Unit 4:

Interpolation

Estimating what goes between samples.

IN-01: Introduction to Interpolation

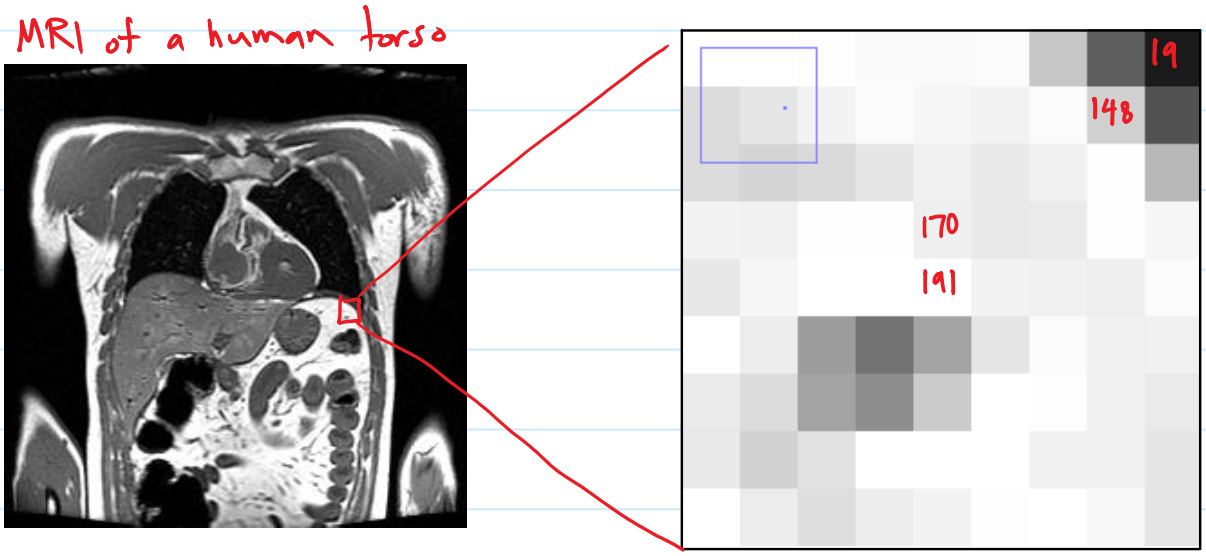
Goal: To find out what interpolation is, and how it is useful.

in·ter·po·late (ĭn-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.

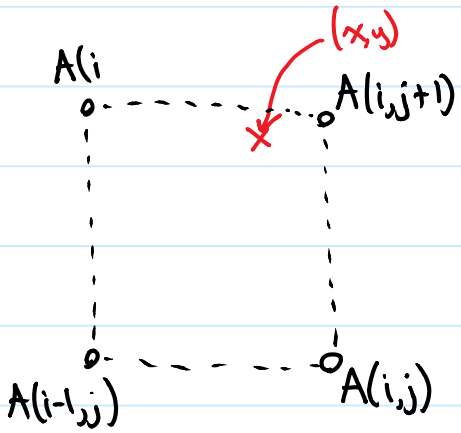
Interpolation in Image Processing

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.

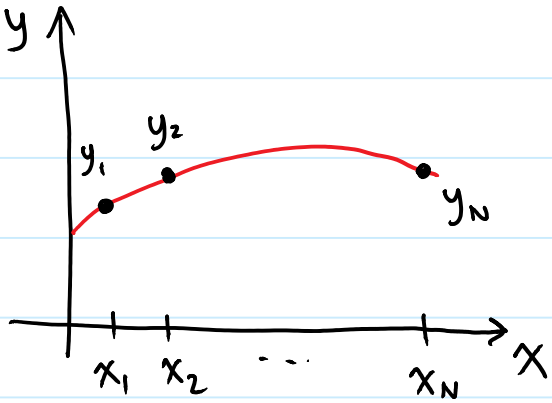
A(i (x,y) Hence, we are forced to approximate



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, $\{x_1, x_2, \ldots, x_N\}$ $(x_i$ distinct$)$ and corresponding set of y-values, $\{y_1, y_2, \ldots, y_N\}$.



Interpolation is the act of finding a function

"such that"

$f(x)$ s.t. $f(x_i) = y_i$, $i = 1, \ldots, N$

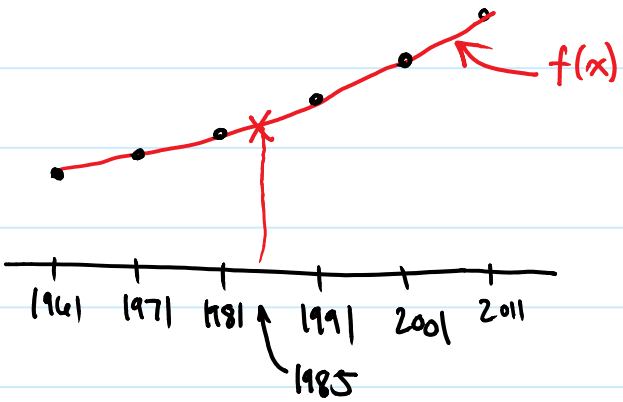
In orther words, the interpolation function ("interpolant") passes through the data points (x_i, y_i) for $i = 1, \ldots, N$.

