Linear interpolation
f = interp1d(t_sim, c_sim)
diff = np.abs(c_exp - f(t_exp))

# Plot the solution
plt.subplot(2, 1, 1)
plt.plot(t_exp, c_exp, '-x', label='c_exp')
plt.plot(t_exp, f(t_exp), '-|', label='c_sim_new')
plt.xlabel('Time [s]'); plt.ylabel('Conc [mol m$^{-3}$]')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t_exp, diff)
plt.xlabel('Time [s]'); plt.ylabel('Difference [mol m$^{-3}$]')
plt.tight_layout()
# plt.show()
plt.savefig('figures/sim_exp_data_interp.pdf')
```



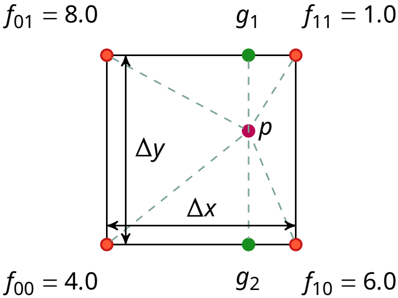
Bi-linear interpolation

When a 2D field of some quantity is known, we can interpolate the solution to an arbitrary position in the 2D domain $p(x,y)$ using 4 field values f_{00}, f_{10}, f_{01} and f_{11}.

$$g_1 = f_{01}\frac{x_1 - x}{x_1 - x_0} + f_{11}\frac{x - x_0}{x_1 - x_0}$$
$$= f_{01}\frac{x_1 - x}{\Delta x} + f_{11}\frac{x - x_0}{\Delta x}$$

$$g_2 = f_{00}\frac{x_1 - x}{\Delta x} + f_{10}\frac{x - x_0}{\Delta x}$$

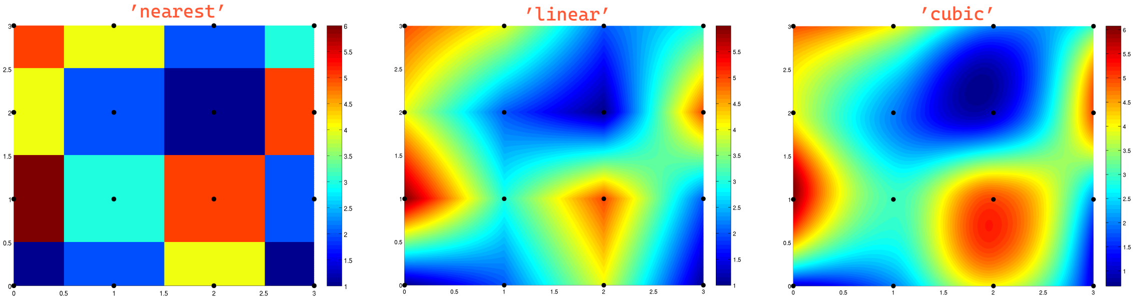
$$p = g_2\frac{y_1 - y}{\Delta y} + g_1\frac{y - y_0}{\Delta y}$$



- The order of interpolation (x or y direction first) does not matter; the results are equal

Higher-dimensional field interpolation in Python

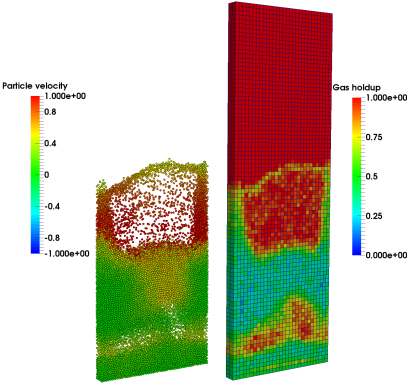
2D or higher-dimensional fields of data can be interpolated in Python using the `scipy.interpolate.interp2d`, `scipy.interpolate.interp3d`, or even `scipy.interpolate.RegularGridInterpolator` functions. The method can be adjusted:



- Also consider tri-linear interpolation (for 3D fields) with `scipy.interpolate.LinearNDInterpolator`, or bicubic interpolation (2D, but third order) with `scipy.interpolate.interp2d`.

A practical example

Field interpolation is used in e.g. CFD simulations, e.g. a fluidized bed simulation using a *discrete particle model*, where particles are found in between the grid nodes used for velocity computation.



Today's outline

Polynomial interpolation

The examples that we have seen, are simplified forms of *Newton polynomials*. We can interpolate our data with a polynomial of degree n:

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

Polynomial interpolation via Vandermonde matrix

Consider the data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, the Vandermonde matrix V, coefficient vector a and function value vector y:

$$V_{m,n} = \begin{pmatrix} x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^{n-1} \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_m^0 & x_m^1 & x_m^2 & \cdots & x_m^{n-1} \end{pmatrix} \quad a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

The coefficients of a polynomial through the data are obtained by solving the linear system $Va = y$.

