HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

Chapter 3: Curve Fitting

Vu Van Thieu, Dinh Viet Sang, Ban Ha Bang

SoICT
Hanoi University of Science and Technology

ONE LOVE. ONE FUTURE.

Introduction

1. Problem
2. Interpolation
   - Lagrange interpolation
   - Spline interpolation
3. Regression
   - Linear regression
   - High-order curve fitting
   - General curve fitting

Problem

The problem is:

Suppose we need to approximate the function $f(x)$ without knowing the formula, but by experiment we can determine the value of $f(x)$ at $n+1$ at discrete points

$$f_i = f(x_i) \quad i = 0, 1, \cdots, N$$

Given the dataset $\{(x_i, f_i), i = 0, 1, ..., N\}$, we need to find a way to approximate the function $f(x)$ by a function (which can be calculated using an analytic formula) $p(x)$.

Problem (continued)

continue...

To solve the problem

We will learn two methods used to construct curve fitting.

- Interpolation: function $p(x)$ must go through all points in the dataset.
- Regression: given the form $p(x)$ with parameters, we must determine the parameters to minimize a certain error criterion (usually the least squares criterion is used).

Introduction

The possible interpolation with functions $p(x)$ is

- Polynomial function
- Rational function
- Fourier series function

Interpolation

Definition: We say the function $p(x)$ is an interpolation function of the dataset $\{(x_i, f_i), i = 0, 1, ..., N\}$ if the following conditions are satisfied:

$$p_i = f_i, i = 0, 1, ..., N$$

This system of $N + 1$ of equations is called **interpolation condition**. The reason that the polynomial function was chosen is because it is easy to calculate values, derivatives, and differentials (the analytic properties of the function are obvious).
The polynomial that interpolates the data set is called the interpolating polynomial.
Definition: We call a polynomial of degree of no more than K has form of:

$$p_K(x) = a_0 + a_1 x + \cdots + a_K x^K$$

where $a_0, a_1, \cdots, a_K$ are constants.

Linear interpolation

We have two data points (x_0, f_0) and (x_1, f_1) which need interpolation by polynomial $p_1(x) \equiv a_0 + a_1 x$

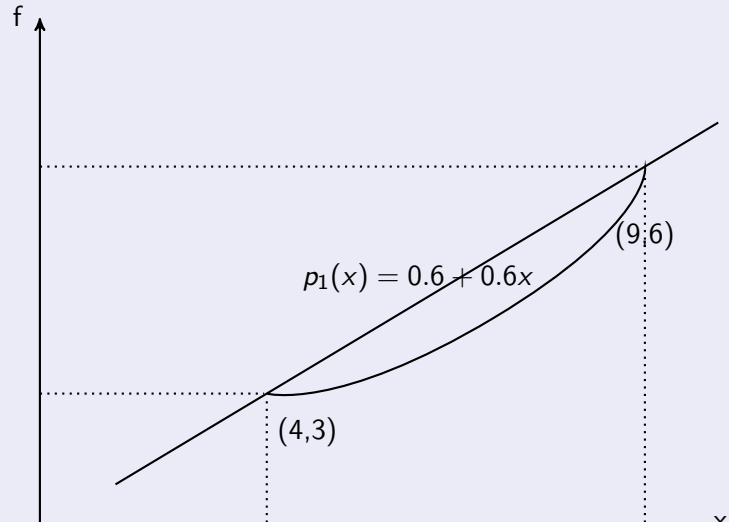
$p_1(x)$ satisfies the system of equations

