

Module 5: Interpolation



Learning Objectives

- Lagrange interpolation
- Newton's divided differences
- Bezier spline
- Cubic B-spline interpolation
- Applications in font representations

Sources

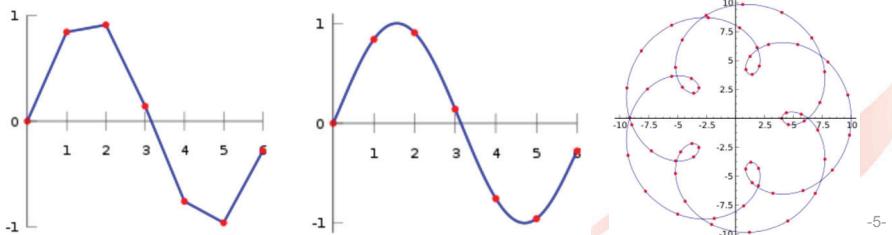
- Textbook (Chapter 3: Interpolation)
- Interpolation
<https://en.wikipedia.org/wiki/Interpolation>
- Metafont (for Tex, Latex)
<https://en.wikipedia.org/wiki/Metafont>
- TrueType (for Mac, Windows)
<https://en.wikipedia.org/wiki/TrueType>
- PostScript (for Adobe)
https://en.wikipedia.org/wiki/PostScript_fonts#Type_1

Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary

Introduction

- Problem: given a set of data pairs (t_i, P_i) , $i = 1, 2, \dots, n$, where
 - t_i are scalar numbers, called parameter values;
 - P_i are scalar or vectors;find a function $f(t)$ such that $f(t_i) = P_i$ for each $1 \leq i \leq n$. Such a function $f(t)$ is said to **interpolate** the data pairs.



Our focus

- Polynomial interpolation
 - Interpolation function is a polynomial.
 - Eg: $f(x) = -0.0001521x^6 - 0.003130x^5 + 0.07321x^4 - 0.3577x^3 + 0.2255x^2 + 0.9038x$.
- Spline interpolation
 - Interpolation function is a spline.
 - Eg:

$$f(x) = \begin{cases} -0.1522x^3 + 0.9937x, & \text{if } x \in [0, 1], \\ -0.01258x^3 - 0.4189x^2 + 1.4126x - 0.1396, & \text{if } x \in [1, 2], \\ 0.1403x^3 - 1.3359x^2 + 3.2467x - 1.3623, & \text{if } x \in [2, 3], \\ 0.1579x^3 - 1.4945x^2 + 3.7225x - 1.8381, & \text{if } x \in [3, 4], \\ 0.05375x^3 - 0.2450x^2 - 1.2756x + 4.8259, & \text{if } x \in [4, 5], \\ -0.1871x^3 + 3.3673x^2 - 19.3370x + 34.9282, & \text{if } x \in [5, 6]. \end{cases}$$

Motivation

- Interpolation is a fundamental numerical technique.
- Wide applications
 - Shipbuilding, aircraft industries (Renault, Citroen, Boeing), automobile industries (General Motors)
 - Computer typesetting
 - Digital signal processing
 - Data compression
 - ...



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

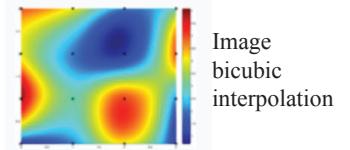
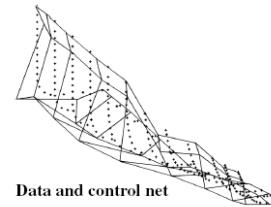
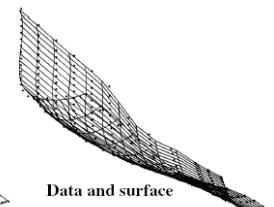


Image
bicubic
interpolation



Data and control net



Data and surface

-6-

Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

-8-

Lagrange interpolation: degree 1

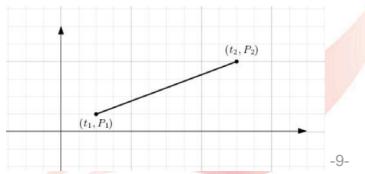
- Lagrange interpolation is an **explicit** formula, for writing down a polynomial of degree $(n-1)$ that interpolates n data points.
- Degree 1 case

Given 2 data points (t_1, P_1) and (t_2, P_2) , the polynomial

$$f_1(t) = \frac{t - t_2}{t_1 - t_2} P_1 + \frac{t - t_1}{t_2 - t_1} P_2$$

is the Lagrange interpolating polynomial.

That is, $f_1(t_1) = P_1$, $f_1(t_2) = P_2$.



Lagrange interpolation: general

- Degree $(n-1)$ case
 - For each k between 1 and n , define the degree $n-1$ polynomial:

$$L_k(t) = \frac{(t - t_1) \cdots (t - t_{k-1})(t - t_{k+1}) \cdots (t - t_n)}{(t_k - t_1) \cdots (t_k - t_{k-1})(t_k - t_{k+1}) \cdots (t_k - t_n)}.$$

- $L_k(t)$ has the following properties:
 - $L_k(t_k) = 1$ and $L_k(t_j) = 0$ for $j \neq k$.
 - $\sum_{k=1}^n L_k(t) \equiv 1$ for any t .
- Define the degree $n-1$ polynomial:

$$f_{n-1}(t) = L_1(t)P_1 + L_2(t)P_2 + \cdots + L_n(t)P_n.$$

It is the Lagrange interpolating polynomial for n data points $(t_1, P_1), (t_2, P_2), \dots, (t_n, P_n)$.