Example: The population of Canada

Suppose Canada does a concensus 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

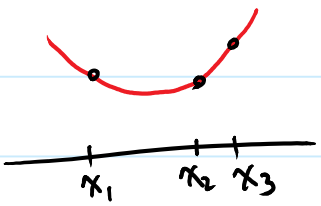
Given n data points (x_i, y_i), $i = 1, \ldots, n$, with $x_i \neq x_j$ if $i \neq j$, $\exists!$ polynomial of degree at most $n-1$

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

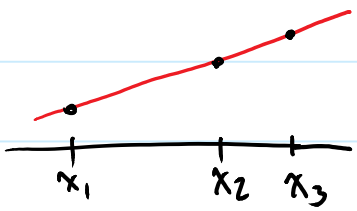
s.t.

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

eg. 3 points

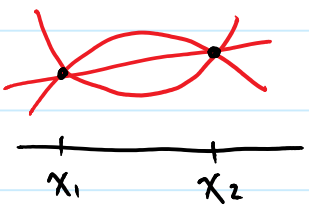


unique parabola (degree = 2)



unique line (degree = 1)

How about 2 points with degree 2?



not unique.

IN-02: Polynomial Interpolation

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

Monomial Form: Vandermonde System

Let $p(x)$ be written

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

By the definition of an interpolant,

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

i.e.

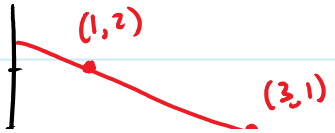
$$\left.\begin{aligned} c_1 + c_2 x_1 + c_3 x_1^2 + \cdots + c_n x_1^{n-1} &= y_1 \\ c_1 + c_2 x_2 + c_3 x_2^2 + \cdots + c_n x_2^{n-1} &= y_2 \\ &\vdots \\ c_1 + c_2 x_n + c_3 x_n^2 + \cdots + c_n x_n^{n-1} &= y_n \end{aligned}\right\} \begin{array}{l} n \text{ equations} \\ n \text{ unknowns} \\ \{c_1, c_2, \ldots, c_n\} \end{array}$$

In matrix form...

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \Rightarrow X\vec{c} = \vec{y}$$

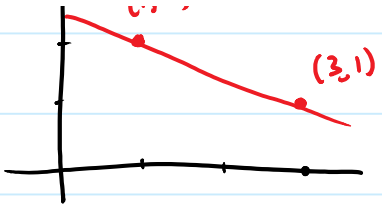
Vandermonde matrix X; $\vec{c}$; $\vec{y}$

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



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

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$X\vec{c} = \vec{y}$$

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

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

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

(demo monomial.m)

Disadvantages of Monial Form

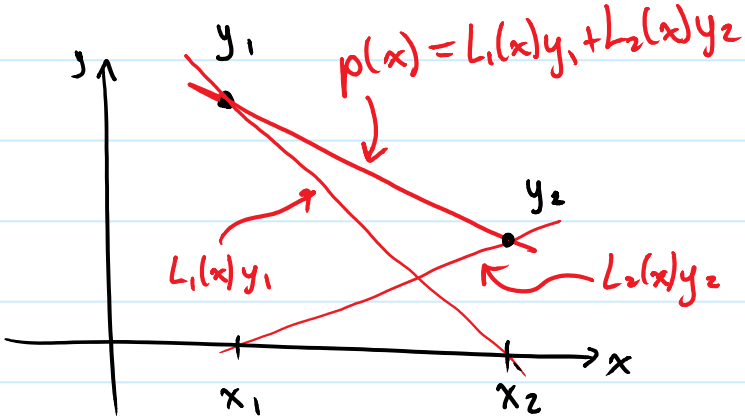
1) Need to solve a linear system
2) The matrix entries become large as n gets bigger
   → matrix X becomes nearly singular
   → difficult to solve accurately

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) $i = 1, \ldots, n$

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

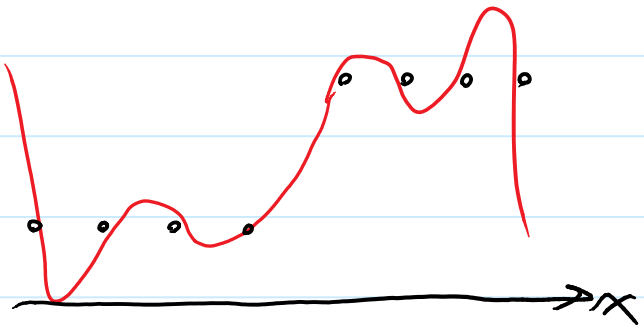
$$p(x) = \sum_{i=1}^{n} L_i(x)\,y_i$$

(Matlab script lagrange.m)

There are <u>problems with monomial interpolation</u>.

