Interpolation – More Flexible Curves

CS 370 - Lecture 8

May 18, 2016

Cubic splines via Hermite approach

Recall: we showed how to use Hermite interpolation ideas to solve for a cubic spline.

1. Set up a linear system to solve for slopes s_i, by requiring matching 2nd derivatives.
2. Use the closed form for Hermite interpolation on each interval to get a_i, b_i, c_i, d_i.

Benefits of this approach over finding a_i, b_i, c_i, d_i directly?

Matrix size

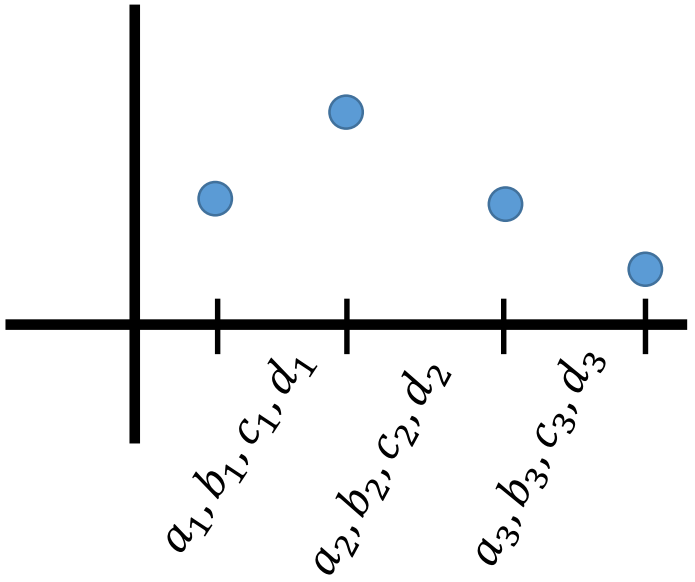
Consider the difference in matrix dimensions.

Old system (for a_i, b_i, c_i, d_i): 4 coefficients *per interval*, so the size is $(4n-4) \times (4n-4)$.

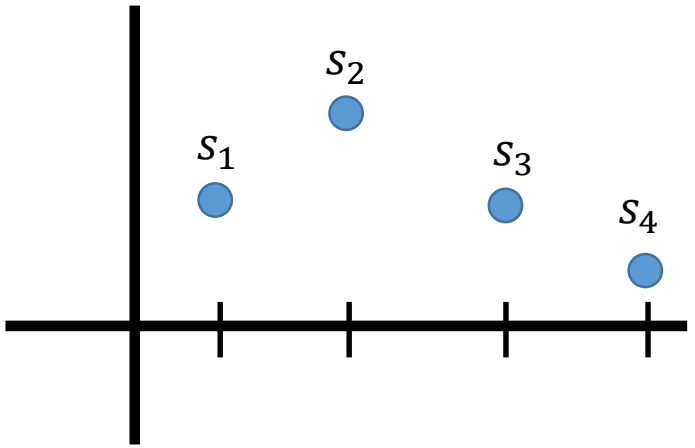
New system (for s_i): one equation *per node*, so size is $n \times n$.

It's *smaller* by a constant factor (i.e. roughly 4X for large n).

Example Comparison



Old: 12 unknowns.



New: 4 unknowns.

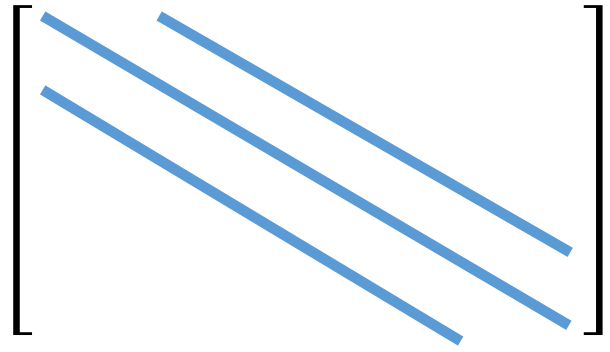
Matrix structure

What about the *structure* of the matrix entries?

i.e, what is the pattern of zeros v.s. non-zeros in T?

The matrix is ***tridiagonal***. Only the entries on the diagonal and its two neighbours are ever non-zero!

$$\begin{bmatrix} X & X & & & 0 \\ X & X & X & & \\ & X & \dots & \dots & \\ & & \dots & X & X \\ 0 & & & X & X \end{bmatrix} = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix}$$

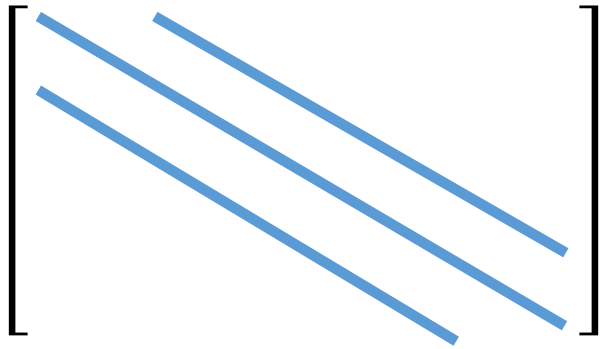


Tridiagonal matrices - Storing

It is often possible to write (vastly) more efficient algorithms by exploiting *matrix structure* of specific problems.

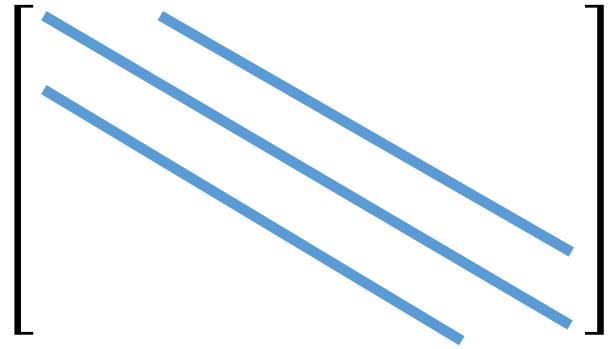
How might you store a tridiagonal matrix?

Store three arrays, one per diagonal. (Or one array of 3-tuples).



Tridiagonal matrices are very “sparse” – they have few non-zero entries. This is one example of exploiting matrix sparsity.

Tridiagonal matrices - Solving



Tridiagonal systems can be solved more quickly than general systems.

Specifically, $O(N)$, where n is the number of rows (i.e., # of points).

Standard Gaussian elimination involves 2 stages:

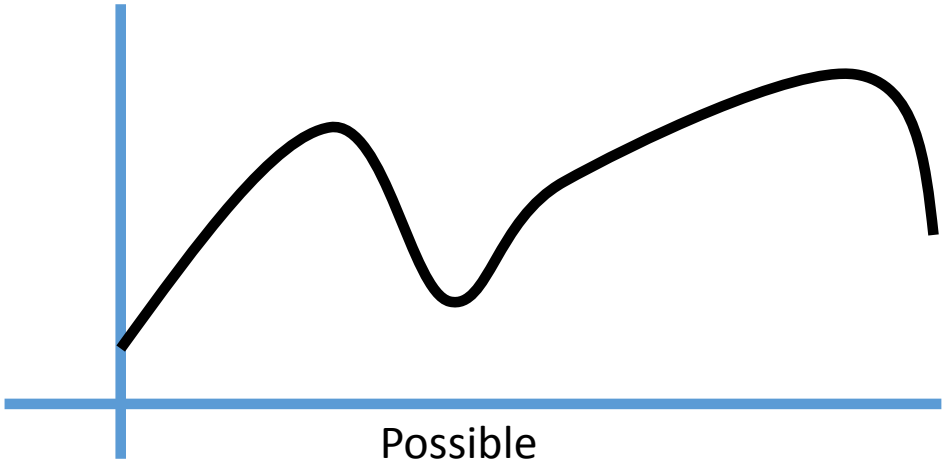
1) Forward elimination. (Eliminate entries *below* the main diagonal.)
   - Only $\mathrm{O}(n)$ such entries for tridiagonal: 1 per row.
2) Back substitution. (Eliminate entries *above* the diagonal.)
   - Only $\mathrm{O}(n)$ such entries for tridiagonal: 1 per row.

Shortcomings of Our Interpolants So Far?

