MATH 350: Introduction to Computational Mathematics

Chapter III: Interpolation

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Spring 2011

Outline

Outline

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

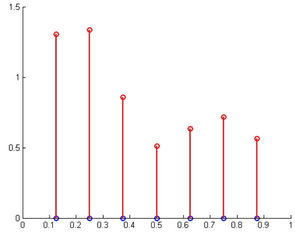
- *exactly* $\rightarrow$ interpolation or curve fitting,

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,

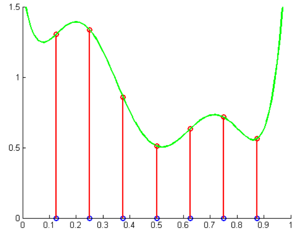


A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,



A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).
Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

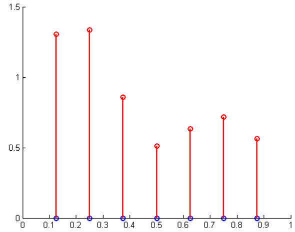
- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

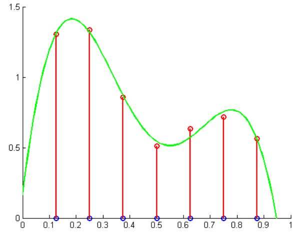


A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

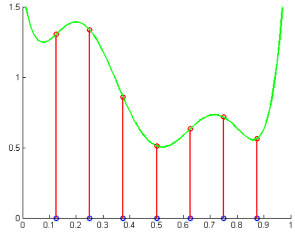
We can fit the data

- *exactly* $\to$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\to$ (least squares) approximation or regression



A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.



We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

The data can be

- *univariate*: measurements of physical phenomenon over time

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

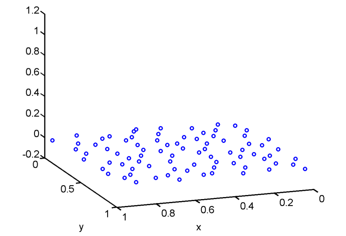
- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

The data can be

- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.



We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

The data can be

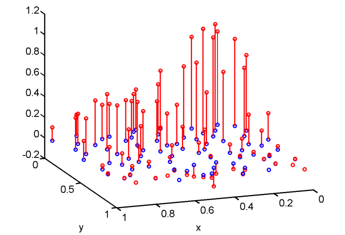
- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

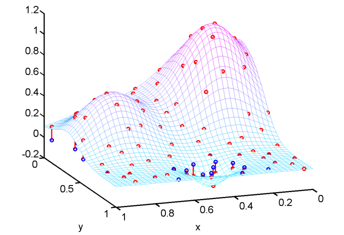
- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression



The data can be

- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).
Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.



We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

The data can be

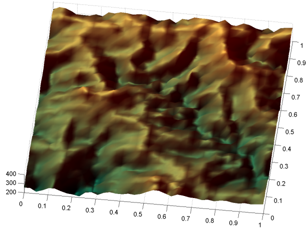
- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.

We can fit the data

- *exactly* $\rightarrow$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\rightarrow$ (least squares) approximation or regression

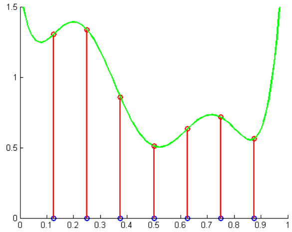


The data can be

- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

A problem that arises in almost all fields of science and engineering is to closely fit a given set of data (obtained, e.g., from experiments or measurements).

Alternatively, we may want to represent a complicated function (that we know only at a few points) by a simpler one. This could come up in the evaluation of integrals with complicated integrands, or the solution of differential equations.



We can fit the data

- *exactly* $\to$ interpolation or curve fitting,
- or only *approximately* (e.g., when the data is noisy) $\to$ (least squares) approximation or regression

The data can be

- *univariate*: measurements of physical phenomenon over time
- *multivariate*: measurements of physical phenomenon over a 2D or 3D spatial domain

We will concentrate on interpolation of univariate data.

Example

Consider the following artificial data

x	3	1	5	6	0
y	1	-3	2	4	2

We can run `InterpolationDemo.m` (which calls the program `interpgui` from [NCM] with this data set) to look at different types of interpolants.

Example

Consider the following time and velocity outputs from the Euler solution of the skydive problem from Computer Assignment 1.

t	v	t	v
0	0	11	23.9383
1	9.8100	12	16.1725
2	18.1795	13	14.1084
3	25.3199	14	13.5598
4	31.4119	15	13.4140
5	36.6093	16	13.3752
6	41.0435	17	13.3649
7	44.8265	18	13.3622
8	48.0541	19	13.3615
9	50.8077	20	13.3613
10	53.1569		

We can continue `InterpolationDemo.m` to see how this set of data is fitted by different methods.

Outline

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.
If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.

If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1)$$

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.

If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1) \qquad \text{or} \qquad p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.

If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1) \qquad \text{or} \qquad p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

- In Lagrange form:

$$p(x) = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2$$

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.
If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1) \qquad \text{or} \qquad p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

- In Lagrange form:

$$\begin{aligned} p(x) &= \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \\ &= L_1(x) y_1 + L_2(x) y_2 \end{aligned}$$

with $L_1(x) = \frac{x - x_2}{x_1 - x_2}$, and $L_2(x) = \frac{x - x_1}{x_2 - x_1}$.

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.

If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1) \qquad \text{or} \qquad p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

- In Lagrange form:

$$\begin{aligned} p(x) &= \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \\ &= L_1(x) y_1 + L_2(x) y_2 \end{aligned}$$

with $L_1(x) = \frac{x - x_2}{x_1 - x_2}$, and $L_2(x) = \frac{x - x_1}{x_2 - x_1}$.

Note that L_1 and L_2 are both polynomials of degree one, so that p is a linear polynomial, and that $L_1(x_1) = 1$, $L_2(x_1) = 0$, $L_1(x_2) = 0$, and $L_2(x_2) = 1$, so that p interpolates the data.

We all know that two distinct points (x_1, y_1) and (x_2, y_2) in the plane uniquely define a line that passes through them.

If $x_1 \neq x_2$ then we can write the interpolant as a linear polynomial:

- In point-slope form:

$$p(x) - y_1 = m(x - x_1) \qquad \text{or} \qquad p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1).$$

- In Lagrange form:

$$\begin{aligned} p(x) &= \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \\ &= L_1(x) y_1 + L_2(x) y_2 \end{aligned}$$

with $L_1(x) = \frac{x - x_2}{x_1 - x_2}$, and $L_2(x) = \frac{x - x_1}{x_2 - x_1}$.

Note that L_1 and L_2 are both polynomials of degree one, so that p is a linear polynomial, and that $L_1(x_1) = 1$, $L_2(x_1) = 0$, $L_1(x_2) = 0$, and $L_2(x_2) = 1$, so that p interpolates the data.
Notation: $L_i(x_j) = \delta_{ij}$, the Kronecker delta symbol.

The Lagrange form can be applied to three distinct points (x_1, y_1), (x_2, y_2), (x_3, y_3) and quadratic interpolation:
The interpolating polynomial is of the form

$$p(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3$$

The Lagrange form can be applied to three distinct points (x_1, y_1), (x_2, y_2), (x_3, y_3) and quadratic interpolation:
The interpolating polynomial is of the form

$$\begin{aligned} p(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 \\ &= L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3 \end{aligned}$$

The Lagrange form can be applied to three distinct points (x_1, y_1), (x_2, y_2), (x_3, y_3) and quadratic interpolation:
The interpolating polynomial is of the form

$$\begin{aligned} p(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 \\ &= L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3 \end{aligned}$$

Again, L_1, L_2, L_3 are quadratic polynomials, and

$$L_i(x_j) = \delta_{ij}, \qquad i,j = 1,2,3,$$

so that p is the (unique) quadratic interpolating polynomial for the given data.

The Lagrange form can be applied to three distinct points (x_1, y_1), (x_2, y_2), (x_3, y_3) and quadratic interpolation:
The interpolating polynomial is of the form

$$\begin{aligned} p(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 \\ &= L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3 \end{aligned}$$

Again, L_1, L_2, L_3 are quadratic polynomials, and

$$L_i(x_j) = \delta_{ij}, \qquad i,j = 1,2,3,$$

so that p is the (unique) quadratic interpolating polynomial for the given data.
The polynomials L_1, L_2 and L_3 are known as the Lagrange basis for quadratic polynomial interpolation.