- 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)

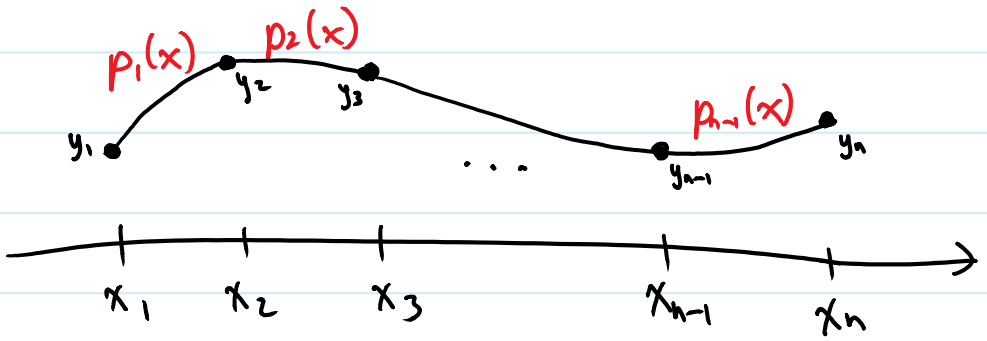


IN-03: Piecewise Polynomial Interpolation

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 **"nodes"**, or **"breakpoints"** or **"knots"**.
The entire piecewise polynomial is denoted $S(x)$.

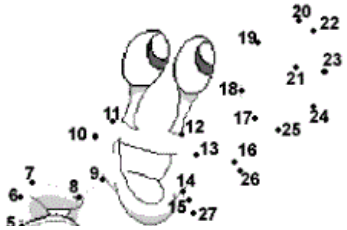
$$S(x) = \begin{cases} p_1(x) & \text{for } x_1 \le x < x_2 \\ p_2(x) & \text{for } x_2 \le x < x_3 \\ \vdots & \\ p_{n-1}(x) & \text{for } x_{n-1} \le x \le x_n \end{cases}$$

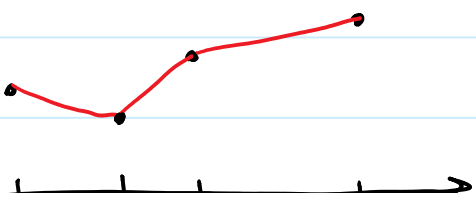
A piecewise polynomial interpolator is:

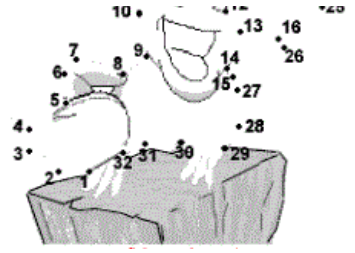
1) an **interpolant** $\Rightarrow S(x_i) = y_i \;,\; i = 1, \ldots, n$
2) a **polynomial** on each subinterval $[x_i, x_{i+1})$
3) **continuous** on the whole interval $[x_1, x_n]$

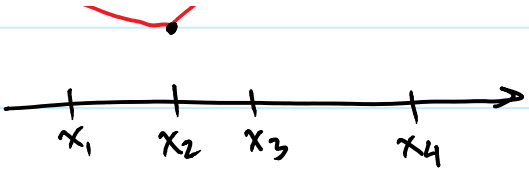
Piecewise Linear Interpolation

Join the dots with lines.





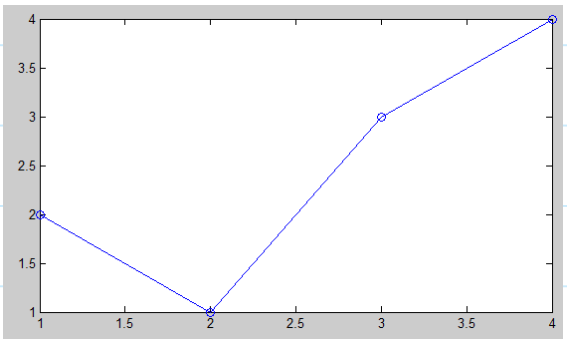




Each $p_i(x)$ is a line on the interval $[x_i, x_{i+1})$.

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 **differentiable (smooth).**

i.e. **first derivative is not continuous**

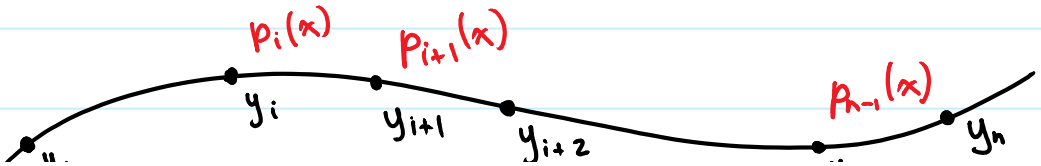
To achieve **smoothness**, we must use polynomial pieces of higher degree.