```python
import numpy as np
x = np.array([0, 1, 2])
y = np.array([1.0000, 3.6667, 2.6667])
V = np.vander(x, increasing=True)
print(V)
```

```
[ 1. 4.50005 -1.83335]
```

So we found the equation:

$$p_2(x) = -1.8333x^2 + 4.5x - 1$$

These Vandermonde-systems are often *ill-conditioned*, so we need another, more stable, method!

Construction of Newton polynomials

Formally, the polynomials $p_n(x)$ are described using prefactors $f[x_0,\ldots,x_k]$ and polynomial terms $w_m(x)$:

$$p_n(x) = \sum_{k=0}^{n} f[x_0,\ldots,x_k]w_k(x)$$

The polynomial terms are computed via:

$$w_0(x) = 1,\ w_1(x) = (x - x_0),\ w_2(x) = (x - x_0)\cdot(x - x_1),$$
$$w_m(x) = (x - x_0)\cdot(x - x_1)\cdots(x - x_{m-1}) = w_{m-1}\cdot(x - x_{m-1})$$
$$w_m(x) = \prod_{j=0}^{m-1}(x - x_j), \qquad m = 0,\ldots,n$$

The prefactors are *forward divided differences*, which can be computed as:

$$f[x_{x-k},\ldots,x_r] \equiv \frac{f[x_{r-k+1},\ldots,x_r] - f[x_{r-k},\ldots,x_{r-1}]}{x_r - x_{r-k}}$$

Construction of Newton polynomials: example

Sample data

x_k	f_k
0	1.00
1	$\frac{11}{3} = 3.67$
2	$\frac{8}{3} = 2.67$

$$p_n(x)=\sum_{k=0}^{n} f[x_0,\ldots,x_k]w_k(x)$$

$$f[x_{x-k},\ldots,x_r]\equiv\frac{f[x_{r-k+1},\ldots,x_r]-f[x_{r-k},\ldots,x_{r-1}]}{x_r-x_{r-k}}$$

$$w_m(x)=\prod_{j=0}^{m-1}(x-x_j)$$

x_k	f_k		
x_0	$f[x_0]=f_0$		
x_1	$f[x_1]=f_1$	$f[x_0,x_1]=\frac{f_1-f_0}{x_1-x_0}$	
x_2	$f[x_2]=f_2$	$f[x_1,x_2]=\frac{f_2-f_1}{x_2-x_1}$	$f[x_0,x_1,x_2]=\frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$

x_k	f_k		
0	1		
1	3.67	$\frac{\frac{11}{3}-1}{1-0}=\frac{8}{3}$	
2	2.67	$\frac{\frac{8}{3}-\frac{11}{3}}{2-1}=\frac{-1}{1}=-1$	$\frac{(-1)-\frac{8}{3}}{2-0}=-\frac{11}{6}$

Construction of Newton polynomials: example

Sample data

x_k	f_k
0	1.00
1	$\frac{11}{3} = 3.67$
2	$\frac{8}{3} = 2.67$

$$p_n(x) = \sum_{k=0}^{n} f[x_0,\ldots,x_k] w_k(x)$$

$$f[x_{x-k},\ldots,x_r] \equiv \frac{f[x_{r-k+1},\ldots,x_r] - f[x_{r-k},\ldots,x_{r-1}]}{x_r - x_{r-k}}$$

$$w_m(x) = \prod_{j=0}^{m-1}(x - x_j)$$

x_k	f_k		
0	1		
1	3.67	$\frac{\frac{11}{3}-1}{1-0} = \frac{8}{3}$	
2	2.67	$\frac{\frac{8}{3}-\frac{11}{3}}{2-1} = \frac{-1}{1} = -1$	$\frac{(-1)-\frac{8}{3}}{2-0} = -\frac{11}{6}$

$$p_2(x) = 1 \cdot w_m(0) + \frac{8}{3} \cdot w_m(1) + \left(-\frac{11}{6}\right) \cdot w_m(2)$$

$$= 1 \cdot 1 + \frac{8}{3} \cdot (x-0) + \left(-\frac{11}{6}\right) \cdot (x-0)(x-1) = -\frac{11}{6}x^2 + 4\frac{1}{2}x + 1$$

Construction of Newton polynomials: example

For each three points, a new polynomial interpolant can be derived:

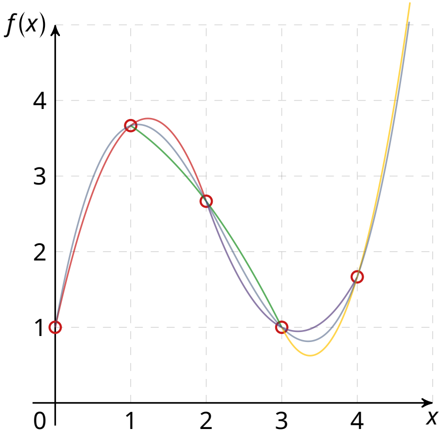
$$p_2(x) = -\frac{11}{6}x^2 + 4\frac{1}{2}x + 1$$

$$p_2(x) = 4 - \frac{x^2}{3}$$

$$p_2(x) = \frac{7x^2}{6} - 7\frac{1}{2}x + 13$$

$$p_2(x) = \frac{8}{3}x^2 - 18x + 31$$

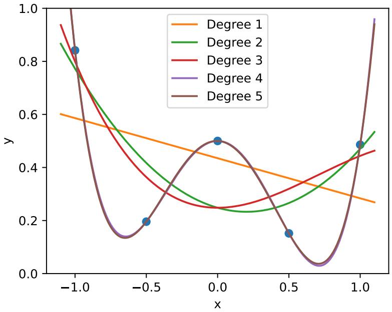
$$f(x) = \frac{x^3}{2} - \frac{10x^2}{3} + \frac{11x}{2} + 1$$



Polynomial fitting in Python: example

Develop the polynomials $p_1(x)$ through $p_5(x)$ using the following data set:

```python
import numpy as np
import matplotlib.pyplot as plt
xdata = np.arange(-1,1.5,0.5)
ydata = [x * np.sin(x)/np.sqrt(x+2) if x != 0 else 0.5 for x in xdata]
plt.plot(xdata,ydata,'o')
```




```python
xc = np.linspace(-1.1,1.1,1001,endpoint=True)
for deg in range(1,6):
    # Fit coefficients
    p_coeffs = np.polyfit(xdata,ydata,deg)
    # Compute function values
    y = np.polyval(p_coeffs,xc)
    # Plot
    plt.plot(xc,y,label=f'Degree {deg}')
```

```
RankWarning: Polyfit may be poorly conditioned
```

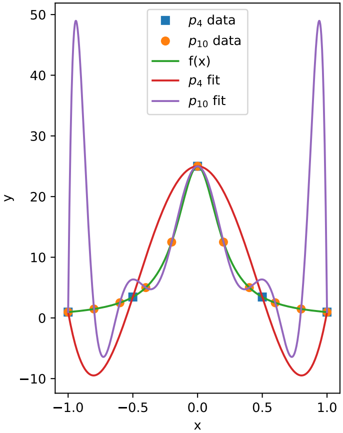
Exercise

Develop the $p_4(x)$ and $p_{10}(x)$ interpolants from the following data sets:

$$f(x) = \frac{1}{x^2 + \frac{1}{25}} \qquad x \in [-1, 1]$$

```python
import numpy as np
import matplotlib.pyplot as plt
f = lambda x: 1/(x**2 + 1/25)
x4,x10,xinf = [np.linspace(-1, 1, n) for n in [5,11,1001]]
y4,y10,yinf = f(x4), f(x10), f(xinf)
```

```python
# Get coefficients for 4th and 10th order polynomial
p4 = np.polyfit(x4, y4, 4)
p10 = np.polyfit(x10, y10, 10)
# Compute function values using fitted coeffs
yinf4 = np.polyval(p4, xinf)
yinf10 = np.polyval(p10, xinf)
```



Final thoughts on polynomial interpolation

- An polynomial interpolant of order n requires $n+1$ data points
  - More data points: interpolant does *not always* cross the points
  - Fewer data points: interpolant is not unique
- Higher-degree polynomials at equidistant points may cause strong oscillatory behaviour (Runge's phenomenon)
  - Mitigation of the problem on Chebyshev (i.e. non uniform grid)...
  - ... or by performing piecewise interpolation (next topic)
- Python functions `np.polyfit(x,y,n)` and `np.polyval(p,x_new)` were demonstrated.

Today's outline

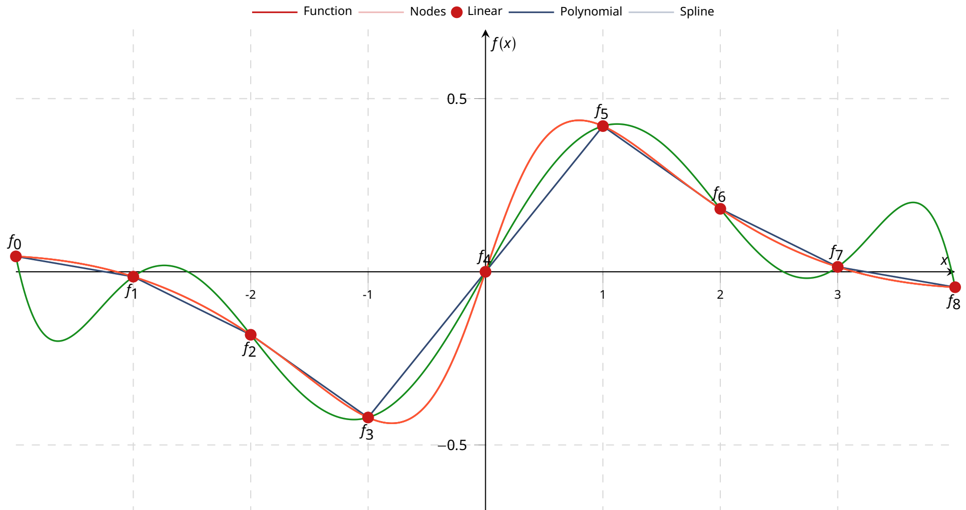
Spline interpolation

A spline is a numerical function that represents a smooth, higher order, piecewise polynomial interpolants of a data set.

- Smooth: the interpolant is continuous in the first and second derivatives
- Higher order: The most common type of splines uses third-order polynomials (cubic splines)
- Piecewise polynomial: The interpolant is constructed between each two consecutive tabulated points

Splines: comparison to other interpolation techniques

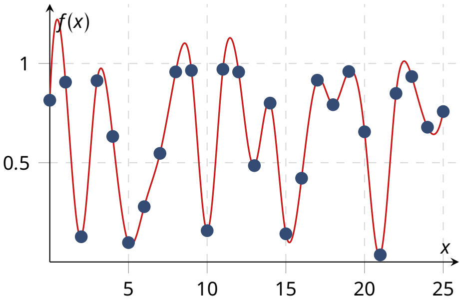
Interpolation of $f(x) = \dfrac{\sin x}{1+x^2}$



Spline interpolation in Python

We can generate a random data set, and interpolate using `scipy.interpolate.interp1d`:

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

# Generate random data set
xdata = np.arange(0, 26)
ydata = np.random.rand(len(xdata))

# Interpolant on a fine mesh
xc = np.linspace(0, 25, 1001)
ifun = make_interp_spline(xdata, ydata)
yc = ifun(xc)

# Plot the data
plt.plot(xdata, ydata, 'o')
plt.plot(xc, yc, '-r')
plt.show()
```



Note: The SciPy Optimize module contains various interpolation methods with a similar interface.

Summary

- Interpolation is used to obtain data between existing data points
  - (Bi-)Linear, polynomial and spline interpolation methods
  - Construction of Newton polynomials
  - Oscillations of high-order polynomials
- Legendre polynomials: alternative way of performing the polynomial interpolation (not discussed here)

Interpolation tutorials

1. In Python, generate the data:

```
x = np.arange(-4, 6, 1)
y = [0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
```

   Interpolate the data using polynomial interpolation (which order do you use?) and a spline. Plot the results together with the original data in a graph.

2. Do the same exercise for the following data. Can you explain your observations?

```
t = [0, 0.1, 0.499, 0.5, 0.6, 1.0, 1.4, 1.5, 1.899, 1.9, 2.0]
y = [0, 0.06, 0.17, 0.19, 0.21, 0.26, 0.29, 0.29, 0.30, 0.31, 0.31]
```

Hint: Use `scipy.interpolate.interp1d(...,kind="...")` to use different splines.