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$L_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$L_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$\begin{aligned} L_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10 \\ L_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} \end{aligned}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$\begin{aligned} L_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10 \\ L_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)} = -\frac{4}{3}x^2 + 8x - \frac{32}{3} \end{aligned}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$\begin{aligned}
L_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10 \\
L_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)} = -\frac{4}{3}x^2 + 8x - \frac{32}{3} \\
L_3(x) &= \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}
\end{aligned}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$\begin{aligned}
L_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10 \\
L_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)} = -\frac{4}{3}x^2 + 8x - \frac{32}{3} \\
L_3(x) &= \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = \frac{(x-2)(x-2.5)}{(4-2)(4-2.5)} = \frac{x^2}{3} - \frac{3}{2}x + \frac{5}{3}
\end{aligned}$$

Example

Consider the data

x	2	2.5	4
y	0.5	0.4	0.25

The quadratic interpolating polynomial is of the form

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3, \tag{1}$$

where

$$\begin{aligned}
L_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = x^2 - \frac{13}{2}x + 10 \\
L_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)} = -\frac{4}{3}x^2 + 8x - \frac{32}{3} \\
L_3(x) &= \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = \frac{(x-2)(x-2.5)}{(4-2)(4-2.5)} = \frac{x^2}{3} - \frac{3}{2}x + \frac{5}{3}
\end{aligned}$$

Plugging these back into (1) together with the given y-values we get

$$\begin{aligned}
p(x) &= \left(x^2 - \tfrac{13}{2}x + 10\right)0.5 + \left(-\tfrac{4}{3}x^2 + 8x - \tfrac{32}{3}\right)0.4 + \left(\tfrac{x^2}{3} - \tfrac{3}{2}x + \tfrac{5}{3}\right)0.25 \\
&= 0.05x^2 - 0.425x + 1.15
\end{aligned}$$

Example

Note that we also could have set up a system of linear equations to find the coefficients a, b, c of a general quadratic polynomial

$$p(x) = ax^2 + bx + c \tag{2}$$

Example

Note that we also could have set up a system of linear equations to find the coefficients a, b, c of a general quadratic polynomial

$$p(x) = ax^2 + bx + c \tag{2}$$

by plugging the three given pairs of x and y-values into (2). This yields

$$\begin{aligned} 0.5 &= a(2)^2 + b(2) + c \\ 0.4 &= a(2.5)^2 + b(2.5) + c \\ 0.25 &= a(4)^2 + b(4) + c \end{aligned}$$

Example

Note that we also could have set up a system of linear equations to find the coefficients a, b, c of a general quadratic polynomial

$$p(x) = ax^2 + bx + c \tag{2}$$

by plugging the three given pairs of x and y-values into (2). This yields

$$\begin{aligned} 0.5 &= a(2)^2 + b(2) + c \\ 0.4 &= a(2.5)^2 + b(2.5) + c \\ 0.25 &= a(4)^2 + b(4) + c \end{aligned}$$

or, in matrix form, $\mathsf{A}\boldsymbol{c} = \boldsymbol{y}$ with

$$\mathsf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 6.25 & 2.5 & 1 \\ 16 & 4 & 1 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} 0.5 \\ 0.4 \\ 0.25 \end{bmatrix}$$

Example

Note that we also could have set up a system of linear equations to find the coefficients a, b, c of a general quadratic polynomial

$$p(x) = ax^2 + bx + c \tag{2}$$

by plugging the three given pairs of x and y-values into (2). This yields

$$\begin{aligned} 0.5 &= a(2)^2 + b(2) + c \\ 0.4 &= a(2.5)^2 + b(2.5) + c \\ 0.25 &= a(4)^2 + b(4) + c \end{aligned}$$

or, in matrix form, $\mathsf{A}\boldsymbol{c} = \boldsymbol{y}$ with

$$\mathsf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 6.25 & 2.5 & 1 \\ 16 & 4 & 1 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} 0.5 \\ 0.4 \\ 0.25 \end{bmatrix}$$

The matrix A is known as a Vandermonde matrix, and the basis $\{x^2, x, 1\}$ is referred to as the monomial basis.

Theorem

Assume data $(x_1, y_1), \ldots, (x_n, y_n)$ with distinct x-values are given. Then there exists a unique polynomial

$$p(x) = \sum_{k=1}^{n} L_k(x) y_k$$

of degree at most $n-1$ with Lagrange basis polynomials

$$L_k(x) = \prod_{j=1, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}, \qquad k = 1, \ldots, n$$

such that p interpolates the data, i.e.,

$$p(x_j) = y_j, \qquad j = 1, \ldots, n.$$

Proof.

Existence is established by the Langrange interpolation formula.

Proof.

Existence is established by the Langrange interpolation formula.

To show uniqueness, we assume that p and q are both interpolating polynomials of degree $n-1$.

Proof.

Existence is established by the Langrange interpolation formula.

To show uniqueness, we assume that p and q are both interpolating polynomials of degree $n-1$.
Then their difference $r = p - q$ is also a polynomial of degree $n-1$.

Proof.

Existence is established by the Langrange interpolation formula.

To show uniqueness, we assume that p and q are both interpolating polynomials of degree $n-1$.
Then their difference $r = p - q$ is also a polynomial of degree $n-1$.
By the *fundamental theorem of algebra* r has $n-1$ roots.

Proof.

Existence is established by the Langrange interpolation formula.

To show uniqueness, we assume that p and q are both interpolating polynomials of degree $n-1$.
Then their difference $r = p - q$ is also a polynomial of degree $n-1$.
By the *fundamental theorem of algebra* r has $n-1$ roots.

On the other hand (since p and q interpolate the data),

$$r(x_j) = p(x_j) - q(x_j) = y_j - y_j = 0, \quad j = 1, \ldots, n,$$

so that r has n roots.

Proof.

Existence is established by the Langrange interpolation formula.

To show uniqueness, we assume that p and q are both interpolating polynomials of degree $n-1$.
Then their difference $r = p - q$ is also a polynomial of degree $n-1$.
By the *fundamental theorem of algebra* r has $n-1$ roots.

On the other hand (since p and q interpolate the data),

$$r(x_j) = p(x_j) - q(x_j) = y_j - y_j = 0, \quad j = 1, \ldots, n,$$

so that r has n roots.

The only way to reconcile this apparent contradiction is if $r \equiv 0$.
However, this means that $p = q$, i.e., the interpolating polynomial is unique. □

The Vandermonde approach works for arbitrary degree interpolation problems. If data $(x_1, y_1), \ldots, (x_n, y_n)$ are given, then the Vandermonde matrix is

$$A = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix}$$

The Vandermonde approach works for arbitrary degree interpolation problems. If data $(x_1, y_1), \ldots, (x_n, y_n)$ are given, then the Vandermonde matrix is

$$A = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix}$$

In MATLAB we can generate a Vandermonde matrix with the command `vander(x)`, where the vector $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ contains the data sites.

The Vandermonde approach works for arbitrary degree interpolation problems. If data $(x_1, y_1), \ldots, (x_n, y_n)$ are given, then the Vandermonde matrix is

$$A = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix}$$

In MATLAB we can generate a Vandermonde matrix with the command `vander(x)`, where the vector $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ contains the data sites.

Note that it is not recommended to work with the Vandermonde matrix (and determine polynomial interpolants via the associated linear system) since the Vandermonde matrix is the prototype of an ill-conditioned matrix.

Polynomial Interpolation in MATLAB

The following function uses the Lagrange form to evaluate the polynomial interpolant of the data $(x_1, y_1), \ldots, (x_n, y_n)$ provided in the vectors `x` and `y` at the points $u_1, \ldots, u_m$ provided in `u`.

Polynomial Interpolation in MATLAB

The following function uses the Lagrange form to evaluate the polynomial interpolant of the data $(x_1, y_1), \ldots, (x_n, y_n)$ provided in the vectors `x` and `y` at the points $u_1, \ldots, u_m$ provided in `u`.

```
function v = polyinterp(x,y,u)
n = length(x);
v = zeros(size(u));
for k = 1:n
   w = ones(size(u));
   for j = [1:k-1 k+1:n]
      w = (u-x(j))./(x(k)-x(j)).*w;      % compute L_k(u)
   end
   v = v + w*y(k);
end
```

Polynomial Interpolation in MATLAB

The following function uses the Lagrange form to evaluate the polynomial interpolant of the data $(x_1, y_1), \ldots, (x_n, y_n)$ provided in the vectors `x` and `y` at the points $u_1, \ldots, u_m$ provided in `u`.

```
function v = polyinterp(x,y,u)
n = length(x);
v = zeros(size(u));
for k = 1:n
   w = ones(size(u));
   for j = [1:k-1 k+1:n]
      w = (u-x(j))./(x(k)-x(j)).*w;      % compute L_k(u)
   end
   v = v + w*y(k);
end
```

Note that while the sum and product of the Lagrange formula are performed with for-loops, the evaluation at the points in `u` is done in parallel.

Polynomial Interpolation in MATLAB

The following function uses the Lagrange form to evaluate the polynomial interpolant of the data $(x_1, y_1), \ldots, (x_n, y_n)$ provided in the vectors `x` and `y` at the points $u_1, \ldots, u_m$ provided in `u`.

```
function v = polyinterp(x,y,u)
n = length(x);
v = zeros(size(u));
for k = 1:n
   w = ones(size(u));
   for j = [1:k-1 k+1:n]
      w = (u-x(j))./(x(k)-x(j)).*w;      % compute L_k(u)
   end
   v = v + w*y(k);
end
```

Note that while the sum and product of the Lagrange formula are performed with for-loops, the evaluation at the points in `u` is done in parallel.

Run `PolyinterpDemo.m` to evaluate our earlier quadratic polynomial.

Outline

Problem

When we interpolated the output data from the skydive problem we saw that polynomial interpolation in general does not work for many data points, i.e., with high degree polynomials[a]*.*
Polynomials are too smooth and therefore give rise to undesired oscillations.