- It is easy to verify that $f_{n-1}(t_j) = P_j$.

Lagrange interpolation: degree 2

- Degree 2 case

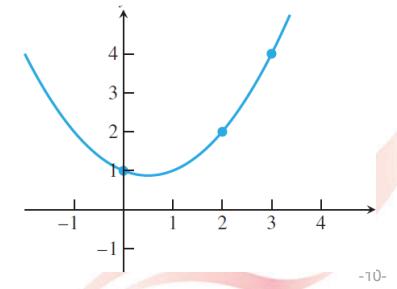
Given 3 data points $(t_1, P_1), (t_2, P_2)$ and (t_3, P_3) , the polynomial

$$f_2(t) = \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)} P_1 + \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)} P_2 + \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)} P_3$$

is the Lagrange interpolating polynomial.

That is, $f_2(t_1) = P_1$, $f_2(t_2) = P_2$, $f_2(t_3) = P_3$.

- Q for you: With reference to the figure, construct the degree 2 Lagrange interpolating polynomial for the 3 data points $(1,1), (2,2)$ and $(3,4)$.

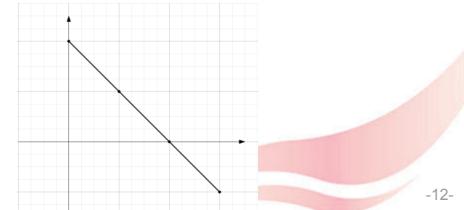


Lagrange interpolation: example

- Example: Find the polynomial of degree 3 or less that interpolates the points $(0,2), (1,1), (2,0)$ and $(3,-1)$.

We treat the first coordinate as parameter t and the second coordinate as P . Then the Lagrange polynomial is as follows:

$$\begin{aligned} f_3(t) &= \frac{(t-1)(t-2)(t-3)}{(0-1)(0-2)(0-3)} 2 + \frac{(t-0)(t-2)(t-3)}{(1-0)(1-2)(1-3)} 1 \\ &\quad + \frac{(t-0)(t-1)(t-3)}{(2-0)(2-1)(2-3)} 0 + \frac{(t-0)(t-1)(t-2)}{(3-0)(3-1)(3-2)} (-1) \\ &= \frac{1}{-3}(t^3 - 6t^2 + 11t - 6) + \frac{1}{2}(t^3 - 5t^2 + 6t) + \frac{1}{-6}(t^3 - 3t^2 + 2t) \\ &= -t + 2 \end{aligned}$$



Discussions

- Uniqueness (Main Theorem of Polynomial Interpolation)

Theorem. Let $(t_1, P_1), \dots, (t_n, P_n)$ be n data pairs with *distinct* t_i . Then there exists one and only one polynomial $f(t)$ of degree $n - 1$ or less that satisfies $f(t_i) = P_i$ for $i = 1, \dots, n$.

- Advantage of Lagrange interpolation

- explicit, constructive, intuitive

- Disadvantage of Lagrange interpolation

- computationally complex
- not progressive



-13-

Outline

- §1. Introduction

- §2. Lagrange interpolation

- §3. Newton's divided differences

- §4. Bezier splines

- §5. Cubic B-spline interpolation

- §6. Applications in font representation

- §7. Homework

- §8. Summary



-14-

Newton's divided differences

- Newton's divided differences provide a *more manageable* and *progressive* form for polynomial interpolation.

- Degree 0 case

Given 1 data pair (t_1, P_1) , we let $f_0(t) = P_1$.

$f_0(t)$ is a constant function satisfying $f_0(t_1) = P_1$.

- Degree 1 case

Given 2 data points (t_1, P_1) and (t_2, P_2) , we let

$$f_1(t) = P_1 + \frac{P_2 - P_1}{t_2 - t_1}(t - t_1).$$

Then, $f_1(t_1) = P_1$, $f_1(t_2) = P_2$.

-15-

Newton's divided differences

- Degree 2 case

Given 3 data points (t_1, P_1) , (t_2, P_2) and (t_3, P_3) , we let

$$f_2(t) = P_1 + \frac{P_2 - P_1}{t_2 - t_1}(t - t_1) + \frac{\frac{P_3 - P_2}{t_3 - t_2} - \frac{P_2 - P_1}{t_2 - t_1}}{t_3 - t_1}(t - t_1)(t - t_2).$$

Then, $f_2(t_1) = P_1$, $f_2(t_2) = P_2$, $f_2(t_3) = P_3$.

