Note that lecture notes Part 1 of this topic was posted last week.

Chapter 10: Polynomial Interpolation

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
A First Course in Numerical Methods (published by SIAM, 2011)
`http://www.ec-securehost.com/SIAM/CS07.html`

This chapter is mainly from Dr. Peter Arbenz Lecture notes at ETH

Topics of today

- What is interpolation
- Monomial interpolation
- Lagrange interpolation
- Newton basis and divided differences

References

- U. Ascher & C. Greif: Numerical methods. SIAM 2011. Chapter 10.1–10.3.

Interpolating data

We are given a collection of data samples $\{(x_i, y_i)\}_{i=0}^{n}$

- The $\{x_i\}_{i=0}^{n}$ are called the abscissae, the $\{y_i\}_{i=0}^{n}$ are called the data values.
- Want to find a function $v(x)$ which can be used to estimate sampled function for $x \neq x_i$.

Interpolation: $v(x_i) = y_i, \quad i = 0, 1, \ldots, n.$

- Why? We often get discrete data from sensors or computation, but we want information as if the function were not discretely sampled.
- Want a *reasonable* looking interpolant. If possible, $v(x)$ should be inexpensive to evaluate for a given x.

Interpolation formulation

Assume a *linear form* of interpolation

$$v(x) = \sum_{j=0}^{n} c_j \phi_j(x) = c_0\phi_0(x) + \cdots + c_n\phi_n(x)$$

where $\{c_i\}_{i=0}^{n}$ are *unknown* coefficients or parameters and $\{\phi_i(x)\}_{i=0}^{n}$ are predetermined basis functions.

The basis functions are assumed to be *linearly independent*.

There are $n+1$ coefficients to be determined by $n+1$ equations.

We assume : number of basis functions = number of data points n+1

Lagrange interpolation

Lagrange polynomials are polynomials of degree n:

Basis functions

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, 1, \ldots, n,$$

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

They satisfy

$$L_i(x_k) = \delta_{ik} = \begin{cases} 1, & \text{if } i = k, \\ 0, & \text{if } i \neq k. \end{cases}$$

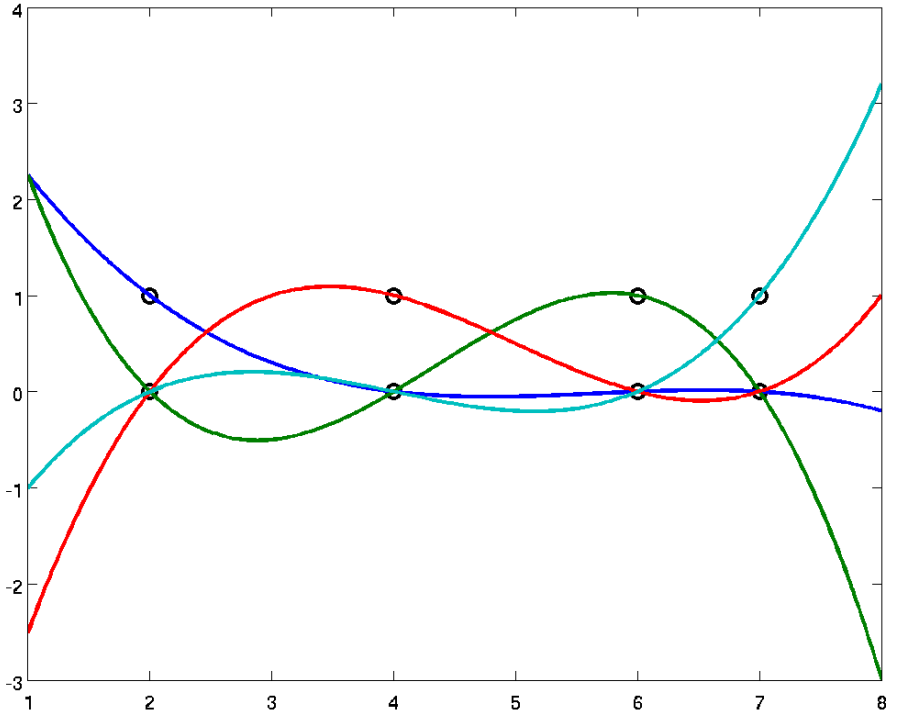
Therefore,

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\, L_j(x), \tag{2}$$

Lagrange interpolation (cont.)

