


L06: Introduction to Interpolation

Goal: To find out what interpolation is, and how it is useful.

in·ter·po·late  (in-tûr'pə-lāt') [Pronunciation Key](#)
v. **in·ter·po·lat·ed**, **in·ter·po·lat·ing**, **in·ter·po·lates**

v. *tr.*

1. To insert or introduce between other elements or parts.
2.
 - a. To insert (material) into a text.
 - b. To insert into a conversation. See Synonyms at [introduce](#).
3. To change or falsify (a text) by introducing new or incorrect material.
4. *Mathematics* To estimate a value of (a function or series) between two known values.

Interpolation in Image Processing

A digital image is an array of pixels (picture elements). Each pixel is drawn as a tiny square on your screen, its colour specified by a number.

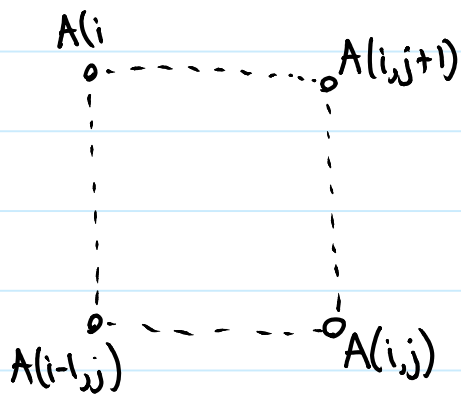


Suppose we have an image, **A**, that we want to alter (shrink, shift, rotate, etc.). In doing so, we create another image, **B**, that is an altered version of the first one. Usually, the pixels from **A** do NOT fall exactly onto pixels in **B**, so we have to estimate values in between the pixels.

$A(x)$

$A(x)$

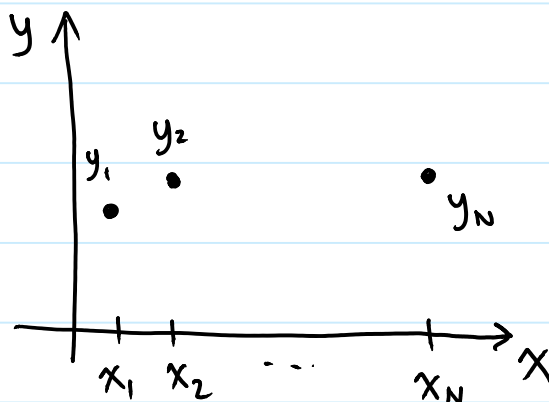
Hence, we are forced to approximate



Hence, we are forced to approximate a value at (x, y) . We interpolate by choosing a value for $A(x, y)$ that is a compromise between the closest pixels.

Interpolating a 1D Function

Suppose we have a set of discrete samples of some unknown function. That is, we are given a set of x -values, $\{x_1, x_2, \dots, x_N\}$ (x_i distinct) and corresponding set of y -values, $\{y_1, y_2, \dots, y_N\}$.



Interpolation is the act of finding a function $f(x)$ s.t.

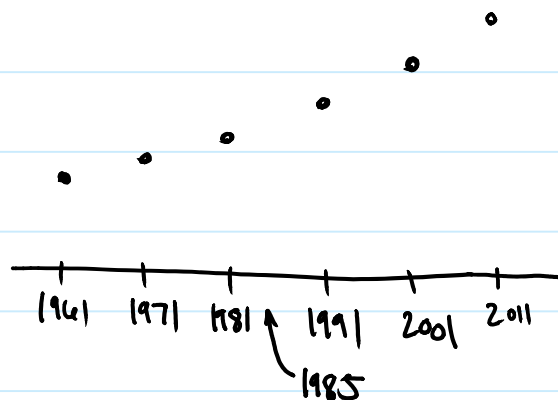
In other words, the interpolation function ("interpolant") passes through the data points (x_i, y_i) for $i = 1, \dots, N$.

Example: The population of Canada

Suppose Canada does a census every 10 years to determine its population.

Year	1961	1971	1981	1991	2001	2011
Pop'n (millions)	18	22	24	27	30	34

What if we want to estimate
Canada's population in 1985?
We need to interpolate.



Polynomial Interpolation

In general, the interpolant $f(x)$ can be any function. If $f(x)$ is chosen to be a polynomial, we call it "polynomial interpolation".