Proof. Expand $f_2(t)$ and check the coefficients of P_1 , P_2 and P_3 :

$$P_3 : \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)}$$

$$P_2 : \frac{t-t_1}{t_2-t_1} - \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)} - \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_2-t_1)} = \frac{(t-t_1)(t-t_3)}{(t_2-t_1)(t_2-t_3)}$$

$$P_1 : 1 - \frac{t-t_1}{t_2-t_1} + \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_2-t_1)} = \frac{(t-t_2)(t-t_3)}{(t_1-t_2)(t_1-t_3)}$$

-16-

Newton's divided differences

- General

Introduce the divided differences:

$$\begin{aligned} f[t_k] &= P_k \\ f[t_k, t_{k+1}] &= \frac{f[t_{k+1}] - f[t_k]}{t_{k+1} - t_k} \\ f[t_k, t_{k+1}, t_{k+2}] &= \frac{f[t_{k+1}, t_{k+2}] - f[t_k, t_{k+1}]}{t_{k+2} - t_k} \\ \text{and so on.} \end{aligned}$$

The Newton's divided difference formula is

$$\begin{aligned} f_{n-1}(t) &= f[t_1] + f[t_1, t_2](t - t_1) \\ &+ f[t_1, t_2, t_3](t - t_1)(t - t_2) \\ &+ f[t_1, t_2, t_3, t_4](t - t_1)(t - t_2)(t - t_3) \\ &+ \dots \\ &+ f[t_1, t_2, \dots, t_n](t - t_1)(t - t_2) \cdots (t - t_{n-1}). \end{aligned}$$

-17-

Newton's divided differences

- Newton form

Let $N_i(t) = \prod_{j=1}^{i-1} (t - t_j)$, $i = 1, \dots, n$, which we call the *Newton basis polynomials*.

Then the interpolating polynomial of degree $n-1$ or less in Newton form is

$$f_{n-1}(t) = \sum_{i=1}^n B_i N_i(t)$$

where the coefficients $B_i = f[t_1, \dots, t_i]$ are the $(i-1)$ -th order *divided differences* of the interpolation data P_i with respect to the parameter values t_i , which result from the recursive definition:

$$\begin{aligned} f[t_i] &= P_i, \\ f[t_i, \dots, t_{i+j}] &= \frac{f[t_{i+1}, \dots, t_{i+j}] - f[t_i, \dots, t_{i+j-1}]}{t_{i+j} - t_i}, \\ i &= 1, \dots, n-j; \quad j = 1, \dots, n-1 \end{aligned}$$

-18-

Newton's divided differences

- Diagram for computing the divided differences

t_1	$f[t_1]$				
t_2	$f[t_2]$	$f[t_1, t_2]$			
t_3	$f[t_3]$	$f[t_2, t_3]$	$f[t_1, t_2, t_3]$		
\vdots	\vdots	\vdots	\ddots		
t_n	$f[t_n]$	$f[t_{n-1}, t_n]$	$f[t_{n-2}, t_{n-1}, t_n]$	\cdots	$f[t_1, \dots, t_n]$

- Algorithm

Once the coefficients B_i have been obtained, the evaluation of the interpolating polynomial at t can be done with $O(n)$ operations via a Horner-like scheme:

```

 $P := B_n$ 
for  $i$  from  $(n-1)$  to 1 by  $(-1)$  do
   $P := (t - t_i)P + B_i$ 
return  $P$ .

```

-19-

Therefore, the degree 2 interpolating polynomial is

$$f_2(x) = 1 + \frac{1}{2}x + \frac{1}{2}x(x-2) = \frac{1}{2}x^2 - \frac{1}{2}x + 1.$$

-20-

Interpolation error

Theorem. Assume that $g(t)$ is a C^n continuous function and $f_{n-1}(t)$ is the (degree $n - 1$ or less) interpolating polynomial passing through the n points $(t_1, g(t_1)), \dots, (t_n, g(t_n))$. The interpolation error is

$$g(t) - f_{n-1}(t) = \frac{(t - t_1)(t - t_2) \cdots (t - t_n)}{n!} g^{(n)}(c)$$

where c lies between the smallest and largest of the numbers t, t_1, \dots, t_n .

Outline of the proof:

- For any u , construct the degree n or less polynomial interpolating $(t_1, g(t_1)), \dots, (t_n, g(t_n))$, and $(u, g(u))$:

$$f_n(t) = f_{n-1}(t) + g[t_1, \dots, t_n, u](t - t_1) \cdots (t - t_n).$$

$$\text{Hence } g(u) = f_n(u) = f_{n-1}(u) + g[t_1, \dots, t_n, u](u - t_1) \cdots (u - t_n).$$

- $f_n(t)$ matches $g(t)$ at at least $n + 1$ points. Therefore there exists a c such that $f_n^{(n)}(c) = g^{(n)}(c)$.
- $g^{(n)}(c) = g[t_1, \dots, t_n, u]n!$

Outline

-
- §1. Introduction
 - §2. Lagrange interpolation
 - §3. Newton's divided differences
 - §4. Bezier splines
 - §5. Cubic B-spline interpolation
 - §6. Applications in font representation
 - §7. Homework
 - §8. Summary



-22-

Background

- Designing complex shapes with a *single* polynomial or Bezier curve is not recommended. This is because it would require choosing a large degree n , which is bad for 2 reasons.
 - the $O(n^2)$ evaluation with the de Casteljau algorithm becomes slow;
 - the local control property is not very effective for large n .
- A common approach is to fit several low-degree Bezier curves together to form a *Bezier spline*. This is actually what common programs like *Adobe Illustrator* and *PowerPoint* do, and what is used for the definition of *TrueType* fonts.