Problem

When we interpolated the output data from the skydive problem we saw that polynomial interpolation in general does not work for many data points, i.e., with high degree polynomials[a].
Polynomials are too smooth and therefore give rise to undesired oscillations.

[a] Things are different if we can optimally choose the data sites.

Problem

When we interpolated the output data from the skydive problem we saw that polynomial interpolation in general does not work for many data points, i.e., with high degree polynomials[a]*.*
Polynomials are too smooth and therefore give rise to undesired oscillations.

Solution

Reduce the smoothness of the interpolant, i.e., use piecewise polynomials.

[a]Things are different if we can optimally choose the data sites.

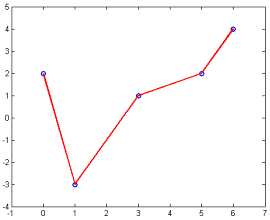
Problem

When we interpolated the output data from the skydive problem we saw that polynomial interpolation in general does not work for many data points, i.e., with high degree polynomials[a]*.*
Polynomials are too smooth and therefore give rise to undesired oscillations.

Solution

Reduce the smoothness of the interpolant, i.e., use piecewise polynomials.

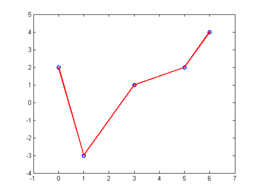
Simplest variant: "connect-the-dots", i.e., piecewise linear interpolation.
Note: this is how MATLAB creates continuous graphs.



[a] Things are different if we can optimally choose the data sites.

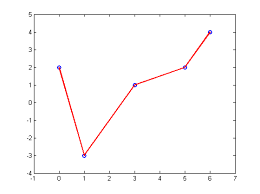
A piecewise function is defined interval-by-interval. For example,

$$\ell(x) = \begin{cases} 2 - 5x, & 0 \le x < 1 \\ -5 + 2x, & 1 \le x < 3 \\ -\frac{1}{2} + \frac{1}{2}x, & 3 \le x < 5 \\ -8 + 2x, & 5 \le x \le 6 \end{cases}$$



A piecewise function is defined interval-by-interval. For example,

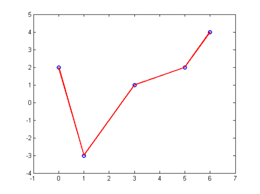
$$\ell(x) = \begin{cases} 2 - 5x, & 0 \le x < 1 \\ -5 + 2x, & 1 \le x < 3 \\ -\frac{1}{2} + \frac{1}{2}x, & 3 \le x < 5 \\ -8 + 2x, & 5 \le x \le 6 \end{cases}$$



In order to be able to evaluate piecewise polynomials and splines efficiently we need to know which piece of the interpolant our evaluation point x lies in.

A piecewise function is defined interval-by-interval. For example,

$$\ell(x) = \begin{cases} 2-5x, & 0 \le x < 1 \\ -5+2x, & 1 \le x < 3 \\ -\frac{1}{2}+\frac{1}{2}x, & 3 \le x < 5 \\ -8+2x, & 5 \le x \le 6 \end{cases}$$



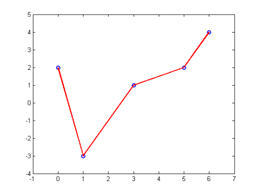
In order to be able to evaluate piecewise polynomials and splines efficiently we need to know which piece of the interpolant our evaluation point x lies in.

We need to find the index k such that $x_k \le x < x_{k+1}$ since

$$\ell(x) = \begin{cases} \ell_1(x), & x_1 \le x < x_2 \\ \ell_2(x), & x_2 \le x < x_3 \\ \quad\vdots \quad , & \\ \ell_{n-1}(x), & x_{n-1} \le x \le x_n \end{cases}$$

A piecewise function is defined interval-by-interval. For example,

$$\ell(x) = \begin{cases} 2-5x, & 0 \le x < 1 \\ -5+2x, & 1 \le x < 3 \\ -\frac{1}{2}+\frac{1}{2}x, & 3 \le x < 5 \\ -8+2x, & 5 \le x \le 6 \end{cases}$$



In order to be able to evaluate piecewise polynomials and splines efficiently we need to know which piece of the interpolant our evaluation point x lies in.

We need to find the index k such that $x_k \le x < x_{k+1}$ since

$$\ell(x) = \begin{cases} \ell_1(x), & x_1 \le x < x_2 \\ \ell_2(x), & x_2 \le x < x_3 \\ \quad\vdots\quad, & \\ \ell_{n-1}(x), & x_{n-1} \le x \le x_n \end{cases}$$

For example, if we want to find $\ell(4)$ above, then we have to evaluate the piece ℓ_3 between $x_3 = 3$ and $x_4 = 5$.

Since a linear function connecting (x_1, y_1) and (x_2, y_2) can be written as

$$y = y_1 + \delta(x - x_1)$$

with slope δ,

Since a linear function connecting (x_1, y_1) and (x_2, y_2) can be written as

$$y = y_1 + \delta(x - x_1)$$

with slope δ, the k-th piece of the piecewise linear interpolant is given by

$$\ell_k(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k).$$

Since a linear function connecting (x_1, y_1) and (x_2, y_2) can be written as

$$y = y_1 + \delta(x - x_1)$$

with slope δ, the k-th piece of the piecewise linear interpolant is given by

$$\ell_k(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k).$$

The points x_k are sometimes called *breakpoints* or *knots*.

MATLAB code `piecelin.m` from [NCM]

The following function evaluates the piecewise linear interpolant to the data provided in the vectors `x` and `y` at all of the points in `u`.

```
function v = piecelin(x,y,u)
%  Compute all the slopes as first divided difference
   delta = diff(y)./diff(x);
%  Find subinterval indices k s.t. x(k) <= u < x(k+1)
   n = length(x);
   k = ones(size(u));
   for j = 2:n-1
      k(x(j) <= u) = j;
   end
%  Evaluate interpolant at all points in u
   s = u - x(k);
   v = y(k) + s.*delta(k);
```

MATLAB code `piecelin.m` from [NCM]

The following function evaluates the piecewise linear interpolant to the data provided in the vectors `x` and `y` at all of the points in `u`.

```
function v = piecelin(x,y,u)
%   Compute all the slopes as first divided difference
    delta = diff(y)./diff(x);
%   Find subinterval indices k s.t. x(k) <= u < x(k+1)
    n = length(x);
    k = ones(size(u));
    for j = 2:n-1
        k(x(j) <= u) = j;
    end
%   Evaluate interpolant at all points in u
    s = u - x(k);
    v = y(k) + s.*delta(k);
```

Note that in the statement `k(x(j) <= u) = j;` *all* entries of `k` whose corresponding entries of `u` are $\geq x_j$ are set to j (see `PiecelinDemo.m`).

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

Just as a linear polynomial naturally matches the two function values given at the endpoints of the subinterval, the four coefficients of the cubic polynomial can be determined by matching function and derivative values at the endpoints.

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

Just as a linear polynomial naturally matches the two function values given at the endpoints of the subinterval, the four coefficients of the cubic polynomial can be determined by matching function and derivative values at the endpoints.

If these derivative values are given, then we can directly use them. The resulting method is known as piecewise cubic Hermite interpolation.

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

Just as a linear polynomial naturally matches the two function values given at the endpoints of the subinterval, the four coefficients of the cubic polynomial can be determined by matching function and derivative values at the endpoints.

If these derivative values are given, then we can directly use them. The resulting method is known as piecewise cubic Hermite interpolation.

If we don't have this additional data, we can somehow approximate it:

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

Just as a linear polynomial naturally matches the two function values given at the endpoints of the subinterval, the four coefficients of the cubic polynomial can be determined by matching function and derivative values at the endpoints.

If these derivative values are given, then we can directly use them. The resulting method is known as piecewise cubic Hermite interpolation.

If we don't have this additional data, we can somehow approximate it:

- The piecewise cubic interpolant, `pchip`, used in MATLAB generates the additional derivative data so that the resulting interpolant is continuously differentiable and shape preserving, i.e., the interpolant does not have any oscillations, over- or undershoots that are not present in the data.

If we want something a bit fancier (i.e., smoother), then we can work with a cubic polynomial piece on each subinterval.

Just as a linear polynomial naturally matches the two function values given at the endpoints of the subinterval, the four coefficients of the cubic polynomial can be determined by matching function and derivative values at the endpoints.

If these derivative values are given, then we can directly use them. The resulting method is known as piecewise cubic Hermite interpolation.

If we don't have this additional data, we can somehow approximate it:

- The piecewise cubic interpolant, `pchip`, used in MATLAB generates the additional derivative data so that the resulting interpolant is continuously differentiable and shape preserving, i.e., the interpolant does not have any oscillations, over- or undershoots that are not present in the data.
- For the cubic spline the derivatives are determined so that the pieces are twice continuously differentiable at the breakpoints.

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

- For $p(x_k)$ we note that $s = 0$, and so $p(x_k) = y_k$.

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

- For $p(x_k)$ we note that $s = 0$, and so $p(x_k) = y_k$.
- For $p(x_{k+1})$ we have $s = h$ and $p(x_{k+1}) = y_{k+1}$.