Theorem

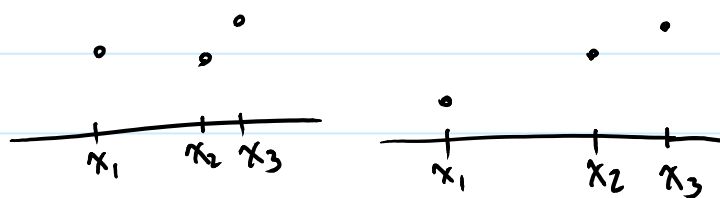
Given n data points (x_i, y_i) , $i=1, \dots, n$, with $x_i \neq x_j$ if $i \neq j$,
 $\exists!$ polynomial of degree at most $n-1$

$$p(x) = c_1 + c_2 x + c_3 x^2 + \dots + c_n x^{n-1}$$

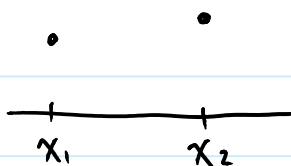
s.t.

$$p(x_i) = y_i, \quad i=1, \dots, n.$$

eg. 3 points



How about 2 points with degree 2?



L07: Polynomial Interpolation

Goal: To design a polynomial that passes through a set of chosen points.

Monomial Form: Vandermonde System

Let $p(x)$ be written

$$p(x) = c_1 + c_2x + c_3x^2 + \dots + c_nx^{n-1}$$

By the definition of an interpolant,

$$p(x_i) = y_i, \quad i = 1, 2, \dots, n.$$

i.e.

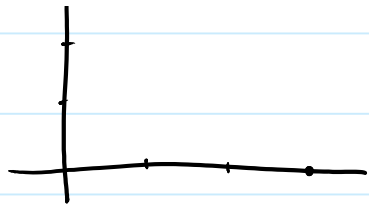
In matrix form...

Example: Points $\{(1,2), (3,1)\}$

|

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

| | | | |



$$\begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

$$X^{-1} = \frac{1}{2} \begin{bmatrix} 3 & -1 \\ -1 & 1 \end{bmatrix}$$

$$\vec{c} = \frac{1}{2} \begin{bmatrix} 3 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ -\frac{1}{2} \end{bmatrix}$$

$$\therefore \text{the interpolant is } p(x) = \frac{5}{2} - \frac{1}{2}x$$

(demo monomial.m)

Disadvantages of Monial Form

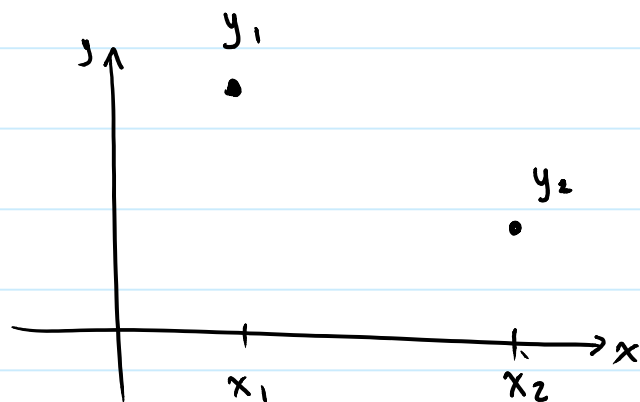
- 1) Need to solve a linear system
- 2) The matrix entries become large as n gets bigger
 - matrix X becomes nearly singular
 - difficult to solve accurately

Lagrange Form

For $n=2$:

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

$$L_2(x) = \frac{x - x_1}{x_2 - x_1}$$



$p(x) = L_1(x)y_1 + L_2(x)y_2$ is the interpolating polynomial.

For $n=3$:

For $n=3$:

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

$$L_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}$$

$$L_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3$$

In general, for $(x_i, y_i) \ i=1, \dots, n$

$$L_i(x) = \frac{(x-x_1) \dots (x-x_{i-1})(x-x_{i+1}) \dots (x-x_n)}{(x_i-x_1) \dots (x_i-x_{i-1})(x_i-x_{i+1}) \dots (x_i-x_n)}$$

$$p(x) = \sum_{i=1}^n L_i(x) y_i$$

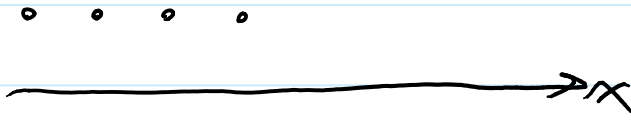
(Matlab script lagrange.m)

There are problems with monomial interpolation.

- The interpolant often does not follow the data in a "reasonable" way.
- The Vandermonde matrix can become difficult to invert as you include more points.
- Especially if two points have similar x-values.