-23-

Bezier splines

-
- Construction of geometrically complex curves
 - Bezier splines
 - Join several Bezier curves of low degrees
 - Guarantee continuous join from one curve to the next



-24-

Bezier curve revisited

- A degree n Bezier curve with control points P_0, \dots, P_n is defined by

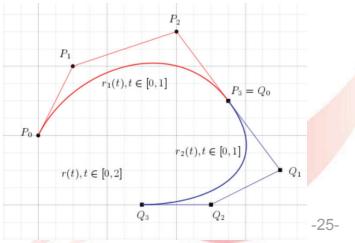
$$r(t) = \sum_{i=0}^n P_i B_i^n(t), \quad t \in [0, 1]$$

- A degree n Bezier curve with control points P_0, \dots, P_n over $[a, b]$ is

$$r(t) = \sum_{i=0}^n P_i B_i^n \left(\frac{t-a}{b-a} \right), \quad t \in [a, b]$$

- Two Bezier curves $r_1(t) = \sum_{i=0}^n P_i B_i^n(t)$ and $r_2(t) = \sum_{i=0}^n Q_i B_i^n(t)$, $t \in [0, 1]$ can be combined into a Bezier spline $r(t)$, $t \in [0, 2]$:

$$r(t) = \begin{cases} \sum_{i=0}^n P_i B_i^n(t), & t \in [0, 1] \\ \sum_{i=0}^n Q_i B_i^n(t-1), & t \in [1, 2] \end{cases}$$



Strategies for tangents

- Choice of tangents T_i ($i = 1, 2, \dots, n-1$)

– FMILL tangents: $T_i = \frac{Q_{i+1} - Q_{i-1}}{2}$

This guarantees that the Bezier spline is C^1 continuous.

This Bezier spline is also called **Catmull-Rom spline**.

- Conditions at the end points Q_0 and Q_n .

– Natural end condition:

$$C''_0(0) = C''_{n-1}(1) = 0$$

which gives $P_{01} = \frac{Q_0 + P_{02}}{2}$ and $P_{(n-1)2} = \frac{P_{(n-1)1} + Q_n}{2}$.

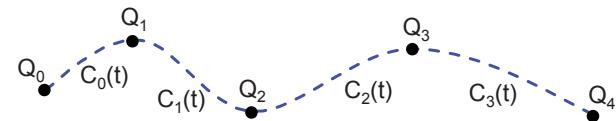
– Simple endpoint condition:

$$C'_0(0) = C'_{n-1}(1) = 0, \quad \text{giving } P_{01} = Q_0, \quad P_{(n-1)2} = Q_n.$$

-27-

C^1 cubic Bezier splines

Problem: Given a sequence of points Q_0, Q_1, \dots, Q_n , we construct n cubic Bezier curves. The two adjacent curves are connected continuously. The i^{th} Bezier curve $C_i(t)$ is the segment from Q_i to Q_{i+1} .

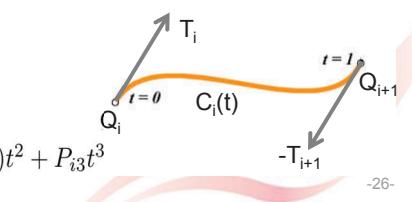


Construction: To define the i^{th} Bezier curve $C_i(t)$, we have to find the four Bezier points P_{i0}, P_{i1}, P_{i2} , and P_{i3} . We assign two tangent vectors T_i and T_{i+1} to P_{i0} and P_{i3} , respectively. Then

$$P_{i0} = Q_i, \quad P_{i1} = Q_i + \frac{1}{3}T_i$$

$$P_{i3} = Q_{i+1}, \quad P_{i2} = Q_{i+1} - \frac{1}{3}T_{i+1}$$

$$C_i(t) = P_{i0}(1-t)^3 + P_{i1}3(1-t)^2t + P_{i2}3(1-t)t^2 + P_{i3}t^3$$



-26-

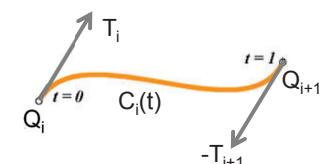
Extension of Catmull-Rom spline

Construction: To define the i^{th} Bezier curve $C_i(t)$, we have to find the four Bezier points P_{i0}, P_{i1}, P_{i2} , and P_{i3} . We assign two tangent vectors T_i and T_{i+1} to P_{i0} and P_{i3} , respectively. Then

$$P_{i0} = Q_i, \quad P_{i1} = Q_i + \frac{1}{3}\lambda_i \frac{T_i}{|T_i|}$$

$$P_{i3} = Q_{i+1}, \quad P_{i2} = Q_{i+1} - \frac{1}{3}\lambda_i \frac{T_{i+1}}{|T_{i+1}|}$$

with $\lambda_i = |Q_{i+1} - Q_i|$.



