# *Lecture Notes on Numerical Analysis*

## Virginia Tech · MATH/CS 5466 · Spring 2016

We model our world with continuous mathematics. Whether our interest is natural science, engineering, even finance and economics, the models we most often employ are functions of real variables. The equations can be linear or nonlinear, involve derivatives, integrals, combinations of these and beyond. The tricks and techniques one learns in algebra and calculus for solving such systems *exactly* cannot tackle the complexities that arise in serious applications. Exact solution may require an intractable amount of work; worse, for many problems, it is impossible to write an exact solution using elementary functions like polynomials, roots, trig functions, and logarithms.

This course tells a marvelous success story. Through the use of clever algorithms, careful analysis, and speedy computers, we can construct *approximate* solutions to these otherwise intractable problems with remarkable speed. Nick Trefethen defines *numerical analysis* to be 'the study of algorithms for the problems of continuous mathematics'.[1] This course takes a tour through many such algorithms, sampling a variety of techniques suitable across many applications. We aim to assess alternative methods based on both accuracy and efficiency, to discern well-posed problems from ill-posed ones, and to see these methods in action through computer implementation.

Perhaps the importance of numerical analysis can be best appreciated by realizing the impact its disappearance would have on our world. The space program would evaporate; aircraft design would be hobbled; weather forecasting would again become the stuff of soothsaying and almanacs. The ultrasound technology that uncovers cancer and illuminates the womb would vanish. Google couldn't rank web pages. Even the letters you are reading, whose shapes are specified by polynomial curves, would suffer. (Several important exceptions involve discrete, not continuous, mathematics: combinatorial optimization, cryptography and gene sequencing.)

On one hand, we are interested in *complexity*: we want algorithms that minimize the number of calculations required to compute a solution. But we are also interested in the *quality* of approximation: since we do not obtain exact solutions, we must understand the accuracy of our answers. Discrepancies arise from approximating a complicated function by a polynomial, a continuum by a discrete grid of points, or the real numbers by a finite set of floating point numbers. Different algorithms for the same problem will differ in the quality of their answers and the labor required to obtain those answers; we will

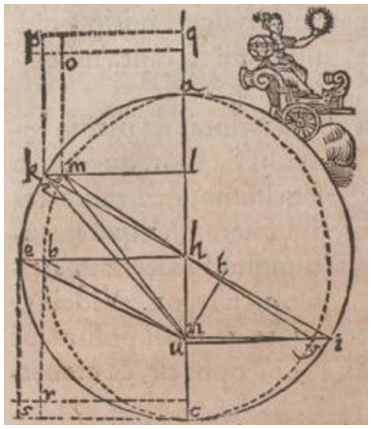


Image from Johannes Kepler's *Astronomia nova*, 1609, (ETH Bibliothek). In this text Kepler derives his famous equation that solves two-body orbital motion,

$$M = E - e \sin E,$$

where $M$ (the mean anomaly) and $e$ (the eccentricity) are known, and one solves for $E$ (the eccentric anomaly). This vital problem spurred the development of algorithms for solving nonlinear equations.

[1] We highly recommend Trefethen's essay, 'The Definition of Numerical Analysis', (reprinted on pages 321–327 of Trefethen & Bau, *Numerical Linear Algebra*), which inspires our present manifesto.

learn how to evaluate algorithms according to these criteria.

Numerical analysis forms the heart of 'scientific computing' or 'computational science and engineering,' fields that also encompass the high-performance computing technology that makes our algorithms practical for problems with millions of variables, visualization techniques that illuminate the data sets that emerge from these computations, and the applications that motivate them.

Though numerical analysis has flourished in the past seventy years, its roots go back centuries, where approximations were necessary in celestial mechanics and, more generally, 'natural philosophy'. Science, commerce, and warfare magnified the need for numerical analysis, so much so that the early twentieth century spawned the profession of 'computers,' people who conducted computations with hand-crank desk calculators. But numerical analysis has always been more than mere number-crunching, as observed by Alston Householder in the introduction to his *Principles of Numerical Analysis*, published in 1953, the end of the human computer era:

> The material was assembled with high-speed digital computation always in mind, though many techniques appropriate only to "hand" computation are discussed... How otherwise the continued use of these machines will transform the computer's art remains to be seen. But this much can surely be said, that their effective use demands a more profound understanding of the mathematics of the problem, and a more detailed acquaintance with the potential sources of error, than is ever required by a computation whose development can be watched, step by step, as it proceeds.

Thus the *analysis* component of 'numerical analysis' is essential. We rely on tools of classical real analysis, such as continuity, differentiability, Taylor expansion, and convergence of sequences and series.

Matrix computations play a fundamental role in numerical analysis. Discretization of continuous variables turns calculus into algebra. Algorithms for the fundamental problems in linear algebra are covered in MATH/CS 5465. If you have missed this beautiful content, your life will be poorer for it; when the methods we discuss this semester connect to matrix techniques, we will provide pointers.

These lecture notes were developed alongside courses that were supported by textbooks, such as *An Introduction to Numerical Analysis* by Süli and Mayers, *Numerical Analysis* by Gautschi, and *Numerical Analysis* by Kincaid and Cheney. These notes have benefited from this pedigree, and reflect certain hallmarks of these books. We have also been significantly influenced by G. W. Stewart's inspiring volumes, *Afternotes on Numerical Analysis* and *Afternotes Goes to Graduate School*. I am grateful for comments and corrections from past students, and welcome suggestions for further repair and amendment.

— Mark Embree

# 1
# *Interpolation*

## LECTURE 1: *Polynomial Interpolation in the Monomial Basis*

AMONG THE MOST FUNDAMENTAL problems in numerical analysis is the construction of a polynomial that approximates a continuous real function $f : [a, b] \to \mathbb{R}$. Of the several ways we might design such polynomials, we begin with *interpolation*: we will construct polynomials that exactly match $f$ at certain fixed points in the interval $[a, b] \subset \mathbb{R}$.

### 1.1 *Polynomial interpolation: definitions and notation*

**Definition 1.1.** The set of continuous functions that map $[a, b] \subset \mathbb{R}$ to $\mathbb{R}$ is denoted by $C[a, b]$. The set of continuous functions whose first $r$ derivatives are also continuous on $[a, b]$ is denoted by $C^r[a, b]$. (Note that $C^0[a, b] \equiv C[a, b]$.)

**Definition 1.2.** The set of polynomials of degree $n$ or less is denoted by $\mathcal{P}_n$.

Note that $C[a, b]$, $C^r[a, b]$ (for any $a < b$, $r \geq 0$) and $\mathcal{P}_n$ are *linear spaces* of functions (since linear combinations of such functions maintain continuity and polynomial degree). Furthermore, note that $\mathcal{P}_n$ is an $n + 1$ dimensional subspace of $C[a, b]$.

The polynomial interpolation problem can be stated as:

> Given $f \in C[a, b]$ and $n + 1$ points $\{x_j\}_{j=0}^n$ satisfying
>
> $$a \leq x_0 < x_1 < \cdots < x_n \leq b,$$
>
> determine some $p_n \in \mathcal{P}_n$ such that
>
> $$p_n(x_j) = f(x_j) \qquad \text{for } j = 0, \ldots, n.$$

We freely use the concept of *vector spaces*. A set of functions $\mathcal{V}$ is a real vector space it is closed under vector addition and multiplication by a real number: for any $f, g \in \mathcal{V}$, $f + g \in \mathcal{V}$, and for any $f \in \mathcal{V}$ and $\alpha \in \mathbb{R}$, $\alpha f \in \mathcal{V}$. For more details, consult a text on linear algebra or functional analysis.

It shall become clear why we require $n+1$ points $x_0, \ldots, x_n$, and no more, to determine a degree-$n$ polynomial $p_n$. (You know the $n = 1$ case well: two points determine a unique line.) If the number of data points were smaller, we could construct infinitely many degree-$n$ interpolating polynomials. Were it larger, there would in general be no degree-$n$ interpolant.

As numerical analysts, we seek answers to the following questions:

- Does such a polynomial $p_n \in \mathcal{P}_n$ *exist*?
- If so, is it *unique*?
- Does $p_n \in \mathcal{P}_n$ behave like $f \in C[a,b]$ at points $x \in [a,b]$ when $x \neq x_j$ for $j = 0, \ldots, n$?
- How can we compute $p_n \in \mathcal{P}_n$ *efficiently* on a computer?
- How can we compute $p_n \in \mathcal{P}_n$ *accurately* on a computer (with floating point arithmetic)?
- If we want to add a new interpolation point $x_{n+1}$, can we easily adjust $p_n$ to give an interpolating polynomial $p_{n+1}$ of one higher degree?
- How should the interpolation points $\{x_j\}$ be chosen?

Regarding this last question, we should note that, in practice, we are not always able to choose the interpolation points as freely as we might like. For example, our 'continuous function $f \in C[a,b]$' could actually be a discrete list of previously collected experimental data, and we are stuck with the values $\{x_j\}_{j=0}^n$ at which the data was measured.

## 1.2 *Constructing interpolants in the monomial basis*

Of course, any polynomial $p_n \in \mathcal{P}_n$ can be written in the form

$$p_n(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

for coefficients $c_0, c_1, \ldots, c_n$. We can view this formula as an expression for $p_n$ as a linear combination of the *basis functions* $1$, $x$, $x^2$, $\ldots$, $x^n$; these basis functions are called *monomials*.

To construct the polynomial interpolant to $f$, we merely need to determine the proper values for the coefficients $c_0, c_1, \ldots, c_n$ in the above expansion. The interpolation conditions $p_n(x_j) = f(x_j)$ for

$j = 0, \ldots, n$ reduce to the equations

$$\begin{aligned} c_0 + c_1 x_0 + c_2 x_0^2 + \cdots + c_n x_0^n &= f(x_0) \\ c_0 + c_1 x_1 + c_2 x_1^2 + \cdots + c_n x_1^n &= f(x_1) \\ &\vdots \\ c_0 + c_1 x_n + c_2 x_n^2 + \cdots + c_n x_n^n &= f(x_n). \end{aligned}$$

Note that these $n+1$ equations are linear in the $n+1$ unknown parameters $c_0, \ldots, c_n$. Thus, our problem of finding the coefficients $c_0, \ldots, c_n$ reduces to solving the linear system

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}, \tag{1.1}$$

which we denote as $\mathbf{Ac} = \mathbf{f}$. Matrices of this form, called *Vandermonde* matrices, arise in a wide range of applications.[1] Provided all the interpolation points $\{x_j\}$ are distinct, one can show that this matrix is invertible.[2] Hence, fundamental properties of linear algebra allow us to confirm that there is exactly one degree-$n$ polynomial that interpolates $f$ at the given $n+1$ distinct interpolation points.

**Theorem 1.1.** Given $f \in C[a,b]$ and distinct points $\{x_j\}_{j=0}^n$, $a \leq x_0 < x_1 < \cdots < x_n \leq b$, there exists a unique $p_n \in \mathcal{P}_n$ such that $p_n(x_j) = f(x_j)$ for $j = 0, 1, \ldots, n$.