1. Moving a node of a cubic spline has a “global” effect.

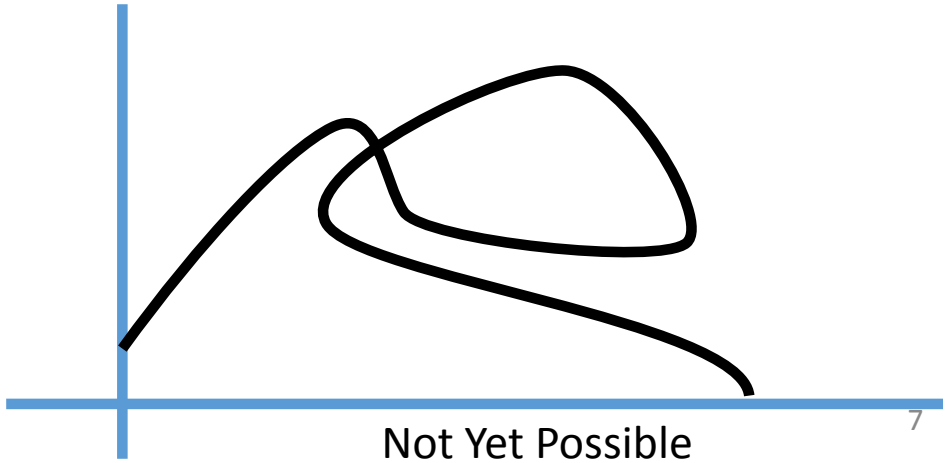
The linear system involves *all nodes together*; a local change can modify the whole curve. This makes local control/editing (e.g., of slopes) tricky.

2. We have only handled functions of the form $y = p(x)$, where one coordinate is a function of the other. What if we want e.g., a curve that folds back over itself?



Possible

v.s.



Not Yet Possible

Shortcomings of Interpolants So Far

Solutions:

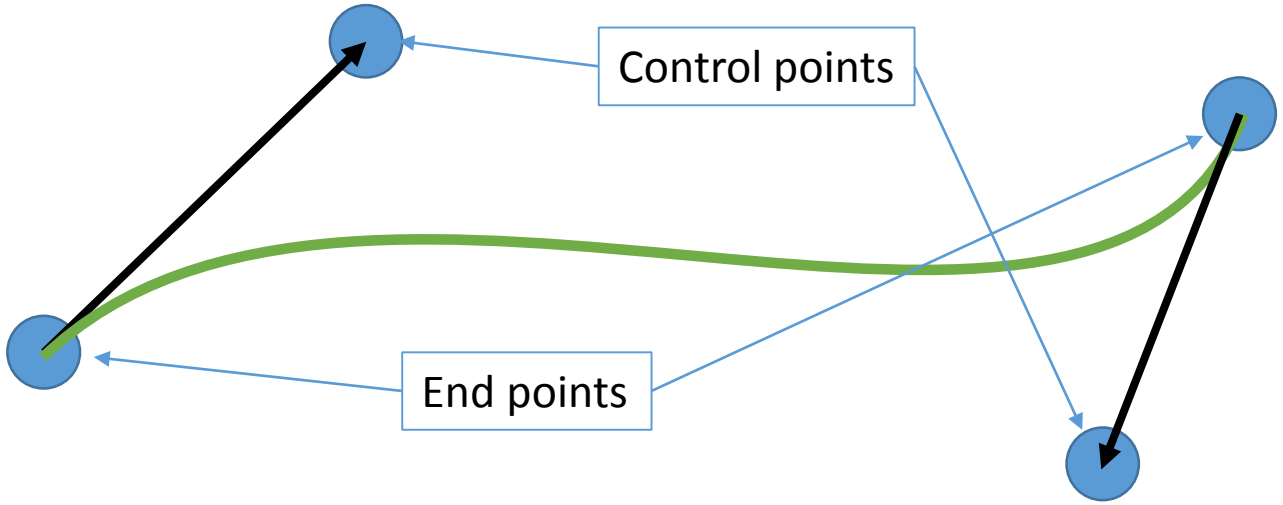
1. New curve types: “*Bezier curves*” and “*B-spline curves*” use *two kinds* of points.

- **interpolation points** that the curve passes right through, as usual.
- **control points** to manipulate curves in a more flexible/local way.

2. The notion of “parametric curves” will let us handle more general curves.

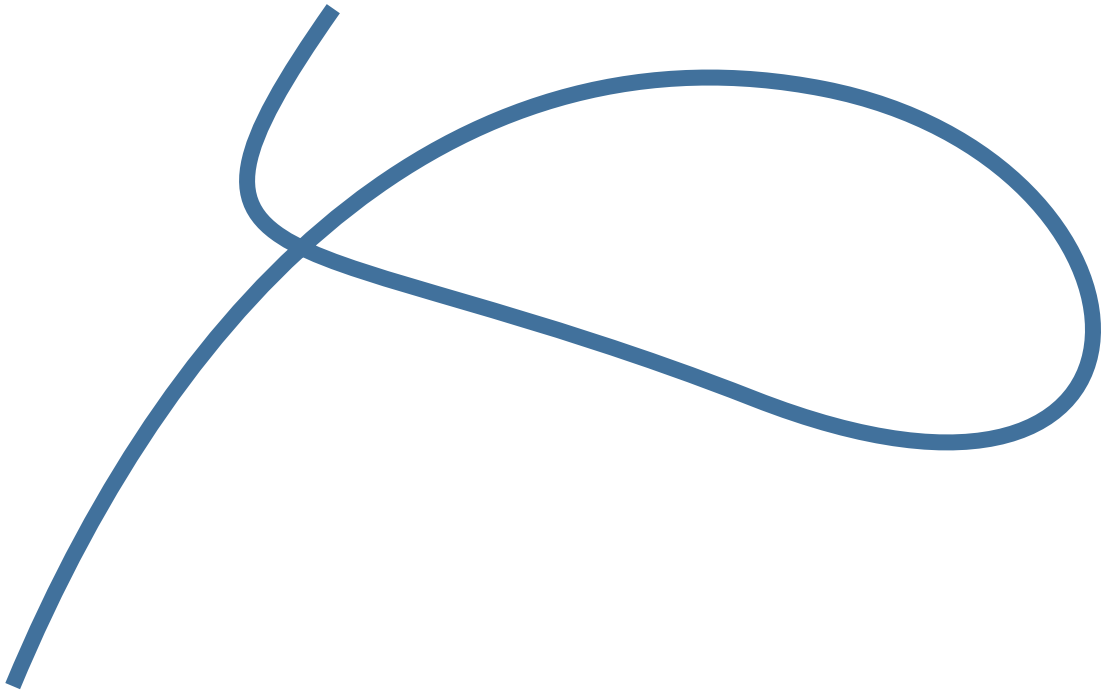
End points v.s. Control points

This curve interpolates the end points. It is affected by the control points, but doesn't interpolate them.



The control points dictate the derivative at the nodes (roughly like in Hermite interpolation) but their *position* also affects the curve shape.

PowerPoint curves example



1. Manipulating ***control points*** modifies the curve locally.

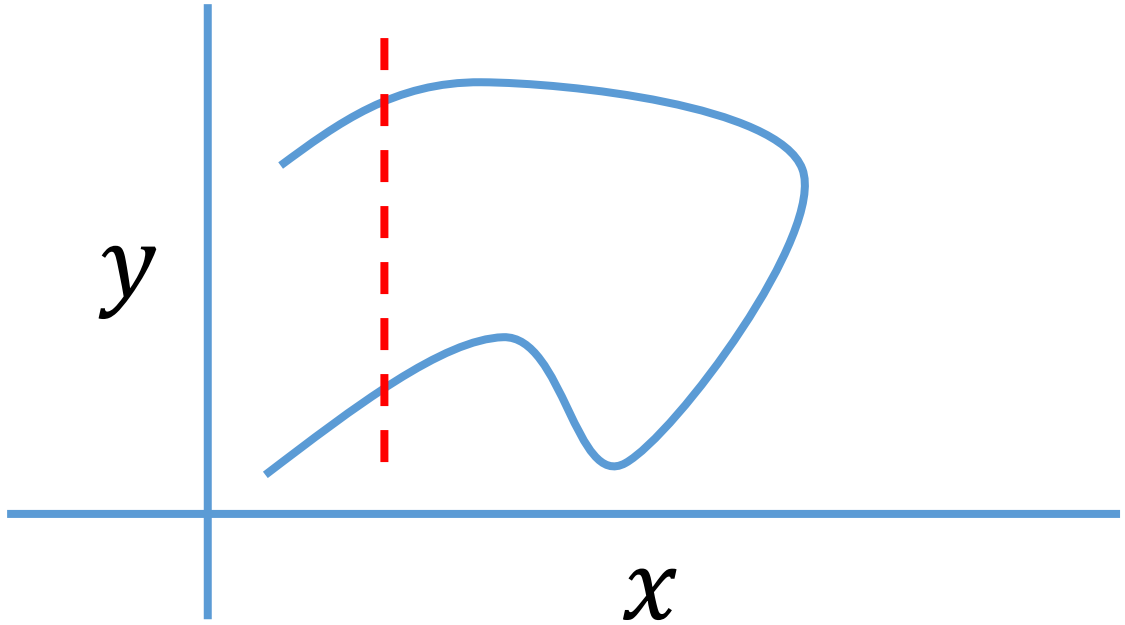
2. We can create curves that fold back and overlap themselves.