Then the Bezier curve is expressed as:

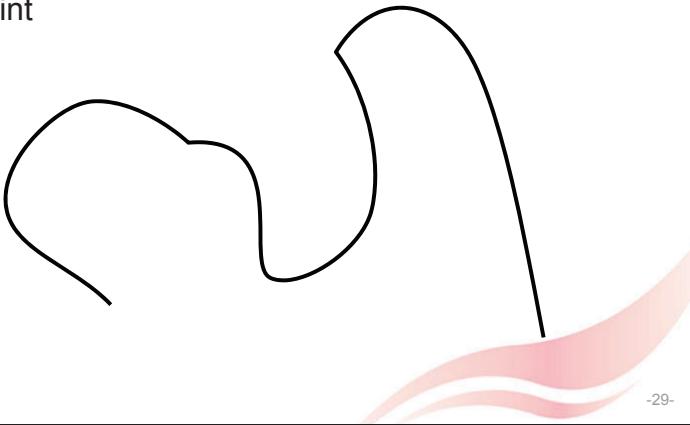
$$C_i(t) = P_{i0}(1-t)^3 + P_{i1}3(1-t)^2t + P_{i2}3(1-t)t^2 + P_{i3}t^3$$

Q for you: Are $C_i(t)$ and $C_{i+1}(t)$ now still C^1 continuous? Are the two adjacent Bezier curve segments connected smoothly?

-28-

Example: freeform curve design

- PowerPoint or MS-Word: AutoShapes
 - Smooth point
 - Straight point
 - Corner point

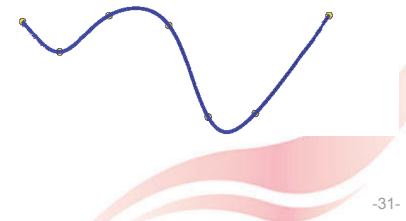


Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary

C² cubic B-spline interpolation

- Problem: Given n+1 data points $\{D_i \mid i=0, 1, \dots, n\}$, find a cubic B-spline curve that interpolates them.
- This involves two basic steps:
 - Step 1. Parameterize the data points: assign appropriate parameter values to data point D_i
 - Step 2. find the B-spline representation.



Step 1. Parameterization

- **Uniform parameterization:** If there are n+1 data points $\{D_i\}$, $i=0, \dots, n$, then n+1 evenly spaced parameter values t_i within [0, 1] are chosen for $\{D_i\}$. That is,

$$\begin{cases} t_0 &= 0 \\ t_i &= i/n \quad \text{for } 1 \leq i \leq n-1 \\ t_n &= 1 \end{cases}$$



Uniform parameterization may result in unwanted wiggles, which is known as *Runge phenomenon* (refer to Chapter 3.2.3).

- **Chord length parameterization:** The parameter values are chosen based on the spatial distribution of the data points.

$$\begin{cases} t_0 &= 0 \\ t_i &= \frac{\sum_{k=1}^i |D_k - D_{k-1}|}{\sum_{k=1}^n |D_k - D_{k-1}|} \quad \text{for } 1 \leq i \leq n-1 \\ t_n &= 1 \end{cases}$$



Step 2. Find the B-spline curve

- Once the parameterization for the data points is done, we now need to determine the B-spline curve, which requires to specify the *degree*, *knots*, and the *de Boor points*.
- The idea is to create a cubic B-spline curve $r(t)$ with n segments whose parameter intervals are $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{n-1}, t_n]$, respectively, satisfying $r(t_i) = D_i$.
 - Set degree = 3
 - Set knots = $\{t_0, t_0, t_0, t_0, t_1, t_2, \dots, t_{n-2}, t_{n-1}, t_n, t_n, t_n, t_n\}$
 - What remains is to determine the $(n+3)$ de Boor points P_0, P_1, \dots, P_{n+2} . This is done by building a set of equations.

-33-

Cubic B-spline basis functions revisited

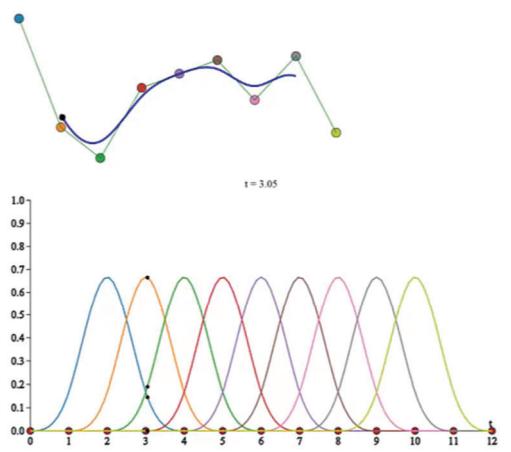
- Cubic B-spline basis

$$N_i^3(u) = \begin{cases} \frac{(u-u_i)^3}{(u_{i+1}-u_i)(u_{i+2}-u_i)(u_{i+3}-u_i)}, & u \in [u_i, u_{i+1}) \\ \frac{(u-u_i)^2(u_{i+2}-u)}{(u_{i+2}-u_{i+1})(u_{i+3}-u_i)(u-u_i)(u-u_{i+1})} + \frac{(u_{i+2}-u_{i+1})(u_{i+3}-u_{i+1})(u_{i+3}-u_i)}{(u_{i+2}-u_{i+1})(u_{i+4}-u_{i+1})(u_{i+3}-u_{i+1})}, & u \in [u_{i+1}, u_{i+2}) \\ \frac{(u-u_i)(u_{i+3}-u)^2}{(u_{i+3}-u_{i+2})(u_{i+3}-u_{i+1})(u_{i+3}-u_i)} + \frac{(u_{i+4}-u)(u-u_{i+1})^2}{(u_{i+3}-u_{i+2})(u_{i+4}-u_{i+2})(u_{i+4}-u_{i+1})}, & u \in [u_{i+2}, u_{i+3}) \\ \frac{(u_{i+4}-u)^3}{(u_{i+4}-u_{i+3})(u_{i+4}-u_{i+2})(u_{i+4}-u_{i+1})}, & u \in [u_{i+3}, u_{i+4}) \end{cases}$$

-34-

Cubic B-spline basis functions revisited

At knot u_i , only 3 basis functions $N_{i-3}^3(u), N_{i-2}^3(u), N_{i-1}^3(u)$ do not vanish.