To determine the coefficients $\{c_j\}$, we could solve the above linear system with the Vandermonde matrix using some variant of Gaussian elimination (e.g., using MATLAB's \ command); this will take $\mathcal{O}(n^3)$ floating point operations. Alternatively, we could (and should) use a specialized algorithm that exploit the Vandermonde structure to determine the coefficients $\{c_j\}$ in only $\mathcal{O}(n^2)$ operations, a vast improvement.[3]

### 1.2.1 *Potential pitfalls of the monomial basis*

Though it is straightforward to see how to construct interpolating polynomials in the monomial basis, this procedure can give rise to some unpleasant numerical problems when we actually attempt to determine the coefficients $\{c_j\}$ on a computer. The primary difficulty is that the monomial basis functions $1$, $x$, $x^2$, $\ldots$, $x^n$ look increasingly alike as we take higher and higher powers. Figure 1.1 illustrates this behavior on the interval $[a,b] = [0,1]$ with $n = 5$ and $x_j = j/5$.

[1] Higham presents many interesting properties of Vandermonde matrices and algorithms for solving Vandermonde systems in Chapter 21 of *Accuracy and Stability of Numerical Algorithms*, 2nd ed., (SIAM, 2002). Vandermonde matrices arise often enough that MATLAB has a built-in command for creating them. If $\mathtt{x} = [x_0, \ldots, x_n]^T$, then $\mathbf{A} = \mathtt{fliplr(vander(x))}$.

[2] In fact, the determinant takes the simple form

$$\det(\mathbf{A}) = \prod_{j=0}^{n} \prod_{k=j+1}^{n} (x_k - x_j).$$

This is evident for $n = 1$; we will not prove if for general $n$, as we will have more elegant ways to establish existence and uniqueness of polynomial interpolants. For a clever proof, see p. 193 of Bellman, *Introduction to Matrix Analysis*, 2nd ed., (McGraw-Hill, 1970).

[3] See Higham's book for details and stability analysis of specialized Vandermonde algorithms.

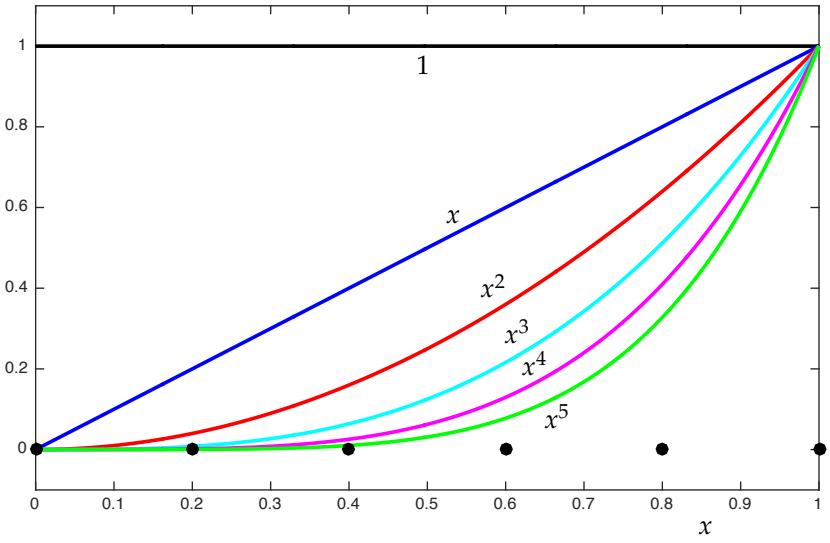



Figure 1.1: The six monomial basis vectors for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (red circles). Note that the basis vectors increasingly align as the power increases: this basis becomes *ill-conditioned* as the degree of the interpolant grows.

Because these basis vectors become increasingly alike, one finds that the expansion coefficients $\{c_j\}$ in the monomial basis can become very large in magnitude even if the function $f(x)$ remains of modest size on $[a, b]$.

Consider the following analogy from linear algebra. The vectors

$$\begin{bmatrix} 1 \\ 10^{-10} \end{bmatrix}, \qquad \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

form a basis for $\mathbb{R}^2$. However, both vectors point in *nearly* the same direction, though of course they are *linearly independent*. We can write the vector $[1, 1]^T$ as a unique linear combination of these basis vectors:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 10,000,000,000 \begin{bmatrix} 1 \\ 10^{-10} \end{bmatrix} - 9,999,999,999 \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{1.2}$$

Although the vector we are expanding and the basis vectors themselves are all have modest size (norm), the expansion coefficients are enormous. Furthermore, small changes to the vector we are expanding will lead to huge changes in the expansion coefficients. This is a recipe for disaster when computing with finite-precision arithmetic.

This same phenomenon can occur when we express polynomials in the monomial basis. As a simple example, consider interpolating $f(x) = 2x + x\sin(40x)$ at uniformly spaced points ($x_j = j/n$, $j = 0, \ldots, n$) in the interval $[0, 1]$. Note that $f \in C^\infty[0, 1]$: this $f$ is a 'nice' function with infinitely many continuous derivatives. As seen in Figures 1.2–1.3, $f$ oscillates modestly on the interval $[0, 1]$, but it

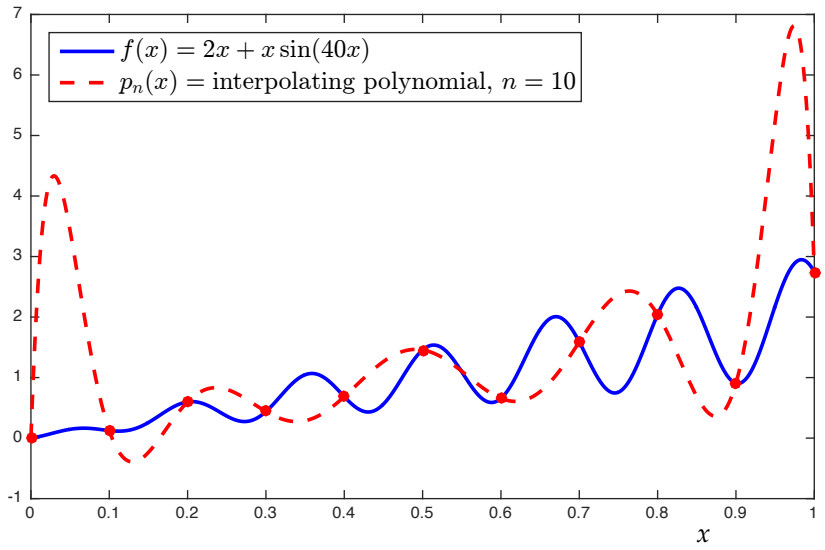



Figure 1.2: Degree $n = 10$ interpolant $p_{10}(x)$ to $f(x) = 2x + x\sin(40x)$ at the uniformly spaced points $x_0, \ldots, x_{10}$ for $x_j = j/10$ over $[a, b] = [0, 1]$. Even though $p_{10}(x_j) = f(x_j)$ at the eleven points $x_0, \ldots, x_n$ (red circles), the interpolant gives a poor approximation to $f$ at the ends of the interval.

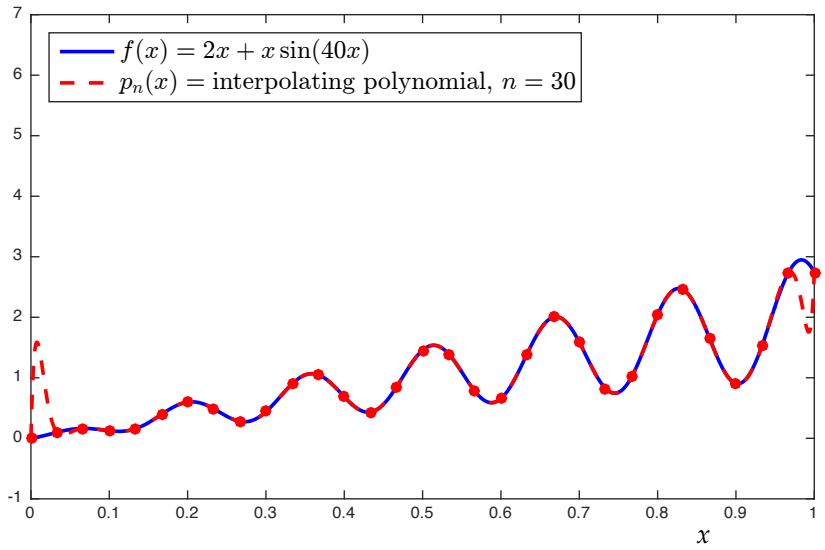



Figure 1.3: Repetition of Figure 1.2, but now with the degree $n = 30$ interpolant at uniformly spaced points $x_j = j/30$ on $[0, 1]$. The polynomial still overshoots $f$ near $x = 0$ and $x = 1$, though by less than for $n = 10$; for this example, the overshoot goes away as $n$ is increased further.

certainly does not grow excessively large in magnitude or exhibit any nasty singularities.

Comparing the interpolants with $n = 10$ and $n = 30$ between the two figures, it appears that, in some sense, $p_n \to f$ as $n$ increases. Indeed, this is the case, in a manner we shall make precise in future lectures.

However, we must address a crucial question:

> *Can we accurately compute the coefficients $c_0, \ldots, c_n$ that specify the interpolating polynomial?*

Use MATLAB's basic Gaussian elimination algorithm to solve the Vandermonde system $\mathbf{A}\mathbf{c} = \mathbf{f}$ for $\mathbf{c}$ via the command `c = A\f`, then evaluate

$$p_n(x) = \sum_{j=0}^{n} c_j x^j$$

e.g., using MATLAB's `polyval` command.

Since $p_n$ was constructed to interpolate $f$ at the points $x_0, \ldots, x_n$, we might (at the very least!) expect

$$f(x_j) - p_n(x_j) = 0, \qquad j = 0, \ldots, n.$$

Since we are dealing with numerical computations with a finite precision floating point system, we should instead be well satisfied if our numerical computations only achieve $|f(x_j) - p_n(x_j)| = \mathcal{O}(\varepsilon_{\text{mach}})$, where $\varepsilon_{\text{mach}}$ denotes the precision of the floating point arithmetic system.[4]

More precisely, we might expect

$$|f(x_j) - p_n(x_j)| \approx \varepsilon_{\text{mach}} \|f\|_{L_\infty},$$

where $\|f\|_{L_\infty} := \max_{x \in [a,b]} |f(x)|$.

Instead, the results of our numerical computations are *remarkably inaccurate* due to the magnitude of the coefficients $c_0, \ldots, c_n$ and the ill-conditioning of the Vandermonde matrix.