(Matlab script monomial.m)

• • • •

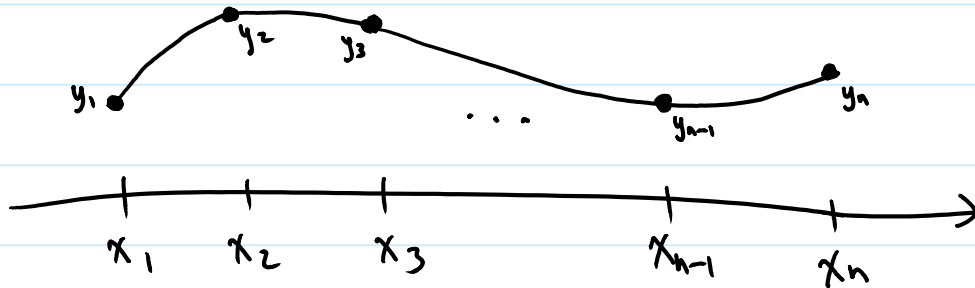


L08: Piecewise Polynomial Interpolation

Goal: To see alternatives to monomial interpolation.

Piecewise Polynomial Interpolation

As the name suggests, we can build a function out of a bunch of polynomial pieces.



The x-values are called $x_1, x_2, x_3, \dots, x_{n-1}, x_n$, or x_i .

The entire piecewise polynomial is denoted $S(x)$.

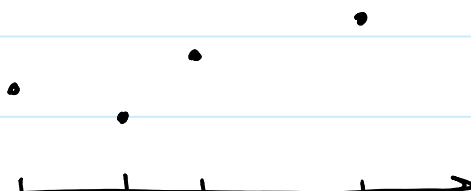
$$S(x) = \begin{cases} p_1(x) & \text{for } x_1 \leq x < x_2 \\ p_2(x) & \text{for } x_2 \leq x < x_3 \\ \vdots & \\ p_{n-1}(x) & \text{for } x_{n-1} \leq x \leq x_n \end{cases}$$

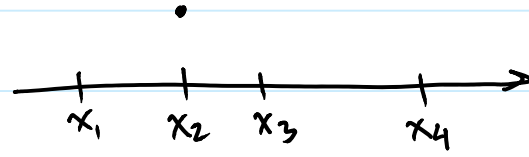
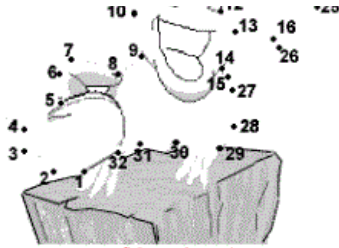
A piecewise polynomial interpolator is:

- 1) an n -point interpolator
- 2) a polynomial of degree $n-1$ on each subinterval
- 3) a continuous function on the whole interval

Piecewise Linear Interpolation

Join the dots with lines.



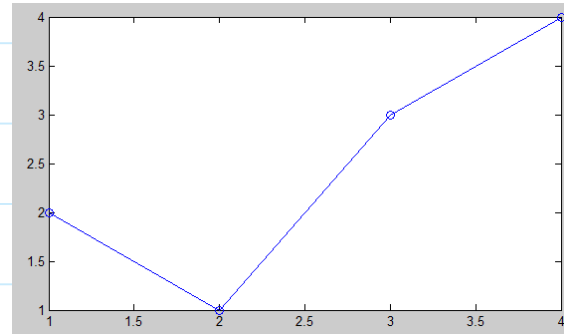


Each $p_i(x)$ is a line on the interval $[x_i, x_{i+1})$.

This is a very common and simple form of interpolation.

Matlab uses it for plotting.

```
>> x = [1 2 3 4];
>> y = [2 1 3 4];
>> plot(x, y, 'o-');
```



A piecewise linear function is continuous, but not

i.e.

To achieve smoothness, we must use polynomial pieces of higher degree.

Cubic Spline Interpolation

Definition:

$S(x)$ is called a cubic spline if

1)

2)

3)



1) Interpolant constraint

2) Differentiability constraint

3) No-Jerk constraint

How many variable vs. how many constraints?

Equations

- Interpolant:
- Differentiability:
- No-Jerk constraint:

Total:

Unknowns

For each cubic piece, there are unknowns.

$$p_k(x) = c_1^{(k)} + c_2^{(k)} x + c_3^{(k)} x^2 + c_4^{(k)} x^3$$

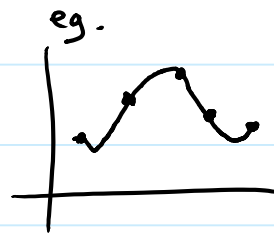
$$p_k(x) = C_1 + C_2 x + C_3 x^2 + C_4 x^3$$

Total:

Thus, we have more unknowns than we have equations. We need more constraints to uniquely determine the interpolator function.