-35-

Step 2. Find the B-spline curve (cont)

- The interpolation requirement gives $n+1$ equations:

For $k = 0, 1, \dots, n$,

$$r(t_k) = \sum_{i=0}^{n+2} P_i N_i^3(t_k) = D_k, \text{ which is}$$

$$N_k^3(t_k)P_k + N_{k+1}^3(t_k)P_{k+1} + N_{k+2}^3(t_k)P_{k+2} = D_k$$

- Endpoint conditions

– Natural conditions: $r''(u)=0$ at t_0 ($=u_3$) and t_n ($=u_{n+3}$).

This gives another two equations:

$$\frac{d^2 N_0^3(t_0)}{du^2} P_0 + \frac{d^2 N_1^3(t_0)}{du^2} P_1 + \frac{d^2 N_2^3(t_0)}{du^2} P_2 = 0$$

$$\frac{d^2 N_n^3(t_n)}{du^2} P_n + \frac{d^2 N_{n+1}^3(t_n)}{du^2} P_{n+1} + \frac{d^2 N_{n+2}^3(t_n)}{du^2} P_{n+2} = 0$$

-36-

Step 2. Find the B-spline curve (cont)

- Thus we obtain $(n+3)$ linear equations in $(n+3)$ variables P_0, P_1, \dots, P_{n+2} :

$$\begin{bmatrix} \frac{d^2 N_0^3(t_0)}{du^2} & \frac{d^2 N_1^3(t_0)}{du^2} & \frac{d^2 N_2^3(t_0)}{du^2} & \cdots & 0 & 0 & 0 \\ N_0^3(t_0) & N_1^3(t_0) & N_2^3(t_0) & \cdots & 0 & 0 & 0 \\ 0 & N_1^3(t_1) & N_2^3(t_1) & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & N_n^3(t_{n-1}) & N_{n+1}^3(t_{n-1}) & 0 \\ 0 & 0 & 0 & \cdots & N_n^3(t_n) & N_{n+1}^3(t_n) & N_{n+2}^3(t_n) \\ 0 & 0 & 0 & \cdots & \frac{d^2 N_n^3(t_n)}{du^2} & \frac{d^2 N_{n+1}^3(t_n)}{du^2} & \frac{d^2 N_{n+2}^3(t_n)}{du^2} \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_n \\ P_{n+1} \\ P_{n+2} \end{bmatrix} = \begin{bmatrix} 0 \\ D_0 \\ D_1 \\ \vdots \\ D_{n-1} \\ 0 \end{bmatrix}$$

-37-

Step 2. Find the B-spline curve (cont)

- Since we choose multiple knots at t_0 and t_n , the first and last control points P_0, P_{n+2} interpolate D_0, D_{n+2} . It is easy to verify that $N_0^3(t_0) = N_{n+2}^3(t_n) = 1, N_1^3(t_0) = N_2^3(t_0) = N_n^3(t_n) = N_{n+1}^3(t_n) = 0$. Swapping the 1st and 2nd rows, the (n+2)-th and (n+3)-th rows gives

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \frac{d^2 N_0^3(t_0)}{du^2} & \frac{d^2 N_1^3(t_0)}{du^2} & \frac{d^2 N_2^3(t_0)}{du^2} & \cdots & 0 & 0 & 0 \\ 0 & N_1^3(t_1) & N_2^3(t_1) & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & N_n^3(t_{n-1}) & N_{n+1}^3(t_{n-1}) & 0 \\ 0 & 0 & 0 & \cdots & \frac{d^2 N_n^3(t_n)}{du^2} & \frac{d^2 N_{n+1}^3(t_n)}{du^2} & \frac{d^2 N_{n+2}^3(t_n)}{du^2} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_n \\ P_{n+1} \\ P_{n+2} \end{bmatrix} = \begin{bmatrix} D_0 \\ 0 \\ D_1 \\ \vdots \\ D_{n-1} \\ 0 \\ D_n \end{bmatrix}$$

- Note that the coefficient matrix is a tri-diagonal matrix. This linear system can be solved very efficiently (see Module 2. Linear Systems or refer to https://en.wikipedia.org/wiki/Tridiagonal_matrix).