Recall from numerical linear algebra that the accuracy of solving the system $\mathbf{A}\mathbf{c} = \mathbf{f}$ depends on the *condition number* $\|\mathbf{A}\| \|\mathbf{A}^{-1}\|$ of $\mathbf{A}$.[5] Figure 1.4 shows that this condition number grows *exponentially* as $n$ increases.[6] Thus, we should expect the computed value of $\mathbf{c}$ to have errors that scale like $\|\mathbf{A}\| \|\mathbf{A}^{-1}\| \varepsilon_{\text{mach}}$. Moreover, consider the entries in $\mathbf{c}$. For $n = 10$ (a typical example), we have

| $j$ | $c_j$ |
|---|---|
| 0 | 0.00000 |
| 1 | 363.24705 |
| 2 | −10161.84204 |
| 3 | 113946.06962 |
| 4 | −679937.11016 |
| 5 | 2411360.82690 |
| 6 | −5328154.95033 |
| 7 | 7400914.85455 |
| 8 | −6277742.91579 |
| 9 | 2968989.64443 |
| 10 | −599575.07912 |

The entries in $\mathbf{c}$ *grow in magnitude and oscillate in sign*, akin to the simple $\mathbb{R}^2$ vector example in (1.2). The sign-flips and magnitude of the coefficients would make

$$p_n(x) = \sum_{j=0}^{n} c_j x^j$$

[4] For the double-precision arithmetic used by MATLAB, $\varepsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$.

[5] For information on conditioning and the accuracy of solving linear systems, see, e.g.., Lecture 12 of Trefethen and Bau, *Numerical Linear Algebra* (SIAM, 1997).

[6] The curve has very regular behavior up until about $n = 20$; beyond that point, where $\|\mathbf{A}\| \|\mathbf{A}^{-1}\| \approx 1/\varepsilon_{\text{mach}}$, the computation is sufficiently unstable that the condition number is no longer computed accurately! For $n > 20$, take all the curves in Figure 1.4 with a healthy dose of salt.

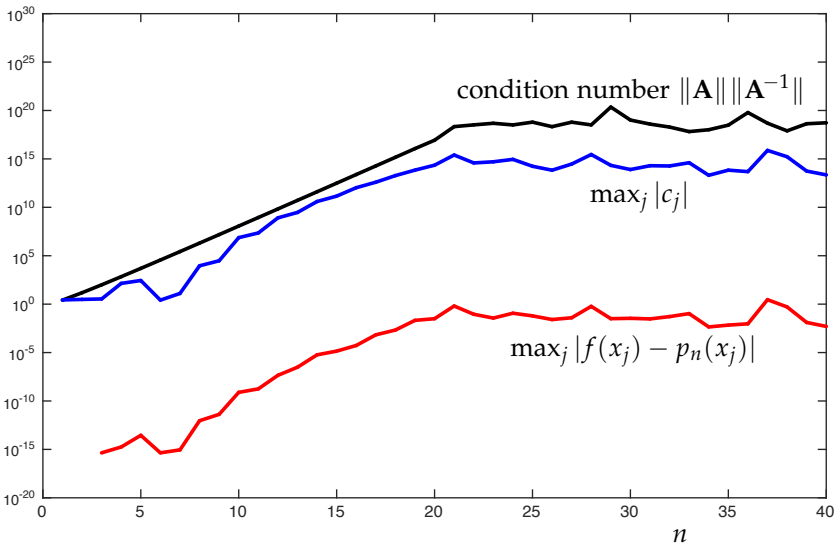



Figure 1.4: Illustration of some pitfalls of working with interpolants in the monomial basis for large $n$: (a) the condition number of $\mathbf{A}$ grows large with $n$; (b) as a result, some coefficients $c_j$ are large in magnitude (blue line) and inaccurately computed; (c) consequently, the computed 'interpolant' $p_n$ is far from $f$ at the interpolation points (red line): *this red curve should be zero*!

difficult to compute accurately for large $n$, even if all the coefficients $c_0, \ldots, c_n$ were given exactly. Figure 1.4 shows how the largest computed value in $\mathbf{c}$ grows with $n$. Finally, this figure also shows the quantity we began discussing,

$$\max_{0 \le j \le n} |f(x_j) - p_n(x_j)|.$$

Rather than being nearly zero, this quantity grows with $n$, until the computed 'interpolating' polynomial differs from $f$ at some interpolation point by roughly $1/10$ for the larger values of $n$: we must have higher standards!

This is an example where a simple problem formulation quickly yields an algorithm, but that algorithm gives unacceptable numerical results.

Perhaps you are now troubled by this entirely reasonable question: If the computations of $p_n$ are as unstable as Figure 1.4 suggests, why should we put any faith in the plots of interpolants for $n = 10$ and, especially, $n = 30$ in Figures 1.2–1.3?

You should trust those plots because I computed them using a much better approach, about which we shall next learn.

# LECTURE 2: *Superior Bases for Polynomial Interpolants*

## 1.3 *Polynomial interpolants in a general basis*

THE MONOMIAL BASIS may seem like the most natural way to write down the interpolating polynomial, but it can lead to numerical problems, as seen in the previous lecture. To arrive at more stable expressions for the interpolating polynomial, we will derive several different bases for $\mathcal{P}_n$ that give superior computational properties: the expansion coefficients $\{c_j\}$ will typically be smaller, and it will be simpler to determine those coefficients. This is an instance of a general principle of applied mathematics: to promote stability, express your problem in a well-conditioned basis.

Suppose we have some *basis* $\{b_j\}_{j=0}^n$ for $\mathcal{P}_n$. We seek the polynomial $p \in \mathcal{P}_n$ that interpolates $f$ at $x_0, \ldots, x_n$. Write $p$ in the basis as

$$p(x) = c_0 b_0(x) + c_1 b_1(x) + \cdots + c_n b_n(x).$$

Recall that $\{b_j\}_{j=0}^n$ is a *basis* if the functions *span* $\mathcal{P}_n$ and are *linearly independent*. The first requirement means that for any polynomial $p \in \mathcal{P}_n$ we can find constants $c_0, \ldots, c_n$ such that

$$p = c_0 b_0 + \cdots + c_n b_n,$$

while the second requirement means that if

$$0 = c_0 b_0 + \cdots + c_n b_n$$

then we must have $c_0 = \cdots = c_n = 0$.

We seek the coefficients $c_0, \ldots, c_n$ that express the interpolant $p$ in this basis. The interpolation conditions are

$$\begin{aligned}
p(x_0) &= c_0 b_0(x_0) + c_1 b_1(x_0) + \cdots + c_n b_n(x_0) = f(x_0) \\
p(x_1) &= c_0 b_0(x_1) + c_1 b_1(x_1) + \cdots + c_n b_n(x_1) = f(x_1) \\
&\vdots \\
p(x_n) &= c_0 b_0(x_n) + c_1 b_1(x_n) + \cdots + c_n b_n(x_n) = f(x_n).
\end{aligned}$$

Again we have $n+1$ equations that are linear in the $n+1$ unknowns $c_0, \ldots, c_n$, hence we can arrange these in the matrix form

$$\begin{bmatrix} b_0(x_0) & b_1(x_0) & \cdots & b_n(x_0) \\ b_0(x_1) & b_1(x_1) & \cdots & b_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ b_0(x_n) & b_1(x_n) & \cdots & b_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}, \tag{1.3}$$

which can be solved via Gaussian elimination for $c_0, \ldots, c_n$.

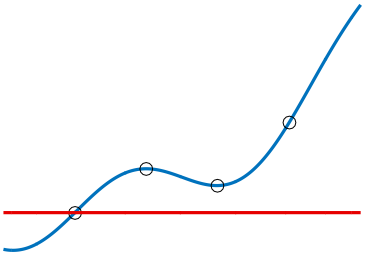
Notice that the linear system for the monomial basis in (1.1) is a special case of the system in (1.3), with the choice $b_j(x) = x^j$. Next we will look at two superior bases that give more stable expressions for the interpolant. We emphasize that when the basis changes, so to do the values of $c_0, \ldots, c_n$, *but the interpolating polynomial $p$ remains the same, regardless of the basis we use to express it*.

## 1.4 *Constructing interpolants in the Newton basis*

To derive our first new basis for $\mathcal{P}_n$, we describe an alternative method for constructing the polynomial $p_n \in \mathcal{P}_n$ that interpolates $f \in C[a,b]$ at the distinct points $\{x_0, \ldots, x_n\} \subset [a,b]$. This approach, called the *Newton form* of the interpolant, builds $p_n$ up from lower degree polynomials that interpolate $f$ at only some of the data points.

Begin by constructing the polynomial $p_0 \in \mathcal{P}_0$ that interpolates $f$ at $x_0$: $p_0(x_0) = f(x_0)$. Since $p_0$ is a zero-degree polynomial (i.e., a constant), it has the simple form

$$p_0(x) = c_0.$$



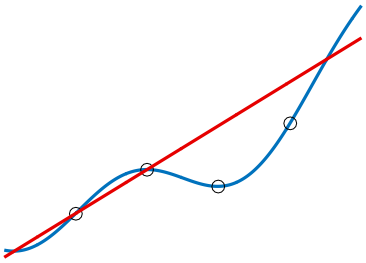
To satisfy the interpolation condition at $x_0$, set $c_0 = f(x_0)$. (We emphasize again: this $c_0$, and the $c_j$ below, will be different from the $c_j$'s obtained in Section 1.2 for the monomial basis.)

Next, use $p_0$ to build the polynomial $p_1 \in \mathcal{P}_1$ that interpolates $f$ at both $x_0$ and $x_1$. In particular, we will require $p_1$ to have the form

$$p_1(x) = p_0(x) + c_1 q_1(x)$$

for some constant $c_1$ and some $q_1 \in \mathcal{P}_1$. Note that

$$\begin{aligned} p_1(x_0) &= p_0(x_0) + c_1 q_1(x_0) \\ &= f(x_0) + c_1 q_1(x_0). \end{aligned}$$



Since we require that $p_1(x_0) = f(x_0)$, the above equation implies that $c_1 q_1(x_0) = 0$. Either $c_1 = 0$ (which can only happen in the special case $f(x_0) = f(x_1)$, and we seek a basis that works for *any* $f$) or $q_1(x_0) = 0$, i.e., $q_1(x_0)$ has a root at $x_0$. Thus, we deduce that $q_1(x) = x - x_0$. It follows that

$$p_1(x) = c_0 + c_1(x - x_0),$$

where $c_1$ is still undetermined. To find $c_1$, use the interpolation condition at $x_1$:

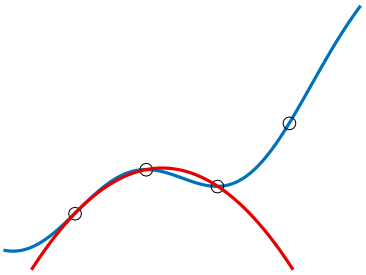
$$f(x_1) = p_1(x_1) = c_0 + c_1(x_1 - x_0).$$

Solving for $c_1$,

$$c_1 = \frac{f(x_1) - c_0}{x_1 - x_0}.$$

Next, find the $p_2 \in \mathcal{P}_2$ that interpolates $f$ at $x_0$, $x_1$, and $x_2$, where $p_2$ has the form

$$p_2(x) = p_1(x) + c_2 q_2(x).$$



Similar to before, the first term, now $p_1(x)$, 'does the right thing' at the first two interpolation points, $p_1(x_0) = f(x_0)$ and $p_1(x_1) = f(x_1)$.