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

- For $p(x_k)$ we note that $s = 0$, and so $p(x_k) = y_k$.
- For $p(x_{k+1})$ we have $s = h$ and $p(x_{k+1}) = y_{k+1}$.
- For the other two conditions we need

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

- For $p(x_k)$ we note that $s = 0$, and so $p(x_k) = y_k$.
- For $p(x_{k+1})$ we have $s = h$ and $p(x_{k+1}) = y_{k+1}$.
- For the other two conditions we need

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

  and see that

  - $p'(x_k) = d_k$ (since $s = 0$),

The Cubic Hermite Interpolant

Assume we now are given function and derivative values, i.e., (x_k, y_k, d_k) and $(x_{k+1}, y_{k+1}, d_{k+1})$.
We can verify (or construct by solving a 4×4 linear system) that the cubic polynomial interpolating this set of data is

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = x_{k+1} - x_k$:

- For $p(x_k)$ we note that $s = 0$, and so $p(x_k) = y_k$.
- For $p(x_{k+1})$ we have $s = h$ and $p(x_{k+1}) = y_{k+1}$.
- For the other two conditions we need

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

  and see that

  - $p'(x_k) = d_k$ (since $s = 0$),
  - and $p'(x_{k+1}) = d_{k+1}$ (since $s = h$).

How does `pchip` set the slopes d_k?

The idea is to avoid over- and undershoots at each x_k.

How does `pchip` set the slopes d_k?

The idea is to avoid over- and undershoots at each x_k.

- If the slopes $\delta_{k-1} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$ and $\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$ to the left and right of x_k have opposite signs, then we set $d_k = 0$.

How does `pchip` set the slopes d_k?

The idea is to avoid over- and undershoots at each x_k.

- If the slopes $\delta_{k-1} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$ and $\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$ to the left and right of x_k have opposite signs, then we set $d_k = 0$.
- If the slopes δ_{k-1} and δ_k have the same sign and the corresponding intervals are of the same length, then we set the slope d_k as the *harmonic mean*:

$$d_k = \frac{2}{\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k}}.$$

How does `pchip` set the slopes d_k?

The idea is to avoid over- and undershoots at each x_k.

- If the slopes $\delta_{k-1} = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}$ and $\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$ to the left and right of x_k have opposite signs, then we set $d_k = 0$.
- If the slopes δ_{k-1} and δ_k have the same sign and the corresponding intervals are of the same length, then we set the slope d_k as the *harmonic mean*:

$$d_k = \frac{2}{\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k}}.$$

- If the slopes δ_{k-1} and δ_k have the same sign, but the corresponding intervals are of different length, then we set the slope d_k as a *weighted harmonic mean*:

$$d_k = \frac{w_1 + w_2}{\frac{w_1}{\delta_{k-1}} + \frac{w_2}{\delta_k}},$$

  where $w_1 = 2h_k + h_{k-1}$, $w_2 = h_k + 2h_{k-1}$, and $h_k = x_{k+1} - x_k$.

How does `pchip` set the slopes d_k? (cont.)

- The slopes at the endpoints are set by slightly different rules.

How does `pchip` set the slopes d_k? (cont.)

- The slopes at the endpoints are set by slightly different rules.

Run `PchipDemo.m` to see an example of the shape-preserving C^1-cubic Hermite interpolant, and view `pchiptx.m` from [NCM] for more details (for example, how the slopes at the endpoints are determined).

Remark

While the derivative of the shape-preserving piecewise cubic Hermite interpolant at the breakpoints will always be continuous, it is in general not differentiable. This means that `pchip` *generates a* C^1*-continuous interpolant.*

Remark

While the derivative of the shape-preserving piecewise cubic Hermite interpolant at the breakpoints will always be continuous, it is in general ***not differentiable****. This means that* `pchip` *generates a C^1-continuous interpolant.*

In order to get a piecewise cubic interpolant that is C^2-continuous at the breakpoints we need to consider ***splines****.*

Remark

While the derivative of the shape-preserving piecewise cubic Hermite interpolant at the breakpoints will always be continuous, it is in general not differentiable. This means that `pchip` *generates a C^1-continuous interpolant.*

In order to get a piecewise cubic interpolant that is C^2-continuous at the breakpoints we need to consider splines.

One reason for wanting a C^2 interpolant is that light reflections appear with a smoothness of one order lower than the reflecting surface, i.e., a C^1 surface will generate non-smooth light reflections. Car manufacturers and owners don't like this!

Remark

While the derivative of the shape-preserving piecewise cubic Hermite interpolant at the breakpoints will always be continuous, it is in general not differentiable. This means that `pchip` *generates a C^1-continuous interpolant.*

In order to get a piecewise cubic interpolant that is C^2-continuous at the breakpoints we need to consider splines.

One reason for wanting a C^2 interpolant is that light reflections appear with a smoothness of one order lower than the reflecting surface, i.e., a C^1 surface will generate non-smooth light reflections. Car manufacturers and owners don't like this!

Remark

While the derivative of the shape-preserving piecewise cubic Hermite interpolant at the breakpoints will always be continuous, it is in general not differentiable. This means that `pchip` *generates a C^1-continuous interpolant.*

In order to get a piecewise cubic interpolant that is C^2-continuous at the breakpoints we need to consider splines.

One reason for wanting a C^2 interpolant is that light reflections appear with a smoothness of one order lower than the reflecting surface, i.e., a C^1 surface will generate non-smooth light reflections. Car manufacturers and owners don't like this!

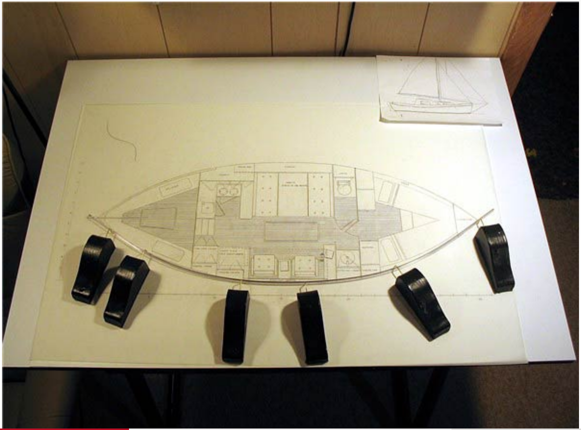
Outline

History of Splines

Mathematical splines originated in the CAD software developed by the aircraft and automobile design industry in the late 1950s and early 1960s and were named after a special wooden or metal drafting tool used in the manual design of ship hulls:

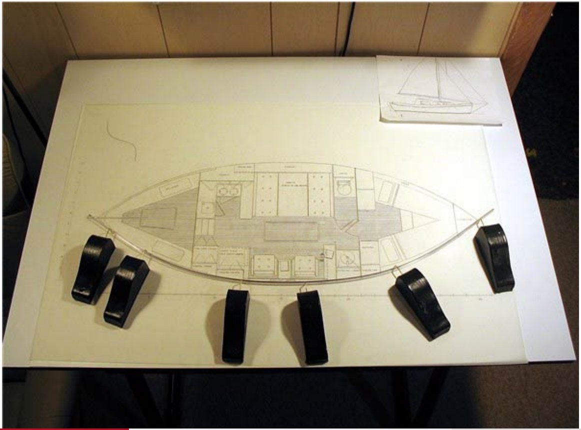
History of Splines

Mathematical splines originated in the CAD software developed by the aircraft and automobile design industry in the late 1950s and early 1960s and were named after a special wooden or metal drafting tool used in the manual design of ship hulls: a spline.



History of Splines

Mathematical splines originated in the CAD software developed by the aircraft and automobile design industry in the late 1950s and early 1960s and were named after a special wooden or metal drafting tool used in the manual design of ship hulls: a spline.



Return

General Splines

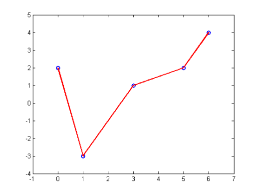
Splines are special piecewise polynomials whose order of smoothness at the breakpoints is always one less than the polynomial degree.

General Splines

Splines are special piecewise polynomials whose order of smoothness at the breakpoints is always one less than the polynomial degree.

Example

- If the pieces are generated by linear functions, then the smoothness is zero, i.e., the pieces join continuously, but the derivatives are in general not continuous. This is the same as the piecewise linear interpolant discussed earlier.

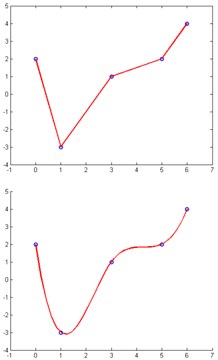


General Splines

Splines are special piecewise polynomials whose order of smoothness at the breakpoints is always one less than the polynomial degree.

Example

- If the pieces are generated by linear functions, then the smoothness is zero, i.e., the pieces join continuously, but the derivatives are in general not continuous. This is the same as the piecewise linear interpolant discussed earlier.
- Cubic splines are required to join with C^2 smoothness (i.e., continuously differentiable first derivative) at the knots. This is more specific than general piecewise cubics.

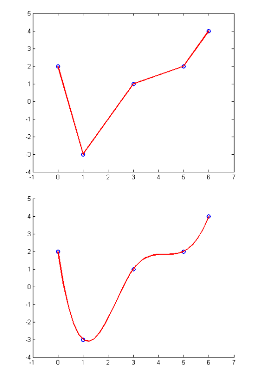


General Splines

Splines are special piecewise polynomials whose order of smoothness at the breakpoints is always one less than the polynomial degree.