Parametric Curves

General Curves: Problem

So far, our curves all had the form $y = p(x)$.

x uniquely dictates y. Therefore two distinct points cannot have the same x coordinate!



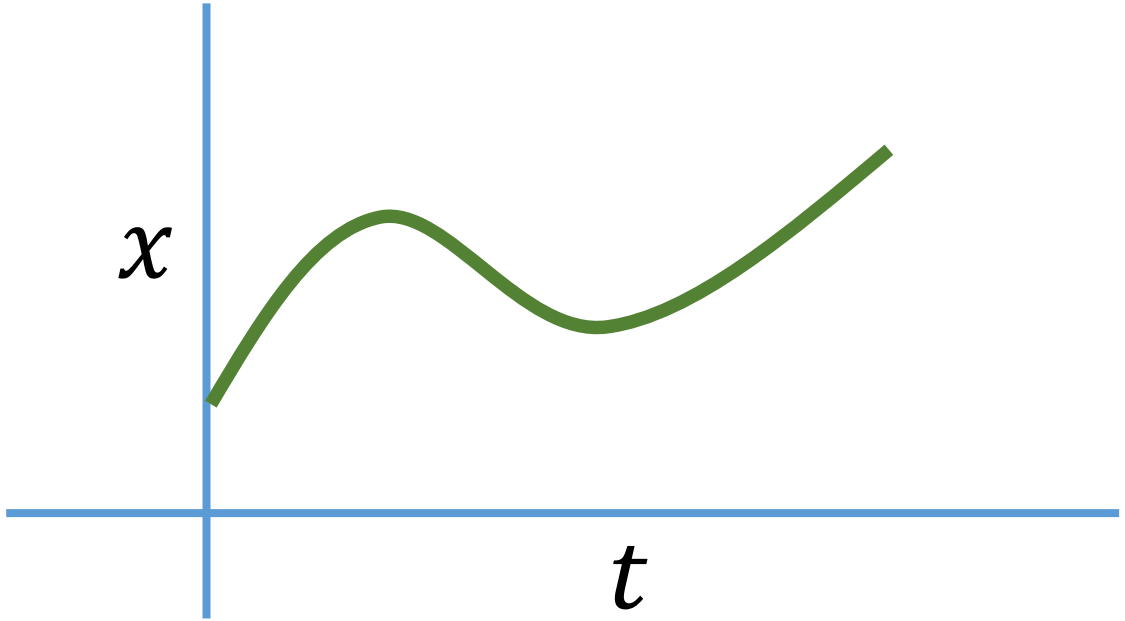
The red line crosses two points with the same x-coordinate, so this configuration is impossible.

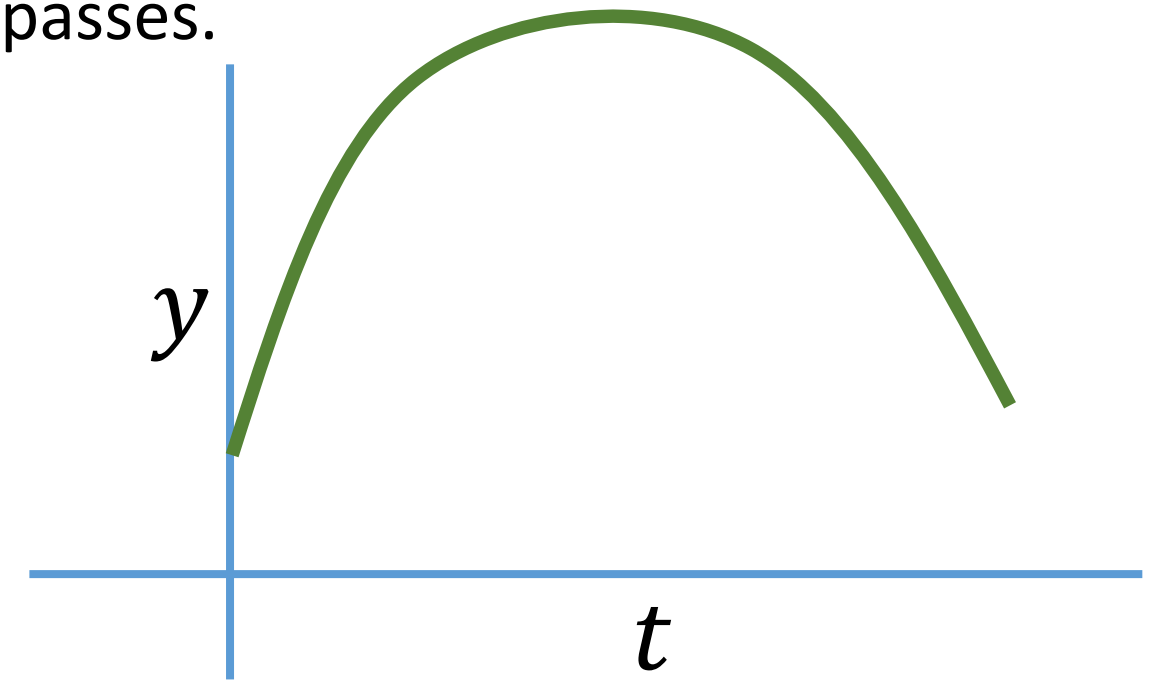
Solution: *Parametric* Curves

Let x and y each be separate functions of a *new* parameter, t.

Then a point's position is given by the vector $\overrightarrow{P(t)} = (x(t), y(t))$.

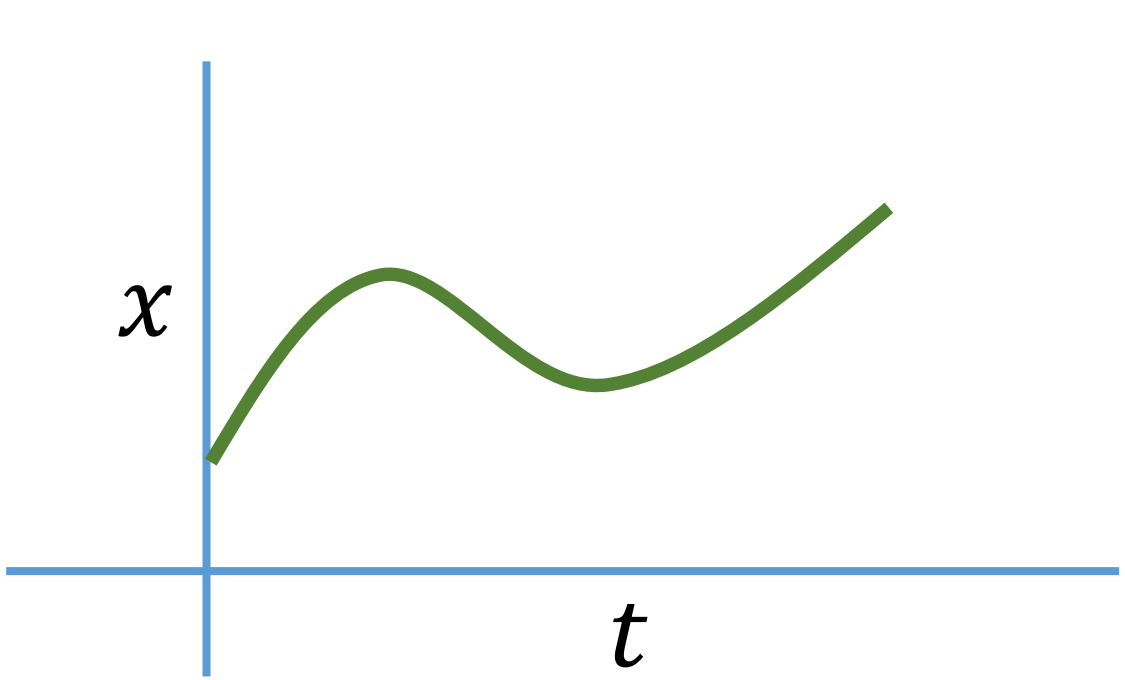
e.g., the parameter t might be *time*, where an object's spatial coordinates, (x, y), change as time passes.