We require that $q_2$ not interfere with $p_1$ at $x_0$ and $x_1$, i.e., $q_2(x_0) = q_2(x_1) = 0$. Thus, we take $q_2$ to have the form

$$q_2(x) = (x - x_0)(x - x_1).$$

The interpolation condition at $x_2$ gives an equation where $c_2$ is the only unknown,

$$f(x_2) = p_2(x_2) = p_1(x_2) + c_2 q_2(x_2),$$

which we can solve for

$$c_2 = \frac{f(x_2) - p_1(x_2)}{q_2(x_2)} = \frac{f(x_2) - c_0 - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}.$$

Follow the same pattern to bootstrap up to $p_n$, which takes the form

$$p_n(x) = p_{n-1}(x) + c_n q_n(x),$$

where

$$q_n(x) = \prod_{j=0}^{n-1} (x - x_j),$$

and, setting $q_0(x) = 1$, we have

$$c_n = \frac{f(x_n) - \sum_{j=0}^{n-1} c_j q_j(x_n)}{q_n(x_n)}.$$

Finally, the desired polynomial takes the form

$$p_n(x) = \sum_{j=0}^{n} c_j q_j(x).$$

The polynomials $q_j$ for $j = 0, \ldots, n$ form a basis for $\mathcal{P}_n$, called the *Newton basis*. The $c_j$ we have just determined are the expansion coefficients for this interpolant in the Newton basis. Figure 1.5 shows the Newton basis functions $q_j$ for $[a, b] = [0, 1]$ with $n = 5$ and $x_j = j/5$, which look considerably more distinct than the monomial basis polynomials illustrated in Figure 1.1.

This entire procedure for constructing $p_n$ can be condensed into a system of linear equations with the coefficients $\{c_j\}_{j=0}^n$ unknown:

$$\begin{bmatrix} 1 & & & & \\ 1 & (x_1 - x_0) & & & \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \cdots & \prod_{j=0}^{n-1}(x_n - x_j) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}, \tag{1.4}$$

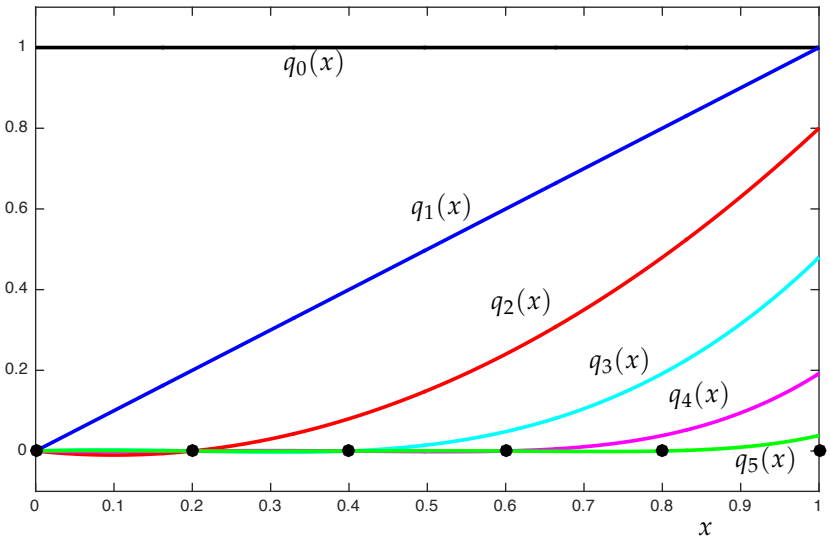



Figure 1.5: The six Newton basis polynomials $q_0, \ldots, q_5$ for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (black dots). Compare these to the monomial basis polynomials in Figure 1.1: these vectors look far more distinct from one another than the monomials.

again a special case of (1.3) but with $b_j(x) = q_j(x)$. (The unspecified entries above the diagonal are zero, since $q_j(x_k) = 0$ when $k < j$.) The system (1.4) involves a triangular matrix, which is simple to solve. Clearly $c_0 = f(x_0)$, and once we know $c_0$, we can solve for

$$c_1 = \frac{f(x_1) - c_0}{x_1 - x_0}.$$

With $c_0$ and $c_1$, we can solve for $c_2$, and so on. This procedure, *forward substitution*, requires roughly $n^2$ floating point operations once the entries are formed.

With this Newton form of the interpolant, one can easily update $p_n$ to $p_{n+1}$ in order to incorporate a new data point $(x_{n+1}, f(x_{n+1}))$, as such a change affects neither the previous values of $c_j$ nor $q_j$. The new data $(x_{n+1}, f(x_{n+1}))$ simply adds a new row to the bottom of the matrix in (1.4), which preserves the triangular structure of the matrix and the values of $\{c_0, \ldots, c_n\}$. If we have already found these coefficients, we easily obtain $c_{n+1}$ through one more step of forward substitution.

## 1.5 *Constructing interpolants in the Lagrange basis*

The monomial basis gave us a linear system (1.1) of the form $\mathbf{Ac} = \mathbf{f}$ in which $\mathbf{A}$ was a *dense* matrix: all of its entries are nonzero. The Newton basis gave a simpler system (1.4) in which $\mathbf{A}$ was a *lower triangular* matrix. Can we go one step further, and find a set of basis functions for which the matrix in (1.3) is *diagonal*?

For the matrix to be diagonal, the $j$th basis function would need to have roots at all the other interpolation points $x_k$ for $k \neq j$. Such func-

tions, denoted $\ell_j$ for $j = 0, \ldots, n$, are called *Lagrange basis polynomials*, and they result in the *Lagrange form* of the interpolating polynomial.

We seek to construct $\ell_j \in \mathcal{P}_n$ with $\ell_j(x_k) = 0$ if $j \neq k$, but $\ell_j(x_k) = 1$ if $j = k$. That is, $\ell_j$ takes the value one at $x_j$ and has roots at all the other $n$ interpolation points.

What form do these basis functions $\ell_j \in \mathcal{P}_n$ take? Since $\ell_j$ is a degree-$n$ polynomial with the $n$ roots $\{x_k\}_{k=0,k\neq j}^{n}$, it can be written in the form

$$\ell_j(x) = \prod_{k=0,k\neq j}^{n} \gamma_k (x - x_k)$$

for appropriate constants $\gamma_k$. We can force $\ell_j(x_j) = 1$ if all the terms in the above product are one when $x = x_j$, i.e., when $\gamma_k = 1/(x_j - x_k)$, so that

$$\ell_j(x) = \prod_{k=0,k\neq j}^{n} \frac{x - x_k}{x_j - x_k}.$$

This form makes it clear that $\ell_j(x_j) = 1$. With these new basis functions, the constants $\{c_j\}$ can be written down immediately. The interpolating polynomial has the form

$$p_n(x) = \sum_{k=0}^{n} c_k \ell_k(x).$$

When $x = x_j$, all terms in this sum will be zero except for one, the $k = j$ term (since $\ell_k(x_j) = 0$ except when $j = k$). Thus,

$$p_n(x_j) = c_j \ell_j(x_j) = c_j,$$

so we can directly write down the coefficients, $c_j = f(x_j)$.

As desired, this approach to constructing basis polynomials leads to a diagonal matrix $\mathbf{A}$ in the equation $\mathbf{Ac} = \mathbf{f}$ for the coefficients. Since we also insisted that $\ell_j(x_j) = 1$, the matrix $\mathbf{A}$ is actually just the *identity* matrix:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

Now the coefficient matrix is simply the identity.

A forthcoming exercise will investigate an important flexible and numerically stable method for constructing and evaluating Lagrange interpolants known as *barycentric interpolation*.

Figure 1.6 shows the Lagrange basis functions for $n = 5$ with $[a, b] = [0, 1]$ and $x_j = j/5$, the same parameters used in the plots of the monomial and Newton bases earlier.

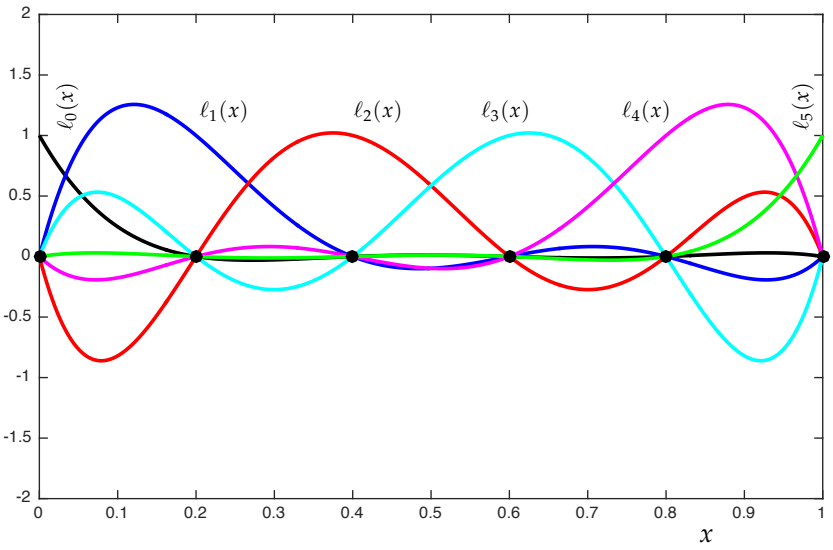



Figure 1.6: The six Lagrange basis polynomials $\ell_0, \ldots, \ell_5$ for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (black dots). Note that each Lagrange polynomial has roots at $n$ of the interpolation points. Compare these polynomials to the monomial and Newton basis polynomials in Figures 1.1 and 1.5 (but note the different vertical scale): these basis vectors look most independent of all.

The fact that these basis functions are not as closely aligned as the previous ones has interesting consequences on the size of the coefficients $\{c_j\}$. For example, if we have $n + 1 = 6$ interpolation points for $f(x) = \sin(10x) + \cos(10x)$ on $[0, 1]$, we obtain the following coefficients:

| | monomial | Newton | Lagrange |
|---|---|---|---|
| $c_0$ | 1.0000000e+00 | 1.0000000e+00 | 1.0000000e+00 |
| $c_1$ | 4.0861958e+01 | -2.5342470e+00 | 4.9315059e-01 |
| $c_2$ | -3.8924180e+02 | -1.7459341e+01 | -1.4104461e+00 |
| $c_3$ | 1.0775024e+03 | 1.1232385e+02 | 6.8075479e-01 |
| $c_4$ | -1.1683645e+03 | -2.9464687e+02 | 8.4385821e-01 |
| $c_5$ | 4.3685881e+02 | 4.3685881e+02 | -1.3830926e+00 |

We emphasize that all three approaches (in exact arithmetic) must yield the same unique polynomial, but they are expressed in different bases. The behavior in floating point arithmetic varies significantly with the choice of basis; the monomial basis is the clear loser.