Example

- If the pieces are generated by linear functions, then the smoothness is zero, i.e., the pieces join continuously, but the derivatives are in general not continuous. This is the same as the piecewise linear interpolant discussed earlier.
- Cubic splines are required to join with C^2 smoothness (i.e., continuously differentiable first derivative) at the knots. This is more specific than general piecewise cubics.



- In general, a spline of degree k will have C^{k-1} smoothness at the breakpoints.

The most interesting — and also most commonly used — splines are the cubic splines.

The most interesting — and also most commonly used — splines are the cubic splines.
We can start exactly as with the `pchip` interpolant, i.e.,

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$, $h = x_{k+1} - x_k$, and d_k and d_{k+1} are slopes at x_k and x_{k+1} (to be determined by the spline method).

The most interesting — and also most commonly used — splines are the cubic splines.
We can start exactly as with the `pchip` interpolant, i.e.,

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$, $h = x_{k+1} - x_k$, and d_k and d_{k+1} are slopes at x_k and x_{k+1} (to be determined by the spline method). Moreover (as before),

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

The most interesting — and also most commonly used — splines are the cubic splines.

We can start exactly as with the `pchip` interpolant, i.e.,

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$, $h = x_{k+1} - x_k$, and d_k and d_{k+1} are slopes at x_k and x_{k+1} (to be determined by the spline method). Moreover (as before),

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

and now also

$$p''(x) = \frac{6h - 12s}{h^3} y_{k+1} - \frac{6h - 12s}{h^3} y_k + \frac{6s - 2h}{h^2} d_{k+1} + \frac{6s - 4h}{h^2} d_k$$

The most interesting — and also most commonly used — splines are the cubic splines.
We can start exactly as with the `pchip` interpolant, i.e.,

$$p(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$, $h = x_{k+1} - x_k$, and d_k and d_{k+1} are slopes at x_k and x_{k+1} (to be determined by the spline method). Moreover (as before),

$$p'(x) = \frac{6hs - 6s^2}{h^3} y_{k+1} - \frac{6hs - 6s^2}{h^3} y_k + \frac{3s^2 - 2sh}{h^2} d_{k+1} + \frac{(s-h)(3s-h)}{h^2} d_k$$

and now also

$$\begin{aligned} p''(x) &= \frac{6h - 12s}{h^3} y_{k+1} - \frac{6h - 12s}{h^3} y_k + \frac{6s - 2h}{h^2} d_{k+1} + \frac{6s - 4h}{h^2} d_k \\ &= \frac{(6h - 12s)\delta_k + (6s - 2h)d_{k+1} + (6s - 4h)d_k}{h^2}, \end{aligned}$$

where $\delta_k = \frac{y_{k+1} - y_k}{h} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$.

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
    - join together continuously,
    - join with a continuous derivative,
    - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓
- Continuous joints

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓
- Continuous joints automatically covered by interpolation ✓

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓
- Continuous joints automatically covered by interpolation ✓
- Continuous derivative

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces $\checkmark$
- Continuous joints automatically covered by interpolation $\checkmark$
- Continuous derivative also covered by construction $\checkmark$

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓
- Continuous joints automatically covered by interpolation ✓
- Continuous derivative also covered by construction ✓
- Continuous second derivative

What conditions does a cubic spline interpolant have to satisfy?

- We need to have cubic pieces between the breakpoints/knots.
- At the knots we need to
  - join together continuously,
  - join with a continuous derivative,
  - join with a continuous second derivative.
- Each piece needs to interpolate the corresponding data, i.e., $p(x_i) = y_i$, $i = k, k+1$.

How far are we on this list?

- Cubic pieces ✓
- Continuous joints automatically covered by interpolation ✓
- Continuous derivative also covered by construction ✓
- Continuous second derivative — still to be done

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

1

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

1

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

We now need to evaluate at $x = x_k$.

1

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

We now need to evaluate at $x = x_k$.

- p''_k: $\quad \longrightarrow \quad s = x - x_k|_{x=x_k} = 0$

1

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

We now need to evaluate at $x = x_k$.

- p''_k: $\longrightarrow$ $s = x - x_k|_{x=x_k} = 0$
- p''_{k-1}: $\longrightarrow$ $s = x - x_{k-1}|_{x=x_k} = x_k - x_{k-1} = h_{k-1}$

1

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

We now need to evaluate at $x = x_k$.

- p''_k: $\longrightarrow$ $s = x - x_k|_{x=x_k} = 0$
- p''_{k-1}: $\longrightarrow$ $s = x - x_{k-1}|_{x=x_k} = x_k - x_{k-1} = h_{k-1}$

Therefore[1],

$$p''_{k-1}(x_k) = \frac{(6h_{k-1} - 12h_{k-1})\delta_{k-1} + (6h_{k-1} - 2h_{k-1})d_k + (6h_{k-1} - 4h_{k-1})d_{k-1}}{h_{k-1}^2}$$

[1] now being more careful with notation and adding subscripts to h (which technically should've been there earlier, but were omitted to prevent notational clutter)

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$p''_{k-1}(x) = \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2},$$

$$p''_k(x) = \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2},$$

We now need to evaluate at $x = x_k$.

- p''_k: $\longrightarrow$ $s = x - x_k|_{x=x_k} = 0$
- p''_{k-1}: $\longrightarrow$ $s = x - x_{k-1}|_{x=x_k} = x_k - x_{k-1} = h_{k-1}$

Therefore[1],

$$\begin{aligned} p''_{k-1}(x_k) &= \frac{(6h_{k-1} - 12h_{k-1})\delta_{k-1} + (6h_{k-1} - 2h_{k-1})d_k + (6h_{k-1} - 4h_{k-1})d_{k-1}}{h^2_{k-1}} \\ &= \frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}}, \end{aligned}$$

[1] now being more careful with notation and adding subscripts to h (which technically should've been there earlier, but were omitted to prevent notational clutter)

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$\begin{aligned} p''_{k-1}(x) &= \frac{(6h-12s)\delta_{k-1} + (6s-2h)d_k + (6s-4h)d_{k-1}}{h^2}, \\ p''_k(x) &= \frac{(6h-12s)\delta_k + (6s-2h)d_{k+1} + (6s-4h)d_k}{h^2}, \end{aligned}$$

We now need to evaluate at $x = x_k$.

- p''_k: $\longrightarrow$ $s = x - x_k|_{x=x_k} = 0$
- p''_{k-1}: $\longrightarrow$ $s = x - x_{k-1}|_{x=x_k} = x_k - x_{k-1} = h_{k-1}$

Therefore[1],

$$\begin{aligned} p''_{k-1}(x_k) &= \frac{(6h_{k-1} - 12h_{k-1})\delta_{k-1} + (6h_{k-1} - 2h_{k-1})d_k + (6h_{k-1} - 4h_{k-1})d_{k-1}}{h_{k-1}^2} \\ &= \frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}}, \\ p''_k(x_k) &= \frac{6h_k\delta_k - 2h_k d_{k+1} - 4h_k d_k}{h_k^2} \end{aligned}$$

[1] now being more careful with notation and adding subscripts to h (which technically should've been there earlier, but were omitted to prevent notational clutter)

How to get continuity of p''

We need to consider two different cubic polynomial pieces:

p_{k-1}, defined on $[x_{k-1}, x_k]$, and p_k, defined on $[x_k, x_{k+1}]$.

From above, we know

$$p''_{k-1}(x) = \frac{(6h - 12s)\delta_{k-1} + (6s - 2h)d_k + (6s - 4h)d_{k-1}}{h^2},$$

$$p''_k(x) = \frac{(6h - 12s)\delta_k + (6s - 2h)d_{k+1} + (6s - 4h)d_k}{h^2},$$

We now need to evaluate at $x = x_k$.

- p''_k: $\longrightarrow$ $s = x - x_k|_{x=x_k} = 0$
- p''_{k-1}: $\longrightarrow$ $s = x - x_{k-1}|_{x=x_k} = x_k - x_{k-1} = h_{k-1}$

Therefore[1],

$$p''_{k-1}(x_k) = \frac{(6h_{k-1} - 12h_{k-1})\delta_{k-1} + (6h_{k-1} - 2h_{k-1})d_k + (6h_{k-1} - 4h_{k-1})d_{k-1}}{h^2_{k-1}}$$

$$= \frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}},$$

$$p''_k(x_k) = \frac{6h_k\delta_k - 2h_k d_{k+1} - 4h_k d_k}{h^2_k} = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}.$$

[1] now being more careful with notation and adding subscripts to h (which technically should've been there earlier, but were omitted to prevent notational clutter)

How to get continuity of p'' (cont.)

To get continuity we now need to ensure $p''_{k-1}(x_k) = p''_k(x_k)$, i.e.,

$$\frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}} = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}.$$

How to get continuity of p'' (cont.)

To get continuity we now need to ensure $p''_{k-1}(x_k) = p''_k(x_k)$, i.e.,

$$\frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}} = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}.$$

Since the h_k and δ_k are differences of the given data values (and therefore known quantities) we isolate them to the right-hand side and get

$$\frac{4d_k + 2d_{k-1}}{h_{k-1}} + \frac{2d_{k+1} + 4d_k}{h_k} = \frac{6\delta_{k-1}}{h_{k-1}} + \frac{6\delta_k}{h_k}$$

How to get continuity of p'' (cont.)