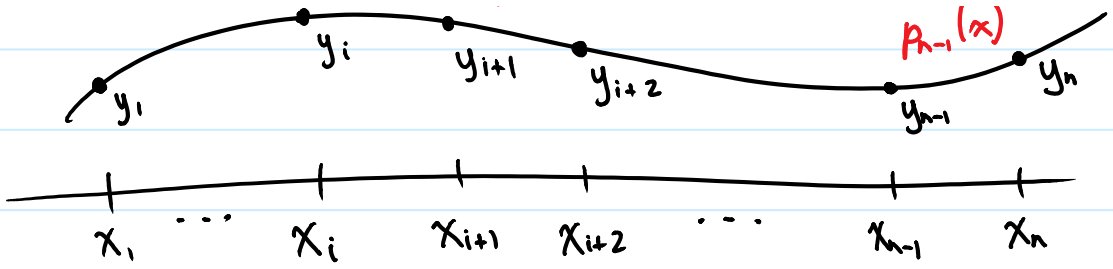
Cubic Spline Interpolation

Definition:

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

1) $S(x)$ is an interpolant
   i.e. $S(x_i) = y_i$, $i = 1, \ldots, n$
2) $S(x)$ is piecewise cubic
3) $S(x)$ is twice differentiable
   i.e. $S'(x)$ and $S''(x)$ are both continuous on (x_1, x_n).





1) Interpolant constraint

$$\left.\begin{aligned} p_k(x_k) &= y_k \\ p_k(x_{k+1}) &= y_{k+1} \end{aligned}\right\} \quad k = 1, \ldots, n-1$$

(guarantees continuity of $S(x)$)

2) Differentiability constraint (Continuity of $S'(x)$.)

$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}) \qquad i = 1, \ldots, n-2$$

3) No-Jerk constraint (Continuity of $S''(x)$)

$$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1}) \qquad i = 1, \ldots, n-2$$

How many variable vs. how many constraints?

<u>Equations</u>

- Interpolant: **2 equations for each piece** $\Rightarrow 2(n-1)$
- Differentiability: **1 equation for each internal node** $\Rightarrow n-2$
- No-Jerk constraint: **1 equation for each internal node** $\Rightarrow n-2$

Total: $4n-6$

<u>Unknowns</u>

For each cubic piece, there are **4** unknowns.

$$p_k(x) = c_1^{(k)} + c_2^{(k)} x + c_3^{(k)} x^2 + c_4^{(k)} x^3 \Rightarrow 4(n-1)$$

prev $= c_1 + c_2 x + c_3 x + c_4 x$ $\Rightarrow 4(n-1)$

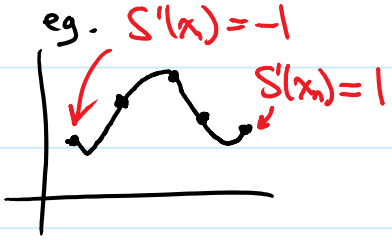
Total: $4n-4$

Thus, we have **2** more unknowns than we have equations. We need **2** 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

$S'(x_1)$ & $S'(x_n)$ are specified (fixed)



Natural spline

$S''(x_1) = S''(x_n) = 0$

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

Periodic spline

$S'(x_1) = S'(x_n)$

$S''(x_1) = S''(x_n)$

(assumes $y_1 = y_n$)

Combinations

eg. $S'(x_1) = 2$

$S''(x_n) = 0$ etc...

IN-04: 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

$$p_i(x) = a_{i-1}\frac{(x_{i+1}-x)^3}{6h_i} + a_i\frac{(x-x_i)^3}{6h_i} + b_i(x_{i+1}-x) + c_i(x-x_i)$$

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

$$p_i'(x) = -3a_{i-1}\frac{(x_{i+1}-x)^2}{6h_i} + 3a_i\frac{(x-x_i)^2}{6h_i} - b_i + c_i$$

$$= -a_{i-1}\frac{(x_{i+1}-x)^2}{2h_i} + a_i\frac{(x-x_i)^2}{2h_i} - b_i + c_i$$

$$p_i''(x) = a_{i-1}\frac{x_{i+1}-x}{h_i} + a_i\frac{x-x_i}{h_i}$$

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

$$p_i(x_i) = y_i$$

$$= a_{i-1}\frac{(x_{i+1}-x_i)^3}{6h_i} + a_i\frac{(x_i-x_i)^3}{6h_i} + b_i(x_{i+1}-x_i) + c_i(x_i-x_i)$$

($(x_i-x_i)^3$ and (x_i-x_i) each $\to 0$)

$$= a_{i-1}\frac{h_i^3}{6h_i} + b_i h_i$$

$$= a_{i-1}\frac{h_i}{6h_i} + b_i h_i$$

$$= a_{i-1}\frac{h_i^2}{6} + b_i h_i \Rightarrow \boxed{b_i = \frac{y_i}{h_i} - a_{i-1}\frac{h_i}{6}}$$