# LECTURE 3: *Interpolation Error Bounds*

## *1.6 Convergence theory for polynomial interpolation*

Interpolation can be used to generate low-degree polynomials that approximate a complicated function over the interval $[a, b]$. One might assume that the more data points that are interpolated (for a fixed $[a, b]$), the more accurate the resulting approximation. In this lecture, we address the behavior of the maximum error

$$\max_{x \in [a,b]} |f(x) - p_n(x)|$$

as the number of interpolation points—hence, the degree of the interpolating polynomial—is increased. We begin with a theoretical result.

---

**Theorem 1.2** (Weierstrass Approximation Theorem)**.**
Suppose $f \in C[a, b]$. For any $\varepsilon > 0$ there exists some polynomial $p_n$ of finite degree $n$ such that $\max_{x \in [a,b]} |f(x) - p_n(x)| \leq \varepsilon$.

---

Unfortunately, we do not have time to prove this in class.[7] As stated, this theorem gives no hint about what the approximating polynomial looks like, whether $p_n$ interpolates $f$ at $n + 1$ points, or merely approximates $f$ well throughout $[a, b]$, nor does the Weierstrass theorem describe the accuracy of a polynomial for a specific value of $n$ (though one could gain insight into such questions by studying the constructive proof).

[7] The typical proof is a construction based on Bernstein polynomials; see, e.g., Kincaid and Cheney, *Numerical Analysis*, 3rd edition, pages 320–323. This result can be generalized to the Stone–Weierstrass Theorem, itself a special case of Bishop's Theorem for approximation problems in operator algebras; see e.g., §5.6–§5.8 of Rudin, *Functional Analysis*, 2nd ed. (McGraw Hill, 1991).

On the other hand, for the interpolation problem studied in the preceding lectures, we can obtain a specific error formula that gives a bound on $\max_{x \in [a,b]} |f(x) - p_n(x)|$. From this bound, we can deduce if interpolating $f$ at increasingly many points will eventually yield a polynomial approximation to $f$ that is accurate to any specified precision.

For any $\widehat{x} \in [a, b]$ that is *not* of the interpolation points, we seek to measure the error

$$f(\widehat{x}) - p_n(\widehat{x}),$$

where $p_n \in \mathcal{P}_n$ is the interpolant to $f$ at the distinct points $x_0, \ldots, x_n \in [a, b]$. We can get a grip on this error from the following perspective. Extend $p_n$ by one degree to give a new polynomial that additionally interpolates $f$ at $\widehat{x}$. This is easy to do with the Newton form of the interpolant; write the new polynomial as

$$p_n(x) + \lambda \prod_{j=0}^{n} (x - x_j),$$

for constant $\lambda$ chosen so that

$$f(\widehat{x}) = p_n(\widehat{x}) + \lambda \prod_{j=0}^{n} (\widehat{x} - x_j).$$

For convenience, we write

$$w(x) := \prod_{j=0}^{n} (x - x_j),$$

so we could solve for $\lambda$ as

$$\lambda = \frac{f(\widehat{x}) - p_n(\widehat{x})}{w(\widehat{x})}. \tag{1.5}$$

Now notice that

$$0 = f(\widehat{x}) - \big(p_n(\widehat{x}) + \lambda w(\widehat{x})\big)$$

implies

$$f(\widehat{x}) - p_n(\widehat{x}) = \lambda w(\widehat{x}), \tag{1.6}$$

which is an expression for the desired error $f(\widehat{x}) - p_n(\widehat{x})$. Unfortunately, the formula (1.5) does not give much insight into how the error behaves as a function of $f$ and the interpolation points.

---

**Theorem 1.3** (Interpolation Error Formula)**.**
Suppose $f \in C^{n+1}[a,b]$ and let $p_n \in \mathcal{P}_n$ denote the polynomial that interpolates $\{(x_j, f(x_j))\}_{j=0}^{n}$ for distinct points $x_j \in [a,b]$, $j = 0, \ldots, n$. Then for every $x \in [a,b]$ there exists $\xi \in [a,b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j).$$

From this formula follows a bound for the worst error over $[a,b]$:

$$\max_{x\in[a,b]} |f(x) - p_n(x)| \le \left( \max_{\xi\in[a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \right) \left( \max_{x\in[a,b]} \prod_{j=0}^{n} |x - x_j| \right). \tag{1.7}$$

---

We shall carefully prove this essential result; it will repay the effort, for this theorem becomes the foundation upon which we shall build the convergence theory for piecewise polynomial approximation and interpolatory quadrature rules for definite integrals.

*Proof.* Consider some arbitrary point $\widehat{x} \in [a,b]$. We seek a descriptive expression for the error $f(\widehat{x}) - p_n(\widehat{x})$. If $\widehat{x} = x_j$ for some $j \in \{0, \ldots, n\}$, then $f(\widehat{x}) - p_n(\widehat{x}) = 0$ and there is nothing to prove. Thus, suppose for the rest of the proof that $\widehat{x}$ is not one of the interpolation points.

To describe $f(\widehat{x}) - p_n(\widehat{x})$, we shall build the polynomial of degree $n+1$ that interpolates $f$ at $x_0, \ldots, x_n$, and also $\widehat{x}$. Of course, this polynomial will give zero error at $\widehat{x}$, since it interpolates $f$ there. From this polynomial we can extract a formula for $f(\widehat{x}) - p_n(\widehat{x})$ by measuring how much the degree $n+1$ interpolant improves upon the degree-$n$ interpolant $p_n$ at $\widehat{x}$.

Since we wish to understand the relationship of the degree $n+1$ interpolant to $p_n$, we shall write that degree $n+1$ interpolant in a manner that explicitly incorporates $p_n$. Given this setting, use of the Newton form of the interpolant is natural; i.e., we write the degree $n+1$ polynomial as

$$p_n(x) + \lambda \prod_{j=0}^{n} (x - x_j)$$

for some constant $\lambda$ chosen to make the interpolant exact at $\widehat{x}$. For convenience, we write

$$w(x) \equiv \prod_{j=0}^{n} (x - x_j)$$

and then denote the error of this degree $n+1$ interpolant by

$$\phi(x) \equiv f(x) - \big(p_n(x) + \lambda w(x)\big).$$

To make the polynomial $p_n(x) + \lambda w(x)$ interpolate $f$ at $\widehat{x}$, we shall pick $\lambda$ such that $\phi(\widehat{x}) = 0$. The fact that $\widehat{x} \notin \{x_j\}_{j=0}^{n}$ ensures that $w(\widehat{x}) \neq 0$, and so we can force $\phi(\widehat{x}) = 0$ by setting

$$\lambda = \frac{f(\widehat{x}) - p_n(\widehat{x})}{w(\widehat{x})}.$$

Furthermore, since $f(x_j) = p_n(x_j)$ and $w(x_j) = 0$ at all the $n+1$ interpolation points $x_0, \ldots, x_n$, we also have $\phi(x_j) = f(x_j) - p_n(x_j) - \lambda w(x_j) = 0$. Thus, $\phi$ is a function with at least $n+2$ zeros in the interval $[a, b]$. Rolle's Theorem[8] tells us that between every two consecutive zeros of $\phi$, there is some zero of $\phi'$. Since $\phi$ has at least $n+2$ zeros in $[a, b]$, $\phi'$ has at least $n+1$ zeros in this same interval. We can repeat this argument with $\phi'$ to see that $\phi''$ must have at least $n$ zeros in $[a, b]$. Continuing in this manner with higher derivatives, we eventually conclude that $\phi^{(n+1)}$ must have at least one zero in $[a, b]$; we denote this zero as $\xi$, so that $\phi^{(n+1)}(\xi) = 0$.

[8] Recall the Mean Value Theorem from calculus: Given $d > c$, suppose $f \in C[c, d]$ is differentiable on $(c, d)$. Then there exists some $\eta \in (c, d)$ such that $(f(d) - f(c))/(d - c) = f'(\eta)$. Rolle's Theorem is a special case: If $f(d) = f(c)$, then there is some point $\eta \in (c, d)$ such that $f'(\eta) = 0$.

We now want a more concrete expression for $\phi^{(n+1)}$. Note that

$$\phi^{(n+1)}(x) = f^{(n+1)}(x) - p_n^{(n+1)}(x) - \lambda w^{(n+1)}(x).$$

Since $p_n$ is a polynomial of degree $n$ or less, $p_n^{(n+1)} \equiv 0$. Now observe that $w$ is a polynomial of degree $n+1$. We could write out all the coefficients of this polynomial explicitly, but that is a bit tedious,

and we do not need all of them. Simply observe that we can write $w(x) = x^{n+1} + q(x)$, for some $q \in \mathcal{P}_n$, and this polynomial $q$ will vanish when we take $n+1$ derivatives:

$$w^{(n+1)}(x) = \left( \frac{d^{n+1}}{dx^{n+1}} x^{n+1} \right) + q^{(n+1)}(x) = (n+1)! + 0.$$

Assembling the pieces, $\phi^{(n+1)}(x) = f^{(n+1)}(x) - \lambda\,(n+1)!$. Since $\phi^{(n+1)}(\xi) = 0$, we conclude that

$$\lambda = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

Substituting this expression into $0 = \phi(\widehat{x}) = f(\widehat{x}) - p_n(\widehat{x}) - \lambda w(\widehat{x})$, we obtain

$$f(\widehat{x}) - p_n(\widehat{x}) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (\widehat{x} - x_j). \qquad \blacksquare$$

This error bound has strong parallels to the remainder term in Taylor's formula. Recall that for sufficiently smooth $h$, the Taylor expansion of $f$ about the point $x_0$ is given by

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \cdots + \frac{(x - x_0)^k}{k!} f^{(k)}(x_0) + \textsc{remainder}.$$

Ignoring the remainder term at the end, note that the Taylor expansion gives a polynomial model of $f$, but one based on local information about $f$ and its derivatives, as opposed to the polynomial interpolant, which is based on global information, but only about $f$, not its derivatives.

An interesting feature of the interpolation bound is the polynomial $w(x) = \prod_{j=0}^{n} (x - x_j)$. This quantity plays an essential role in approximation theory, and also a closely allied subdiscipline of complex analysis called *potential theory*. Naturally, one might wonder what choice of points $\{x_j\}$ minimizes $|w(x)|$: We will revisit this question when we study approximation theory in the near future. For now, we simply note that the points that minimize $|w(x)|$ over $[a, b]$ are called *Chebyshev points*, which are clustered more densely at the ends of the interval $[a, b]$.