To get continuity we now need to ensure $p''_{k-1}(x_k) = p''_k(x_k)$, i.e.,

$$\frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}} = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}.$$

Since the h_k and δ_k are differences of the given data values (and therefore known quantities) we isolate them to the right-hand side and get

$$\begin{aligned} \frac{4d_k + 2d_{k-1}}{h_{k-1}} + \frac{2d_{k+1} + 4d_k}{h_k} &= \frac{6\delta_{k-1}}{h_{k-1}} + \frac{6\delta_k}{h_k} \\ \iff \quad (2d_k + d_{k-1})h_k + (d_{k+1} + 2d_k)h_{k-1} &= 3\delta_{k-1}h_k + 3\delta_k h_{k-1} \end{aligned}$$

How to get continuity of p'' (cont.)

To get continuity we now need to ensure $p''_{k-1}(x_k) = p''_k(x_k)$, i.e.,

$$\frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}} = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}.$$

Since the h_k and δ_k are differences of the given data values (and therefore known quantities) we isolate them to the right-hand side and get

$$\begin{aligned} \frac{4d_k + 2d_{k-1}}{h_{k-1}} + \frac{2d_{k+1} + 4d_k}{h_k} &= \frac{6\delta_{k-1}}{h_{k-1}} + \frac{6\delta_k}{h_k} \\ \iff \quad (2d_k + d_{k-1})h_k + (d_{k+1} + 2d_k)h_{k-1} &= 3\delta_{k-1}h_k + 3\delta_k h_{k-1} \\ \iff \quad h_k d_{k-1} + 2(h_{k-1} + h_k)d_k + h_{k-1}d_{k+1} &= 3(h_{k-1}\delta_k + h_k\delta_{k-1}). \end{aligned}$$

How to get continuity of p'' (cont.)

Note that the equation, which we derived for an arbitrary knot x_k,

$$h_k d_{k-1} + 2(h_{k-1} + h_k) d_k + h_{k-1} d_{k+1} = 3(h_{k-1}\delta_k + h_k \delta_{k-1})$$

needs to hold for all interior knots $x_2, x_3, \ldots, x_{n-1}$,

How to get continuity of p'' (cont.)

Note that the equation, which we derived for an arbitrary knot x_k,

$$h_k d_{k-1} + 2(h_{k-1} + h_k) d_k + h_{k-1} d_{k+1} = 3(h_{k-1}\delta_k + h_k \delta_{k-1})$$

needs to hold for all interior knots $x_2, x_3, \ldots, x_{n-1}$, i.e., we have the $(n-2) \times n$ system of linear equations $\mathsf{A}\boldsymbol{d} = \boldsymbol{r}$ with tridiagonal

$$\mathsf{A} = \begin{bmatrix} h_2 & 2(h_1 + h_2) & h_1 & & & \\ & h_3 & 2(h_2 + h_3) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-1} & 2(h_{n-2} + h_{n-1}) & h_{n-2} \end{bmatrix},$$

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \qquad \boldsymbol{r} = 3 \begin{bmatrix} h_1\delta_2 + h_2\delta_1 \\ h_2\delta_3 + h_3\delta_2 \\ \vdots \\ h_{n-2}\delta_{n-1} + h_{n-1}\delta_{n-2} \end{bmatrix}$$

How to get continuity of p'' (cont.)

Note that the equation, which we derived for an arbitrary knot x_k,

$$h_k d_{k-1} + 2(h_{k-1} + h_k) d_k + h_{k-1} d_{k+1} = 3(h_{k-1}\delta_k + h_k \delta_{k-1})$$

needs to hold for all interior knots $x_2, x_3, \ldots, x_{n-1}$, i.e., we have the $(n-2) \times n$ system of linear equations $\mathsf{A}\boldsymbol{d} = \boldsymbol{r}$ with tridiagonal

$$\mathsf{A} = \begin{bmatrix} h_2 & 2(h_1+h_2) & h_1 & & & \\ & h_3 & 2(h_2+h_3) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-1} & 2(h_{n-2}+h_{n-1}) & h_{n-2} \end{bmatrix},$$

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \qquad \boldsymbol{r} = 3 \begin{bmatrix} h_1\delta_2 + h_2\delta_1 \\ h_2\delta_3 + h_3\delta_2 \\ \vdots \\ h_{n-2}\delta_{n-1} + h_{n-1}\delta_{n-2} \end{bmatrix}$$

Problem: We don't have enough conditions to determine all of the n unknown slope values $d_1, \ldots, d_n$!

End Conditions

There are many different types of cubic splines. They differ by which two equations we add to the linear system $A\boldsymbol{d} = \boldsymbol{r}$ to determine the slopes at the endpoints x_1 and x_n.

End Conditions

There are many different types of cubic splines. They differ by which two equations we add to the linear system $A\boldsymbol{d} = \boldsymbol{r}$ to determine the slopes at the endpoints x_1 and x_n.

For example,

- cubic natural splines: use zero second derivative at ends,
- cubic not-a-knot splines: use a single cubic on first two and last two intervals,
- cubic clamped (or complete) splines: specify first derivative values at ends,
- cubic periodic splines: ensure that value of function, first and second derivative are same at both ends.

Cubic Natural Splines

We set $p''(x) = 0$ when x is one of the endpoints, x_1 or x_n.

Cubic Natural Splines

We set $p''(x) = 0$ when x is one of the endpoints, x_1 or x_n. Physically, this has the effect of forcing the interpolating spline to have zero curvature at the ends, i.e., its behavior is similar to that of a thin rod

see mechanical spline.

Cubic Natural Splines

We set $p''(x) = 0$ when x is one of the endpoints, x_1 or x_n. Physically, this has the effect of forcing the interpolating spline to have zero curvature at the ends, i.e., its behavior is similar to that of a thin rod see mechanical spline.

Left end (enforce $p_1''(x_1) = 0$): From above we know

$$p_1''(x_1) = \frac{6\delta_1 - 2d_2 - 4d_1}{h_1},$$

Cubic Natural Splines

We set $p''(x) = 0$ when x is one of the endpoints, x_1 or x_n. Physically, this has the effect of forcing the interpolating spline to have zero curvature at the ends, i.e., its behavior is similar to that of a thin rod ▸ see mechanical spline.

Left end (enforce $p_1''(x_1) = 0$): From above we know

$$p_1''(x_1) = \frac{6\delta_1 - 2d_2 - 4d_1}{h_1},$$

so that we have the condition

$$4d_1 + 2d_2 = 6\delta_1. \tag{3}$$

Cubic Natural Splines

We set $p''(x) = 0$ when x is one of the endpoints, x_1 or x_n. Physically, this has the effect of forcing the interpolating spline to have zero curvature at the ends, i.e., its behavior is similar to that of a thin rod see mechanical spline.

Left end (enforce $p_1''(x_1) = 0$): From above we know

$$p_1''(x_1) = \frac{6\delta_1 - 2d_2 - 4d_1}{h_1},$$

so that we have the condition

$$4d_1 + 2d_2 = 6\delta_1. \tag{3}$$

Right end (enforce $p_{n-1}''(x_n) = 0$): From above

$$p_{n-1}''(x_n) = \frac{-6\delta_{n-1} + 4d_n + 2d_{n-1}}{h_{n-1}},$$

so that

$$2d_{n-1} + 4d_n = 6\delta_{n-1}. \tag{4}$$

Cubic Natural Splines (cont.)

The final linear system $A\boldsymbol{d} = \boldsymbol{r}$ to be solved for the unknown slopes of the cubic natural spline is obtained by adding equations (3) and (4) to the generic $(n-2) \times n$ tridiagonal linear system we derived earlier.

Cubic Natural Splines (cont.)

The final linear system $\mathsf{A}\boldsymbol{d} = \boldsymbol{r}$ to be solved for the unknown slopes of the cubic natural spline is obtained by adding equations (3) and (4) to the generic $(n-2) \times n$ tridiagonal linear system we derived earlier. Its components are

$$\mathsf{A} = \begin{bmatrix} 2 & 1 & & & & \\ h_2 & 2(h_1+h_2) & h_1 & & & \\ & h_3 & 2(h_2+h_3) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-1} & 2(h_{n-2}+h_{n-1}) & h_{n-2} \\ & & & & 1 & 2 \end{bmatrix},$$

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \qquad \boldsymbol{r} = 3 \begin{bmatrix} \delta_1 \\ h_1\delta_2 + h_2\delta_1 \\ h_2\delta_3 + h_3\delta_2 \\ \vdots \\ h_{n-2}\delta_{n-1} + h_{n-1}\delta_{n-2} \\ \delta_{n-1} \end{bmatrix}$$

Cubic Natural Splines (cont.)

Remark

The cubic natural spline is that interpolating C^2 function which minimizes the model for the bending energy of a thin rod. Thus the name seems justified.

Cubic Natural Splines (cont.)

Remark

The cubic natural spline is that interpolating C^2 function which minimizes the model for the bending energy of a thin rod. Thus the name seems justified.