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

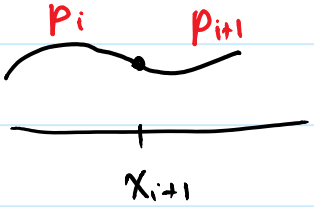
$$= a_{i-1}\frac{(x_{i+1}-x_{i+1})^3}{6h_i} + a_i\frac{(x_{i+1}-x_i)^3}{6h_i} + b_i(x_{i+1}-x_{i+1}) + c_i(x_{i+1}-x_i)$$

$$= a_i\frac{h_i^2}{6} + c_i h_i \Rightarrow \boxed{c_i = \frac{y_{i+1}}{h_i} - a_i\frac{h_i}{6}}$$

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

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

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



$$a_{i-1}\frac{x_{i+1}-x_{i+1}}{h_i} + a_i\frac{x_{i+1}-x_i}{h_i} = a_i\frac{x_{i+2}-x_{i+1}}{h_{i+1}} + a_{i+1}\frac{x_{i+1}-x_{i+1}}{h_{i+1}}$$

$$a_i\frac{h_i}{h_i} = a_i\frac{h_{i+1}}{h_{i+1}} \Rightarrow 1 = 1$$

Phew! It's true!

Thus, $S''(x)$ is **continuous by design**; we have no choice!

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

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

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

the a-values.

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

$$a_{i-1}\frac{h_i}{6} + a_i\frac{h_i+h_{i+1}}{3} + a_{i+1}\frac{h_{i+1}}{6} = \frac{y_{i+2}-y_{i+1}}{h_{i+1}} - \frac{y_{i+1}-y_i}{h_i}$$

$i = 1, 2, \ldots, n-2$

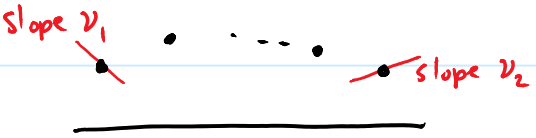
$n-2$ equations for n unknowns $\{a_0, a_1, \ldots, a_{n-1}\}$.

Bring in two boundary conditions.

eg. Clamped BCs

$$p_1'(x_1) = v_1$$

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



$$p_1'(x) = -a_0\frac{(x_2-x)^2}{2h_1} + a_1\frac{(x-x_1)^2}{2h_1} - b_1 + c_1$$

$$p_1'(x_1) = -a_0\frac{h_1}{2} - b_1 + c_1$$

$$= -a_0\frac{h_1}{2} - \left(\frac{y_1}{h_1} - a_0\frac{h_1}{6}\right) + \left(\frac{y_2}{h_1} - a_1\frac{h_1}{6}\right) = v_1$$

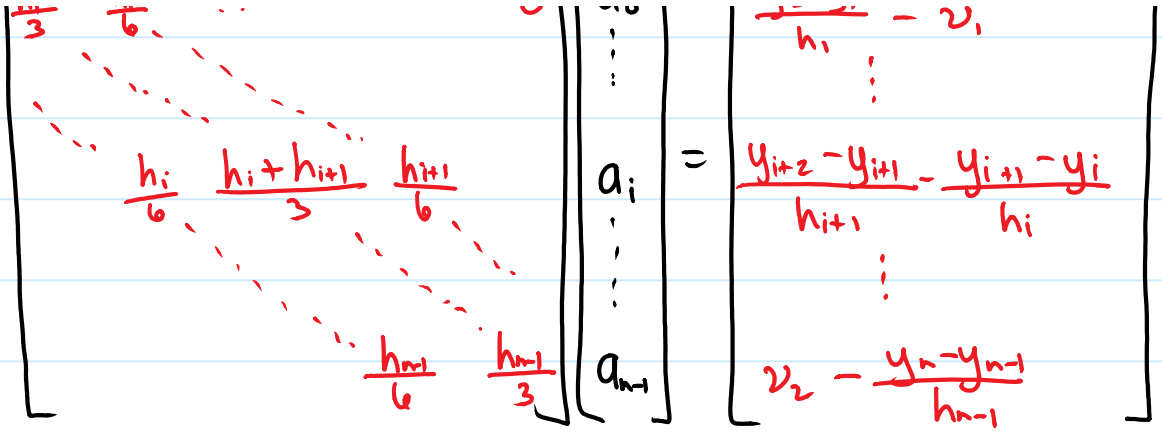
$$\Rightarrow \quad a_0\frac{h_1}{3} + a_1\frac{h_1}{6} = \frac{y_2-y_1}{h_1} - v_1$$

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

$$a_{n-2}\frac{h_{n-1}}{6} + a_{n-1}\frac{h_{n-1}}{3} = v_2 - \frac{y_n-y_{n-1}}{h_{n-1}}$$

Let's put it into one big matrix system!

$$\begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} & & \\ & \ddots & \ddots & \\ & & & \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ \end{bmatrix} = \begin{bmatrix} \frac{y_2-y_1}{h_1} - v_1 \\ \vdots \\ \end{bmatrix}$$



Note: Natural BCs are even easier!