$$p_1(x_0) \equiv a_0 + a_1 x_0 = f_0$$
$$p_1(x_1) \equiv a_0 + a_1 x_1 = f_1$$

Solve this system, we have

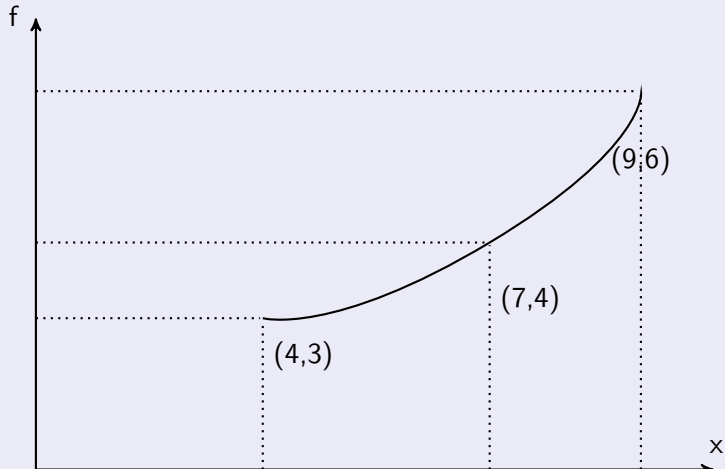
- When $x_0 \neq x_1$ occurs two cases
  - if $f_0 \neq f_1$ then $p_1(x) = \frac{x_1 f_0 - x_0 f_1}{x_1 - x_0} + \frac{f_1 - f_0}{x_1 - x_0} x$
  - if $f_0 = f_1$ then the solution is constant $\Rightarrow$ polynomial of 0-th degree.

Linear interpolation (continued)



Lagrange Interpolation Formula

It is necessary to construct a curve that passes through 3, 4 or more points (see figure).



Lagrange interpolation formula (continued)

Consider polynomial

$$p_M(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_M x^M$$

The polynomial $p_M(x)$ interpolates the dataset $\{(x_i, f_i), i = 0, 1, ..., N\}$ if we have the interpolation conditions:

$$\begin{aligned} p_M(x_0) &\equiv a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_M x_0^M = f_0 \\ p_M(x_1) &\equiv a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_M x_1^M = f_1 \\ &\cdots \\ p_M(x_N) &\equiv a_N + a_1 x_N + a_2 x_N^2 + \cdots + a_M x_N^M = f_N \end{aligned}$$

This is a system of linear equations, so we can represent it using a matrix formula.

Lagrange interpolation formula (continued)

Thus the matrix of the interpolation conditions is as follows:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^M \\ 1 & x_1 & \cdots & x_1^M \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^M \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_M \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_N \end{pmatrix}$$

Assuming that $x_0, x_1, \cdots, x_N$ is distinct, then according to the theory of solving the system of linear equations in the previous chapter, we have

- N=M: system of equations with unique solution
- M<N: can choose the dataset so that it has no solution
- M>N: if the system has a solution, it has infinitely many solutions

Lagrange interpolation formula (continued)

When $N = M$ the coefficient matrix of the above system of equations is a Vandermonde matrix (Not Voldemort)

$$V_N = \begin{pmatrix} 1 & x_0 & \cdots & x_0^M \\ 1 & x_1 & \cdots & x_1^M \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^M \end{pmatrix}$$

The determinant of V_n is:

$$V_N = \prod_{i>j} (x_i - x_j)$$

infer that the system has a solution when $x_0, x_1, \cdots, x_N$ is pairwise distinct.

Lagrange interpolation formula (continued)

Theorem (Uniqueness of Interpolated Polynomials): If the data nodes $x_0, x_1, \cdots, x_N$ are pairwise different, then there exists a unique interpolation polynomial $p_N(x)$ of no more than N degree that interpolates the dataset $\{(x_i, f_i), i = 0, 1, ..., N\}$.

The degree of the interpolated polynomial can be less than N

For example, when the three data points $(x_0, f_0), (x_1, f_1), (x_3, f_3)$ line up, the interpolation equation becomes of no more than two degree because it is a straight line.

Example 1:

Determine the coefficients of the interpolation polynomial $p_2(x) = a_0 + a_1x + a_2x^2$ interpolating the dataset {(-1,0),(0,1),(1,3)}, we have interpolation condition

$$\begin{aligned} a_0 + (-1)a_1 + 1a_2 &= 0 \\ a_0 + 0a_1 + 0a_2 &= 1 \\ a_0 + \quad 1a_1 + 1a_2 &= 3 \end{aligned}$$

Solving the system of equations we have $a_0 = 1, a_1 = 1.5$ and $a_2 = 0.5$. So the interpolated polynomial is

$$p_2(x) = 1 + 1.5x + 0.5x^2$$

Lagrange interpolation formula (continued)

Directly solving a system of interpolated conditional equations using Gaussian elimination, for example, requires a lot of computations $\mathcal{O}(N^3)$. In addition, because the power of the coefficient x can be very high, the error is even larger due to rounding effects.
$\Rightarrow$ we can use **Lagrange interpolation formula** to calculate the constant of the interpolated polynomial without solving the above system of linear equations.