**Example 1.1 ($f(x) = \sin(x)$).** We shall apply the interpolation bound to $f(x) = \sin(x)$ on $x \in [-5, 5]$. Since $f^{(n+1)}(x) = \pm \sin(x)$ or $\pm \cos(x)$, we have $\max_{x \in [-5,5]} |f^{(n+1)}(x)| = 1$ for all $n$. The interpolation result we just proved then implies that *for any choice of distinct interpolation points in* $[-5, 5]$,

$$\prod_{j=0}^{n} |x - x_j| < 10^{n+1},$$

the worst case coming if all the interpolation points are clustered at an end of the interval $[-5, 5]$. Now our theorem ensures that

$$\max_{x \in [-5,5]} |\sin(x) - p_n(x)| \leq \frac{10^{n+1}}{(n+1)!}.$$

For small values of $n$, this bound will be very large, but eventually $(n+1)!$ grows much faster than $10^{n+1}$, so we conclude that our error must go to zero as $n \to \infty$ *regardless of where in* $[-5, 5]$ *we place our interpolation points*! The error bound is shown in the first plot below.

Consider the following specific example: Interpolate $\sin(x)$ at points uniformly selected in $[-1, 1]$. At first glance, you might think there is no reason that we should expect our interpolants $p_n$ to converge to $\sin(x)$ for all $x \in [-5, 5]$, since we are only using data from the subinterval $[-1, 1]$, which is only 20% of the total interval and does not even include one entire period of the sine function. (In fact, $\sin(x)$ attains neither its maximum nor minimum on $[-1, 1]$.) Yet the error bound we proved above ensures that the polynomial interpolant must converge throughout $[-5, 5]$. This is illustrated in the first plot below. The next plots show the interpolants $p_4(x)$ and $p_{10}(x)$ generated from these interpolation points. Not surprisingly, these interpolants are most accurate near $[-1, 1]$, the location of the interpolation points (shown as circles), but we still see convergence well beyond $[-1, 1]$, in the same way that the Taylor expansion for $\sin(x)$ at $x = 0$ will converge everywhere.

**Example 1.2** (Runge's Example)**.** The error bound (1.7) suggests those functions for which interpolants might fail to converge as $n \to \infty$: beware if higher derivatives of $f$ are large in magnitude over the interpolation interval. The most famous example of such behavior is due to Carl Runge, who studied convergence of interpolants for $f(x) = 1/(1 + x^2)$ on the interval $[-5, 5]$. This function looks beautiful: it resembles a bell curve, with no singularities in sight on $\mathbb{R}$, as Figure 1.8 shows. However, the interpolants to $f$ at uniformly spaced points over $[-5, 5]$ do not seem to converge even for $x \in [-5, 5]$.

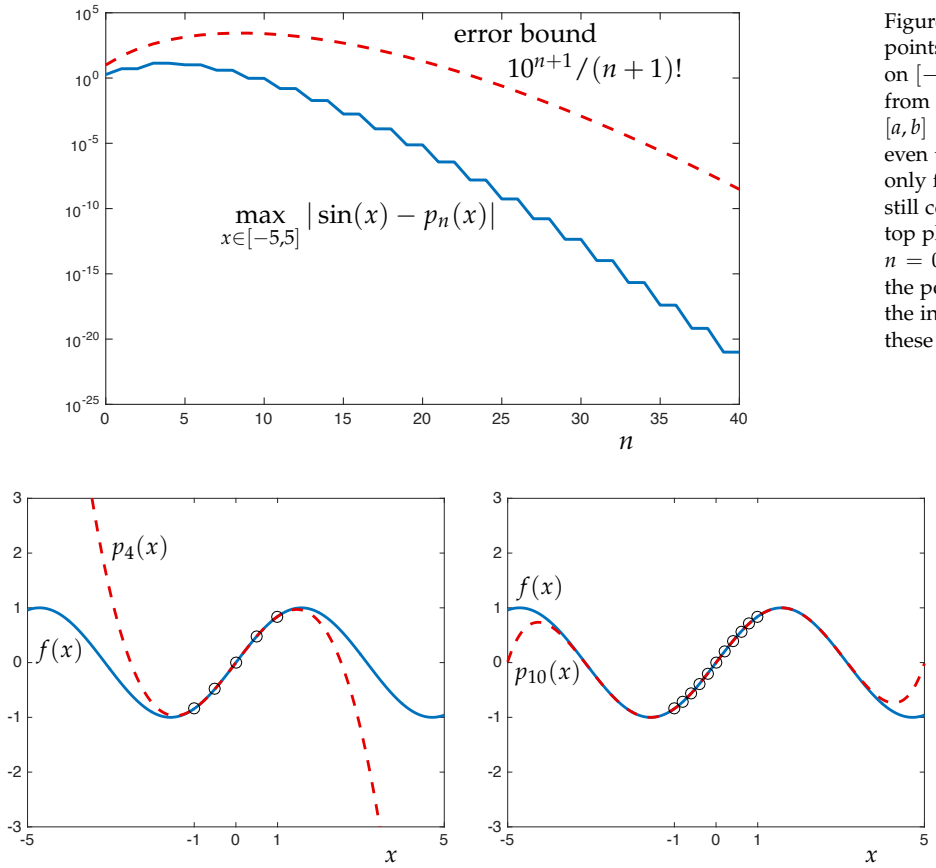



Figure 1.7: Interpolation of $\sin(x)$ at points $x_0, \ldots, x_n$ uniformly distributed on $[-1, 1]$. We develop an error bound from Theorem 1.3 for the interval $[a, b] = [-5, 5]$. The bound proves that even though the interpolation points only fall in $[-1, 1]$, the interpolant will still converge throughout $[-5, 5]$. The top plot shows this convergence for $n = 0, \ldots, 40$; the bottom plots show the polynomials $p_4$ and $p_10$, along with the interpolation points that determine these polynomials (black circles).

Look at successive derivatives of $f$; they expose its crucial flaw:

$$f'(x) = -\frac{2x}{(1+x^2)^2}$$

$$f''(x) = \frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2}$$

$$f'''(x) = -\frac{48x^3}{(1+x^2)^4} + \frac{24x}{(1+x^2)^3}$$

$$f^{(iv)}(x) = \frac{348x^4}{(1+x^2)^5} - \frac{288x^2}{(1+x^2)^4} + \frac{24}{(1+x^2)^3}$$

$$f^{(vi)}(x) = \frac{46080x^6}{(1+x^2)^7} - \frac{57600x^4}{(1+x^2)^6} + \frac{17280x^2}{(1+x^2)^5} - \frac{720}{(1+x^2)^4}.$$

At certain points on $[-5, 5]$, $f^{(n+1)}$ blows up more rapidly than $(n+1)!$, and the interpolation bound (1.7) suggests that $p_n$ *will not* converge to $f$ on $[-5, 5]$ as $n$ gets large. Not only does $p_n$ fail to converge to $f$; the error between certain interpolation points gets enormous as $n$ increases.

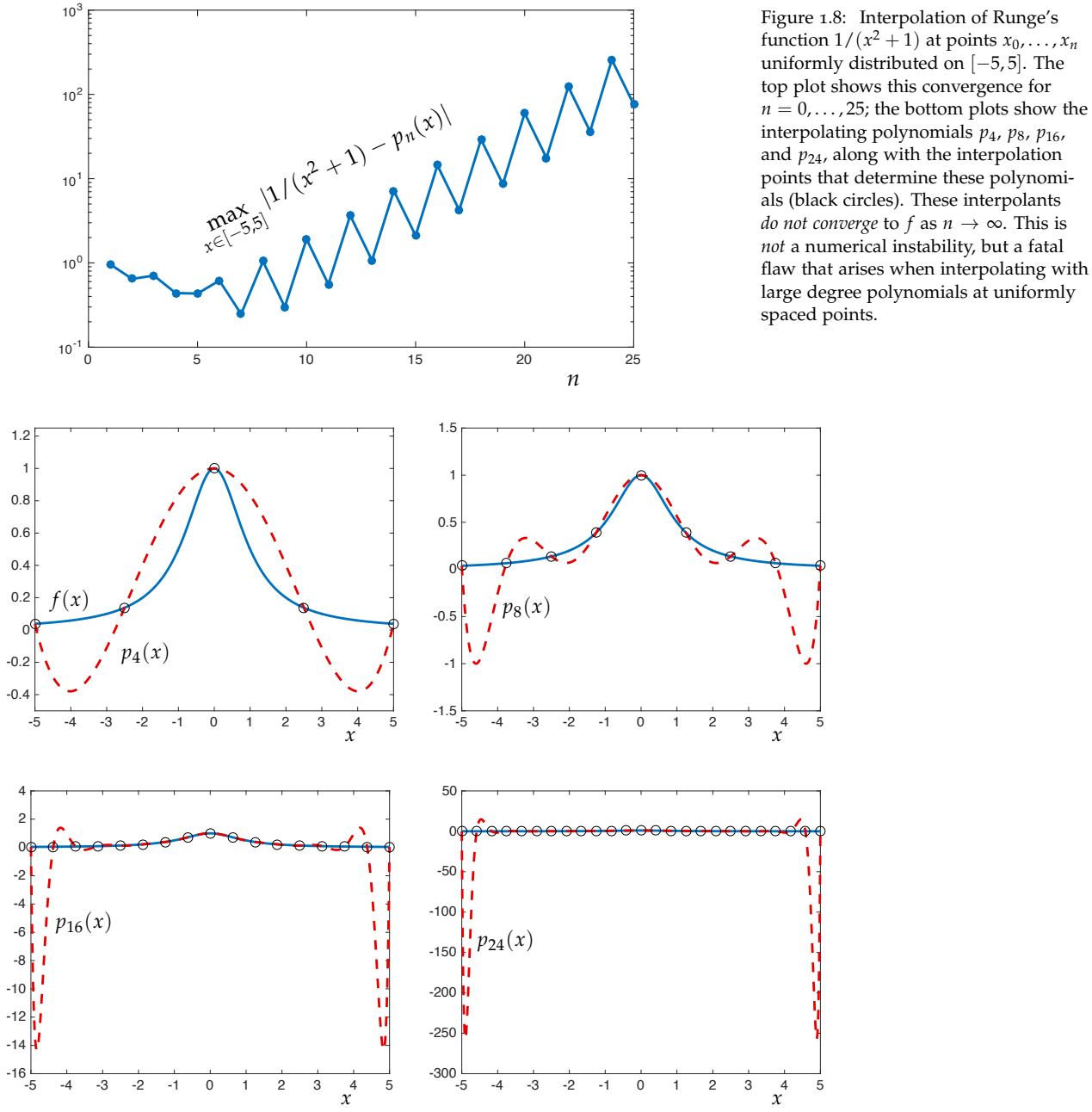



Figure 1.8: Interpolation of Runge's function $1/(x^2+1)$ at points $x_0, \ldots, x_n$ uniformly distributed on $[-5,5]$. The top plot shows this convergence for $n = 0, \ldots, 25$; the bottom plots show the interpolating polynomials $p_4$, $p_8$, $p_{16}$, and $p_{24}$, along with the interpolation points that determine these polynomials (black circles). These interpolants *do not converge* to $f$ as $n \to \infty$. This is *not* a numerical instability, but a fatal flaw that arises when interpolating with large degree polynomials at uniformly spaced points.

