Applied Math 205

- Homework schedule now posted. Deadlines at 5 PM on Sep 26, Oct 10, Oct 24, Nov 7, Dec 2. First assignment will be uploaded by Monday Sep 15.
- Take-home midterm: Nov 13–14
- Last time: polynomial interpolation for discrete and continuous data
- Today: piecewise polynomial interpolation, least-squares fitting

Lebesgue Constant

The 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)
- $p_n^* \in \mathbb{P}_n$ is unique
- In general, p_n^* is unknown

Bernstein interpolation

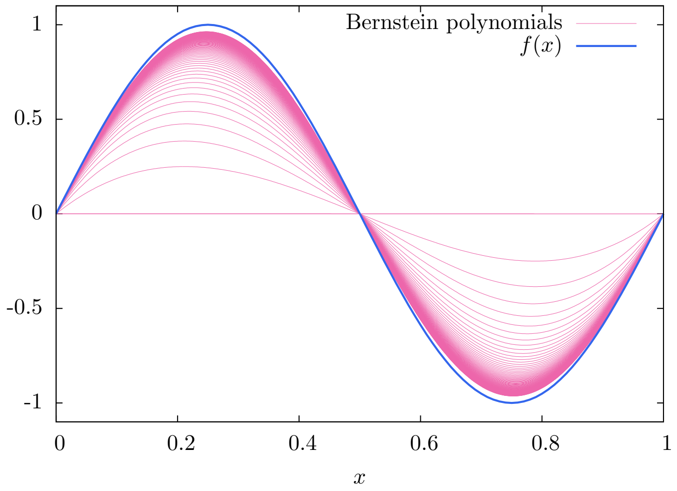
The Bernstein polynomials on $[0, 1]$ are

$$b_{m,n}(x) = \left(\begin{array}{c} n \\ m \end{array} \right) x^m (1-x)^{n-m}.$$

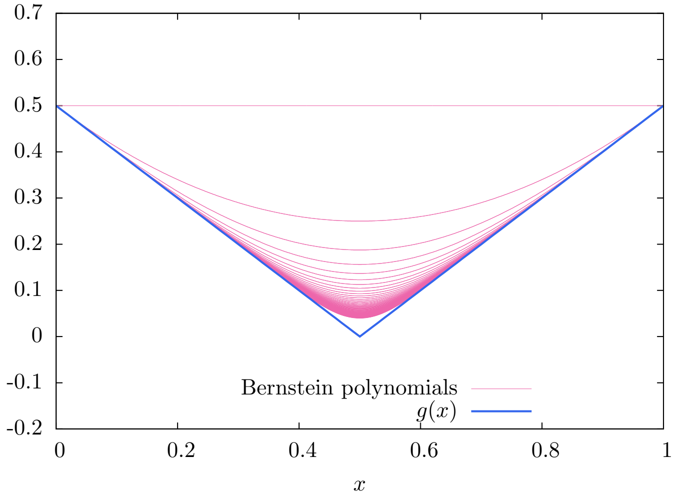
For a function f on the $[0, 1]$, the approximating polynomial is

$$B_n(f)(x) = \sum_{m=0}^{n} f\left(\frac{m}{n}\right) b_{m,n}(x).$$

Bernstein interpolation



Bernstein interpolation



Lebesgue Constant

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

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

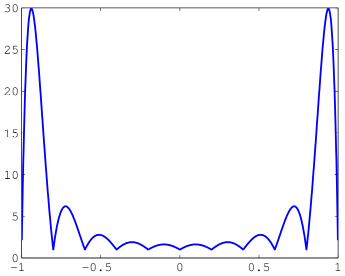
Lebesgue Constant

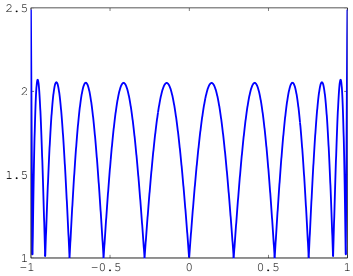
Small Lebesgue constant means that our interpolation can't be much worse that the best possible polynomial approximation!