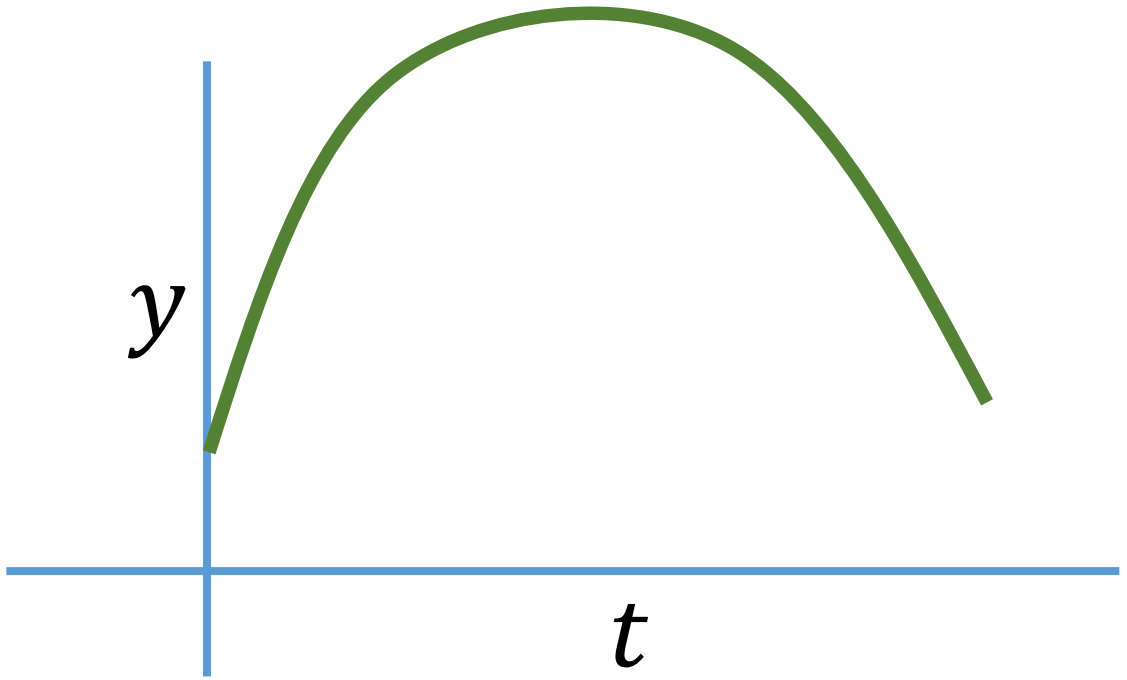




Solution: *Parametric* Curves

The parameter t increases monotonically along the curve, but x and y may increase and decrease as needed to describe *any* shape.





Parametric Curves

Note: The notion of parametric curves is not tied to any specific *type* of curve or interpolating function (like piecewise linear, cubic splines, Bezier curves, etc.)

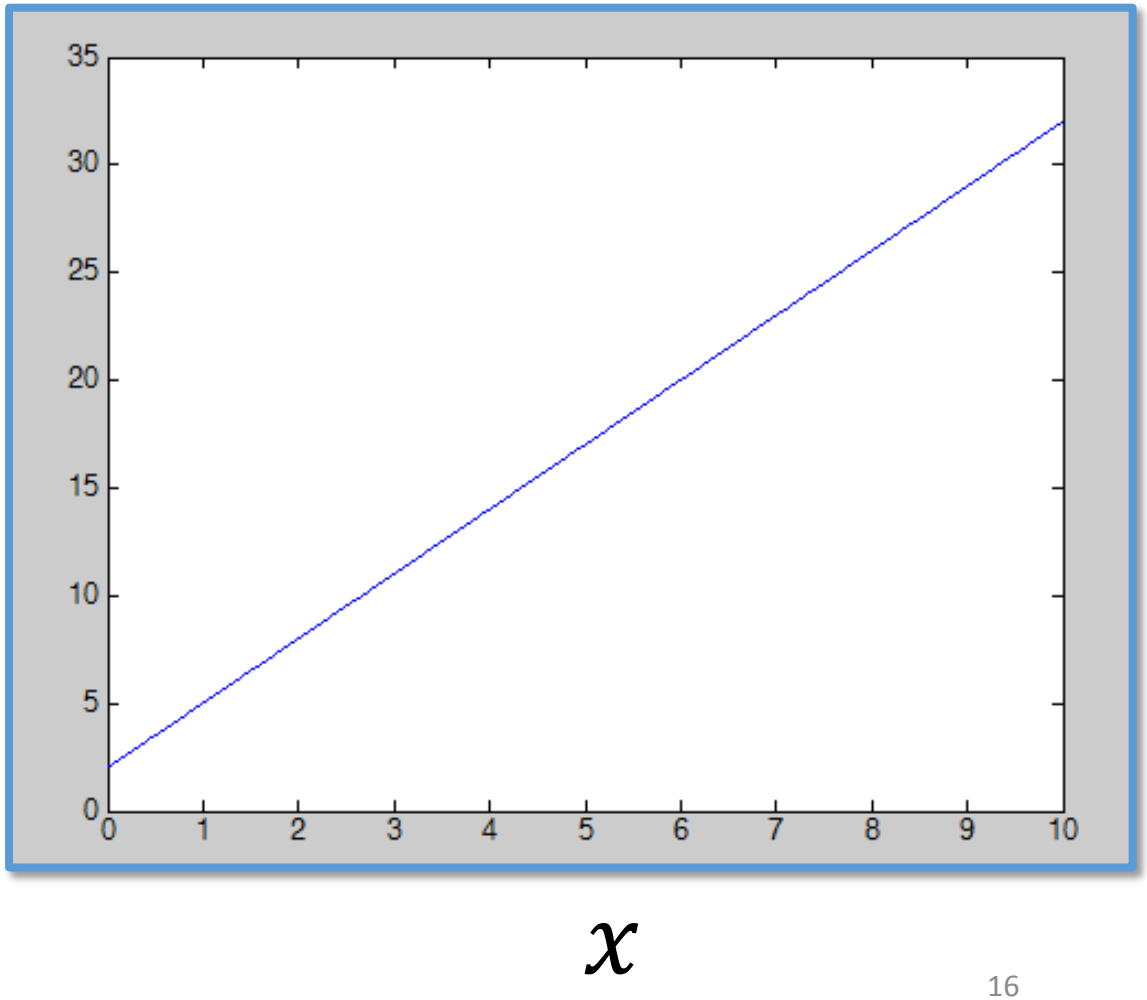
It's a more powerful/flexible way of describing curves in general.

We say that the curve is "parameterized" by t. i.e., the (x, y) position on the curve is dictated by parameter t.

Parametric Curves – Line Example

The simple line $y = 3x + 2$ can equivalently be described by the two coordinate functions:

$$x(t) = t,$$
$$y(t) = 3t + 2.$$



Parametric Curves – Semi-Circle Example

Consider a curve along a semi-circle in the upper half plane, oriented from $(1, 0)$ to $(-1, 0)$.