Lagrange interpolation formula (continued)

Consider the dataset $\{(x_i, f_i), i = 0, 1, ..., N\}$. The Lagrangian form of the interpolation polynomial for the dataset is as follows:

$$p_N(x) = f_0 V_0(x) + f_1 V_1(x) + \cdots + f_N V_N(x)$$

where $V_i(x) : i = 0, 1, \cdots, N$ are polynomials of degree N satisfying the condition:

$$V_i(x_j) = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

The family of polynomials $V_i(x) : i = 0, 1, \cdots, N$ is called the family of basis polynomials. It is easy to check that:

$$p_N(x_i) = f_i \quad i = 0, 1, \cdots, N$$

Lagrange interpolation formula (continued)

One of the base polynomial class that can be built according to the following formula:

$$V_i(x) = \frac{(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

It is clearly satisfied

$$V_i(x_j) = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

So we can summarize the above formula as follows:

$$V_i(x) = \frac{\prod_{i \neq j}(x - x_j)}{\prod_{i \neq j}(x_i - x_j)}$$

Lagrange interpolation formula (continued)

So the Lagrange interpolation formula

$$
\begin{aligned}
p_N(x) = & \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_0-x_1)\cdots(x_0-x_N)} f_0 \\
& \cdots \\
& + \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} f_i \\
& \cdots \\
& + \frac{(x-x_0)(x-x_1)\cdots(x-x_{N-1})}{(x_N-x_0)\cdots(x_N-x_{N-1})} f_N
\end{aligned}
\tag{1}
$$

Formula (1) is long but easier to remember and calculate.

Install $V_i(x)$ using Matlab

$v = polyinterp(x, y, u)$ calculates $v(j) = P(u(j))$ where P is a polynomial of degree $d = length(x) - 1$ such that $P(x(i)) = y(i)$. Use Lagrange interpolation formula. Two vectors x, y represent the coordinates of the data points.

```
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 = (ux(j))./(x(k)-x(j)).*w;
    end
    v = v + w*y(k);
end
```

Install $V_i(x)$ using Matlab (continued)

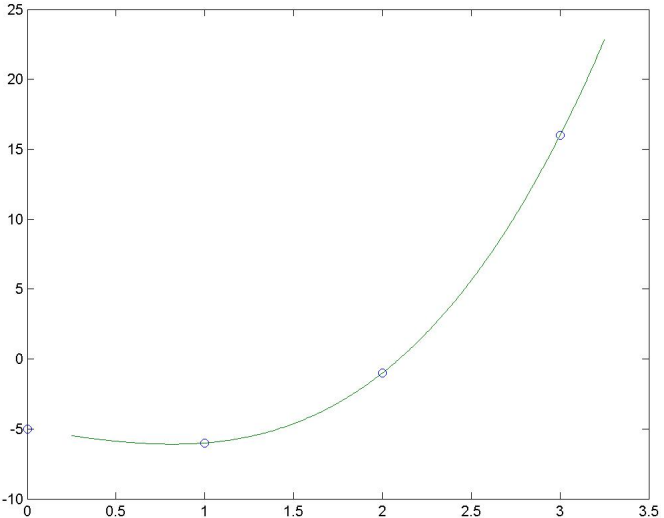
```
function v = polyinterp(x,y,u)
n = length(x);
v = zeros(size(u));
for k = 1:n
    w = ones(size(u));
    for j = [1:k-1 k+1:n]
        w = (ux(j))./(x(k)-x(j)).*w;
    end
    v = v + w*y(k);
end
```

- The two vectors **x,y** represent the coordinates of the data points.
- **u** is a vector containing the coordinates of the points to be calculated according to the Lagrange interpolation formula.
- **v** is the vector containing the interpolation function values at the given points in **u**

Example 2:

```
x = [0 1 2 3];
y = [-5 -6 -1 16];
u= [0.25:0.01:3.25];
v = polyinterp(x,y,u);
plot(x,y,'o',u,v,'-')
```

Interpolation



Interpolation error

Definition: The error of the interpolated polynomial $g(x)$ is determined by the formula:

$$e(x) = f(x) - g(x)$$

The magnitude of e(x) depends on

- The magnitude of the data points.
- Size of the interpolation domain $D = x_n - x_0$.
- Degree of interpolation polynomial