Interpolation is guaranteed:

$$p_n(x_i) = \sum_{j=0}^{n} f(x_j)\, L_j(x_i) = f(x_i)\, L_i(x_i) = f(x_i).$$



Lagrange polynomials of degree 4, with abszissae at $x = 2, 4, 6, 7$, and value 1 at one of these x.

Construction and evaluation

Construction: Construct auxiliary quantities independent of x

$$\rho_j = \prod_{i \neq j} (x_j - x_i), \quad w_j = \frac{1}{\rho_j}, \qquad j = 0, \ldots, n.$$

The quantities w_j are called *barycentric weights*.

Evaluation:

$$\psi(x) = \prod_{i=0}^{n} (x - x_i).$$

Then,

$$p(x) = \psi(x) \sum_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}.$$

Construction and evaluation (cont.)

If we interpolate the constant function $f(x) \equiv 1$, then we get

$$1 = \psi(x) \sum_{j=0}^{n} \frac{w_j \cdot 1}{(x - x_j)}.$$

Therefore, $\psi(x) = \left(\sum_{j=0}^{n} \frac{w_j \cdot 1}{(x - x_j)} \right)^{-1}$ and

$$p(x) = \frac{\sum_{j=0}^{n} \frac{w_j y_j}{(x - x_j)}}{\sum_{j=0}^{n} \frac{w_j}{(x - x_j)}}.$$

Lagrange basis assessment

- Not so simple.
- Matrix A is the identity: the coefficients are immediately obtained.
- Construction cost $\mathcal{O}(n^2)$ flops (OK even if n is large)
- Evaluation cost $\mathcal{O}(n)$ flops (low but not lowest)
- Coefficients c_j indicative of data and useful for function manipulation such as integration and differentiation.
- Stable even if degree is large or abscissae spread apart.

Summary until now:

* Monomial basis:
  relatively difficult procedure to construct p_n but an easy procedure to evaluate the polynomial.

* Lagrange polynomial basis:
  construction stage is easy, but the evaluation stage is relatively involved.

Yet another polynomial representation but has a purpose: add interpolation data one pair at a time

Newton interpolation

For Newton interpolation we represent the interpolation polynomial in a different way:

$$p_n(x) := c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}).$$

All terms but the first vanish at x_0:

$$P_n(x_0) = c_0 \quad \Longrightarrow \quad c_0 = y_0.$$

All terms but the first two vanish at x_1:

$$p_n(x_1) = c_0 + c_1(x_1 - x_0) \quad \Longrightarrow \quad c_1 = \frac{y_1 - y_0}{x_1 - x_0}.$$

i.e. to evaluate c0, we only used 1st data pt; for c1, we used 1st and 2nd data pt;...

Newton interpolation (cont.)

All terms but the first three vanish at x_2:

$$p_n(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1)$$

$$\Longrightarrow \quad c_2 = \frac{y_2 - y_0 - \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$\Longrightarrow \quad c_2 = \frac{\frac{y_2 - y_0}{x_2 - x_0} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_1}$$

The c_i are defined recursively: they are ratios of differences of previously computed ratios. These so-called divided differences can be defined recursively as follows.

Example: quadratic polynomial

Ex 10.3 in the textbook

Given data points:

$$\{(x_i, y_i)\} = \{(1,1),(2,3),(4,3)\}$$

Newton Polynomial Basis:

$$\boxed{\phi_0(x) = 1,\ \ \phi_1(x) = x - x_0,\ \ \phi_2(x) = (x - x_0)(x - x_1).}$$

$$1 = p(1) = c_0\phi_0(1) + c_1\phi_1(1) + c_2\phi_2(1) = c_0 \cdot 1 + c_1 \cdot 0 + c_2 \cdot 0 = c_0.$$

$$3 = p(2) = 1 + c_1(2-1) + c_2 \cdot 0 \quad \Longrightarrow \quad c_1 = 2.$$

$$3 = p(4) = 1 + 2(4-1) + c_2(4-1)(4-2) \quad \Longrightarrow \quad c_2 = -\frac{2}{3}.$$

$$p_2(x) = 1 + 2(x-1) - \frac{2}{3}(x-1)(x-2).$$

Note: the first 2 terms form the linear interpolant p1(x): property of Newton!

Example: cubic polynomial

$$\{(x_i, y_i)\} = \{(2, 14), (6, 24), (4, 25), (7, 15)\}$$

Use Newton basis.

- Four basis functions:

$$\phi_0(x) = 1, \quad \phi_1(x) = (x - 2),$$
$$\phi_2(x) = (x - 2)(x - 6), \quad \phi_3(x) = (x - 2)(x - 6)(x - 4).$$

- Construct linear system

Exercise: Construct the A matrix by inserting into the basis functions

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 1 & 2 & -4 & 0 \\ 1 & 5 & 5 & 15 \end{pmatrix}$$

Lower triangular

and solve $A\mathbf{c} = \mathbf{y}$ to find $\mathbf{c} \approx [14, 2.5, -1.5, -0.2667]^T$

- Same polynomial as before, but different representation!

Newton interpolation

Zeroth divided difference:

$$f[x_i] := f(x_i).$$

First divided difference:

$$f[x_0, x_1] := \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

k-th divided difference:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] := \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

where $k = 2, 3, \ldots, n$ and $i = 0, 1, \ldots, n - k$.

Newton interpolation (cont.)

For example, the first divided difference would evaluate to

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

The second divided difference would amount to

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}.$$

The third divided difference is

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}.$$

Newton interpolation (cont.)

The divided difference are best arranged in a triangular array:

$$
\begin{array}{c|cccc}
x_0 & f[x_0] = c_0 & & & \\
 & & f[x_0, x_1] = c_1 & & \\
x_1 & f[x_1] & & f[x_0, x_1, x_2] = c_2 & \\
 & & f[x_1, x_2] & & f[x_0, x_1, x_2, x_3] = c_3 \\
x_2 & f[x_2] & & f[x_1, x_2, x_3] & \\
 & & f[x_2, x_3] & & \\
x_3 & f[x_3] & & &
\end{array}
$$

Here is the case $n = 3$.

Theorem: Divided difference and derivative

Theorem

Let the function f be defined and have k bounded derivatives in an interval $[a, b]$ and let $z_0, z_1, \ldots, z_k$ be $k+1$ distinct points in $[a, b]$. Then there is a point $\zeta \in [a, b]$ such that

$$f[z_0, z_1, \ldots, z_k] = \frac{f^{(k)}(\zeta)}{k!}$$

Textbook, page 312

MATLAB code

```
function [c] = coeffnewton(x,y)
% COEFFNEWTON computes the divided differences needed for
%             constructing the interpolating polynomial
%             through (x_i,y_i) 'in place'
n = length(x)-1;    % degree of interpolating polynomial
for k=1:n+1         % divided differences
   c(k) = y(k);
end
for k=1:n
   for i=n+1:-1:k+1
      c(i)=(c(i)-c(i-1))/(x(i)-x(i-k));
   end
end
c=c(:);
```

Evaluation of Newton polynomial by Horner scheme

We evaluate the interpolation polynomial for a new argument z by a procedure similar to Horner's rule

$$p_n(z) = c_0 + (z - x_0)\,(c_1 + (z - x_1)\,(c_2 + \cdots \\ +(z - x_{n-1})\,(c_n)\cdots))$$

by the recurrence

$$p := c_n$$
$$p := (z - x_i)p + c_i, \qquad i = n-1, n-2, \ldots, 0.$$

Complexity. The computation of the Newton coefficients costs $3n^2/2$ flops. The evaluation of the Newton polynomial costs $3n$ flops per data point.

MATLAB code

```
function y = intnewton(x,c,z)
% INTNEWTON evaluates the Newton interpolating polynomial
%           at the new points z: y = P_n(z) using the
%           Horner form and the diagonal c of the
%           divided difference scheme.

n = length(x)-1;
y = c(n+1);
for i= n:-1:1
   y = y.*(z - x(i)) + c(i);
end
```

MATLAB example

We want to compute $\log(1.57) = 0.451075619360217$. We have a table with 4 values of the natural logarithm:

x	y
1.4	0.336472236621213
1.5	0.405465108108164
1.6	0.470003629245736
1.7	0.530628251062170

With the above MATLAB functions we can easily verify that

indices	$p^*_{\ldots}(1.57)$	error
1,2,3	0.451109779691149	$3.416 \cdot 10^{-5}$
2,3,4	0.451053032333184	$-2.259 \cdot 10^{-5}$
1,2,3,4	0.451077622854969	$2.003 \cdot 10^{-6}$

Algorithms comparison

Basis name	$\phi_j(x)$	construction cost	evaluation cost	selling feature
Monomial	x^j	$\frac{2}{3}n^3$	$2n$	simple
Lagrange	$L_j(x)$	n^2	$5n$	$c_j = y_j$ most stable
Newton	$\prod_{i=0}^{j-1}(x - x_i)$	$\frac{3}{2}n^2$	$2n$	adaptive

Theorem: polynomial interpolation error

If p_n interpolates f at the $n+1$ points $x_0, \ldots, x_n$ and f has $n+1$ bounded derivatives on an interval $[a, b]$ containing these points, then for each $x \in [a, b]$ there is a point $\xi = \xi(x) \in [a, b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

By consequence

$$|e_n(x)| \leq \max_{a \leq t \leq b} \frac{|f^{(n+1)}(t)|}{(n+1)!} \max_{a \leq s \leq b} \left| \prod_{j=0}^{n} (s - x_j) \right|$$

$$\|e_n\|_\infty \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} (b-a)^{n+1}$$

Whats wrong with polynomial interpolation?

We are (still) given a collection of data samples $\{(x_i, y_i)\}_{i=0}^{n}$ looking for a function $v(x)$ that satisfies

$$v(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

- The error bound may not be small if $\frac{\|f^{(n+1)}\|}{(n+1)!}$ is not.
- High order polynomials tend to oscillate "unreasonably".
- Data often suggest only piecewise smooth, whereas polynomials are infinitely differentiable.
- No locality: changing any one data value may drastically alter the entire interpolant.

Piecewise polynomial interpolation

- Divide $[a, b]$ into subintervals

$$a = t_0 < t_1 < t_2 < \cdots < t_r = b$$

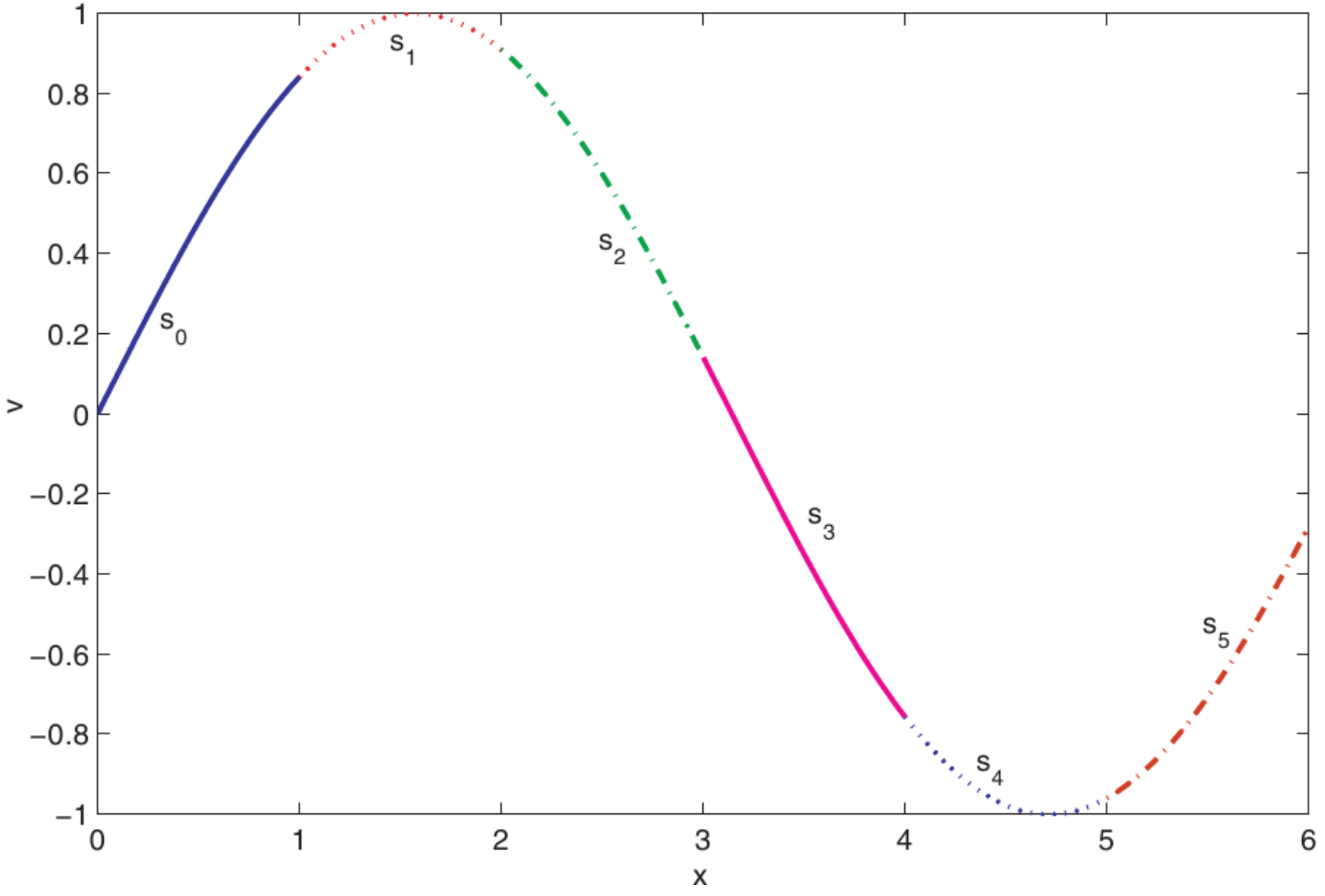
where t_i are the "breakpoints", or "knots".

- Construct interpolant

$$v(x) = s_i(x), \qquad t_i \le x \le t_{i+1}, \quad i = 0, \ldots, r-1$$

where each $s_i(x)$ is a polynomial of low degree (e.g., piecewise cubic: $m = 3$).

- As before $v(x)$ must satisfy the interpolation conditions. In addition, require a global smoothness property.



A piecewise polynomial function with break points $t_i = i$, $i = 0, 1, \ldots, 6$.

t_i and x_i

- Distinguish between abscissae x_i where data are specified

$$a \leq x_0 \leq x_1 \leq \cdots \leq x_{n-1} \leq x_n \leq b$$

- and knots (break points) of the piecewise polynomial

$$a = t_0 < t_1 < t_2 < \cdots < t_r = b$$

- For future purposes, set

$$h = \max_{1 \leq i \leq r} (t_i - t_{i-1}).$$

- Rely on h being small (while $b - a$ is not small) for quality approximation.

Piecewise linear interpolation

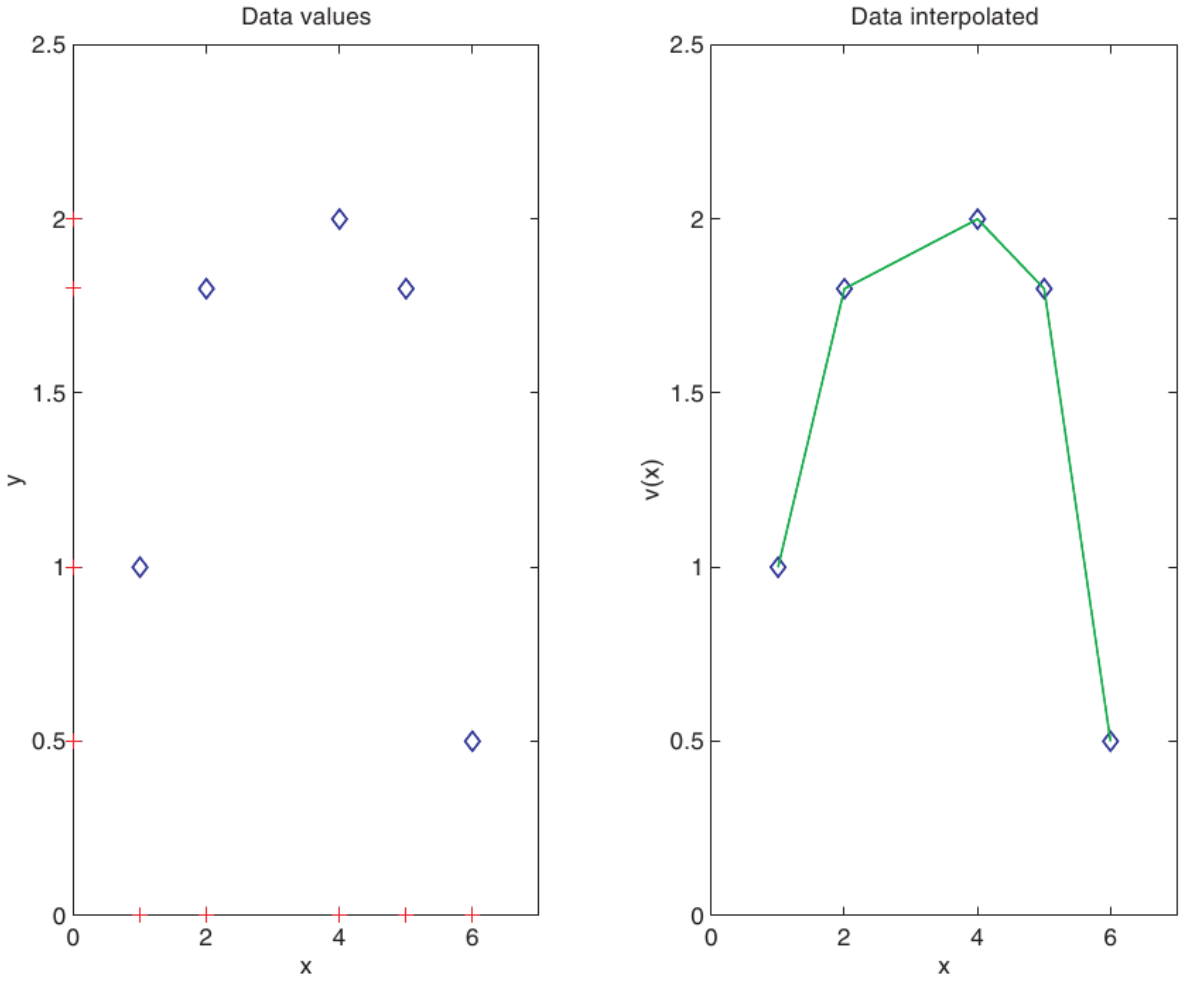
The simplest continuous interpolant.

- Assuming $\{x_i\}_{i=0}^{n}$ are distinct and ordered, identify $t_i = x_i$, $y_i = f(x_i)$.
- Determine $s_i(x)$ as the straight line interpolant at x_i and x_{i+1}. Thus, $m = 1$ and

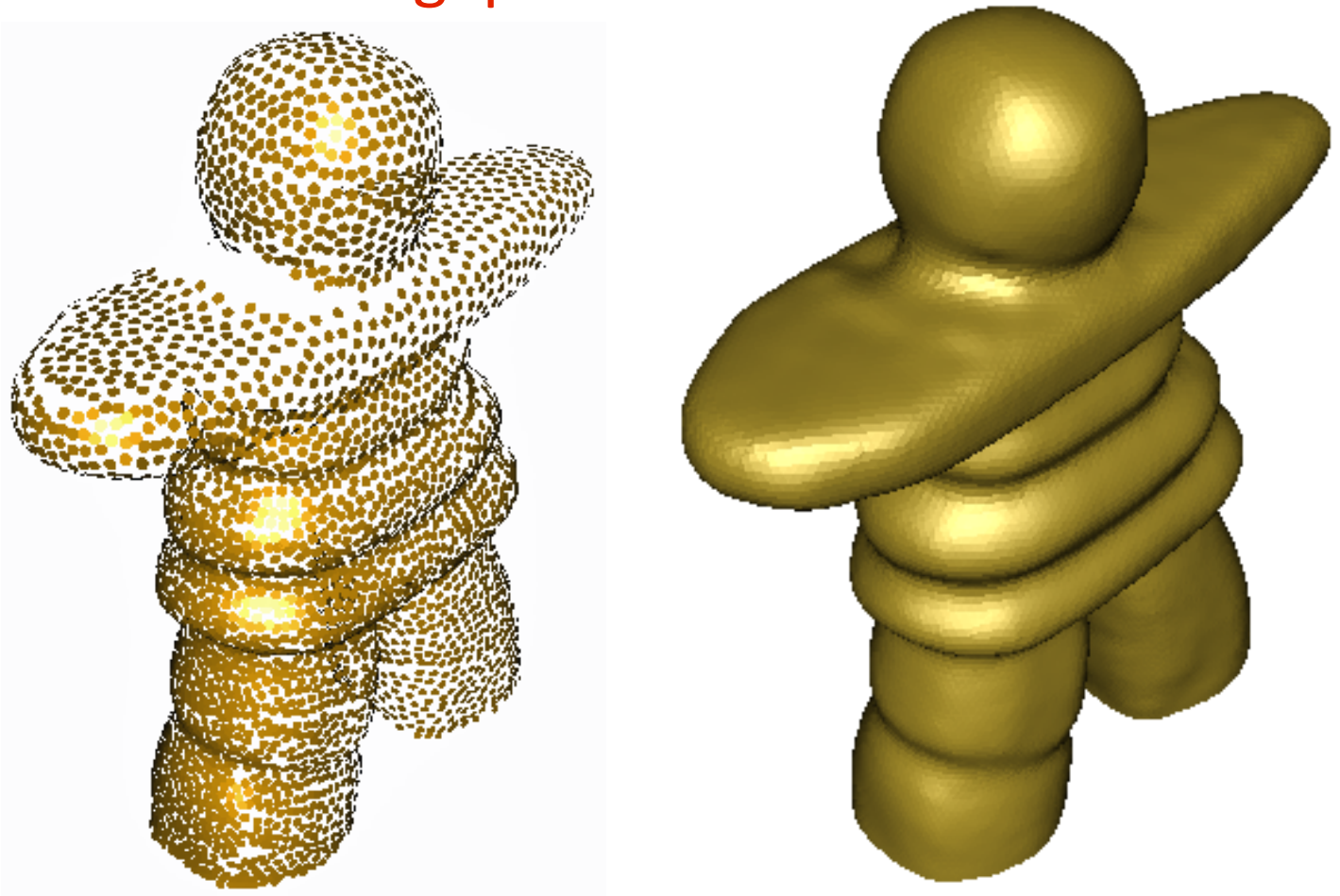
$$v(x) = s_i(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i),$$

  with $x_i \leq x \leq x_{i+1}$, $0 \leq i \leq n-1$.
- Inherently, $v(x)$ is continuous but not smoother than that.
- Shape preservation. No new minima or maxima are "invented" by $v(x)$ for $f(x)$.

Data and broken line interpolation



Interpolation in 3D: e.g. point cloud obtained with laser scanner



(a) consolidated point cloud (b) RBF surface

FIGURE: RBF interpolation of an upsampling of a consolidated point cloud.

RBF: Radial basis functions