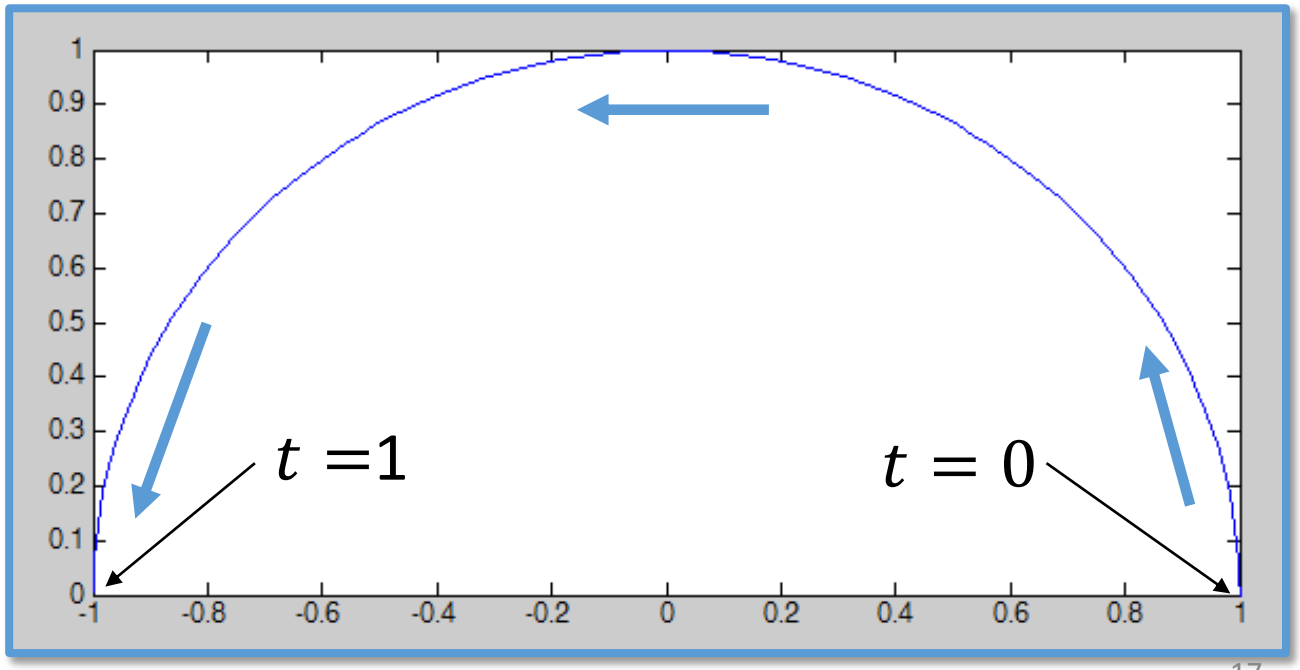
The usual *implicit* equation for a unit circle is $x^2 + y^2 = 1$.

One parametric form is:

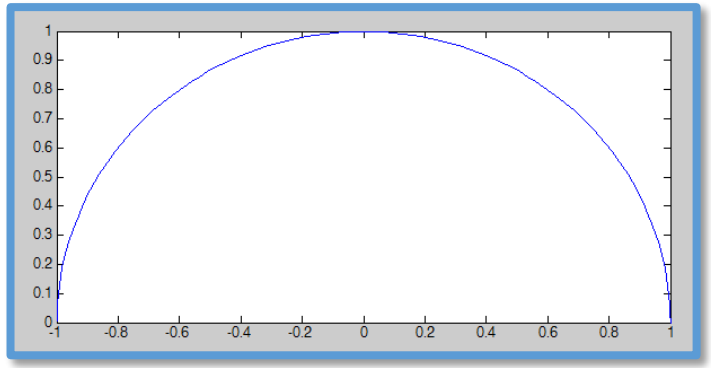
$x(t) = \cos(\pi t)$

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

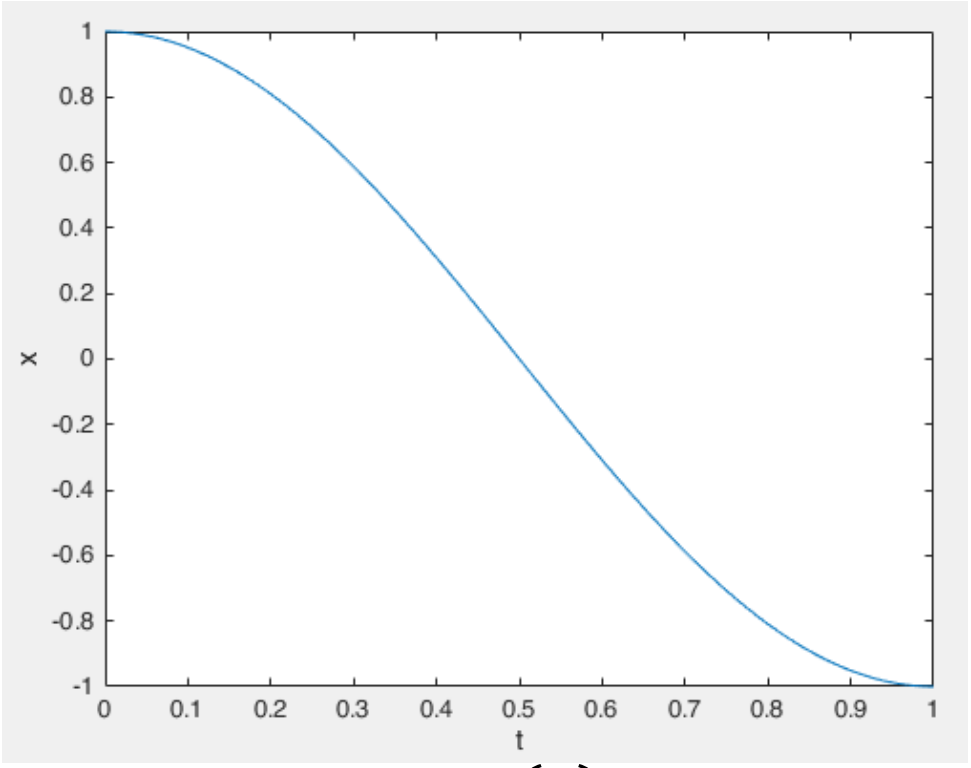
for $0 \leq t \leq 1$.



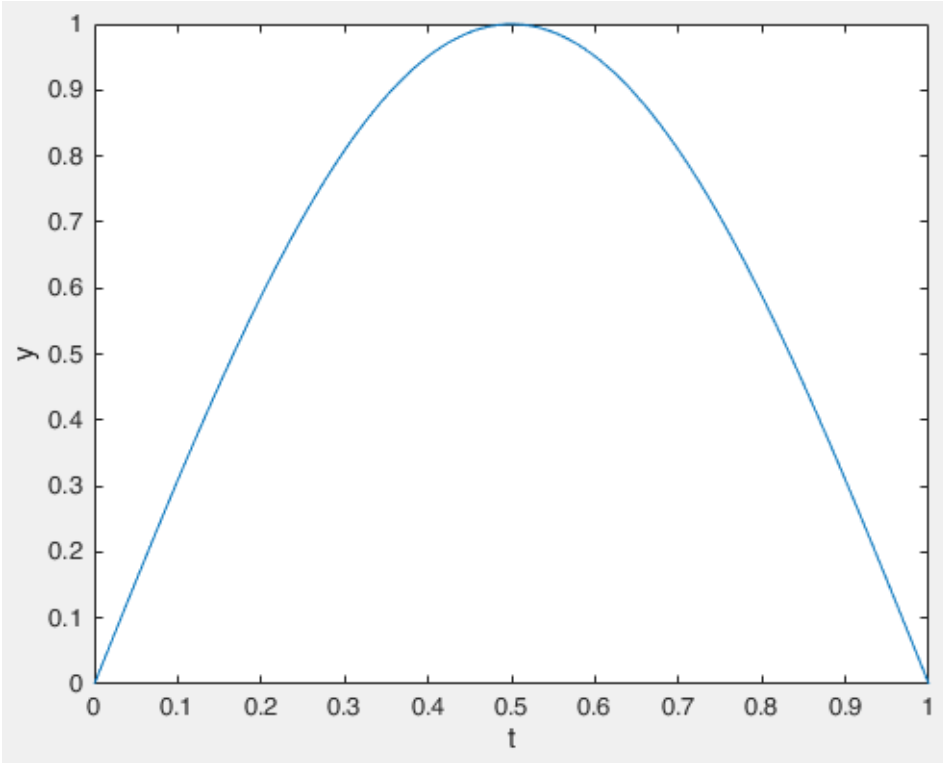
Parametric Curves – Semi-Circle



The two coordinate curves $x(t), y(t)$ are just sine and cosine functions, together creating the semi-circle. (See sample code.)