The modified version `splinetx_natural.m` of the [NCM] routine `splinetx.m` performs cubic spline interpolation with natural end conditions.

Cubic Natural Splines (cont.)

Remark

The cubic natural spline is that interpolating C^2 function which minimizes the model for the bending energy of a thin rod. Thus the name seems justified.

The modified version `splinetx_natural.m` of the [NCM] routine `splinetx.m` performs cubic spline interpolation with natural end conditions.

See `SplineDemo.m` for an example.

Cubic Not-a-Knot Splines

Since the generic linear system is missing two equations (and we may not have any more data than the function values at the break points), we condense the representation and use two subintervals near each end (instead of one) to generate the cubic pieces.

Cubic Not-a-Knot Splines

Since the generic linear system is missing two equations (and we may not have any more data than the function values at the break points), we condense the representation and use two subintervals near each end (instead of one) to generate the cubic pieces.

Left end: define one cubic piece on $x_1 \leq x < x_3$, i.e., x_2 is not a knot.

Without giving the details[2] this leads to

$$h_2 d_1 + (h_1 + h_2) d_2 = \frac{(3h_1 + 2h_2) h_2 \delta_1 + h_1^2 \delta_2}{h_1 + h_2}. \qquad (5)$$

[2] we need to ensure that $p_1'''(x_2) = p_2'''(x_2)$

Cubic Not-a-Knot Splines

Since the generic linear system is missing two equations (and we may not have any more data than the function values at the break points), we condense the representation and use two subintervals near each end (instead of one) to generate the cubic pieces.

Left end: define one cubic piece on $x_1 \leq x < x_3$, i.e., x_2 is not a knot. Without giving the details[2] this leads to

$$h_2 d_1 + (h_1 + h_2) d_2 = \frac{(3h_1 + 2h_2)h_2\delta_1 + h_1^2\delta_2}{h_1 + h_2}. \tag{5}$$

Right end: define one cubic piece on $x_{n-2} \leq x \leq x_n$, i.e., x_{n-1} is not a knot. Thus,

$$(h_{n-1} + h_{n-2}) d_{n-1} + h_{n-2} d_n = \frac{h_{n-1}^2 \delta_{n-2} + (2h_{n-2} + 3h_{n-1}) h_{n-2}\delta_{n-1}}{h_{n-2} + h_{n-1}}. \tag{6}$$

[2] we need to ensure that $p_1'''(x_2) = p_2'''(x_2)$

Cubic Not-a-Knot Splines (cont.)

The final linear system $A\boldsymbol{d} = \boldsymbol{r}$ to be solved for the unknown slopes of the cubic not-a-knot spline is obtained by adding equations (5) and (6) to the tridiagonal $(n-2) \times n$ linear system we derived earlier.

Cubic Not-a-Knot Splines (cont.)

The final linear system $A\boldsymbol{d} = \boldsymbol{r}$ to be solved for the unknown slopes of the cubic not-a-knot spline is obtained by adding equations (5) and (6) to the tridiagonal $(n-2) \times n$ linear system we derived earlier. Its components are

$$
A = \begin{bmatrix}
h_2 & h_1 + h_2 & & & & \\
h_2 & 2(h_1 + h_2) & h_1 & & & \\
 & h_3 & 2(h_2 + h_3) & h_2 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & h_{n-1} & 2(h_{n-2} + h_{n-1}) & h_{n-2} \\
 & & & & h_{n-1} + h_{n-2} & h_{n-2}
\end{bmatrix},
$$

$$
\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \qquad
\boldsymbol{r} = \begin{bmatrix}
\frac{(3h_1 + 2h_2)h_2\delta_1 + h_1^2\delta_2}{h_1 + h_2} \\
3(h_1\delta_2 + h_2\delta_1) \\
3(h_2\delta_3 + h_3\delta_2) \\
\vdots \\
3(h_{n-2}\delta_{n-1} + h_{n-1}\delta_{n-2}) \\
\frac{h_{n-1}^2\delta_{n-2} + (2h_{n-2} + 3h_{n-1})h_{n-2}\delta_{n-1}}{h_{n-2} + h_{n-1}}
\end{bmatrix}
$$

Splines in MATLAB

[NCM] includes the function `splinetx.m` that works similarly to `pchiptx.m` discussed earlier. It is a simplified version of the built-in `spline` function and evaluates a cubic not-a-knot interpolating spline.

Splines in MATLAB

[NCM] includes the function `splinetx.m` that works similarly to `pchiptx.m` discussed earlier. It is a simplified version of the built-in `spline` function and evaluates a cubic not-a-knot interpolating spline.

MATLAB's built-in `spline` function allows us to perform cubic clamped spline interpolation by setting the values of the derivatives of the spline at the left and right ends in the vector `y` (which now must contain two more entries than `x`).

Splines in MATLAB

[NCM] includes the function `splinetx.m` that works similarly to `pchiptx.m` discussed earlier. It is a simplified version of the built-in `spline` function and evaluates a cubic not-a-knot interpolating spline.

MATLAB's built-in `spline` function allows us to perform cubic clamped spline interpolation by setting the values of the derivatives of the spline at the left and right ends in the vector `y` (which now must contain two more entries than `x`).

See the script `SplineDemo.m` for an example of cubic not-a-knot, natural, and clamped spline interpolation.

Splines in MATLAB

[NCM] includes the function `splinetx.m` that works similarly to `pchiptx.m` discussed earlier. It is a simplified version of the built-in `spline` function and evaluates a cubic not-a-knot interpolating spline.

MATLAB's built-in `spline` function allows us to perform cubic clamped spline interpolation by setting the values of the derivatives of the spline at the left and right ends in the vector `y` (which now must contain two more entries than `x`).

See the script `SplineDemo.m` for an example of cubic not-a-knot, natural, and clamped spline interpolation.

It can be shown that for generic interpolation problems (when we don't know much about the behavior near the endpoints) the cubic not-a-knot spline is the most accurate of the three cubic spline methods discussed here.

Splines in MATLAB

[NCM] includes the function `splinetx.m` that works similarly to `pchiptx.m` discussed earlier. It is a simplified version of the built-in `spline` function and evaluates a cubic not-a-knot interpolating spline.

MATLAB's built-in `spline` function allows us to perform cubic clamped spline interpolation by setting the values of the derivatives of the spline at the left and right ends in the vector `y` (which now must contain two more entries than `x`).

See the script `SplineDemo.m` for an example of cubic not-a-knot, natural, and clamped spline interpolation.

It can be shown that for generic interpolation problems (when we don't know much about the behavior near the endpoints) the cubic not-a-knot spline is the most accurate of the three cubic spline methods discussed here.

There is also an entire toolbox for splines written by Carl de Boor, one of the leaders in the field (see also [de Boor]).

Related Methods

There are many other related interpolation methods such as

- B-splines,
- Bézier splines,
- splines with non-uniform knots,
- rational splines,
- rational splines with non-uniform knots (NURBS)

Related Methods

There are many other related interpolation methods such as

- B-splines,
- Bézier splines,
- splines with non-uniform knots,
- rational splines,
- rational splines with non-uniform knots (NURBS)

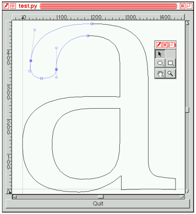
If the curves are more complicated, then we can use all methods in parametric form. This has applications, e.g., in the design of typesetting fonts.

Related Methods

There are many other related interpolation methods such as

- B-splines,
- Bézier splines,
- splines with non-uniform knots,
- rational splines,
- rational splines with non-uniform knots (NURBS)

If the curves are more complicated, then we can use all methods in parametric form. This has applications, e.g., in the design of typesetting fonts.



Outline

What happens in higher dimensions?

Theorem (Mairhuber-Curtis)

If we fix $n \geq 2$ basis functions $B_1, \ldots, B_n$ in two or more space dimensions, then we may always be able to find n data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ such that the Vandermonde-like interpolation matrix

$$\begin{bmatrix} B_1(\boldsymbol{x}_1) & B_2(\boldsymbol{x}_1) & \ldots & B_n(\boldsymbol{x}_1) \\ B_1(\boldsymbol{x}_2) & B_2(\boldsymbol{x}_2) & \ldots & B_n(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_1(\boldsymbol{x}_n) & B_2(\boldsymbol{x}_n) & \ldots & B_n(\boldsymbol{x}_n) \end{bmatrix}$$

with entries $B_k(\boldsymbol{x}_j)$ is singular.

What does the Mairhuber-Curtis theorem actually say?

- The M-C theorem implies that we can't choose our basis independent of the data locations, i.e., the basis has to be chosen after the data sites. It has to be a data-dependent basis.

What does the Mairhuber-Curtis theorem actually say?

- The M-C theorem implies that we can't choose our basis independent of the data locations, i.e., the basis has to be chosen after the data sites. It has to be a data-dependent basis.
- As a consequence we can no longer use (multivariate) polynomials for arbitrary data in higher dimensions.

What does the Mairhuber-Curtis theorem actually say?

- The M-C theorem implies that we can't choose our basis independent of the data locations, i.e., the basis has to be chosen after the data sites. It has to be a data-dependent basis.
- As a consequence we can no longer use (multivariate) polynomials for arbitrary data in higher dimensions.
- Radial basis functions present one way to circumvent the problem presented by the Mairhuber-Curtis theorem.