For Lagrange interpolated polynomials, the error is determined by the formula:

$$e(x) = f(x) - g(x) = L(x) f^{(N+1)}(\xi)$$

where $f^{(N+1)}$ is the derivative of $N+1$ degree of the function $f(x)$ and $L(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(N+1)!}$

Interpolation error (continued)

formula:

$$e(x) = f(x) - g(x) = L(x)f^{(N+1)}(\xi)$$

ξ depends on x but satisfies the condition $x_0 < \xi < x_N$. According to the above formula, if f(x) is a polynomial of degree no more than N, then the derivative $f^{(N+1)}$ will be zero, that is, the error is zero. Since $L(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(N+1)!}$ we have the following conclusions

- With an equidistant grid, the amplitude of oscillations of $L(x)$ is smallest at the center and increases towards the boundary of the interpolation domain.
- The interpolation size increases, the magnitude of the oscillation increases rapidly.

Lagrange interpolation

Spline interpolation

Problem

In this case we consider interpolation by a set of low-order polynomials instead of a single higher-order polynomial

- Interpolation by linear spline
- Interpolation by cubic spline

Recall the Lagrange interpolation error evaluation formula

$$f(x) - p_N(x) = [w_{N+1}(x)/(N+1)!]f^{(N+1)}(\xi_x)$$

where $w_{N+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_N)$ and ξ_x is a value between (x_0, x_N).

Problem (continued)

Assuming every data point $x_i \in [a, b]$ then we have

$$\max_{x\in[a,b]} |f(x) - p_N(x)| \leq |b-a|^{N+1} \times \frac{\max_{x\in[a,b]} |f^{(N+1)}(x)|}{(N+1)!}$$

This evaluation indicates that if we want to reduce the error while keeping the number of data points N, we need to find a way to reduce the size of $|b - a|$.
Our approach is piecewise polynominal approximation: Thus the segment $[a, b]$ will be divided into many small non-intersecting segments and **different polynomials will approximated on each sub-segment**.

Snippet Linear Spline

For the dataset

$$\{(x_i, f_i) : i = 0, 1, ..., N\}$$

Inside

$$a = x_0 < x_1 < \cdots < x_N = b, h \equiv \max_i |x_i - x_{i-1}|,$$

linear spline $S_{1,N}(x)$ is a continuous function that interpolates given data and is built from linear functions defined by two data point interpolation polynomials:

Snippet Linear Spline (continued)

$$S_{1,N}(x) = \begin{cases} \frac{f_1 - f_0}{x_1 - x_0}(x - x_1) + f_1, & \text{if } x \in [x_0, x_1] \\ \vdots \\ \frac{f_i - f_{i-1}}{x_i - x_{i-1}}(x - x_i) + f_i, & \text{if } x \in [x_{i-1}, x_i] \\ \vdots \\ \frac{f_N - f_{N-1}}{x_N - x_{N-1}}(x - x_N) + f_N, & \text{if } x \in [x_{N-1}, x_N] \end{cases}$$

Easy to see

$$L_i(x) = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}(x - x_i) + f_i$$

is the equation of the line through two points (x_i, f_i) and (x_{i-1}, f_{i-1})

Snippet Linear Spline (continued)

Using error formula for interpolated polynomial with $x \in [a, b]$ for each segment $[x_{i-1}, x_i]$ we get

$$\begin{aligned}\max_{z\in[x_{i-1},x_i]} |f(z) - S_{1,N}(z)| &\leq \frac{|x_i - x_{i-1}|^2}{8} \times \max_{x\in[x_{i-1},x_i]} |f^{(2)}(x)| \\ &\leq \frac{h^2}{8} \times \max_{x\in[x_{i-1},x_i]} |f^{(2)}(x)|\end{aligned}$$

where $h \equiv \max_i |x_i - x_{i-1}|$.

Example 3:

Given the dataset {(-1,0),(0,1),(1,3)}, we build a spline calculated as follows

$$S_{1,2}(x) = \begin{cases} \frac{f_1-f_0}{x_1-x_0}(x-x_1)+f_1 = \frac{f_1-f_0}{x_1-x_0}(x-x_0)+f_0, & \text{if } x \in [x_0, x_1] \\ \frac{f_2-f_1}{x_2-x_1}(x-x_2)+f_2 = \frac{f_2-f_1}{x_2-x_1}(x-x_1)+f_1, & \text{if } x \in [x_1, x_2] \end{cases}$$

infer with the corresponding data set

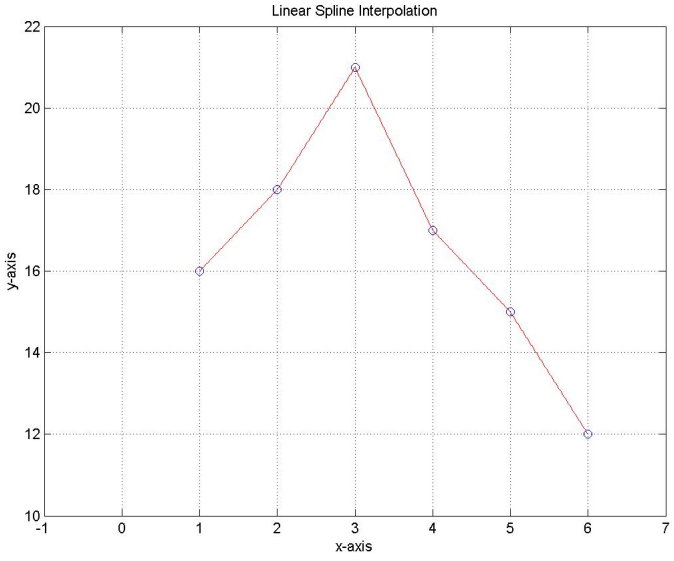
$$S_{1,2}(x) = \begin{cases} \frac{1-0}{0-(-1)}x+1 = x+1, & \text{if } x \in [-1, 0] \\ \frac{3-1}{1-0}(x-1)+3 = 2x+1, & \text{if } x \in [0, 1] \end{cases}$$

Spline interpolation

Example 4:

```
» clear;
» x=1:6;
» y=[16 18 21 17 15 12];
» plot(x,y,'o')
» hold on; grid on;
» axis([-1 7 10 22])
» xx = [1:0.01:6];
» yy = piecelin(x,y,xx);
» plot(xx,yy,'r')
» hold off
» xlabel('x-axis');ylabel('y-axis');
» title('Linear Spline Interpolation');
```

Spline interpolation



Third-order splines

Instead of line segments, third-order splines use 3rd degree polynomials to approximate the segment polynomial. I also have the dataset

$$\{(x_i, f_i) : i = 0, 1, ..., N\}$$

Inside

$$a = x_0 < x_1 < \cdots < x_N = b, h \equiv \max_i |x_i - x_{i-1}|,$$

Third-order spline (continued)

Definition: A third-order spline $S_{3,N}(x)$ interpolates a given dataset

- $S_{3,N}(x)$ is a 3rd order function between two nodes $[x_{i-1}, x_i]$

$$S_{3,N}(x) = \begin{cases} p_1(x) = a_{1,0} + a_{1,1}x + a_{1,2}x^2 + a_{1,3}x^3, & \text{if } x \in [x_0, x_1] \\ \vdots \\ p_i(x) = a_{i,0} + a_{i,1}x + a_{i,2}x^2 + a_{i,3}x^3, & \text{if } x \in [x_{i-1}, x_i] \\ \vdots \\ p_N(x) = a_{N,0} + a_{N,1}x + a_{N,2}x^2 + a_{N,3}x^3, & \text{if} x \in [x_{N-1}, x_N] \end{cases}$$

- $S_{3,N}(x)$ is a continuous function with continuous first and second derivatives on the segment $[x_0, x_N]$ (including boundaries)

Third-order spline (continued)

For $x_{i-1} < x < x_i$, $S_{3,N}(x)$ has value $p_i(x) = a_{i,0} + a_{i,1}x + a_{i,2}x^2 + a_{i,3}x^3$	For $x_i < x < x_{i+1}$, $S_{3,N}(x)$ has value $p_{i+1}(x) = a_{i+1,0} + \cdots + a_{i+1,3}x^3$
Value of $S_{3,N}(x)$ when $x \to x_i^-$	Value of $S_{3,N}(x)$ when $x \to x_i^+$
$p_i(x_i)$	$p_{i+1}(x_i)$
$p_i'(x_i)$	$p_{i+1}'(x_i)$
$p_i''(x_i)$	$p_{i+1}''(x_i)$

Comment:

- The smoothness condition at the point x_i then

$$p_i'(x_i) = p_{i+1}'(x_i); \quad p_i''(x_i) = p_{i+1}''(x_i); i = 1, 2, \cdots, N-1$$

- Guaranteed data interpolation condition

$$p_i(x_i) = p_{i+1}(x_i) = f_i; i = 0, 1, 2, \cdots, N-1$$

Third-order spline (continued)

- Natural Boundary Condition

$$p_1''(x_0) = 0; p_N''(x_N) = 0;$$

- Second Derivative Condition

$$p_1''(x_0) = f''(x_0); p_N''(x_N) = f''(x_N);$$

- Not-a-knot Condition

$$p_1'''(x_1) = p_2'''(x_1); p_{N-1}'''(x_{N-1}) = p_N'''(x_{N-1});$$

Spline interpolation

Spline function for 3rd order interpolation in MatLab:

p=spline(x,y): returns the segmented polynomial of the third order spline for the dataset (x,y).

v=ppval(p,x): Calculates the value of the interpolated polynomial at input x (p is determined by the spline function).

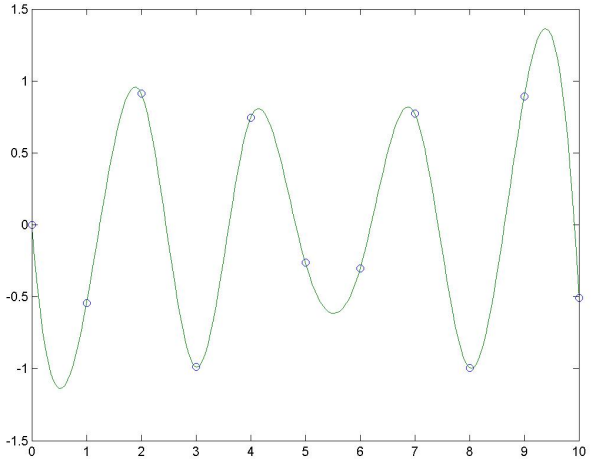
yy=spline(x,y,xx): Defines the 3rd order interpolated spline for the dataset (x,y) and returns yy as the vector of the values of the spline interpolated at points in the vector xx .

Spline interpolation

Example 4:

```
» x = 0:10; y = sin(10*x);
» p = spline(x,y);
» xx = linspace(0,10,200);
» yy = ppval(p,xx);
» plot(x,y,'o',xx,yy)
```

Level 3 spline interpolation

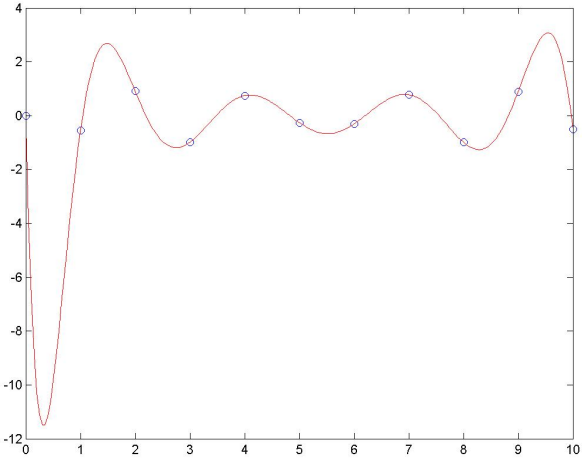


Spline interpolation

Example 5:

```
» x = 0:10; y = sin(10*x);
» xx = linspace(0,10,200);
» coef = polyfit(x,y,10);
Warning: Polynomial is badly conditioned. Remove repeated data points or try
centering and scaling as described in HELP POLYFIT. (Type "warning off
MATLAB:polyfit:RepeatedPointsOrRescale" to suppress this warning.)
» yy = polyval(coef,xx);
» plot(x,y,'o',xx,yy,'r')
```

Level 3 spline interpolation



So when we interpolate with spline of order 3, it is better than polynomial interpolation with