$x(t)$



$y(t)$

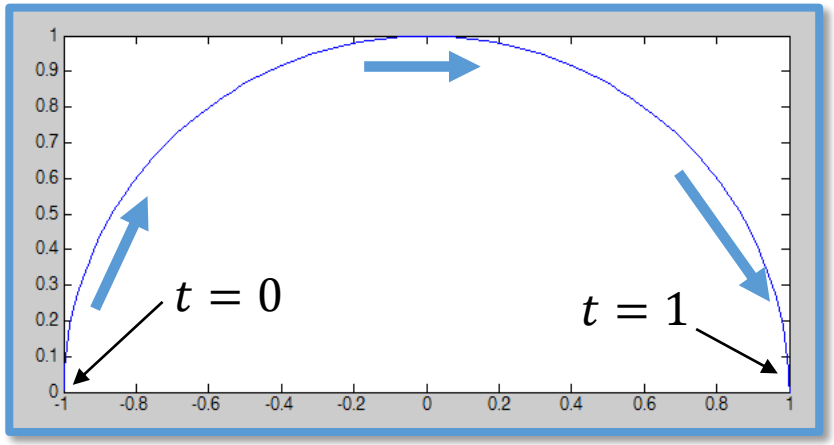
Parametric Curves – Non-Uniqueness

A given curve can be *parameterized* in different ways, while yielding the exact same curve.

1) $x(t) = \cos(\pi t)\,, y(t) = \sin(\pi t)$ for $0 \leq t \leq 1$.

2) $x(t) = \cos(\pi(1-t))\,, y(t) = \sin(\pi(1-t))$ for $0 \leq t \leq 1$.

Parameterization (2) traverses the curve in the *opposite direction* (left to right) as t goes from 0 to 1.



Parametric Curves – Speeds

2 parameterizations can also traverse the curve in the same direction, but at **different speeds/rates**.

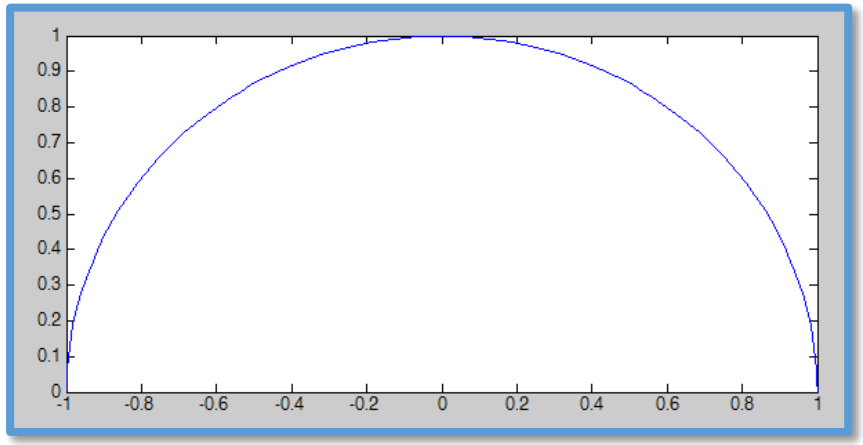
e.g.,

1) $x_1(t) = \cos(\pi t)\,, y_1(t) = \sin(\pi t)$ for $0 \le t \le 1$.

2) $x_2(t) = \cos(\pi t^2)\,, y_2(t) = \sin(\pi t^2)$ for $0 \le t \le 1$.

(2) covers the same curve, in the same direction, but returns different points for any given value of t.

e.g $x_1(0.5) = 0$ v.s., $x_2(0.5) = 1/\sqrt{2}$.



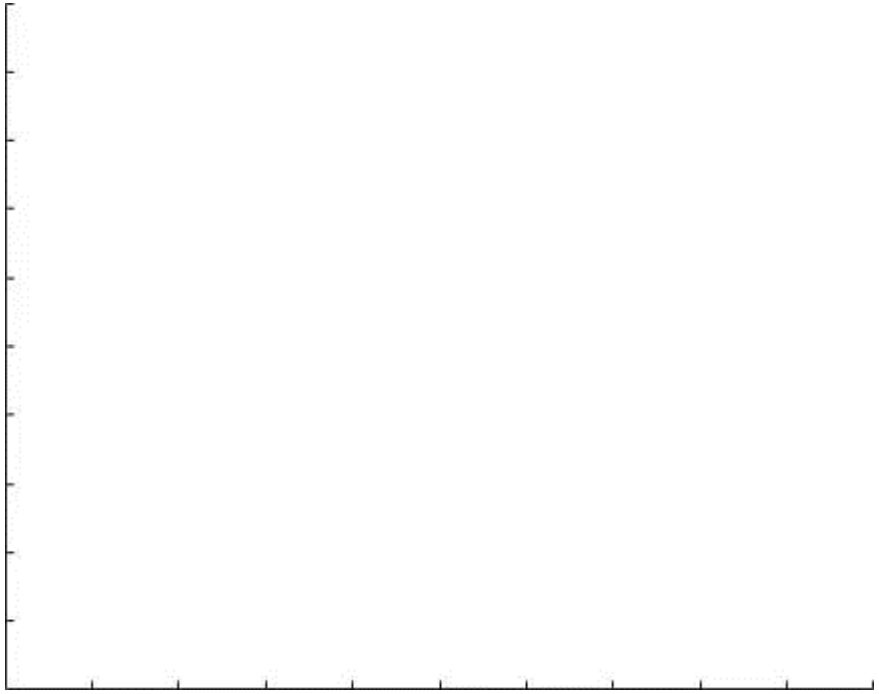
Parameterization Speeds in Semi-Circle Case

By animating the curve over time (t), we can see how the different speeds of the parameterizations behave.

1) Red = constant speed.

2) Green = irregular.

(See sample code.)

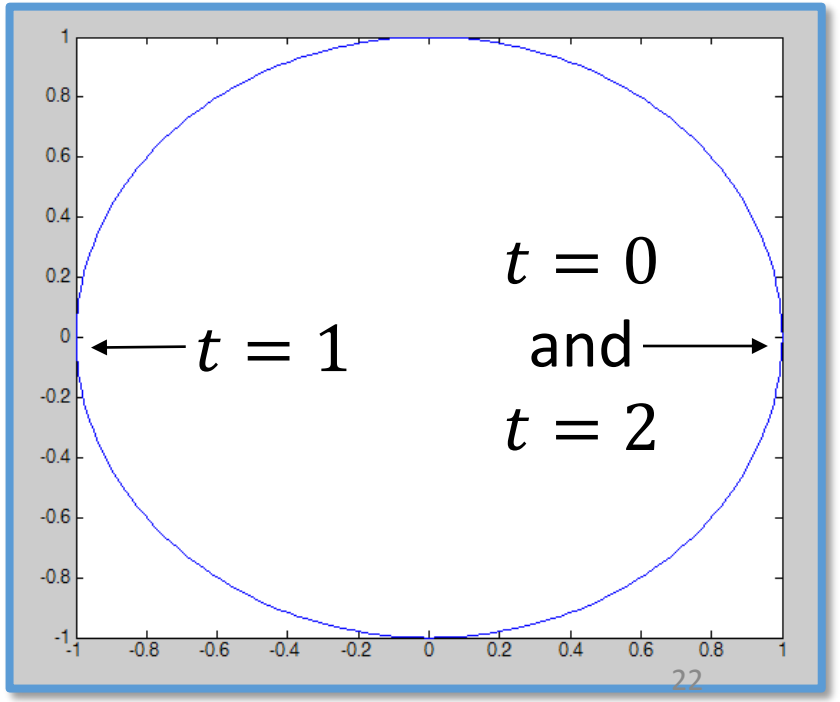


Parametric Curves – Full Circle Example

Parametric curves allow multiple points to have the same x (or y) coordinate:

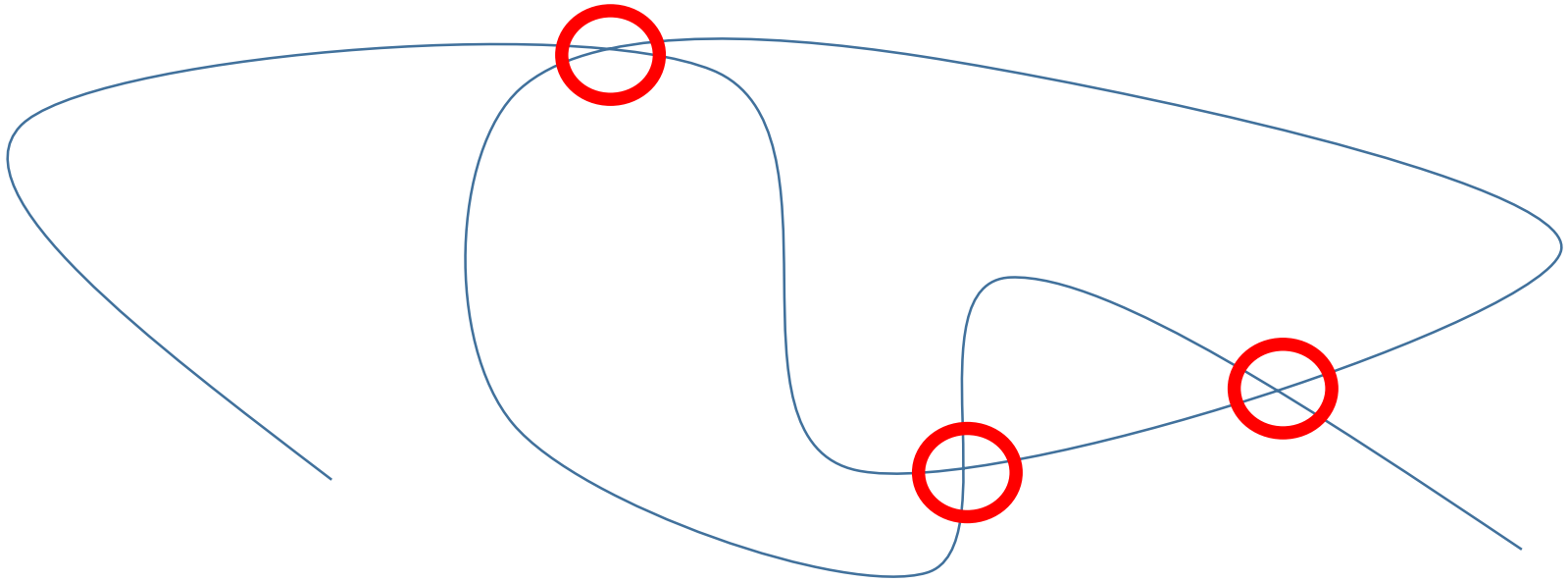
$x(t) = \cos(\pi t)\,, y(t) = \sin(\pi t)$ for $0 \leq t < 2$.

For even larger values of $t(\geq 2)$ the circle continues to wrap back over itself.



Parametric Curves – Repeated Points

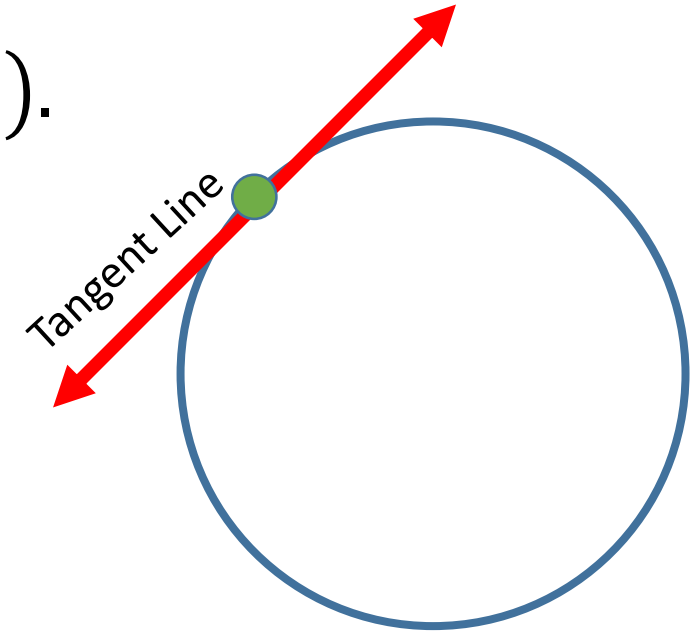
Likewise, curves with self-intersections (i.e. points where both x and y coordinates match) need no special handling.



Parametric Curves – Tangent Lines

A vector giving the direction of the *tangent line* at t $= t_0$ is the derivative of the curve's components with respect to t:

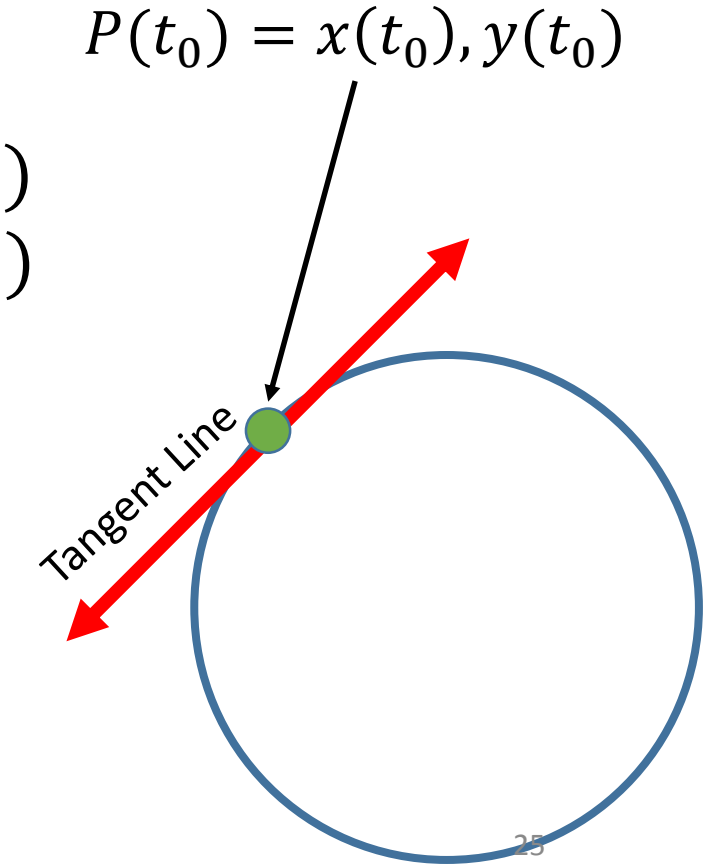
$$\frac{d\vec{P}(t_0)}{dt} = \vec{P}'(t_0) = \big(x'(t_0), y'(t_0)\big).$$



Parametric Curves – Tangent Lines

The tangent line *itself* can be expressed as a parametric curve, using another parameter s:

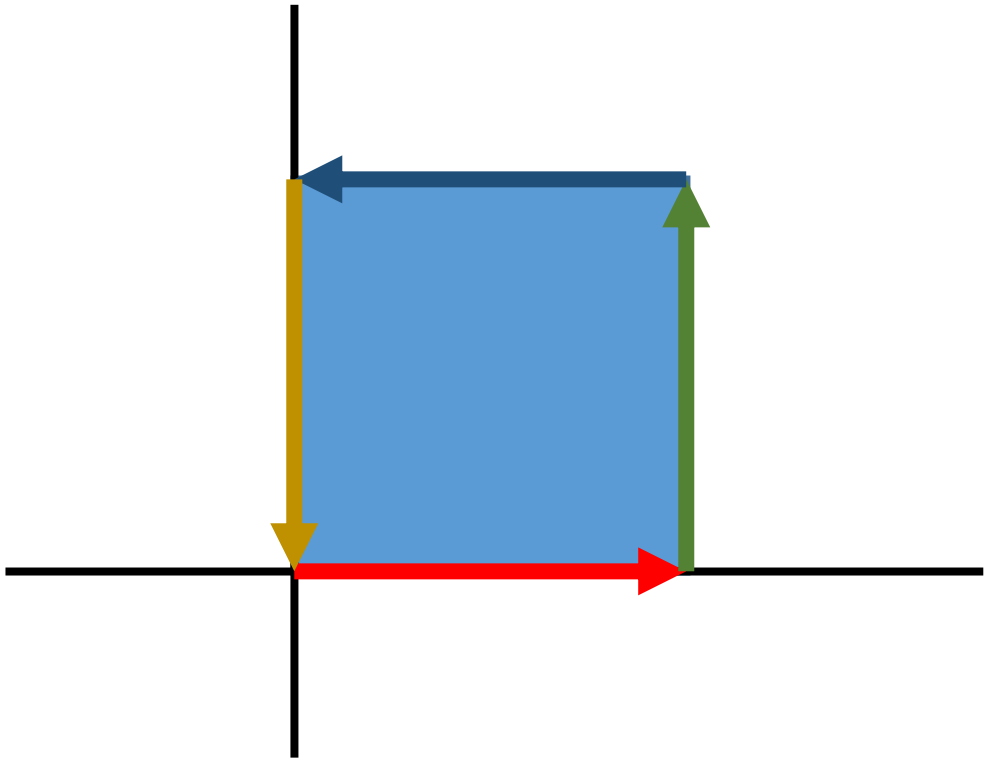
$$x_{tangent}(s) = x(t_0) + x'(t_0)(s - t_0)$$
$$y_{tangent}(s) = y(t_0) + y'(t_0)(s - t_0)$$



Piecewise Parametric Curves

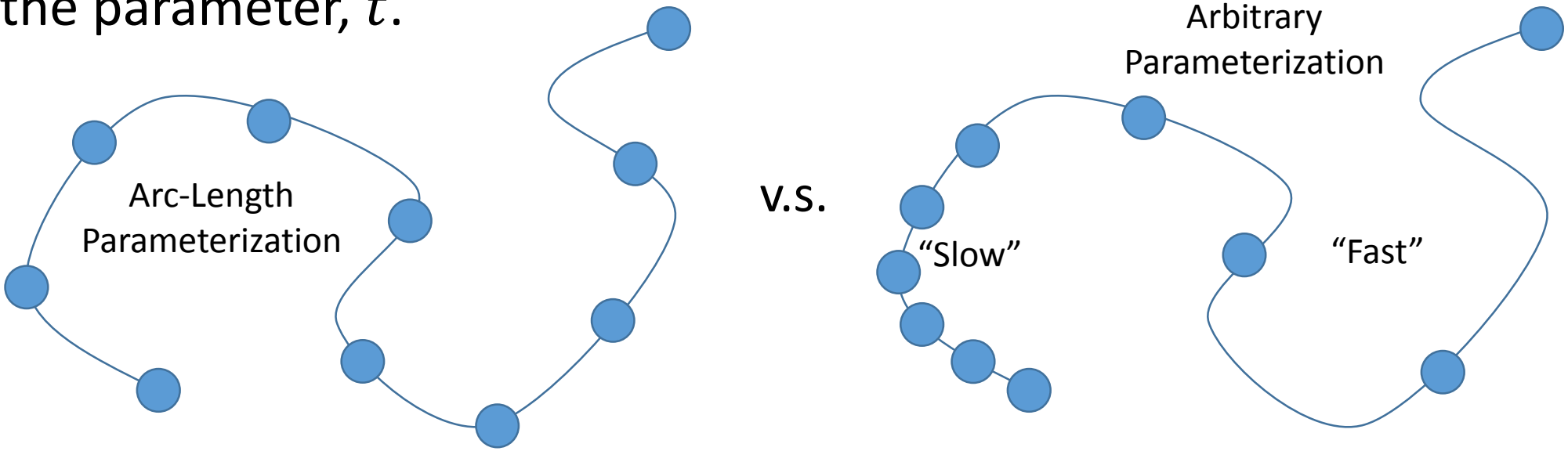
e.g. The unit square:

1. $x(t) = t, y(t) = 0, 0 \leq t \leq 1$
2. $x(t) = 1, y(t) = t - 1, 1 \leq t \leq 2$
3. $x(t) = 1 - (t - 2), y(t) = 1, 2 \leq t \leq 3$
4. $x(t) = 0, y(t) = 1 - (t - 3), 3 \leq t \leq 4$



Arc-Length Parameterization

A common parameterization is to use the *distance along the curve* as the parameter, t.



This gives a “unit speed“ traversal of the curve, i.e. travel 1 unit of distance in 1 unit of time. (Blue disks are evenly spaced in t.)

Parametric Curves using Interpolation

We can apply all our existing interpolant types with parametric curves, by considering $x(t)$ and $y(t)$, separately.

e.g.,

- Use two Lagrange polynomials, $x(t), y(t)$, to fit a small set of (t_i, x_i, y_i) point data.
- Use Hermite interpolation for $x(t), y(t)$ given $(t_i, x_i, y_i, sx_i, sy_i)$ point/slope data for many points.
- Fit separate cubic splines to $x(t), y(t)$, given many points.
- Etc!

Parameterizations for Piecewise Polynomials

Given ordered (x_i, y_i) point data, we don't yet have a parameterization.

- We need data for t, to form (t_i, x_i) and (t_i, y_i) pairs to fit curves to.

Option 1: Use the node index as the parameterization, i.e. $t_i = i$ at each node.

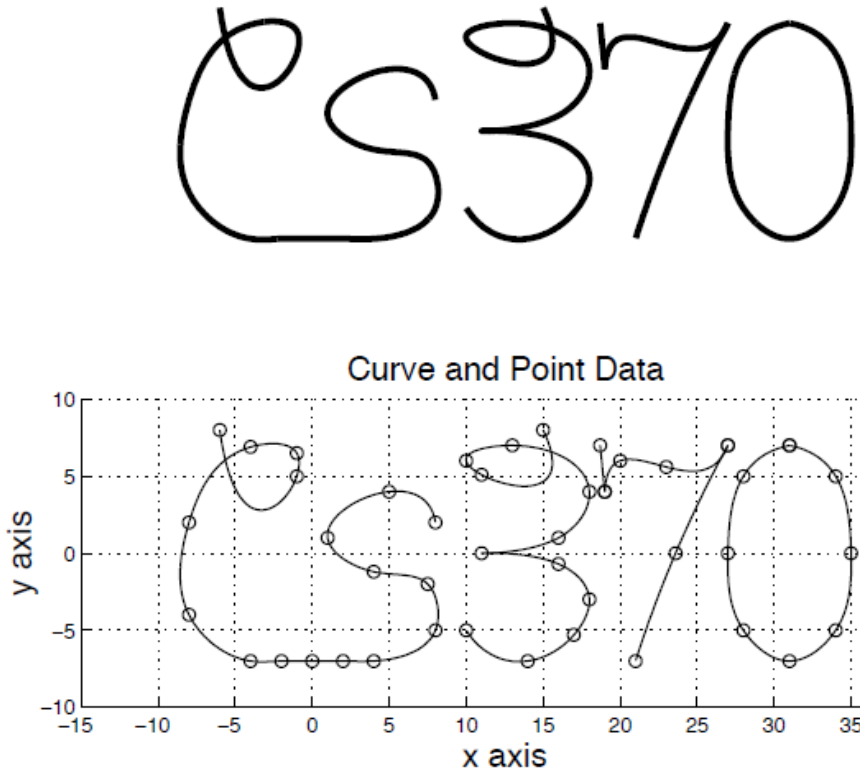
Option 2 (approximate arc-length parameterization):

- Set $t_1 = 0$ at first node.
- Recursively compute $t_{i+1} = t_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$

Parametric Curves with Cubic Splines

Assignment #1 requires you to apply Matlab's cubic spline functionality to create parametric curves for smooth shapes, using an *approximate* arc-length parameterization.

See course notes (3.1 & 3.2) for additional details.



Smooth handwriting with (parametric) cubic splines.

Matlab spline demo

- Blue line is piecewise linear plot of data.
- Red line is the cubic spline, sampled at 2 extra points in each interval.

See sample code on Piazza (General Resources).