The following code uses MATLAB's Symbolic Toolbox to compute higher derivatives of the Runge function. Several of the resulting plots follow.[9] *Note how the scale on the vertical axis changes from plot to plot!*

[9] Not all versions of MATLAB have the Symbolic Toolbox, but you should be able to run this code on any Student Edition or on copies on Virginia Tech network.

```
% rungederiv.m
% routine to plot derivatives of Runge's example,
%     f(x) = 1/(1+x^2) on [-5,5]
```

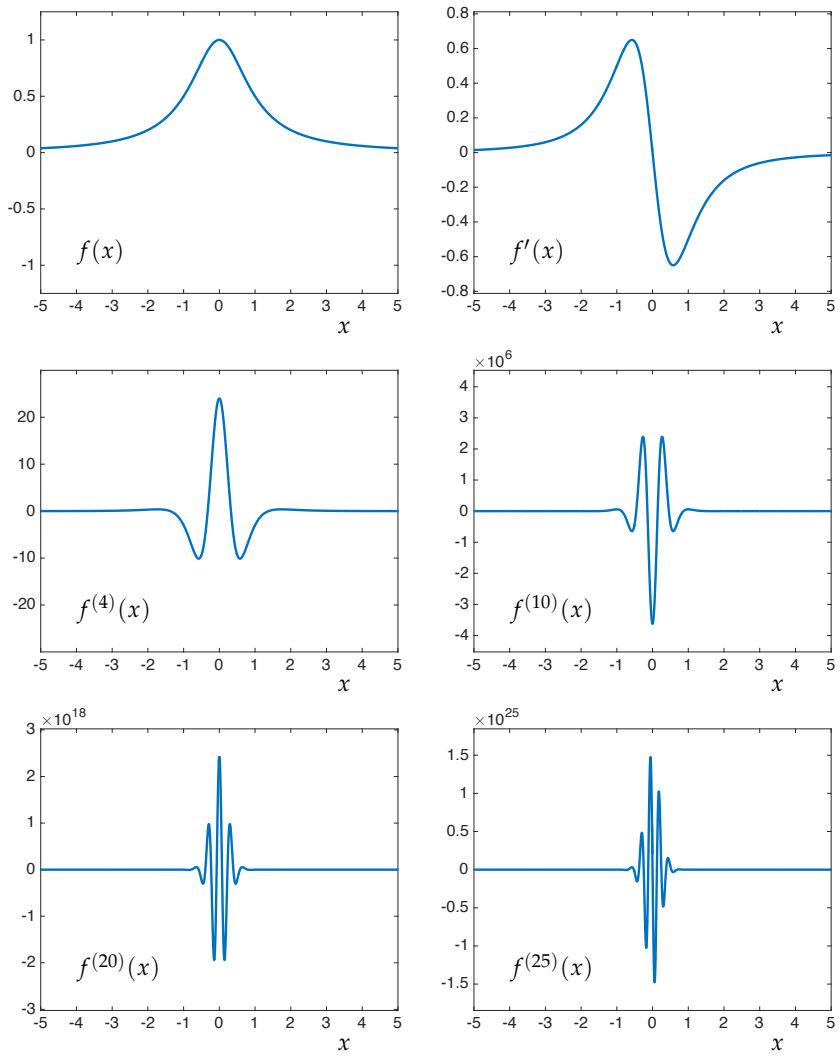



Figure 1.9: Runge's function

$$f(x) = \frac{1}{1+x^2}$$

and a few of its derivatives on $x \in [-5,5]$. Notice how large the derivatives grow in magnitude: the vertical scale on the plot for $f^{(25)}$ (bottom-right) is $10^{25}$.

```
figure(1), clf, set(gca,'fontsize',18)
for j=0:25
   syms x
   fj = vectorize(diff(1/(x^2+1),j));            % compute derivative (Symbolic Toolbox)
   x = linspace(-5,5,1000); fjx = eval(fj);      % evaluate on a grid of points
   plot(x,fjx,'b-','linewidth',2);               % plot derivative
   title(sprintf('Runge''s Example: f^{(%d)}(x)',j),'fontsize',14)
   input(' ')
end
```

Some improvement can be made by a careful selection of the interpolation points $\{x_0\}$. In fact, if one interpolates Runge's example, $f(x) = 1/(1+x^2)$, at the *Chebyshev points* for $[-5,5]$,

$$x_j = 5\cos\Big(\frac{j\pi}{n}\Big), \qquad j = 0,\ldots,n,$$

then the interpolant will converge!

As a general rule, interpolation at Chebyshev points is greatly preferred over interpolation at uniformly spaced points for reasons we shall understand in a few lectures. However, even this set is not perfect: there exist functions for which the interpolants at Chebyshev points do not converge. Examples to this effect were constructed by Marcinkiewicz and Grunwald in the 1930s. We close with two results of a more general nature.[10] We require some general notation to describe a family of interpolation points that can change as the polynomial degree increases. Toward this end, let $\{x_j^{[n]}\}_{j=0}^n$ denote the set of interpolation points used to construct the degree-$n$ interpolant. As we are concerned here with the behavior of interpolants as $n \to \infty$, so we will speak of the *system of interpolation points* $\{\{x_j^{[n]}\}_{j=0}^n\}_{n=0}^\infty$.

[10] An excellent exposition of these points is given in volume 3 of I. P. Natanson, *Constructive Function Theory* (Ungar, 1965).

Our first result is bad news.

---

**Theorem 1.4** (Faber's Theorem)**.**
Let $\{\{x_j^{[n]}\}_{j=0}^n\}_{n=0}^\infty$ be any system of interpolation points with $x_j^{[n]} \in [a,b]$ and $x_j^{[n]} \neq x_\ell^{[n]}$ for $j \neq \ell$ (i.e., distinct interpolation points for each polynomial degree). Then there exists some function $f \in C[a,b]$ such that the polynomials $p_n$ that interpolate $f$ at $\{x_j^{[n]}\}_{j=0}^n$ do not converge uniformly to $f$ in $[a,b]$ as $n \to \infty$.

---

The good news is that there always exists a suitable set of interpolation points for any given $f \in C[a,b]$.

---

**Theorem 1.5** (Marcinkiewicz's Theorem)**.**
Given any $f \in C[a,b]$, there exist a system of interpolation points with $x_j^{[n]} \in [a,b]$ such that the polynomials $p_n$ that interpolate $f$ at $\{x_j^{[n]}\}_{j=0}^n$ converge uniformly to $f$ in $[a,b]$ as $n \to \infty$.

---

These results are both quite abstract; for example, the construction of the offensive example in Faber's Theorem is not nearly as concrete as Runge's nice example for uniformly spaced points discussed above. We will revisit the question of the convergence of interpolants in a few weeks when we discuss Chebyshev polynomials. Then we will be able to say something much more positive: there exists a nice set of points that works for all but the ugliest functions in $C[a,b]$.

# LECTURE 4: *Constructing Finite Difference Formulas*

## *1.7 Application: Interpolants for Finite Difference Formulas*

The most obvious use of interpolants is to construct polynomial models of more complicated functions. However, numerical analysts rely on interpolants for many other numerical chores. For example, in a few weeks we shall see that common techniques for approximating definite integrals amount to exactly integrating a polynomial interpolant. Here we turn to a different application: the use of interpolating polynomials to derive finite difference formulas that approximate derivatives, the to use those formulas to construct approximations of differential equation boundary value problems.

### *1.7.1 Derivatives of Interpolants*

Theorem 1.3 from the last lecture showed how well the interpolant $p_n \in \mathcal{P}_n$ approximates $f$. Here we seek deeper connections between $p_n$ and $f$.

> How well do derivatives of $p_n$ approximate derivatives of $f$?

Let $p \in \mathcal{P}_n$ denote the degree-$n$ polynomial that interpolates $f$ at the distinct points $x_0, \ldots, x_n$. We want to derive a bound on the error $f'(x) - p'(x)$. Let us take the proof of Theorem 1.3 as a template, and adapt it to analyze the error in the derivative.

For simplicity, assume that $\widehat{x} \in \{x_0, \ldots, x_n\}$, i.e., assume that $\widehat{x}$ is one of the interpolation points. Suppose we extend $p(x)$ by one degree so that the derivative of the resulting polynomial at $\widehat{x}$ matches $f'(\widehat{x})$. To do so, use the Newton form of the interpolant, writing the new polynomial as

$$p(x) + \lambda w(x),$$

again with

$$w(x) := \prod_{j=0}^{n} (x - x_j).$$

The derivative interpolation condition at $\widehat{x}$ is

$$f'(\widehat{x}) = p'(\widehat{x}) + \lambda w'(\widehat{x}), \tag{1.8}$$

and since $w(x_j) = 0$ for $j = 0, \ldots, n$, the new polynomial maintains the standard interpolation at the $n + 1$ interpolation points:

$$f(x_j) = p(x_j) + \lambda w(x_j), \qquad j = 0, \ldots, n. \tag{1.9}$$

Here we must tweak the proof of Theorem 1.3 slightly. As in that proof, define the error function

$$\phi(x) := f(x) - \big(p(x) + \lambda w(x)\big).$$

Because of the standard interpolation conditions (1.9) at $x_0, \ldots, x_n$, $\phi$ must have $n+1$ zeros. Now Rolle's theorem implies that $\phi'$ has (at least) $n$ zeros, each of which occurs strictly between every two consecutive interpolation points. But in addition to these points, $\phi'$ must have another root at $\widehat{x}$ (which we have required to be one of the interpolation points, and thus distinct from the other $n$ roots). Thus, $\phi'$ has $n+1$ distinct zeros on $[a,b]$.

Now, repeatedly apply Rolle's theorem to see that $\phi''$ has $n$ distinct zeros, $\phi'''$ has $n-1$ distinct zeros, etc., to conclude that $\phi^{(n+1)}$ has a zero: call it $\xi$. That is,

$$0 = \phi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \big(p^{(n+1)}(\xi) + \lambda w^{(n+1)}(\xi)\big). \tag{1.10}$$

We must analyze

$$\phi^{(n+1)}(x) = f^{(n+1)}(x) - \big(p^{(n+1)}(x) + \lambda w^{(n+1)}(x)\big).$$

Just as in the proof of Theorem 1.3, note that $p^{(n+1)} = 0$ since $p \in \mathcal{P}_n$ and $w^{(n+1)}(x) = (n+1)!$. Thus from (1.10) conclude

$$\lambda = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

From (1.8) we arrive at

$$f'(\widehat{x}) - p'(\widehat{x}) = \lambda w'(\widehat{x}) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w'(\widehat{x}).$$

To arrive at a concrete estimate, perhaps we should say something more specific about $w'(\widehat{x})$. Expanding $w$ and computing $w'$ explicitly will take us far into the weeds; it suffices to invoke an interesting result from 1889.

---

**Lemma 1.1** (Markov brothers' inequality for first derivatives)**.**
For any polynomial $q \in \mathcal{P}_n$,

$$\max_{x\in[a,b]} |q'(x)| \le \frac{2n^2}{b-a} \max_{x\in[a,b]} |q(x)|.$$

---

We can thus summarize our discussion as the following theorem, an analogue of Theorem 1.3.