Proof.

Assume that we have a basis $\{B_1, \ldots, B_n\}$ with $n \geq 2$ such for arbitrary data that the interpolation is non-singular, i.e.

$$\det\left(B_k(\boldsymbol{x}_j)\right) \neq 0 \tag{7}$$

for any distinct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

Proof.

Assume that we have a basis $\{B_1, \ldots, B_n\}$ with $n \geq 2$ such for arbitrary data that the interpolation is non-singular, i.e.

$$\det\left(B_k(\boldsymbol{x}_j)\right) \neq 0 \tag{7}$$

for any distinct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

We need to show that this leads to a contradiction.

Proof.

Assume that we have a basis $\{B_1, \ldots, B_n\}$ with $n \geq 2$ such for arbitrary data that the interpolation is non-singular, i.e.

$$\det\left(B_k(\boldsymbol{x}_j)\right) \neq 0 \tag{7}$$

for any distinct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

We need to show that this leads to a contradiction.

▸ Mairhuber-Curtis Movie

Proof.

Assume that we have a basis $\{B_1, \ldots, B_n\}$ with $n \geq 2$ such for arbitrary data that the interpolation is non-singular, i.e.

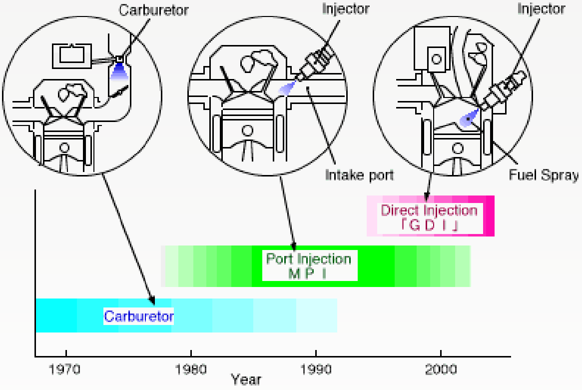
$$\det \left(B_k(\boldsymbol{x}_j)\right) \neq 0 \qquad (7)$$

for any distinct $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.
We need to show that this leads to a contradiction.

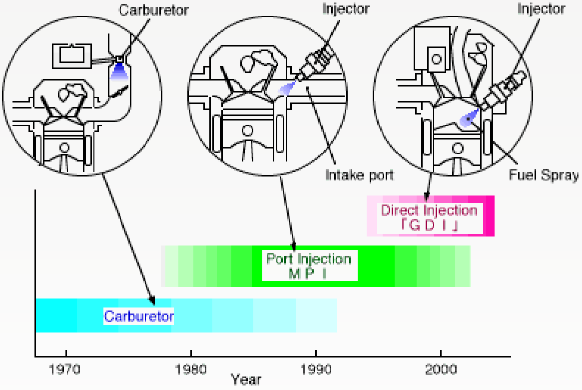
Mairhuber-Curtis Movie

Since the determinant is a continuous function of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ we must have had $\det = 0$ at some point along the continuous path in our interpolation domain. This contradicts (7). □

Gasoline Engine Design



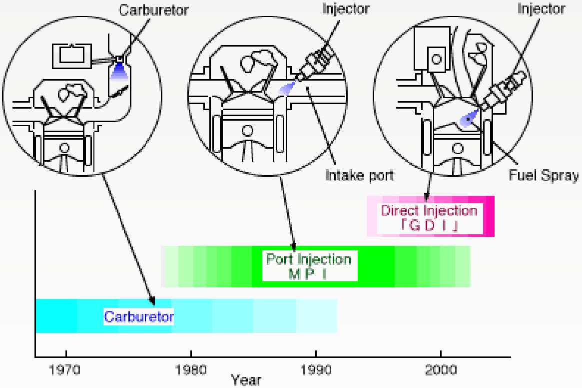
Gasoline Engine Design



Variables:

spark timing
speed
load
air-fuel ratio

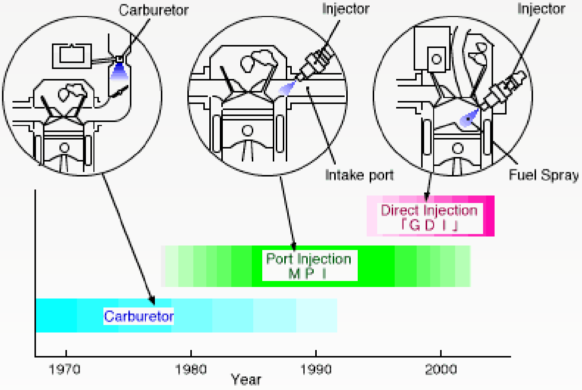
Gasoline Engine Design



Variables:

spark timing	exhaust gas re-circulation rate
speed	intake valve timing
load	exhaust valve timing
air-fuel ratio	

Gasoline Engine Design



Variables:

spark timing
speed
load
air-fuel ratio

exhaust gas re-circulation rate
intake valve timing
exhaust valve timing

fuel injection timing

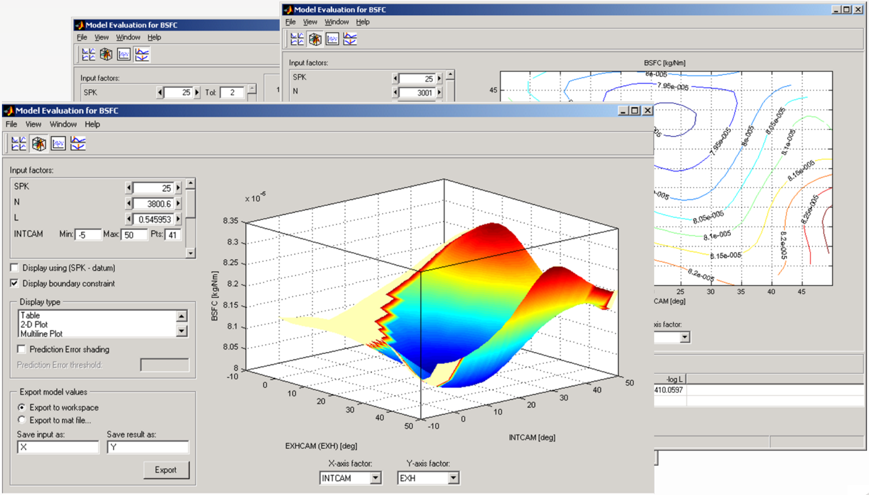
Engine Data Fitting

Find a function (model) that fits the “input” variables and “output” (fuel consumption), and use the model to decide which variables lead to an optimal fuel consumption.

input ? output

$$\left\{ \begin{array}{lcl} x_1 & = & \text{spark timing} \\ x_2 & = & \text{speed} \\ x_3 & = & \text{load} \\ x_4 & = & \text{air-fuel ratio} \end{array} \right\} \overset{f(x_1,x_2,x_3,x_4)}{\Longrightarrow} \text{fuel consumption}$$

Fuel Consumption Model



Tanya Morton, The MathWorks

Rapid Prototyping

An important application of interpolation (or very good approximation) methods is the creation of computer models by scanning physical objects such as historic artifacts or even household appliance parts, and then using interpolation to produce a surface or solid model that can be fed into the manufacturing process.

Rapid Prototyping

An important application of interpolation (or very good approximation) methods is the creation of computer models by scanning physical objects such as historic artifacts or even household appliance parts, and then using interpolation to produce a surface or solid model that can be fed into the manufacturing process.

Special 3D printers can then be used to quickly and easily generate “clones” of the original object.

Rapid Prototyping

An important application of interpolation (or very good approximation) methods is the creation of computer models by scanning physical objects such as historic artifacts or even household appliance parts, and then using interpolation to produce a surface or solid model that can be fed into the manufacturing process.

Special 3D printers can then be used to quickly and easily generate "clones" of the original object.

See, e.g.,

- [Prof. Qian's website] in IIT's MMAE department,
- [The Digital Michelangelo Project] at Stanford University,
- this article [about 3D printers].

Special Movie Effects `http://www.fastscan3d.com`

trollscanning.mpeg

Source is here [FastSCAN]. Also look at this [Lord of the Rings].

References I

C. de Boor.
A Practical Guide to Splines.
Springer, revised edition, 2001.

G. E. Fasshauer.
Meshfree Approximation Methods with MATLAB.
World Scientific Publishers, 2007.

C. Moler.
Numerical Computing with MATLAB.
SIAM, Philadelphia, 2004.
Also http://www.mathworks.com/moler/.

L. L. Schumaker.
Spline Functions: Basic Theory (3rd ed.).
Cambridge University Press, 2007.

'Printers' that can make 3-D solid objects soon to enter mainstream.
http://www.sciencedaily.com/releases/2007/09/070925081418.htm.

References II

X. Qian.
Computational Design and Manufacturing Lab.
http://mmae.iit.edu/cadcam/research.htm.

The Digital Michelangelo Project.
http://graphics.stanford.edu/projects/mich/.

FastSCAN.
http://www.fastscan3d.com/media/trollscanning.mpg.

The Lord of the Rings.
Enter “The Prologue” and select “Digital Scanning” from “Orcs”; and “Digital Lands” from “The Battlefield” http://www.lordoftherings.net/effects/.