$\Rightarrow p_1''(x_1) = 0 \Rightarrow a_0 = 0$

$p_{n-1}''(x_n) = 0 \Rightarrow a_{n-1} = 0$

} 1st & last eq'ns are trivial to solve.

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 solved 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]
```

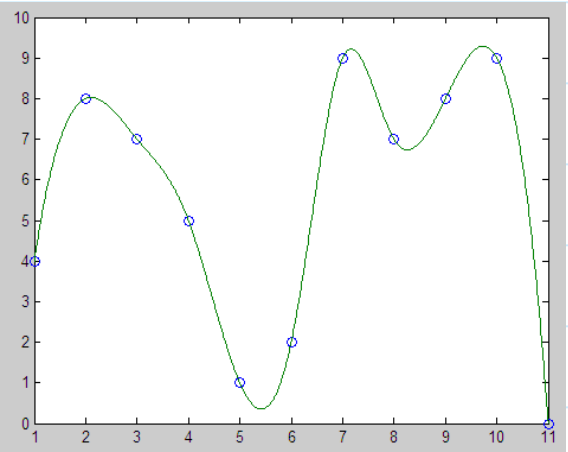
y-values

```
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)
```

generates a corresponding list of values for the independent variable, t.

solves the spline system

finer sampling of t... for plotting a nice smooth spline.

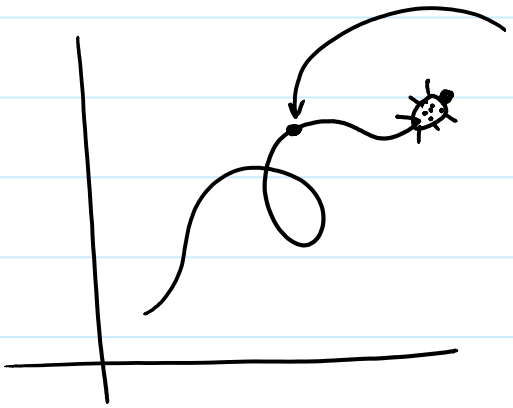


(Matlab demo spline_example.m)

IN-05: Parametric Curves

Goal: To lay the groundwork for drawing curves using interpolators.

Imagine a bug crawling around on a table… Oh, and you happen to have a stopwatch.



Location of bug at time t is $(x(t), y(t))$

This a parametric curve, where t is the parameter.

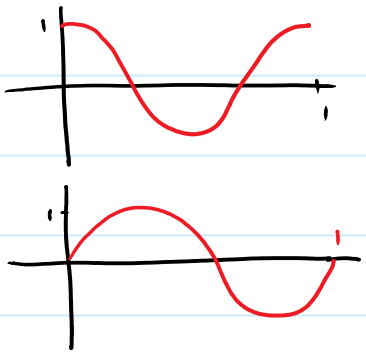
A parametric curve is given by two (or more) functions $x(t)$ and $y(t)$ of a common parameter t, $a \le t \le b$.
In the case of the bug, t could be thought of as time.

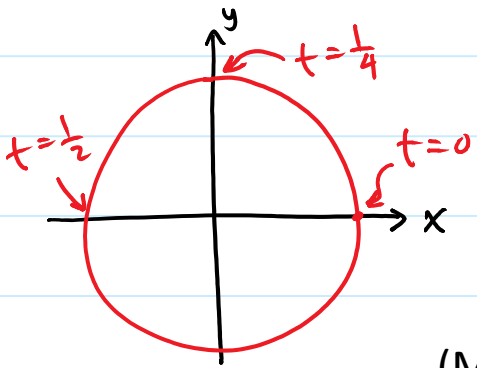
eg. Circle

$x(t) = \cos(2\pi t)$

$y(t) = \sin(2\pi t)$

$0 \le t \le 1$





t	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1
x	1	0	-1	0	1
y	0	1	0	-1	0

(Matlab demo - crawling_bug.m)

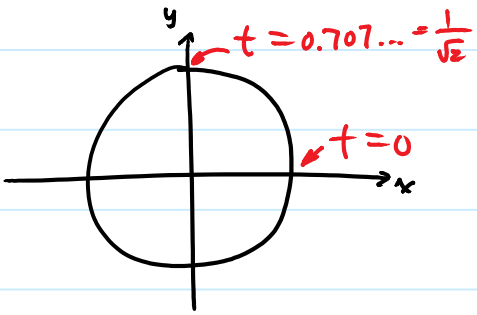
Different parameterizations can give the same parametric curve. For example, the bug's speed could be different for different

traversals of the same path. They bug could even speed up or slow down, but stay on the same curve.

Consider,

$$x(t) = \cos(2\pi t^4)$$
$$y(t) = \sin(2\pi t^4)$$

t	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1
x	1	0.9997	0.9239	-0.4052	1
y	0	$-0.000\ldots$	0.3827	0.9142	0



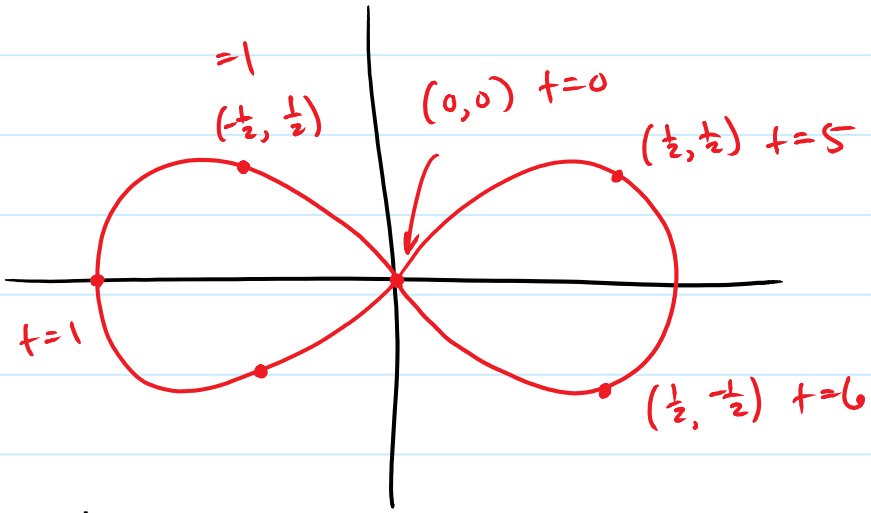
Same curve, different parameterization.

The Reverse Process

Given a curve, find and that approximately draw that curve.

Here's how we'll approach it.

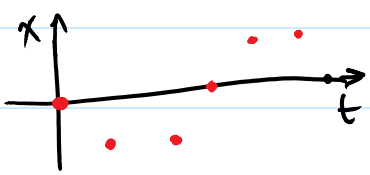
1) Draw the curve
2) Choose points
3) Assign parameter values

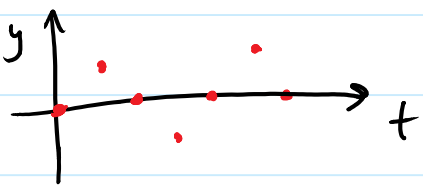


4) Separate x and y components

t	0	1	2	3	$\ldots$
x	0	$-\frac{1}{2}$	-1	$-\frac{1}{2}$	$\ldots$

t	0	1	2	3	$\ldots$
y	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	$\ldots$





5) Interpolate each of the sets of points.
6) Draw $(x(t), y(t))$ as a parametric curve.

5) Interpolate each of the sets of points.
6) Draw $(x(t), y(t))$ as a parametric curve.

(Matlab demo - curve_demo.m)

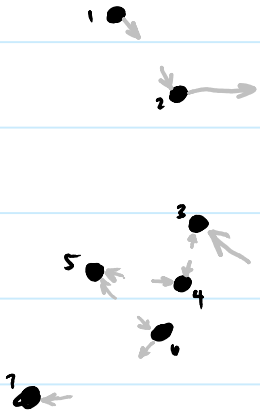
IN-06: Bézier Curves

Goal: To find out about a common type of interpolatin curve used in industry, Bezier curves.

Bézier Curves

Suppose we have a set of interpolation points, and we also want to specify the direction at which our curve approaches each point.

eg.



Introducing... Bézier Curves.

Bézier curves are based on Bernstein polynomials.

$$B_{i,N}(t) = \binom{N}{i} t^i (1-t)^{N-i} \qquad i = 0, \ldots, N$$