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 $\mathcal{X}_{\mathrm{equi}}$ and $\mathcal{X}_{\mathrm{cheb}}$ (11 pts in [-1,1])



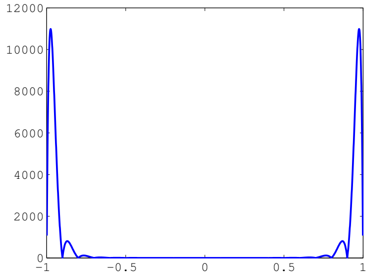


$\Lambda_{10}(\mathcal{X}_{\mathrm{equi}}) \approx 29.9$

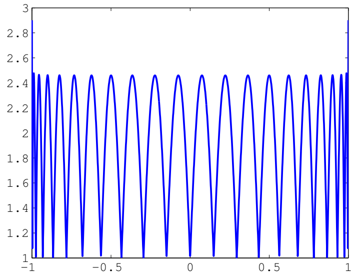
$\Lambda_{10}(\mathcal{X}_{\mathrm{cheb}}) \approx 2.49$

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])



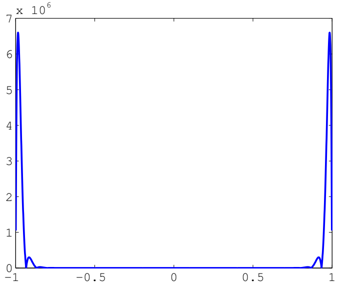
$\Lambda_{20}(\mathcal{X}_{\text{equi}}) \approx 10{,}987$



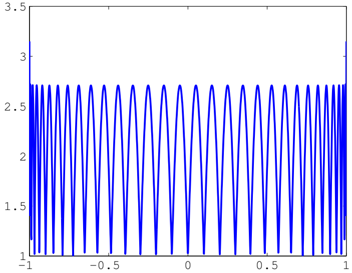
$\Lambda_{20}(\mathcal{X}_{\text{cheb}}) \approx 2.9$

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[1]

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

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

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 $\Lambda_n(\mathcal{X})$)?

[1] 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 (diagonal system, well-conditioned)

2. Polynomial interpolation for approximating continuous functions:

- For a given set of interpolating points, uses the methodology from 1 above to construct the interpolant
- But now interpolation points play a crucial role in determining the magnitude of the error $\|f - \mathcal{I}_n(f)\|_\infty$

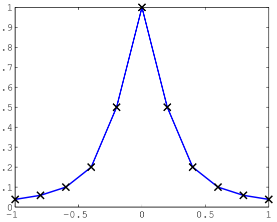
Piecewise Polynomial Interpolation

Piecewise Polynomial Interpolation

We can't always choose our interpolation points to be Chebyshev, so another way to avoid "blow up" 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 clearly lose smoothness of the interpolant

Also, can’t do better than algebraic convergence[2]

[2]Recall that for smooth functions Chebyshev interpolation gives exponential convergence with n

Splines

Splines are a popular type of piecewise polynomial interpolant

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

Splines solve the “loss of smoothness” issue to some extent since they have continuous derivatives

Splines are the basis of CAD software (AutoCAD, SolidWorks), also used in vector graphics, fonts, *etc.*[3]

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

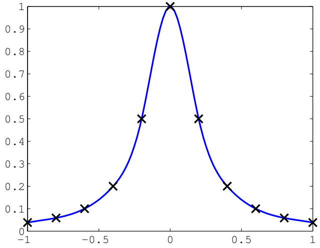
[3]CAD software uses NURB splines, font definitions use Bézier splines

Splines

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

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

Example: Cubic spline interpolation of Runge 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"[4]: 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)[5]

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

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

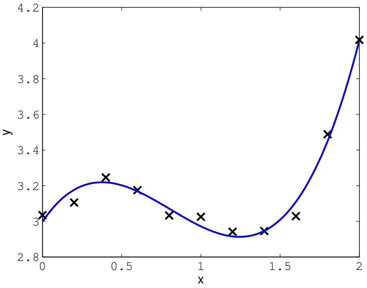
Unit I: Data Fitting

Chapter I.3: Linear Least Squares

The Problem Formulation

Recall that it can be advantageous to not fit data points exactly (*e.g.* due to experimental error), we don't want to "overfit"

Suppose we want to fit a cubic polynomial to 11 data points



Question: How do we do this?