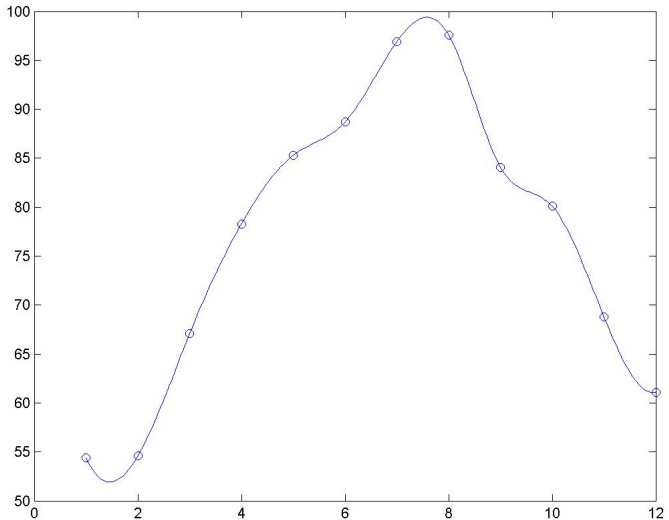
Example 6:

The power consumption of a device in 12 hours is as follows

1	2	3	4	5	6
54.4	54.6	67.1	78.3	85.3	88.7
7	8	9	10	11	12
96.9	97.6	84.1	80.1	68.8	61.1

Using the third-order spline to interpolate the above data point set.

Third-order spline interpolation



Level 3 spline interpolation

Problem

Given the dataset $\{(x_i, f_i), i = 1, \cdots, N\}$, we need to find a way to approximate the function $f(x)$ by a function $p(x)$. We need to take care of measuring the total error over the entire segment.

- Maximum error $E_\infty(f) = \max_{1 \le k \le n} |f(x_k) - y_k|$
- Average error $E_1(f) = \frac{1}{n} \sum_{k=1}^{n} |f(x_k) - y_k|$
- Square root error (least square)

$$E_2(f) = \sqrt{\frac{1}{n} \sum_{k=1}^{n} (f(x_k) - y_k)^2}$$

Linear regression

Linear regression is finding a line that fits the data points in the least squares sense.

- Given a set of N pairs of data points $\{(x_i, f_i) : i = 1, \cdots, N\}$
- Find the coefficient m and the free constant b of the line

$$y(x) = mx + b$$

such that this line fits the data according to the least squares criterion.

$$L(m, b) = \sum_{i=1}^{N} (f_i - (mx_i + b))^2$$

Linear regression (continued)

That is, we need to find m and b to minimize the function

$$L(m, b) = \sum_{i=1}^{N} (f_i - (mx_i + b))^2$$

To find this minimum, we need to solve the system of equations to find the stopping point

$$\frac{\partial L}{\partial m} = \sum_{i=1}^{N} 2(f_i - (mx_i + b))(-x_i) = 0$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} 2(f_i - (mx_i + b))(-1) = 0$$

Linear regression (continued)

$$\left[\sum_{i=1}^{N} x_i^2\right] m + \left[\sum_{i=1}^{N} x_i\right] b = \sum_{i=1}^{N} x_i f_i$$
$$\left[\sum_{i=1}^{N} x_i\right] m + \left[\sum_{i=1}^{N} 1\right] b = \sum_{i=1}^{N} f_i$$

or in matrix form,

$$\begin{pmatrix} \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & N \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} x_i f_i \\ \sum_{i=1}^{N} f_i \end{pmatrix}$$

Solving the system of equations we get the coefficients m and b.

Example 6:

Construct the fitting curve for the following dataset

i	1	2	3	4
x_i	1	3	4	5
f_i	2	4	3	1

So the system of equations to be solved is

$$13m + 4b = 10$$
$$51m + 13b = 31$$

then find $b = 107/35$ and $m = -6/35$, the line to find is $y = (-6/35)x + 107/35$

High-order curve fitting

Build the curve fitting

$$p_M(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_M x^M$$

matches the dataset $\{(x_i, f_i) i = 1, \cdots, N\}$ according to least squares criteria

$$\sigma_M \equiv (p_M(x_1) - f_1)^2 + (p_M(x_2) - f_2)^2 + \cdots + (p_M(x_N) - f_N)^2$$

Solve the system of equations to find the stopping point

$$\frac{\partial}{\partial a_0}\sigma_M = 0; \quad \frac{\partial}{\partial a_1}\sigma_M = 0; \cdots \frac{\partial}{\partial a_M}\sigma_M = 0;$$

High-order curve fitting (continued)

Write in matrix form the system of standard equations

$$\begin{pmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i & \cdots & \sum_{i=1}^{N} x_i^M \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \cdots & \sum_{i=1}^{N} x_i^{M+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{N} x_i^M & \sum_{i=1}^{N} x_i^{M+1} & \cdots & \sum_{i=1}^{N} x_i^{2M} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ am \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} f_i \\ \sum_{i=1}^{N} f_i x_i \\ \vdots \\ \sum_{i=1}^{N} f_i x_i^C ODE \end{pmatrix}$$

The special feature of the above matrix is that it is definite symmetry, so we can use Gaussian elimination without changing the line to solve the system of standard equations.

Example 7:

Determine the parameters a, b, cc of the curve

$$y = ax^2 + bx + c$$

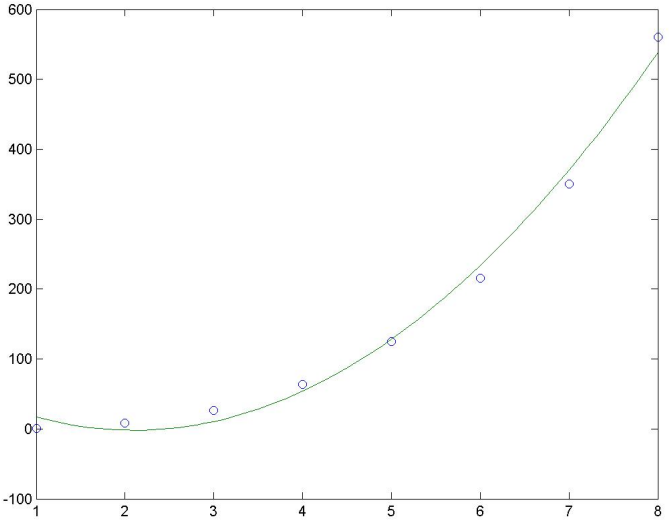
matches the following dataset

x	1	2	3	4	5	6	7	8
y	1	8	27	64	125	216	350	560

Example 7: (continued)

```
x = [1 2 3 4 5 6 7 8];
y = [1 8 27 64 125 216 350 560];
s0 = length(x); s1 = sum(x); s2 = sum(x.^2);
s3 = sum(x.^3); s4 = sum(x.^4);
A = [s4 s3 s2; s3 s2 s1; s2 s1 s0];
b = [sum(x.^2.*y); sum(x.*y);sum(y)];
c0 = A\b;
c = polyfit(x,y,2); % Find a regression polynomial that matches the data
c0 % Returns the calculated coefficient
c % Returns the coefficient found by polyfit
xx = 1:0.1:8;
yy = polyval(c0,xx);
plot(x,y,'o',xx,yy)
```

Regression



General curve fitting

Suppose we want to match N multipoint for $\{(x_i, f_i) : i = 1, \cdots, N\}$ because they include m linearly independent function $\varphi_j(x), j = 1, 2, \cdots, m$, i.e. the function $f(x)$ to look for has the form

$$f(x) = \sum_{j=1}^{m} c_j \varphi_j(x)$$

For example, the following family of functions

- $\varphi_j(x) = x^j, j = 1, 2, \cdots, m$
- $\varphi_j(x) = sin(jx), j = 1, 2, \cdots, m$
- $\varphi_j(x) = cos(jx), j = 1, 2, \cdots, m$

General curve fitting (continued)

We need to determine the numbers $c_1, c_2, \cdots, c_m$ to minimize the error of the square root of the sum of squares of errors:

$$E(c_1, c_2, \cdots, c_m) = \sum_{k=1}^{n} (f(x_k) - f_k)^2 = \sum_{k=1}^{n} \left[\left(\sum_{j=1}^{m} c_j \varphi_j(x) \right) - f_k \right]^2$$

To find the coefficients $c_1, c_2, \cdots, c_m$, we need to solve the system of equations

$$\frac{\partial E}{\partial c_j} = 0, j = 1, 2, \cdots, m$$

Regression

Summary

- Curve fitting problem
- Definition of interpolation
  - Lagrange Interpolation
  - Spline Interpolation
- Definition of regression
  - Linear Regression
  - Higher order polynomial regression