-38-

Algorithm

Input: $n+1$ data points D_0, \dots, D_n

Output: A cubic B-spline curve that goes through all D_i in the given order

Algorithm:

```

computing a set of  $n+1$  parameters  $t_0, \dots, t_n$ ;
set knot vector  $T$ ;
for  $i := 0$  to  $n$  do
    for  $j := 0$  to  $n+2$  do
        Evaluate  $N_j^3(t_i)$  into row  $i+2$  and column  $j+1$  of matrix [N];
    for  $i := 0$  to  $n$  do
        Place data point  $D_i$  on row  $i+2$  of matrix [D];
    add the boundary conditions into the second row and the (last-1) row
Use an efficient linear system solver to solve for [P] from  $[N] \cdot [P] = [D]$ .
Row  $i$  of [P] is control point  $P_{i-1}$ ; Control points  $P_0, \dots, P_{n+2}$ , and knot vector T
determine an interpolating cubic B-spline curve.
```

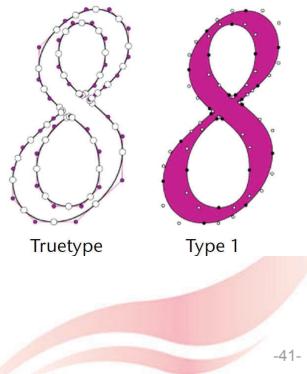
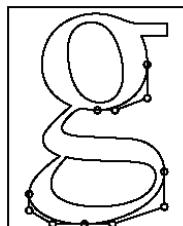
-39-

Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary

Applications in font definition

- A common use for Bezier curves is in font definition.
 - Type 1: cubic Bezier curves (PostScript, Adobe)
 - TrueType: quadratic Bezier curves (Mac, Windows)
 - MetaFont: cubic Bezier curves (D Knuth, Tex)
- Adobe illustrator
 - Cubic Bezier curves
- PowerPoint
 - Cubic Bezier curves



-41-

Fonts from Bezier curves

- Modern fonts are built directly from Bezier curves, which were a fundamental part of the PostScript language from its start in the 1980s.
- The PostScript commands have been migrated to the PDF format.



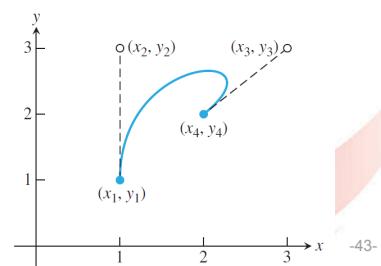
-42-

A complete PDF file

```
%PDF-1.7
1 0 obj
<<
/Length 2 0 R
>>
stream
<0 300 300 300 200 200 c
S
endstream
endobj
2 0 obj
1000
endobj
4 0 obj
<<
/Type /Page
/Parent 5 0 R
/Contents 1 0 R
>>
endobj
5 0 obj
<<
/Kids [ 4 0 R ]
/Count 1
/Type /Pages
/MediaBox [ 0 0 612 792 ]
>>
endobj
```

The lines between **stream** and **endstream** identify the Bezier curve.

- m: set the current point
- c: accept another 3 points for a curve
- S: draw the curve

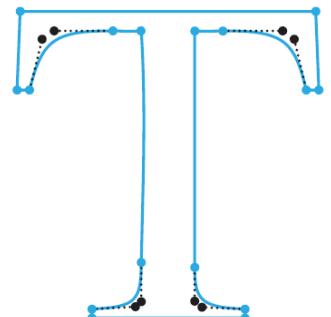


-43-

Example: Times-Roman T made with Bezier splines

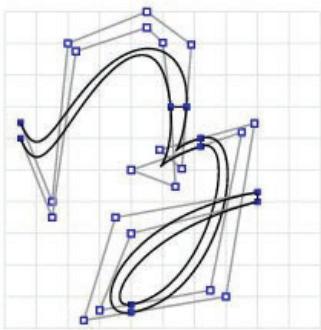
A Bezier spline consists of 16 Bezier curves.

```
237 620 237 620 237 120 237 120;
237 120 237 35 226 24 143 19;
143 19 143 19 143 0 143 0;
143 0 143 0 435 0 435 0;
435 0 435 0 435 19 435 19;
435 19 353 23 339 36 339 109;
339 109 339 108 339 620 339 620;
339 620 339 620 393 620 393 620;
393 620 507 620 529 602 552 492;
552 492 552 492 576 492 576 492;
576 492 576 492 570 662 570 662;
570 662 570 662 6 662 6 662;
6 662 6 662 0 492 0 492;
0 492 0 492 24 492 24 492;
24 492 48 602 71 620 183 620;
183 620 183 620 237 620 237 620;
```



To create a PDF file, one needs to add commands:
237 620 m
237 620 237 120 237 120 c
...
-44-

