Applied Mathematics 205

Unit I: Data Fitting

Lecturer: Dr. David Knezevic

Unit I: Data Fitting

Chapter I.2: Polynomial Interpolation

The Problem Formulation

Let $\mathbb{P}_n$ denote the set of all polynomials of degree n on $\mathbb{R}$

i.e. if $p(\cdot; b) \in \mathbb{P}_n$, then

$$p(x; b) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$$

for $b \equiv [b_0, b_1, \ldots, b_n]^T \in \mathbb{R}^{n+1}$

The Problem Formulation

Suppose we have the data $\mathcal{S} \equiv \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, where the $\{x_0, x_1, \ldots, x_n\}$ are called interpolation points

Goal: Find a polynomial that passes through every data point in $\mathcal{S}$

Therefore, we must have $p(x_i; b) = y_i$ for each $(x_i, y_i) \in \mathcal{S}$, i.e. $n + 1$ equations

For uniqueness, we should look for a polynomial with $n + 1$ parameters, i.e. look for $p \in \mathbb{P}_n$

Vandermonde Matrix

Then we obtain the following system of $n+1$ equations in $n+1$ unknowns

$$\begin{aligned} b_0 + b_1 x_0 + b_2 x_0^2 + \ldots + b_n x_0^n &= y_0 \\ b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n &= y_1 \\ &\vdots \\ b_0 + b_1 x_n + b_2 x_n^2 + \ldots + b_n x_n^n &= y_n \end{aligned}$$

Vandermonde Matrix

This can be written in Matrix form $Vb = y$, where

$$b = [b_0, b_1, \ldots, b_n]^T \in \mathbb{R}^{n+1},$$

$$y = [y_0, y_1, \ldots, y_n]^T \in \mathbb{R}^{n+1}$$

and $V \in \mathbb{R}^{(n+1)\times(n+1)}$ is the Vandermonde matrix:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$$

Existence and Uniqueness

Let's prove that if the $n+1$ interpolation points are distinct, then $Vb = y$ has a unique solution

We know from linear algebra that for a square matrix A
if $Az = 0 \implies z = 0$, then $Ab = y$ has a unique solution

If $Vb = 0$, then $p(\cdot; b) \in \mathbb{P}_n$ vanishes at $n+1$ distinct points

Therefore we must have $p(\cdot; b) = 0$, or equivalently $b = 0 \in \mathbb{R}^{n+1}$

Hence $Vb = 0 \implies b = 0$, so that $Vb = y$ has a unique solution for any $y \in \mathbb{R}^{n+1}$

Vandermonde Matrix

This tells us that we can find the polynomial interpolant by solving the Vandermonde system $Vb = y$

In general, however, this is a bad idea since V is ill-conditioned; $\text{cond}(V)$ grows rapidly with n

```
>> x = -1:0.2:1;
>> cond(vander(x))

ans =

    1.3952e+04

>> x = -1:0.1:1;
>> cond(vander(x))

ans =

    8.3138e+08
```

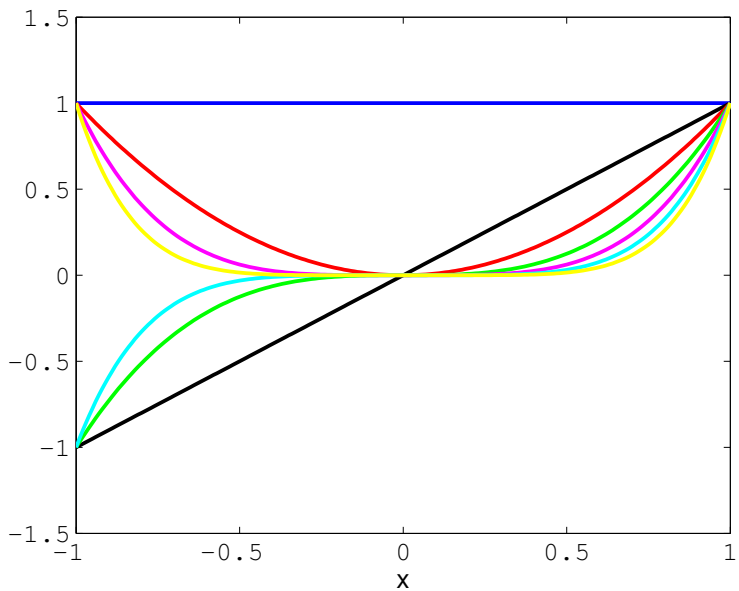
Monomial Interpolation

The problem here is that Vandermonde matrix corresponds to interpolation using the monomial basis

Monomial basis for $\mathbb{P}_n$ is $\{1, x, x^2, \ldots, x^n\}$

Monomial basis functions become increasingly similar

Vandermonde columns become nearly linearly-dependent
$\Longrightarrow$ ill-conditioned matrix!



Vandermonde Matrix: Interpolating on $[a, b]$

We often interpolate on an interval $[a, b]$ by reformulating in terms of interpolation on $[-1, 1]$ (see Assignment 1)

But of course, we can also interpolate on $[a, b]$ directly

Note that the Vandermonde matrix is ill-conditioned for any interval $[a, b]$, e.g. 11 interpolation points in $[-8, -5]$

```
>> x = -8:0.3:-5;
>> cond(vander(x))

ans =

    7.848e+18
```

Monomial Basis

Question: What is the practical consequence of this ill-conditioning?

Answer:

- A backward stable method for solving $Vb = y$ gives $\hat{b}$ where $V\hat{b} = (y + \Delta y)$, i.e. we can't avoid perturbations in y

- With large $\text{cond}(V)$, a small perturbation in y can give a large error in b!

The goal is to get a good approximation for b (since that defines our interpolant) hence large $\text{cond}(V)$ is a problem!

Monomial Basis

These sensitivities are directly analogous to what happens with an ill-conditioned basis in $\mathbb{R}^n$, e.g. consider a basis $\{v_1, v_2\}$ of $\mathbb{R}^2$:

$$v_1 = [1, 0]^T, \qquad v_2 = [1, 0.0001]^T$$

Then, let's express $y = [1, 0]^T$ and $\tilde{y} = [1, 0.0005]^T$ in terms of this basis

We can do this by solving a 2x2 linear system in each case, and hence we get

$$b = [1, 0]^T, \qquad \tilde{b} = [-4, 5]^T$$

Hence the answer is highly sensitive to perturbations in y!

Interpolation

We would like to avoid these kinds of sensitivities to perturbations... How can we do better?

Try to construct a basis such that the interpolation matrix is the identity matrix rather than the Vandermonde matrix

This gives a condition number of 1, and as an added bonus we also avoid inverting a dense $(n+1) \times (n+1)$ matrix

Lagrange Interpolation

Key idea: Construct basis $\{L_k \in \mathbb{P}_n, k = 0, \ldots, n\}$ such that

$$L_k(x_i) = \begin{cases} 0, & i \neq k, \\ 1, & i = k. \end{cases}$$

The polynomials that achieve this are called Lagrange polynomials[1]

See Lecture: These polynomials are given by:

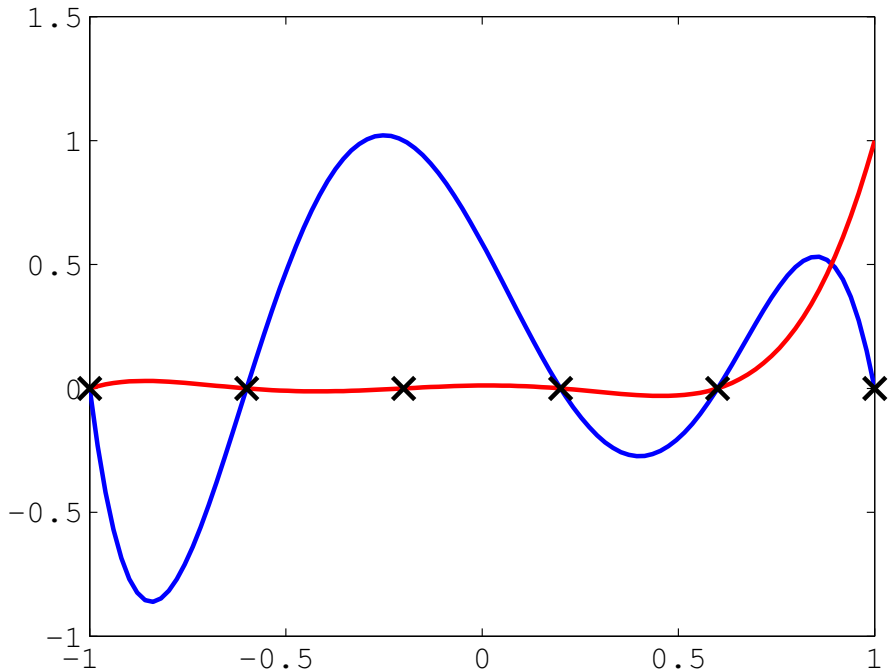
$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

and then the interpolant can be expressed as
$p_n(x) = \sum_{k=0}^{n} y_k L_k(x)$

[1] Joseph-Louis Lagrange, 1736-1813

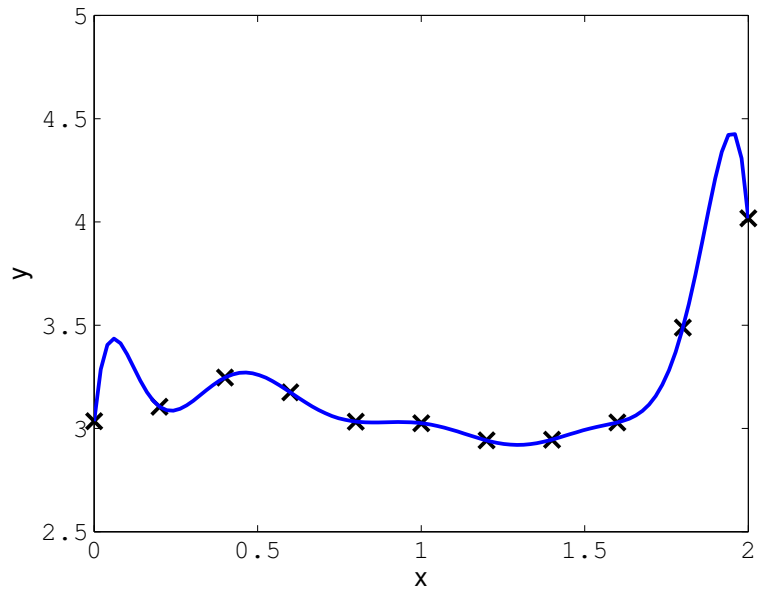
Lagrange Interpolation

Two Lagrange polynomials of degree 5



Lagrange Interpolation

Hence we can use Lagrange polynomials to interpolate discrete data (recall plot from I.1)



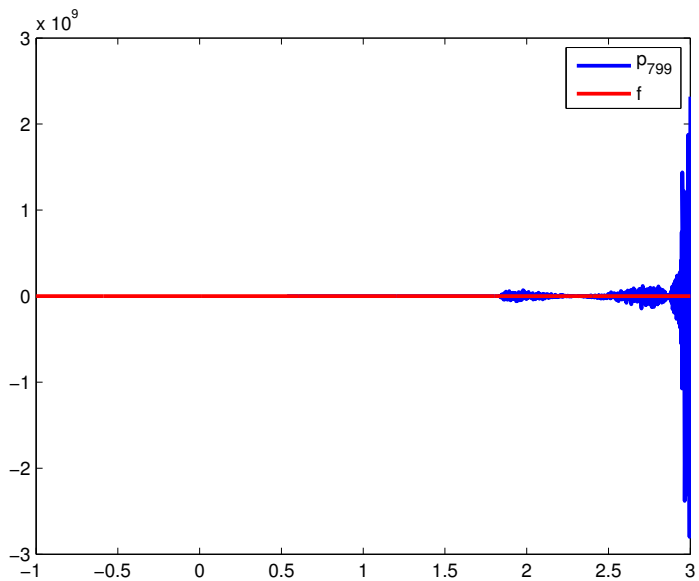
Summary: With Lagrange polynomials we can construct an interpolant of discrete data with condition number of 1

Lagrange Interpolation: Numerical Stability

The Lagrange interpolation formula $p_n(x) = \sum_{k=0}^{n} y_k L_k(x)$ works well in practice (as you'll see in Assignment 1)

But for very large n (e.g. > 500) it has issues with rounding error when we evaluate the formula, e.g. for plotting purposes

For example, interpolant of $\sin(5.5x)$ at 800 points:



Lagrange Interpolation: Barycentric Interpolation

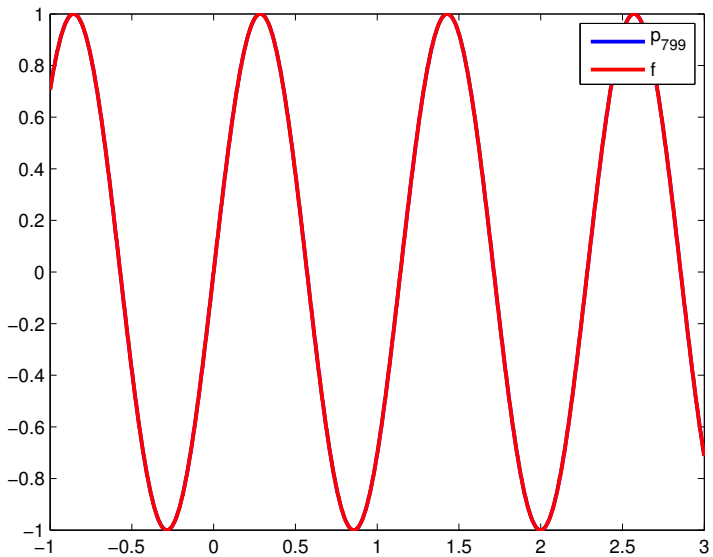
An alternative stable formula exists, called the barycentric interpolation formula:

$$p_n(x) = \frac{\sum_{j=0}^{n} \frac{w_j}{x-x_j} y_j}{\sum_{j=0}^{n} \frac{w_j}{x-x_j}}, \quad \text{where} \quad w_j = \frac{1}{\prod_{i=0, i\neq j}^{n} (x_j - x_i)}$$

The two approaches are mathematically identical, but they have different properties with regard to rounding error

Lagrange Interpolation: Barycentric Interpolation

Barycentric interpolation is "rock solid" for any n



Interpolation for Function Approximation

Interpolation for Function Approximation

We now turn to a different (and much deeper) question: Can we use interpolation to accurately approximate continuous functions?

Suppose the interpolation data come from samples of a continuous function f on $[a, b] \subset \mathbb{R}$

Then we'd like the interpolant to be "close to" f on $[a, b]$

The error in this type of approximation can be quantified from the following theorem due to Cauchy[2]:

$$f(x) - p_n(x) = \frac{f^{n+1}(\theta)}{(n+1)!}(x - x_0)\dots(x - x_n) \text{ for some } \theta \in (a, b)$$

[2] Augustin-Louis Cauchy, 1789-1857

Polynomial Interpolation Error

We prove this result in the case $n = 1$

Let $p_1 \in \mathbb{P}_1[x_0, x_1]$ interpolate $f \in C^2[a, b]$ at $\{x_0, x_1\}$

For some $\lambda \in \mathbb{R}$, let

$$q(x) \equiv p_1(x) + \lambda(x - x_0)(x - x_1),$$

here q is quadratic and interpolates f at $\{x_0, x_1\}$

Fix an arbitrary point $\hat{x} \in (x_0, x_1)$ and set $q(\hat{x}) = f(\hat{x})$ to get

$$\lambda = \frac{f(\hat{x}) - p_1(\hat{x})}{(\hat{x} - x_0)(\hat{x} - x_1)}$$

Goal: Get an expression for λ, since then we obtain an expression for $f(\hat{x}) - p_1(\hat{x})$

Polynomial Interpolation Error

Now, let $e(x) \equiv f(x) - q(x)$

- e has 3 roots in $[x_0, x_1]$, i.e. at $x = x_0, \hat{x}, x_1$
- Therefore e' has 2 roots in (x_0, x_1) (by Rolle's theorem)
- Therefore e'' has 1 root in (x_0, x_1) (by Rolle's theorem)

Let $\theta \in (x_0, x_1)$ be such that[3] $e''(\theta) = 0$

Then

$$\begin{aligned} 0 &= e''(\theta) = f''(\theta) - q''(\theta) \\ &= f''(\theta) - p_1''(\theta) - \lambda \frac{\mathrm{d}^2}{\mathrm{d}\theta^2}(\theta - x_0)(\theta - x_1) \\ &= f''(\theta) - 2\lambda \end{aligned}$$

hence $\lambda = \frac{1}{2} f''(\theta)$

[3] Note that θ is a function of $\hat{x}$

Polynomial Interpolation Error

Hence, we get:

$$f(\hat{x}) - p_1(\hat{x}) = \lambda(\hat{x} - x_0)(\hat{x} - x_1) = \frac{1}{2}f''(\theta)(\hat{x} - x_0)(\hat{x} - x_1)$$

for any $\hat{x} \in (x_0, x_1)$ (recall that $\hat{x}$ was chosen arbitrarily)

This argument can be generalized to $n > 1$ to give:

$$f(x) - p_n(x) = \frac{f^{n+1}(\theta)}{(n+1)!}(x - x_0)\dots(x - x_n) \text{ for some } \theta \in (a, b)$$

Polynomial Interpolation Error

For any $x \in [a, b]$, this theorem gives us the error bound

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \max_{x \in [a,b]} |(x - x_0) \dots (x - x_n)|,$$

where $M_{n+1} = \max_{\theta \in [a,b]} |f^{n+1}(\theta)|$

If $1/(n+1)! \to 0$ faster than

$$M_{n+1} \max_{x \in [a,b]} |(x - x_0) \dots (x - x_n)| \to \infty$$

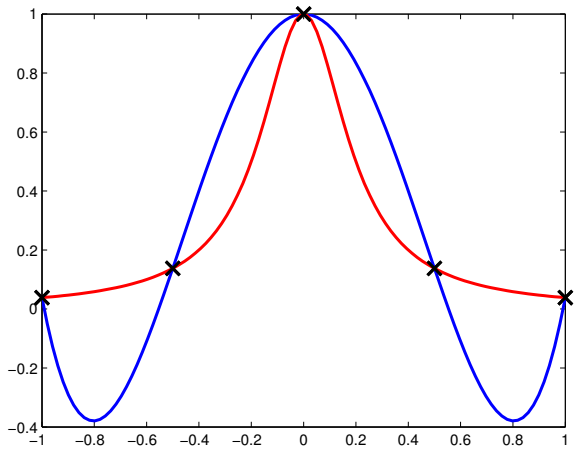
then $p_n \to f$

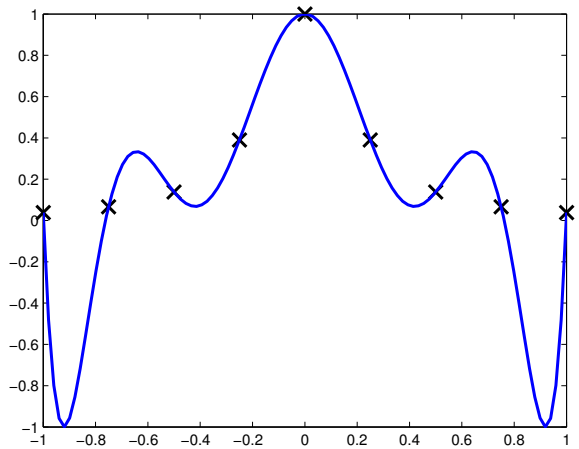
Unfortunately, this is not always the case!

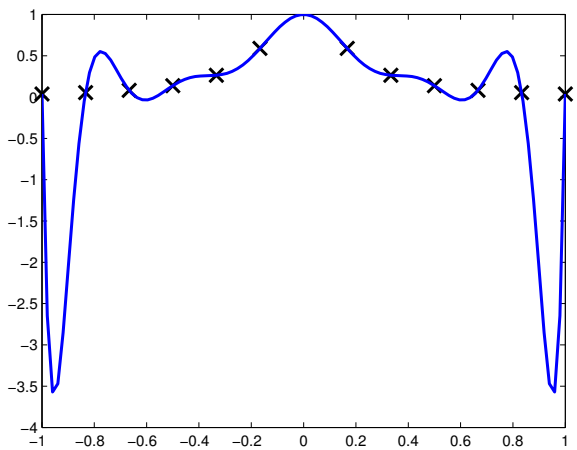
Runge's Phenomenon

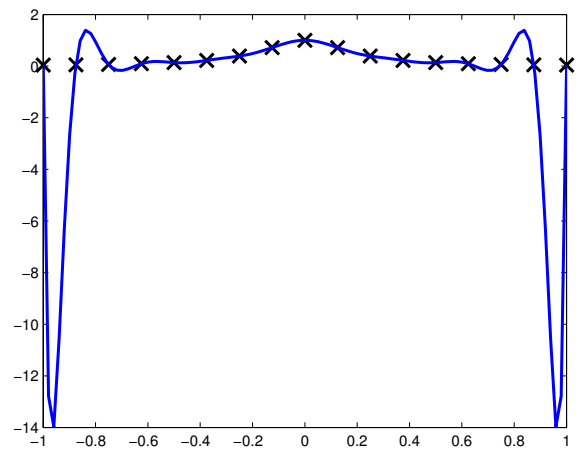
A famous pathological example of the difficulty of interpolation at equally spaced points is Runge's Phenomenon

Consider $f(x) = 1/(1 + 25x^2)$ for $x \in [-1, 1]$









Runge's Phenomenon

Note that of course p_n fits the evenly spaced samples exactly

But we are now also interested in the maximum error between f and its polynomial interpolant p_n

That is, we want $\max_{x\in[-1,1]} |f(x) - p_n(x)|$ to be small!

This is generally referred to as the "infinity norm" or the "max norm":

$$\|f - p_n\|_\infty \equiv \max_{x\in[-1,1]} |f(x) - p_n(x)|$$

Runge's Phenomenon

Interpolating Runge's function at evenly spaced points leads to infinity norm error that grows exponentially with n

We would like to construct an interpolant of f such that this kind of pathological behavior is impossible

Minimizing Interpolation Error

In general we don't know how to construct a p_n that minimizes $\|f - p_n\|_\infty$

Optimal polynomial approximation in the ∞-norm is a deep subject in approximation theory, e.g. see Chapter 8 of Suli & Mayers

However, we can construct a "near minimizer" that works very well in practice

Minimizing Interpolation Error

To do this, we recall our error equation

$$f(x) - p_n(x) = \frac{f^{n+1}(\theta)}{(n+1)!}(x - x_0)\dots(x - x_n)$$

We focus our attention on the polynomial $(x - x_0)\dots(x - x_n)$, since we can choose the interpolation points

Intuitively, we should choose $x_0, x_1, \dots, x_n$ such that $\|(x - x_0)\dots(x - x_n)\|_\infty$ is as small as possible

Interpolation at Chebyshev Points

Result from Approximation Theory:
For $x \in [-1, 1]$, the minimum value of $\|(x - x_0) \dots (x - x_n)\|_\infty$ is $1/2^n$, achieved by the polynomial $T_{n+1}(x)/2^n$

$T_{n+1}(x)$ is the Chebyshev poly. (of the first kind) of order $n + 1$

T_{n+1} has leading coefficient of 2^n, hence $T_{n+1}(x)/2^n$ is monic, and so is $(x - x_0) \dots (x - x_n)$

Hence to set $(x - x_0) \dots (x - x_n) = T_{n+1}(x)/2^n$, we choose interpolation points to be the roots of T_{n+1}

Interpolation at Chebyshev Points

Chebyshev polynomials are defined for $x \in [-1, 1]$ by
$T_n(x) = \cos(n \cos^{-1} x), n = 0, 1, 2, \ldots$

Or equivalently[4], the recurrence relation,
$T_0(x) = 1$,
$T_1(x) = x$,
$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),\ n = 1, 2, 3, \ldots$

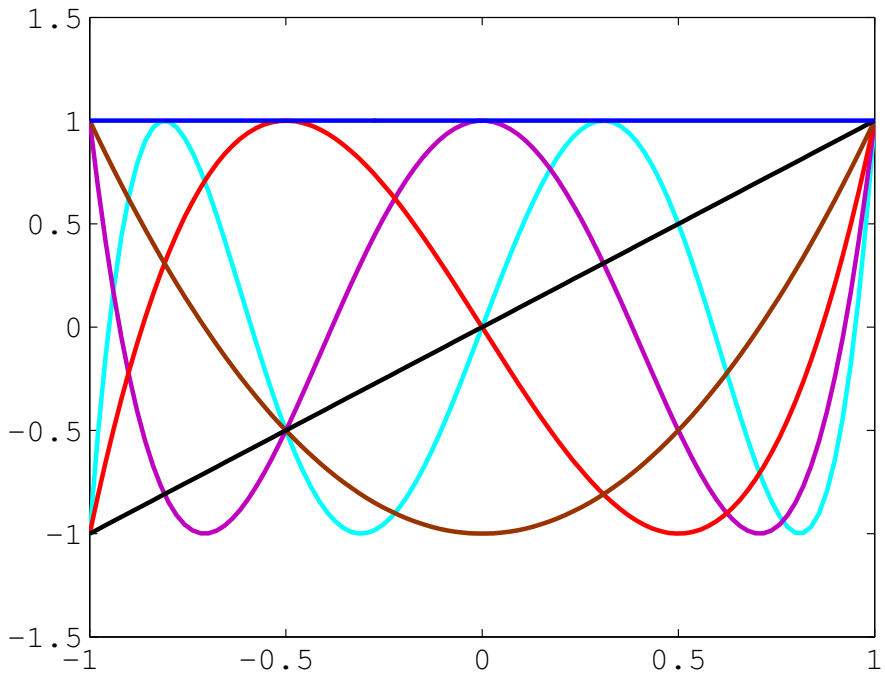
Exercise: Show that the roots of T_n are given by
$x_j = \cos((2j - 1)\pi/2n),\ j = 1, \ldots, n$

The roots of T_n are called Chebyshev points

[4] Equivalence can be shown using trig. identities for T_{n+1} and T_{n-1}

Interpolation at Chebyshev Points

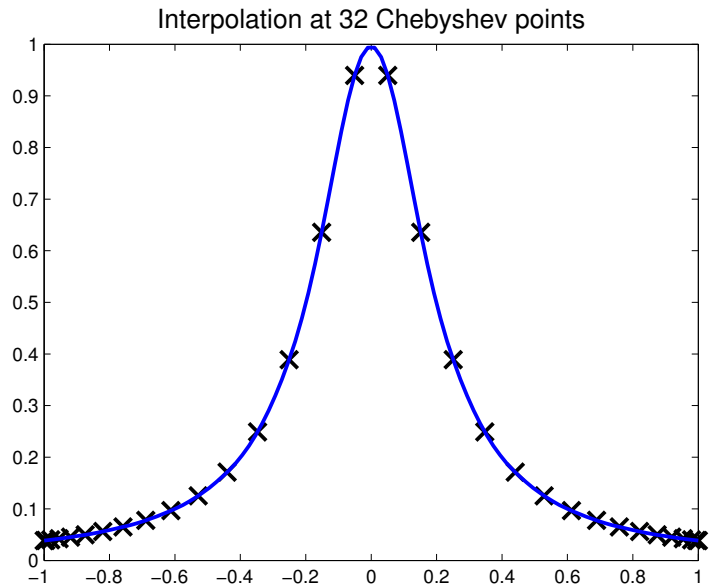
Chebyshev polys "equi-oscillate" between -1 and 1, hence it's not surprising that they are related to the minimum infinity norm



Interpolation at Chebyshev Points

We can combine the preceding results to derive an error bound for interpolation at Chebyshev points (see Assignment 1)

Generally speaking, with Chebyshev interpolation, p_n converges to any smooth f very rapidly! e.g. Runge function:



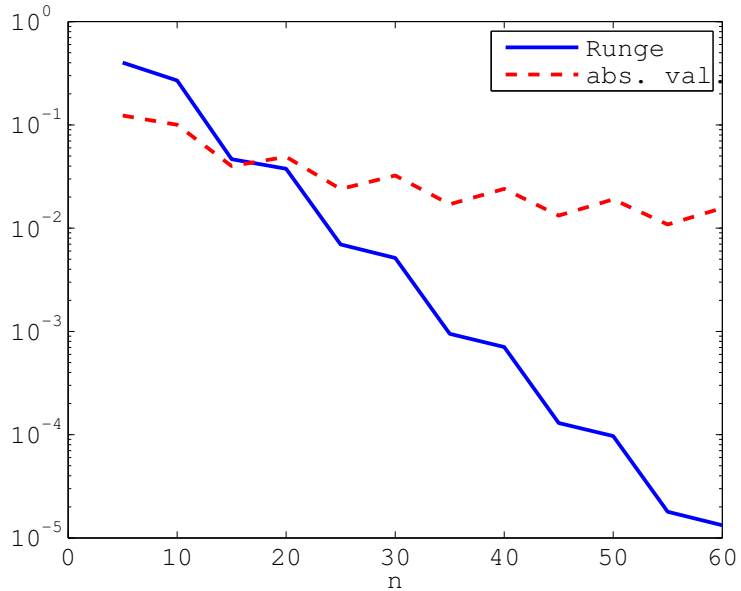
Chebyshev points are defined on $[-1, 1]$, but we can interpolate on $[a, b]$ by mapping the interval (see Assignment 1)

Interpolation at Chebyshev Points

Note that convergence rates depend on smoothness of f — precise statements about this can be made, outside the scope of AM205

In general, smoother $f \implies$ faster convergence[5]

e.g. compare convergence of Chebyshev interpolation of Runge's function (smooth) and $f(x) = |x|$ (not smooth)



[5] For example, if f is analytic, we get exponential convergence!

Another View on Interpolation Accuracy

We have seen that the interpolation points we choose have an enormous effect on how well our interpolant approximates f

The choice of Chebyshev interpolation points was motivated by our interpolation error formula for $f(x) - p_n(x)$

But this formula depends on f — we would prefer to have a measure of interpolation accuracy that is independent of f

This would provide a more general way to compare the quality of interpolation points... This is provided by the Lebesgue constant

Lebesgue Constant

Let $\mathcal{X}$ denote a set of interpolation points,
$\mathcal{X} \equiv \{x_0, x_1, \ldots, x_n\} \subset [a, b]$

A fundamental property of $\mathcal{X}$ is its Lebesgue constant, $\Lambda_n(\mathcal{X})$,

$$\Lambda_n(\mathcal{X}) \equiv \max_{x \in [a,b]} \sum_{k=0}^{n} |L_k(x)|$$

The $L_k \in \mathbb{P}_n$ are the Lagrange polynomials associated with $\mathcal{X}$,
hence Λ_n is also a function of $\mathcal{X}$

$\Lambda_n(\mathcal{X}) \geq 1$, why?

Lebesgue Constant

Think of polynomial interpolation as a map, $\mathcal{I}_n$, where $\mathcal{I}_n : C[a,b] \to \mathbb{P}_n[a,b]$

$\mathcal{I}_n(f)$ is the degree n polynomial interpolant of $f \in C[a,b]$ at the interpolation points $\mathcal{X}$

Exercise: Convince yourself that $\mathcal{I}_n$ is linear (e.g. use the Lagrange interpolation formula)

The reason that the Lebesgue constant is interesting is because it bounds the "operator norm" of $\mathcal{I}_n$:

$$\sup_{f \in C[a,b]} \frac{\|\mathcal{I}_n(f)\|_\infty}{\|f\|_\infty} \leq \Lambda_n(\mathcal{X})$$

Lebesgue Constant

Proof:

$$\begin{aligned}
\|\mathcal{I}_n(f)\|_\infty &= \|\sum_{k=0}^{n} f(x_k)L_k\|_\infty = \max_{x\in[a,b]}\left|\sum_{k=0}^{n} f(x_k)L_k(x)\right| \\
&\leq \max_{x\in[a,b]}\sum_{k=0}^{n}|f(x_k)||L_k(x)| \\
&\leq \left(\max_{k=0,1,\ldots,n}|f(x_k)|\right)\max_{x\in[a,b]}\sum_{k=0}^{n}|L_k(x)| \\
&\leq \|f\|_\infty \max_{x\in[a,b]}\sum_{k=0}^{n}|L_k(x)| \\
&= \|f\|_\infty \Lambda_n(\mathcal{X})
\end{aligned}$$

Hence

$$\frac{\|\mathcal{I}_n(f)\|_\infty}{\|f\|_\infty} \leq \Lambda_n(\mathcal{X}), \quad \text{so} \quad \sup_{f\in C[a,b]}\frac{\|\mathcal{I}_n(f)\|_\infty}{\|f\|_\infty} \leq \Lambda_n(\mathcal{X}).$$

Lebesgue Constant

Lebesgue constant allows us to bound interpolation error in terms of the smallest possible error from $\mathbb{P}_n$

Let $p_n^* \in \mathbb{P}_n$ denote the best infinity-norm approximation to f, i.e. $\|f - p_n^*\|_\infty \leq \|f - w\|_\infty$ for all $w \in \mathbb{P}_n$

Some facts about p_n^*:

- $\|p_n^* - f\|_\infty \to 0$ as $n \to \infty$ for any continuous f! (Weierstrass approximation theorem[6])
- $p_n^* \in \mathbb{P}_n$ is unique
- In general, p_n^* is unknown

[6] Any continuous f can be uniformly approximated by polynomials. Bernstein polynomials can be used to give a constructive proof, but these polynomials do not yield p_n^* and they converge too slowly for practical purposes.

Lebesgue Constant

Then, we can relate interpolation error to $\|f - p_n^*\|_\infty$ as follows:

$$
\begin{aligned}
\|f - \mathcal{I}_n(f)\|_\infty &\leq \|f - p_n^*\|_\infty + \|p_n^* - \mathcal{I}_n(f)\|_\infty \\
&= \|f - p_n^*\|_\infty + \|\mathcal{I}_n(p_n^*) - \mathcal{I}_n(f)\|_\infty \\
&= \|f - p_n^*\|_\infty + \|\mathcal{I}_n(p_n^* - f)\|_\infty \\
&= \|f - p_n^*\|_\infty + \frac{\|\mathcal{I}_n(p_n^* - f)\|_\infty}{\|p_n^* - f\|_\infty}\|f - p_n^*\|_\infty \\
&\leq \|f - p_n^*\|_\infty + \Lambda_n(\mathcal{X})\|f - p_n^*\|_\infty \\
&= (1 + \Lambda_n(\mathcal{X}))\|f - p_n^*\|_\infty
\end{aligned}
$$

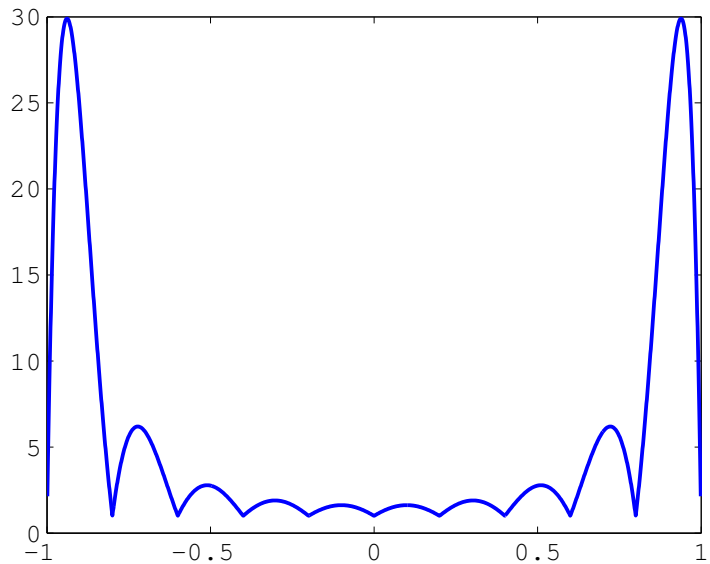
Lebesgue Constant

Small Lebesgue constant means that our interpolation can't be much worse that the best possible polynomial approximation!

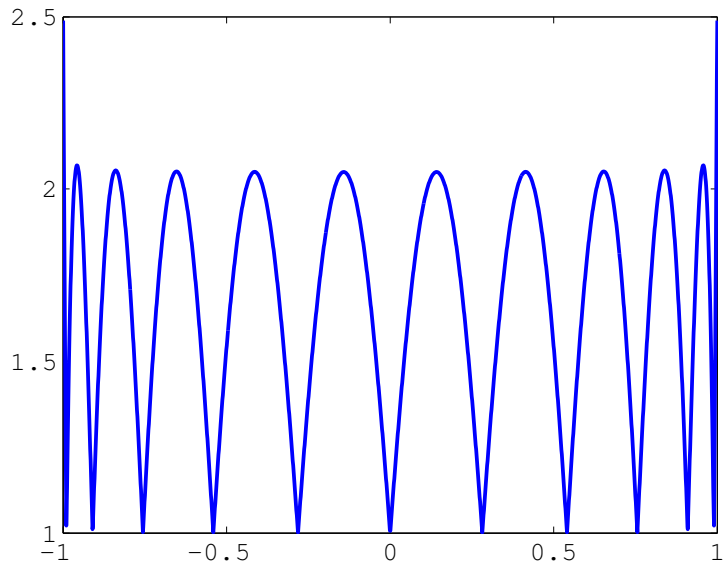
Now let's compare Lebesgue constants for equispaced ($\mathcal{X}_{\text{equi}}$) and Chebyshev points ($\mathcal{X}_{\text{cheb}}$)

Lebesgue Constant

Plot of $\sum_{k=0}^{10} |L_k(x)|$ for $\mathcal{X}_{\text{equi}}$ and $\mathcal{X}_{\text{cheb}}$ (11 pts in [-1,1])



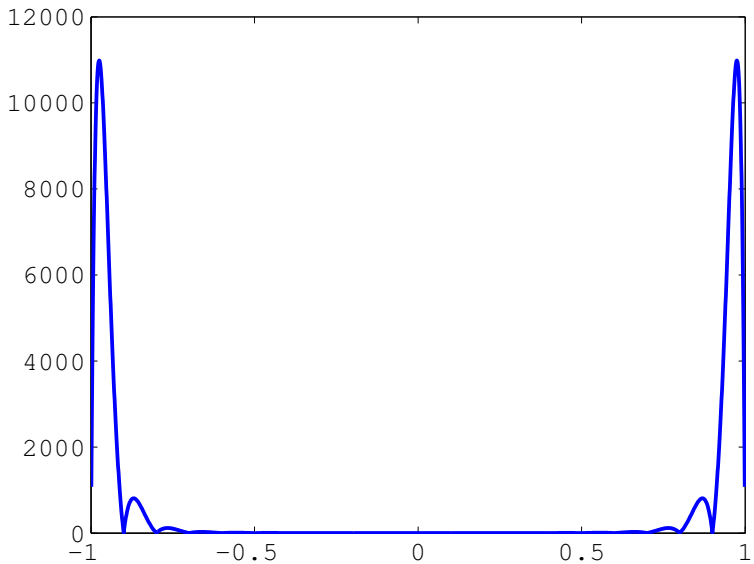
$\Lambda_{10}(\mathcal{X}_{\text{equi}}) \approx 29.9$

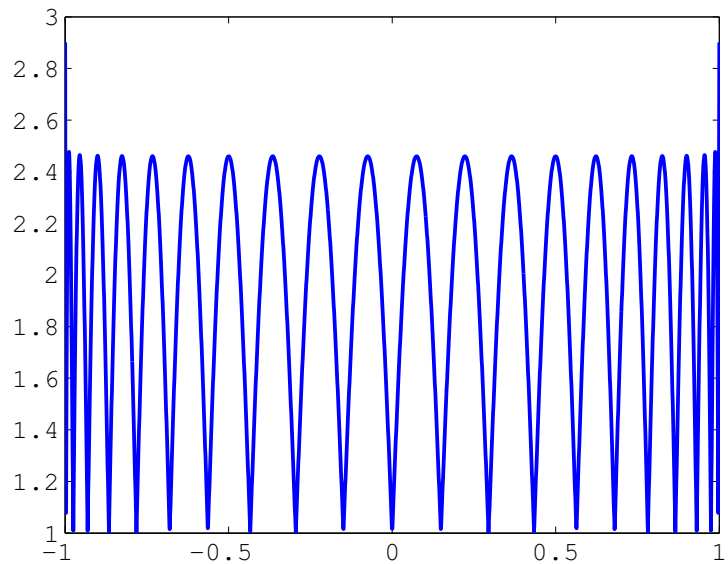


$\Lambda_{10}(\mathcal{X}_{\text{cheb}}) \approx 2.49$

Lebesgue Constant

Plot of $\sum_{k=0}^{20} |L_k(x)|$ for $\mathcal{X}_{\text{equi}}$ and $\mathcal{X}_{\text{cheb}}$ (21 pts in [-1,1])



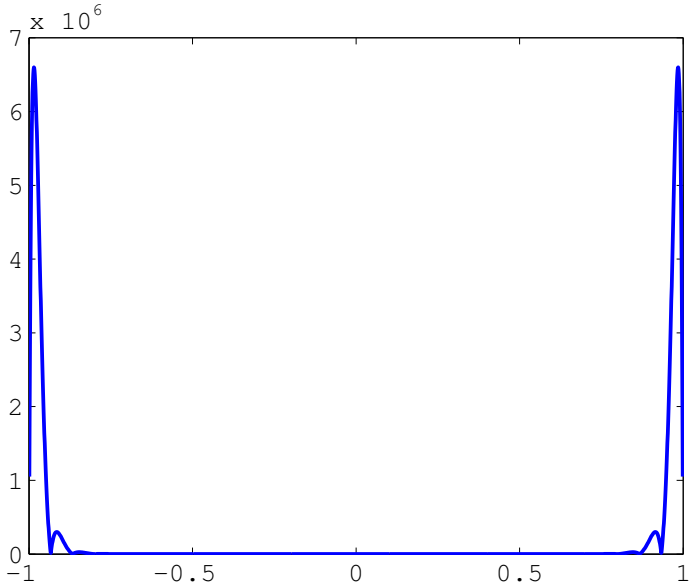


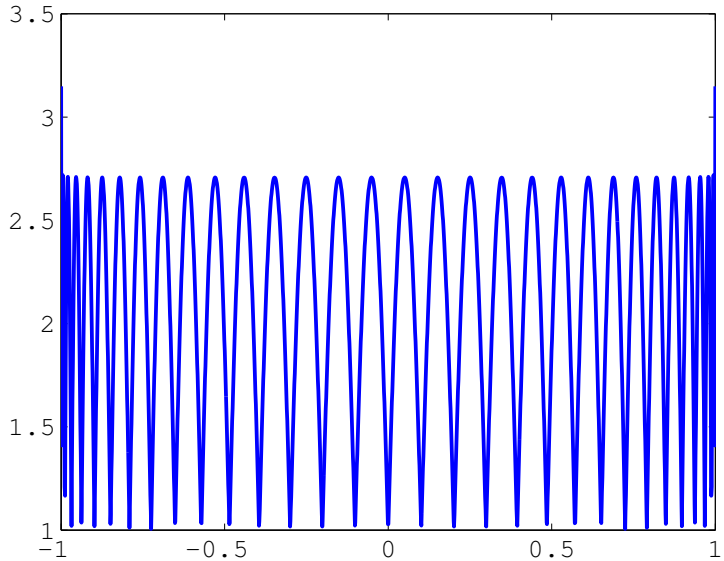
$\Lambda_{20}(\mathcal{X}_{\text{equi}}) \approx 10{,}987$

$\Lambda_{20}(\mathcal{X}_{\text{cheb}}) \approx 2.9$

Lebesgue Constant

Plot of $\sum_{k=0}^{30} |L_k(x)|$ for $\mathcal{X}_{\text{equi}}$ and $\mathcal{X}_{\text{cheb}}$ (31 pts in [-1,1])





$\Lambda_{30}(\mathcal{X}_{\text{equi}}) \approx 6{,}600{,}000$

$\Lambda_{30}(\mathcal{X}_{\text{cheb}}) \approx 3.15$

Lebesgue Constant

The explosive growth of $\Lambda_n(\mathcal{X}_{\text{equi}})$ is an explanation for Runge's phenomenon[7]

It has been shown that as $n \to \infty$,

$$\Lambda_n(\mathcal{X}_{\text{equi}}) \sim \frac{2^n}{en \log n} \qquad \text{BAD!}$$

whereas

$$\Lambda_n(\mathcal{X}_{\text{cheb}}) < \frac{2}{\pi} \log(n+1) + 1 \qquad \text{GOOD!}$$

Important open mathematical problem: What is the optimal set of interpolation points (i.e. what $\mathcal{X}$ minimizes $\Lambda_n(\mathcal{X})$)?

[7] Runge's function $f(x) = 1/(1+25x^2)$ excites the "worst case" behavior allowed by $\Lambda_n(\mathcal{X}_{\text{equi}})$

Summary

It is helpful to compare and contrast the two key topics we've considered so far in this chapter

1. Polynomial interpolation for fitting discrete data:

- We get "zero error" regardless of the interpolation points, i.e. we're guaranteed to fit the discrete data
- Should use Lagrange polynomial basis (ideally via the barycentric formula!)

2. Polynomial interpolation for approximating continuous functions:

- For a given set of interp. pts., uses the methodology from 1. above to construct the interpolant
- Interpolation points determine the magnitude of the error $\|f - \mathcal{I}_n(f)\|_\infty$; Chebyshev points are an excellent choice

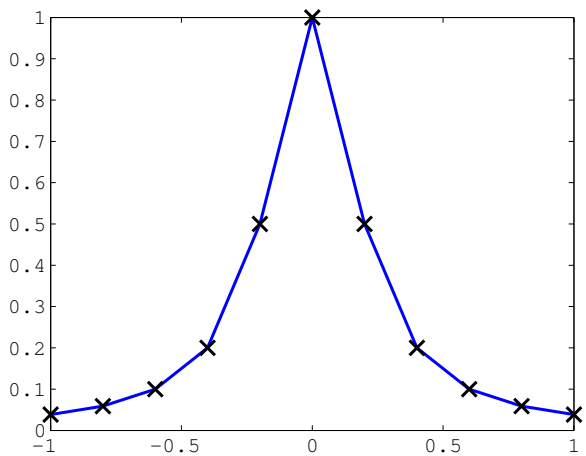
Piecewise Polynomial Interpolation

Piecewise Polynomial Interpolation

We can't always choose our interpolation points to be Chebyshev, so another way to get good results is via piecewise polynomials

Idea is simple: Break domain into subdomains, apply polynomial interpolation on each subdomain (interp. pts. now called "knots")

Recall piecewise linear interpolation, also called "linear spline"



Piecewise Polynomial Interpolation

With piecewise polynomials, we avoid high-order polynomials hence we avoid "blow-up"

However, we limit the convergence rate based on the polynomial degree

See lecture: Piecewise polynomial error analysis

In general for sufficiently smooth f, degree n piecewise polynomial interpolation gives convergence rate h^{n+1} as $h \to 0$

Splines

Splines are a popular type of piecewise polynomial interpolant that mitigate "loss of smoothness" of piecewise approximation

(The name "spline" comes from a tool used by ship designers to draw smooth curves by hand)

In general, a spline of degree k is a piecewise polynomial that is continuously differentiable $k-1$ times

Splines are the basis of CAD software (e.g. AutoCAD, SolidWorks), also used in vector graphics, fonts etc[8]

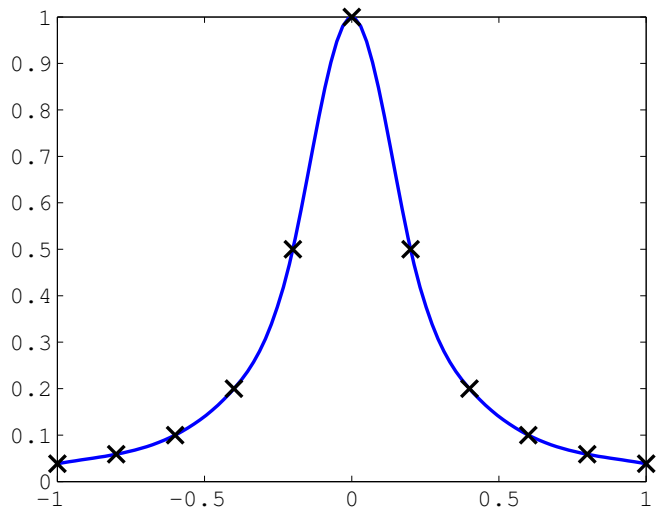
[8] CAD software uses NURB splines, font definitions use Bézier splines

Splines

We focus on a popular type of spline: Cubic spline $\in C^2[a, b]$

Continuous second derivatives $\implies$ looks smooth to the eye

For example, cubic spline interpolation of Runge function (using Matlab's `spline` function)



Cubic Splines: Formulation

Suppose we have knots $x_0, \ldots, x_n$, then cubic on each interval $[x_{i-1}, x_i] \Longrightarrow 4n$ parameters in total

Let s denote our cubic spline, and suppose we want to interpolate the data $\{f_i, i = 0, 1, \ldots, n\}$

We must interpolate at $n+1$ points, $s(x_i) = f_i$, which provides two equations per interval $\Longrightarrow 2n$ equations for interpolation

Also, $s'_-(x_i) = s'_+(x_i)$, $i = 1, \ldots, n-1 \Longrightarrow n-1$ equations for continuous first derivative

And, $s''_-(x_i) = s''_+(x_i)$, $i = 1, \ldots, n-1 \Longrightarrow n-1$ equations for continuous second derivative

Hence $4n-2$ equations in total

Cubic Splines

We are short by two conditions! There are many ways to make up the last two, e.g.

- Natural cubic spline: Set $s''(x_0) = s''(x_n) = 0$
- "Not-a-knot spline"[9]: Set $s'''_-(x_1) = s'''_+(x_1)$ and $s'''_-(x_{n-1}) = s'''_+(x_{n-1})$
- Or we can choose any other two equations we like (e.g. set two of the spline parameters to zero)[10]

Example: See lecture

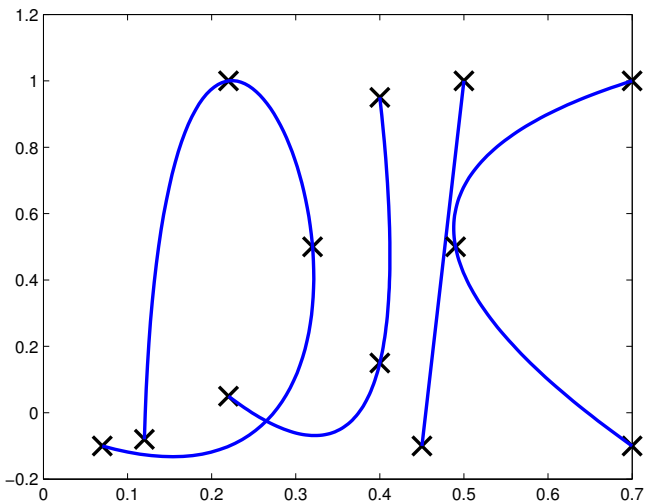
[9] "Not-a-knot" because all derivatives of s are continuous at x_1 and x_{n-1}

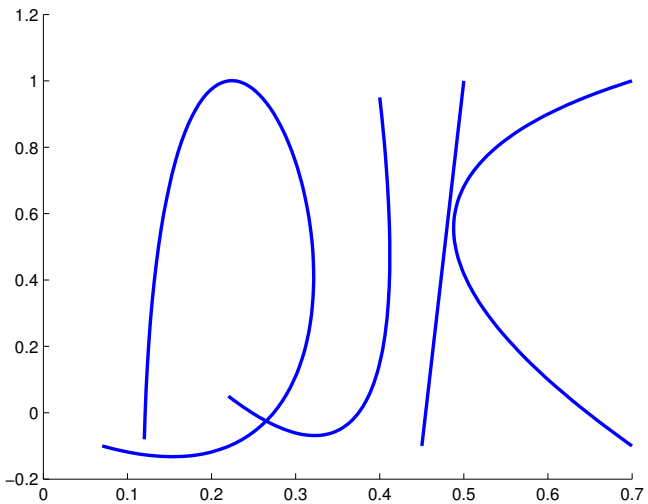
[10] As long as they are linearly independent from the first $4n - 2$ equations

Cubic Splines

Splines are well-suited to designing parametrized curves since we have a lot of flexibility in placing knots (AKA control points)

For example, curves in $\mathbb{R}^2$: $t \to (s^x(t), s^y(t))$ where s^x and s^y are splines, can be used for fonts





Cubic Splines

Also, once we have a parametrized description of a font, we can (for example) change font size by scaling control points