The Problem Formulation

Suppose we have m constraints and n parameters with $m > n$ (e.g. $m = 11$, $n = 4$ on previous slide)

In terms of linear algebra, this is an overdetermined system $Ab = y$, where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^n$ (parameters), $y \in \mathbb{R}^m$ (data)

$$\begin{bmatrix} & & \\ & & \\ & A & \\ & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} \\ b \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ y \\ \\ \\ \\ \end{bmatrix}$$

i.e. we have a “tall, thin” matrix A

The Problem Formulation

In general, cannot be solved exactly (hence we will write $Ab \simeq y$); instead our goal is to minimize the residual, $r(b) \in \mathbb{R}^m$

$$r(b) \equiv y - Ab$$

A very effective approach for this is the method of least squares:[6]
Find parameter vector $b \in \mathbb{R}^n$ that minimizes $\|r(b)\|_2$

As we shall see, we minimize the 2-norm above since it gives us a differentiable function (we can then use calculus)

[6] Developed by Gauss and Legendre for fitting astronomical observations with experimental error

The Normal Equations

Goal is to minimize $\|r(b)\|_2$, recall that $\|r(b)\|_2 = \sqrt{\sum_{i=1}^n r_i(b)^2}$

The minimizing b is the same for $\|r(b)\|_2$ and $\|r(b)\|_2^2$, hence we consider the differentiable "objective function" $\phi(b) = \|r(b)\|_2^2$

$$\begin{aligned}\phi(b) &= \|r\|_2^2 = r^T r = (y - Ab)^T(y - Ab) \\ &= y^T y - y^T Ab - b^T A^T y + b^T A^T Ab \\ &= y^T y - 2b^T A^T y + b^T A^T Ab\end{aligned}$$

where last line follows from $y^T Ab = (y^T Ab)^T$, since $y^T Ab \in \mathbb{R}$

ϕ is a quadratic function of b, and is non-negative, hence a minimum must exist, (but not nec. unique, e.g. $f(b_1, b_2) = b_1^2$)

The Normal Equations

To find minimum of $\phi(b) = y^T y - 2b^T A^T y + b^T A^T A b$, differentiate wrt b and set to zero[7]

First, let's differentiate $b^T A^T y$ wrt b

That is, we want $\nabla(b^T c)$ where $c \equiv A^T y \in \mathbb{R}^n$:

$$b^T c = \sum_{i=1}^{n} b_i c_i \implies \frac{\partial}{\partial b_i}(b^T c) = c_i \implies \nabla(b^T c) = c$$

Hence $\nabla(b^T A^T y) = A^T y$

[7]We will discuss numerical optimization of functions of many variables in detail in Unit IV

The Normal Equations

Next consider $\nabla(b^T A^T A b)$ (note $A^T A$ is symmetric)

Consider $b^T M b$ for symmetric matrix $M \in \mathbb{R}^{n \times n}$

$$b^T M b = b^T \left(\sum_{j=1}^{n} m_{(:,j)} b_j \right)$$

From the product rule

$$\begin{aligned} \frac{\partial}{\partial b_k}(b^T M b) &= e_k^T \sum_{j=1}^{n} m_{(:,j)} b_j + b^T m_{(:,k)} \\ &= \sum_{j=1}^{n} m_{(k,j)} b_j + b^T m_{(:,k)} \\ &= m_{(k,:)} b + b^T m_{(:,k)} \\ &= 2 m_{(k,:)} b, \end{aligned}$$

where the last line follows from symmetry of M

Therefore, $\nabla(b^T M b) = 2Mb$, so that $\nabla(b^T A^T A b) = 2A^T A b$

The Normal Equations

Putting it all together, we obtain

$$\nabla\phi(b) = -2A^T y + 2A^T Ab$$

We set $\nabla\phi(b) = 0$ to obtain

$$-2A^T y + 2A^T Ab = 0 \Longrightarrow A^T Ab = A^T y$$

This square $n \times n$ system $A^T Ab = A^T y$ is known as the normal equations