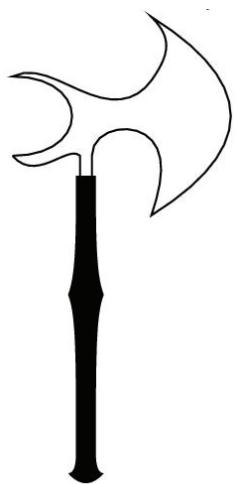
Example: font design



-45-



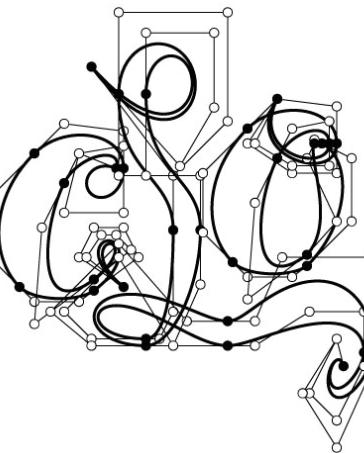
Example: font design



-47-



Example: font design



-46-

Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary



-48-

Homework

If we define the polygon (blue) with vertices

$$P_0 = \begin{pmatrix} 200 \\ 200 \end{pmatrix},$$

$$P_1 = \begin{pmatrix} 200 \\ 500 \end{pmatrix},$$

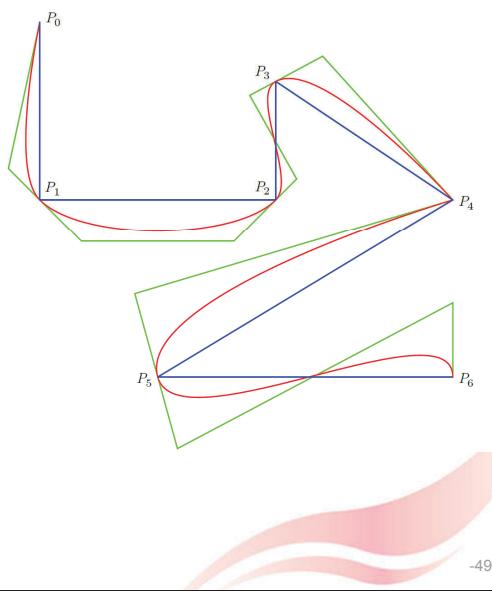
$$P_2 = \begin{pmatrix} 600 \\ 500 \end{pmatrix},$$

$$P_3 = \begin{pmatrix} 600 \\ 300 \end{pmatrix},$$

$$P_4 = \begin{pmatrix} 900 \\ 500 \end{pmatrix},$$

$$P_5 = \begin{pmatrix} 400 \\ 800 \end{pmatrix},$$

$$P_6 = \begin{pmatrix} 900 \\ 800 \end{pmatrix}$$



-49-



Homework (cont)

in *Adobe Illustrator* and convert the points P_1, P_2, P_3, P_5, P_6 from *corner* to *smooth*, then we get a cubic Bézier spline (red) with control points (green)

$$\begin{array}{llll} C_{0,0} = \begin{pmatrix} 200 \\ 200 \end{pmatrix}, & C_{0,1} = \begin{pmatrix} 200 \\ 200 \end{pmatrix}, & C_{0,2} = \begin{pmatrix} 146.97 \\ 446.97 \end{pmatrix}, & C_{0,3} = \begin{pmatrix} 200 \\ 500 \end{pmatrix}, \\ C_{1,0} = \begin{pmatrix} 200 \\ 500 \end{pmatrix}, & C_{1,1} = \begin{pmatrix} 270.71 \\ 570.71 \end{pmatrix}, & C_{1,2} = \begin{pmatrix} 529.29 \\ 570.71 \end{pmatrix}, & C_{1,3} = \begin{pmatrix} 600 \\ 500 \end{pmatrix}, \\ C_{2,0} = \begin{pmatrix} 600 \\ 500 \end{pmatrix}, & C_{2,1} = \begin{pmatrix} 635.36 \\ 464.64 \end{pmatrix}, & C_{2,2} = \begin{pmatrix} 555.92 \\ 323.59 \end{pmatrix}, & C_{2,3} = \begin{pmatrix} 600 \\ 300 \end{pmatrix}, \\ C_{3,0} = \begin{pmatrix} 600 \\ 300 \end{pmatrix}, & C_{3,1} = \begin{pmatrix} 679.47 \\ 257.47 \end{pmatrix}, & C_{3,2} = \begin{pmatrix} 900 \\ 500 \end{pmatrix}, & C_{3,3} = \begin{pmatrix} 900 \\ 500 \end{pmatrix}, \\ C_{4,0} = \begin{pmatrix} 900 \\ 500 \end{pmatrix}, & C_{4,1} = \begin{pmatrix} 900 \\ 500 \end{pmatrix}, & C_{4,2} = \begin{pmatrix} 361.09 \\ 659.52 \end{pmatrix}, & C_{4,3} = \begin{pmatrix} 400 \\ 800 \end{pmatrix}, \\ C_{5,0} = \begin{pmatrix} 400 \\ 800 \end{pmatrix}, & C_{5,1} = \begin{pmatrix} 433.37 \\ 920.46 \end{pmatrix}, & C_{5,2} = \begin{pmatrix} 900 \\ 675 \end{pmatrix}, & C_{5,3} = \begin{pmatrix} 900 \\ 800 \end{pmatrix}. \end{array}$$

Figure out which strategy *Adobe* has adopted for defining the tangents at the points that are converted to *smooth* and how they handle the situation at points that remain *corners*.

Describe the strategy and how you found it in detail, either with words or formulas.



-50-

Outline

- §1. Introduction
- §2. Lagrange interpolation
- §3. Newton's divided differences
- §4. Bezier splines
- §5. Cubic B-spline interpolation
- §6. Applications in font representation
- §7. Homework
- §8. Summary

-51-



Summary

- Polynomial interpolation
 - Explicit: Lagrange interpolation
 - Progressive: Newton's divided differences
- Spline interpolation
 - Bezier splines
 - Cubic B-splines
- Application in font representation

52



End



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE