2.0 Interpolation

If we relax the approximation requirement in such a way, that minimal distance is only required at certain discrete and disjoint points the problem reduces to an interpolation problem.

This is studied in this section.

2.1 Setting up the problem

Let $f : [a, b] \to \mathbb{R}$ and consider disjoint gridpoints

$$X = \{x_0, x_1, \ldots, x_n\}$$

then we define a restriction operator $R_X : \mathcal{C}([a, b]) \to \mathbb{R}^n$ by

$$R_X f = f_X = (f(x_0), f(x_1), \ldots, f(x_n)) \in \mathbb{R}^n$$

The size of f_X is measured by a norm in $\mathbb{R}^n$, e.g.

$$\|f_X\|_2 = \Big(\sum_{i=0}^{n} f(x_i)^2\Big)^{1/2}$$

.

2.2 Setting up the problem (Cont.)

Consider now $\mathcal{S}_n := \mathcal{P}^n$. By constructing a polynomial $p^* \in \mathcal{P}^n$ with $p_X^* = f_X$ we get

$$0 = \|R_X(p^* - f)\| = \|p_X^* - f_X\|$$

p^* is called the interpolation polynomial of f.

2.3 The interpolation conditions as linear equation system

The interpolation conditions can be written as a linear equation system for the polynomial coefficients

Ansats: $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots a_1 x + a_0$

Interpolation conditions

$$p(x_j) = a_n x_i^n + a_{n-1} x_i^{n-1} + \cdots a_1 x_i + a_0 = f(x_i) \qquad 0 \le i \le n$$

In matrix form

$$\begin{pmatrix} x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ & & \cdots & & \\ x_n^n & x_n^{n-1} & \cdots & x_n & 1 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \quad \texttt{or} \quad Va = f_X$$

V is called a Vandermonde matrix.

2.4 Lagrange Polynomials

We take now another approach to compute the interpolation polynomial

Definition. [–] *The polynomials $L_i \in \mathcal{P}^n$ with the property*

$$L_i(x_j) = \begin{cases} 0 & \text{if} \quad i \neq j \\ 1 & \text{if} \quad i = j \end{cases}$$

are called Lagrange polynomials.

2.5 Lagrange Polynomials (Cont.)

It is easy to check, that

$$L_i(x) = \prod_{\substack{j=0 \\ i \neq j}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

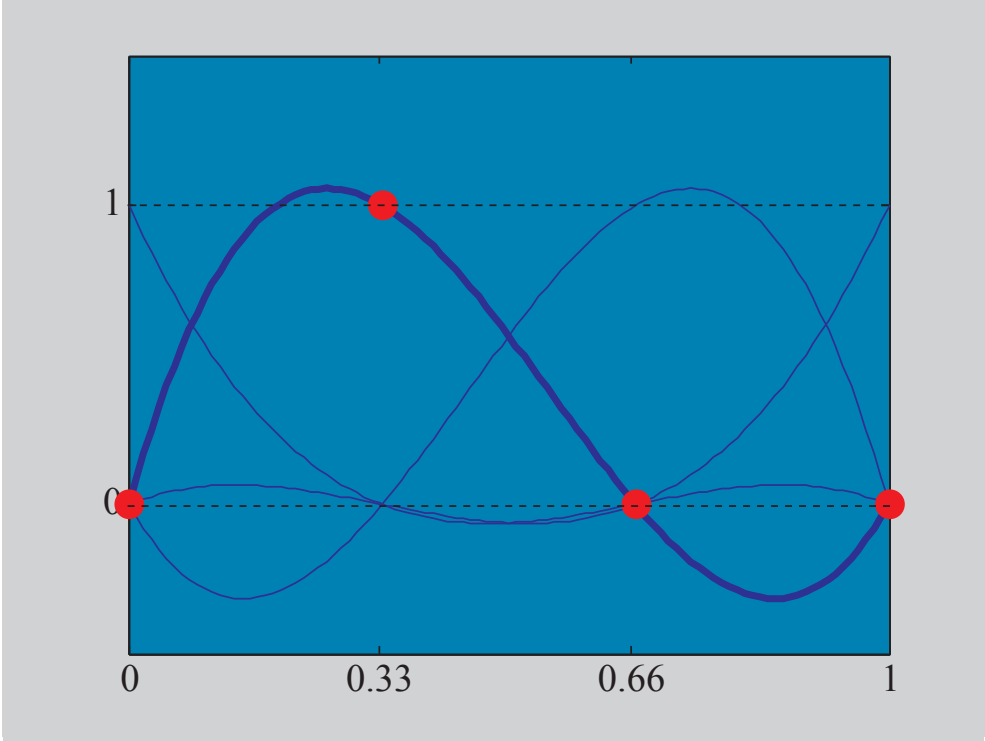
The interpolation polynomial p can be written as

$$p(x) = \sum_{i=0}^{n} y_i L_i(x)$$

Check that it indeed fulfils the interpolation conditions!

2.6 Lagrange Polynomials: Example

Lagrange polynomials of degree 3:



2.7 Lagrange Polynomials in Python

```
def lagrange(x,i,xm):
    "\""
    Evaluates the i-th Lagrange polynomial at x
    based on grid data xm
    "\""
    n=len(xm)-1
    y=1.
    for j in range(n+1):
        if i!=j:
            y*=(x-xm[j])/(xm[i]-xm[j])
    return y
```

and an interpolation function

```
def interpolation(x,xm,ym):
    n=len(xm)-1
    lagrpoly=array([lagrange(x,i,xm) for i in range(n+1)])
    return dot(ym,lagrpoly)
```

2.8 Example (cont.)

Interpolation data and a plot

```
xm = array([1,2,3,4,5,6])
ym = array([-3,0,-1,2,1,4])
xplot = \
linspace(0.9,6.1,100)
yplot = \
[interpolation(x,xm,ym) for x in xpl
```