A Bezier curve for the set of points $\{x_0, x_1, \ldots, x_N\}$ is a sum of Bernstein polynomials

$$P(t) = \sum_{i=0}^{N} x_i B_{i,N}(t)$$

Then!

- $P(0) = x_0$ since $B_{0,N}(0) = 1$ and $B_{i,N}(0) = 0$ for $i = 1, \ldots, N$

- $P(1) = x_N$ since $B_{N,N}(1) = 1$ and $B_{i,N}(1) = 0$ for $i = 0, \ldots, N-1$

- $P'(t) = \frac{d}{dt}\sum_{i=0}^{N} x_i B_{i,N}(t)$

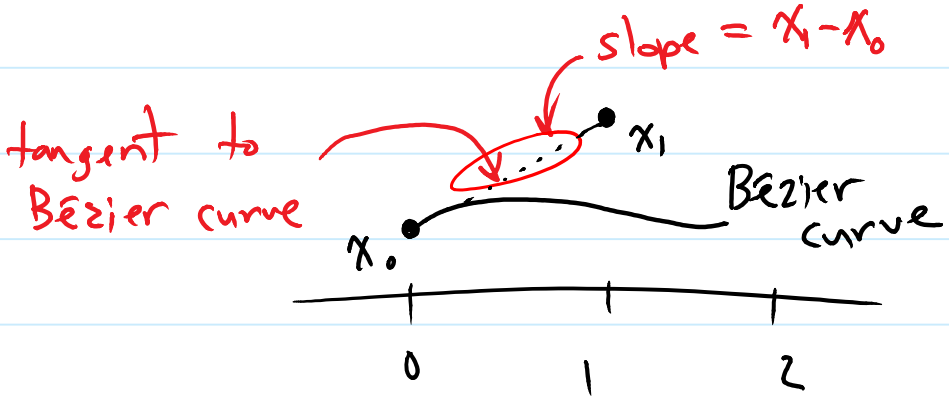
$$= \sum_{i=0}^{N} x_i N\left(B_{i-1,N-1}(t) - B_{i,N-1}(t)\right)$$

At $t=0 \Rightarrow P'(0) = N\sum_{i=0}^{N} x_i\left(B_{i-1,N-1}(0) - B_{i,N-1}(0)\right)$

$$= N\left[x_0\left(B_{-1,N-1}(0) - B_{0,N-1}(0)\right)\right.$$

$$\left. + x_1\left(B_{0,N-1}(0) - B_{1,N-1}(0)\right)\right]$$

$$= N(x_1 - x_0)$$



- $P'(t)\Big|_{t=1} = N(x_N - x_{N-1})$

Hence, for a set of points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)\}$
The Bezier curve:

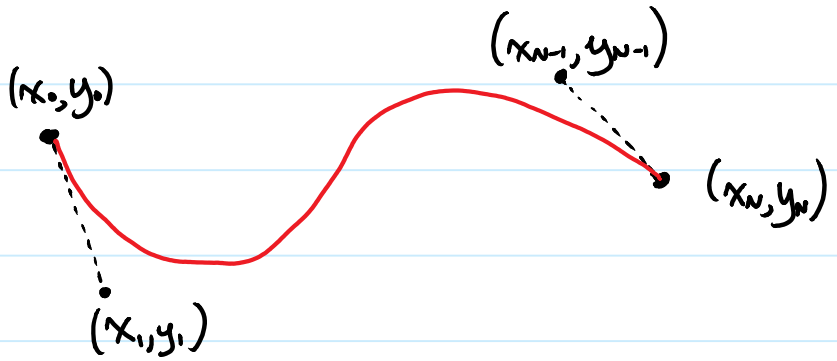
- Passes through (x_0, y_0) & (x_N, y_N)
- Is tangent to

- Is tangent to

$$(x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \text{ at } (x_0, y_0)$$

- Is tangent to

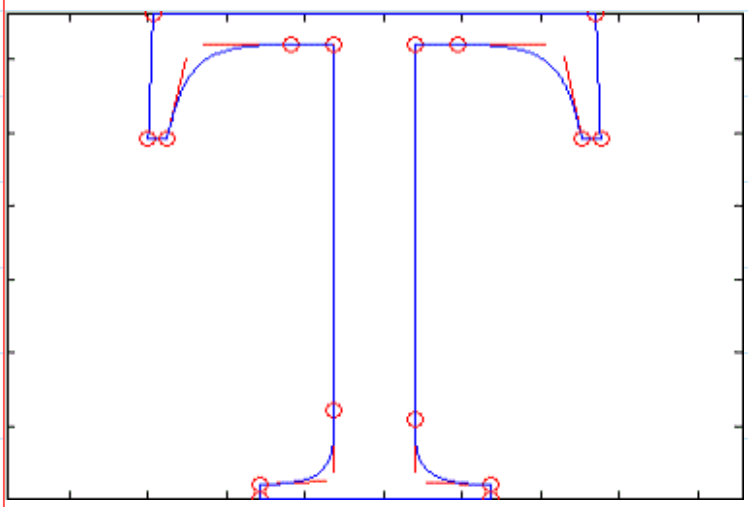
$$(x_N, y_N) + t(x_N - x_{N-1}, y_N - y_{N-1}) \text{ at } (x_N, y_N)$$



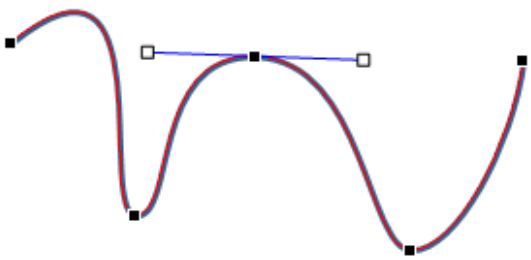
Moreover, the positions of the points "pull" the curve, although the curve does not necessarily pass through the points (except for the first and last).

<u>Application</u>

Bézier curves are used in TrueType fonts.



Microsoft Word, and other drawing applications.



(Matlab demo - Bezier.m)