Boundary Conditions refer to constraints placed on the spline at the first and last nodes.

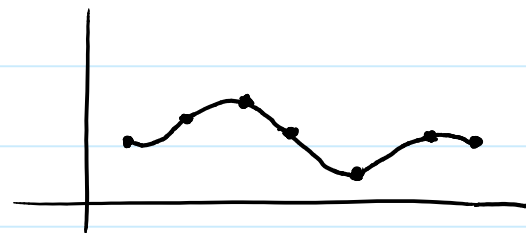
Clamped spline



Natural spline

Corresponds to the minimum energy of an elastic (flexible) band.

Periodic spline



Combinations

eg.

L09: Cubic Splines 2

Goal: To learn an alternate and more efficient representation for cubic splines.

Cubic Splines: Alternate Representation

Monomial form is

$$p_i(x) = c_1^{(i)} + c_2^{(i)}x + c_3^{(i)}x^2 + c_4^{(i)}x^3$$

Instead, we'll use

$$p_i(x) =$$

where $h_i = x_{i+1} - x_i$, $i = 1, \dots, n-1$.

$$p_i'(x) =$$

$$=$$

$$p_i''(x) =$$

We have to choose all the a 's, b 's and c 's so that $S(x)$ satisfies all the constraints of a cubic spline.

Interpolation Constraint

$$p_i(x_i) = y_i$$

$$=$$

=

=

$$p_i(x_{i+1}) = y_{i+1}$$

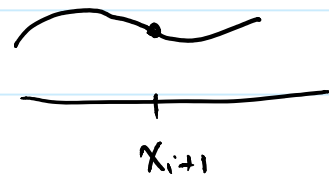
=

=

Thus, if we know the a_i 's, we can calculate the b_i 's and c_i 's.

No-Jerk Constraint (continuity of $S''(x)$)

$$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1})$$



.
.
.
.
.
.
.

Thus, $S''(x)$ is ; we have no choice!

Differentiability (continuity of $S'(x)$)

$$\text{i.e. } p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}), \quad i = 1, \dots, n-2$$

This will give us a system of equations that we will solve to get

the a -values.

We'll move all the a -terms to the left-hand-side, and the rest to the right-hand-side.

$n-2$ equations for n unknowns $\{a_0, a_1, \dots, a_{n-1}\}$.

Bring in two boundary conditions.

eg. Clamped BCs

$$p'_1(x_1) = v_1$$

$$p'_{n-1}(x_n) = v_2$$

$$\begin{array}{ccccccc} \cdot & \cdot & \cdots & \cdot & \cdot \\ \hline \end{array}$$

$$p'_i(x) =$$

$$p'_i(x_i) =$$

$$=$$

\Rightarrow

Likewise, from $p'_{n-1}(x_n) = v_2$ we get

Let's put it into one big matrix system!

$$\begin{bmatrix} \vdots \\ a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

Note: Natural BCs are even easier!

$$\left. \begin{aligned} \Rightarrow p_1''(x_1) = 0 &\Rightarrow a_0 = 0 \\ p_{n-1}''(x_n) = 0 &\Rightarrow a_{n-1} = 0 \end{aligned} \right\} \begin{array}{l} 1^{\text{st}} \text{ \& last eq'ns} \\ \text{are trivial to} \\ \text{solve.} \end{array}$$

Advantages of using this formulation

- $S''(x)$ is continuous by design, so those constraints do not factor into the problem.
 \Rightarrow smaller linear system to solve
- Many of the equations are largely decoupled, so the b 's and c 's are easy to compute once you know the a 's.
 \Rightarrow smaller linear system to solve
- The system involving the a 's is tri-diagonal. In general, an $n \times n$ system takes $O(n^3)$ floating-point operations (flops) to solve. A tri-diagonal system can be solve in $O(n)$ flops.

\therefore this formulation is much easier and faster to compute.

Splines in Matlab

Matlab has a basic spline function, as well as more flexible implementations in its curve-fitting toolbox.

```
y = [4 8 7 5 1 2 9 7 8 9 0]
```

```
N = length(y)
```

```
t = 1:N
```

```
y_cs = spline(t,y)
```

```
tt = linspace(1,N,1000);
```

```
yy = ppval(tt,y_cs);
```

```
plot(t,y,'o', tt,yy)
```

(Matlab demo
spline_example.m)