Lemma 1.1 was proved by Andrey Markov in 1889, generalizing a result for $n = 2$ that was obtained by the famous chemist Mendeleev in his research on specific gravity. Markov's younger brother Vladimir extended it to higher derivatives (with a more complicated right-hand side) in 1892. The interesting history of this inequality (and extensions into the complex plane) is recounted in a paper by Ralph Boas, Jr. on 'Inequalities for the derivatives of polynomials,' *Math. Magazine* 42 (4) 1969, 165–174. The result is called the 'Markov brothers' inequality' to distinguish it from the more famous 'Markov's inequality' in probability theory (named, like 'Markov chains,' for Andrey; Vladimir died of tuberculosis at the age of 25 in 1897).

**Theorem 1.6** (Bound on the derivative of an interpolant)**.**
Suppose $f \in C^{(n+1)}[a,b]$ and let $p_n \in \mathcal{P}_n$ denote the polynomial that interpolates $\{(x_j, f(x_j))\}_{j=0}^n$ at distinct points $x_j \in [a,b]$, $j = 0, \ldots, n$. Then for every $x_k \in \{x_0, \ldots, x_n\}$, there exists some $\xi \in [a,b]$ such that

$$f'(x_k) - p_n'(x_k) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w'(x_k),$$

where $w(x) = \prod_{j=0}^n (x - x_j)$. From this formula follows the bound

$$|f'(x_k) - p_n'(x_k)| \leq \frac{2n^2}{b-a} \left( \max_{\xi \in [a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \right) \left( \max_{x \in [a,b]} \prod_{j=0}^n |x - x_j| \right). \tag{1.11}$$

Why don't we simply 'take a derivative of Theorem 1.3'? The subtlety is the $f^{(n+1)}(\xi)$ term in Theorem 1.3. Since $\xi$ depends on $x$, taking the derivative of $f^{(n+1)}(\xi(x))$ via the chain rule would require explicit knowledge of $\xi(x)$. We don't want to work out a formula for $\xi(x)$ for each $f$ and interval $[a,b]$.

Contrast the bound (1.11) with (1.7) from Theorem 1.3: the bounds are the same, aside from the leading constant $2n^2/(b-a)$ inherited from Lemma 1.1.

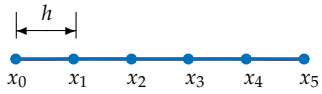
For our later discussion it will help to get a rough bound for he case where the interpolation points are uniformly distributed, i.e.,

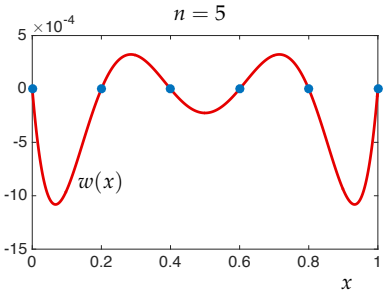
$$x_j = a + jh, \qquad j = 0, \ldots, n$$

with spacing equal to $h := (b-a)/n$. We seek to bound

$$\max_{x \in [a,b]} \prod_{j=0}^n |x - x_j|,$$

i.e., maximize the product of the distances of $x$ from each of the interpolation points. Consider the sketch in the margin. Think about how you would place $x \in [x_0, x_n]$ so as to make $\prod_{j=0}^n |x - x_j|$ as large as possible. Putting $x$ somewhere toward the ends, but not too near one of the interpolation points, will maximize product. Convince yourself that, regardless of where $x$ is placed within $[x_0, x_n]$:




Notice that for $n = 5$ uniformly spaced points on $[0,1]$, $w(x)$ takes its maximum magnitude between the two interpolation points on each end of the domain.

- at least one interpolation point is no more than $h/2$ away from $x$;
- a different interpolation point is no more than $h$ away from $x$;
- a different interpolation point is no more than $2h$ away from $x$;

  $\vdots$
- the last remaining (farthest) interpolation point is no more than $nh = b - a$ away from $x$.

This reasoning gives the bound

$$\max_{x \in [a,b]} \prod_{j=0}^n |x - x_j| \leq \frac{h}{2} \cdot h \cdot 2h \cdots nh = \frac{h^{n+1} n!}{2}. \tag{1.12}$$

Substituting this into (1.11) and using $b - a = nh$ gives the following result.

**Corollary 1.1** (The derivative of an interpolant at equispaced points)**.** Suppose $f \in C^{(n+1)}[a,b]$ and let $p_n \in \mathcal{P}_n$ denote the polynomial that interpolates $\{(x_j, f(x_j))\}_{j=0}^n$ at equispaced points $x_j = a + jh$ for $h = (b-a)/n$. Then for every $x_k \in \{x_0, \ldots, x_n\}$,

$$|f'(x_k) - p_n'(x_k)| \leq \frac{nh^n}{n+1}\left(\max_{\xi\in[a,b]} |f^{(n+1)}(\xi)|\right). \tag{1.13}$$

### 1.7.2 *Finite difference formulas*

The preceding analysis was toward a very specific purpose: to use interpolating polynomials to develop formulas that approximate derivatives of $f$ from the value of $f$ at a few points.

**Example 1.3** (First derivative)**.** We begin with the simplest case: formulas for the first derivative $f'(x)$. Pick some value for $x_0$ and some spacing parameter $h > 0$.

First construct the linear interpolant to $f$ at $x_0$ and $x_1 = x_0 + h$. Using the Newton form, we have

$$\begin{aligned} p_1(x) &= f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \\ &= f(x_0) + \frac{f(x_1) - f(x_0)}{h}(x - x_0). \end{aligned}$$

Take a derivative of the interpolant:

$$p_1'(x) = \frac{f(x_1) - f(x_0)}{h}, \tag{1.14}$$

which is precisely the conventional definition of the derivative, if we take the limit $h \to 0$. But how accurate an approximation is it? Appealing to Corollary 1.1 with $n = 1$ and $[a,b] = [x_0, x_1] = x_0 + [0,h]$, we have

$$|f'(x_k) - p_1'(x_k)| \leq \left(\frac{1}{2}\max_{\xi\in[x_0,x_1]} |f''(\xi)|\right) h \tag{1.15}$$

Does the bound (1.15) improve if we use a quadratic interpolant to $f$ through $x_0$, $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$? Again using the Newton form, write

$$\begin{aligned} p_2(x) &= f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + \frac{f(x_2) - f(x_0) - \frac{f(x_1)-f(x_0)}{x_1-x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}(x - x_0)(x - x_1) \\ &= f(x_0) + \frac{f(x_1) - f(x_0)}{h}(x - x_0) + \frac{f(x_0) - 2f(x_1) + f(x_2)}{2h^2}(x - x_0)(x - x_1). \end{aligned} \tag{1.16}$$

Taking a derivative of this interpolant with respect to $x$ gives

$$p_2'(x) = \frac{f(x_1) - f(x_0)}{h} + \frac{f(x_0) - 2f(x_1) + f(x_2)}{2h^2}(2x - x_0 - x_1).$$

Evaluate this at $x = x_0$, $x = x_1$, and $x = x_2$ and simplify as much as possible to get:

$$p_2'(x_0) = \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h} \tag{1.17}$$

$$p_2'(x_1) = \frac{f(x_2) - f(x_0)}{2h} \tag{1.18}$$

$$p_2'(x_2) = \frac{f(x_0) - 4f(x_1) + 3f(x_2)}{2h}. \tag{1.19}$$

These beautiful formulas are *right-looking*, *central*, and *left-looking* approximations to $f'$. Though we used an interpolating polynomial to derive these formulas, those polynomials are now nowhere in sight: they are merely the scaffolding that lead to these formulas. How accurate are these formulas? Corollary 1.1 with $n = 2$ and $[a, b] = [x_0, x_2] = x_0 + [0, 2h]$ gives

These formulas can also be derived by strategically combining Taylor expansions for $f(x+h)$ and $f(x-h)$. That is an easier route to simple formulas like (1.18), but is less appealing when more sophisticated approximations like (1.17) and (1.19) (and beyond) are needed.

$$|f'(x_k) - p_2'(x_k)| \leq \left(\frac{2}{3} \max_{\xi \in [x_0, x_2]} |f'''(\xi)|\right) h^2. \tag{1.20}$$

Notice that these approximations indeed scale with $h^2$, rather than $h$, and so the quadratic interpolant leads to a much better approximation to $f'$, at the cost of evaluating $f$ at three points (for $f'(x_0)$ and $f'(x_2)$), rather than two.

**Example 1.4** (Second derivative)**.** While we have only proved a bound for the error in the first derivative, $f'(x) - p'(x)$, you can see that similar bounds should hold when higher derivatives of $p$ are used to approximate corresponding derivatives of $f$. Here we illustrate with the second derivative.

Since $p_1$ is linear, $p_1''(x) = 0$ for all $x$, and the linear interpolant will not lead to any meaningful bound on $f''(x)$. Thus, we focus on the quadratic interpolant to $f$ at the three uniformly spaced points $x_0$, $x_1$, and $x_2$. Take two derivatives of the formula (1.16) for $p_2(x)$ to obtain

$$p_2''(x) = \frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2}, \tag{1.21}$$

which is a famous approximation to the second derivative that is often used in the finite difference discretization of differential equations. One can show that, like the approximations $p_2'(x_k)$, this formula is accurate to order $h^2$.

**Example 1.5** (*Mathematica* code for computing difference formulas)**.** *Code to follow....*

# LECTURE 5: *Finite Difference Methods for Differential Equations*

### *1.7.3 Application: Boundary Value Problems*

**Example 1.6** (Dirichlet boundary conditions)**.** Suppose we want to solve the differential equation

$$-u''(x) = g(x), \qquad x \in [0,1]$$

for the unknown function $u$, subject to the *Dirichlet* boundary conditions

$$u(0) = u(1) = 0.$$

One common approach to such problems is to approximate the solution $u$ on a uniform grid of points

$$0 = x_0 < x_1 < \cdots < x_n = 1$$

with $x_j = j/N$.

We seek to approximate the solution $u(x)$ at each of the grid points $x_0, \ldots, x_n$. The Dirichlet boundary conditions give the end values immediately:

$$u(x_0) = 0, \qquad u(x_n) = 0.$$

At each of the interior grid points, we require a local approximation of the equation

$$-u''(x_j) = g(x_j), \qquad j = 1, \ldots, n-1.$$

For each, we will (implicitly) construct the quadratic interpolant $p_{2,j}$ to $u(x)$ at the points $x_{j-1}$, $x_j$, and $x_{j+1}$, and then approximate

$$-p''_{2,j}(x_j) \approx -u''(x_j) = g(x_j).$$

Aside from some index shifting, we have already constructed $p''_{2,j}$ in equation (1.21):

$$p''_{2,j}(x) = \frac{u(x_{j-1}) - 2u(x_j) + u(x_{j+1})}{h^2}.$$

Just one small caveat remains: we cannot construct $p''_{2,j}(x)$, *because we do not know the values of* $u(x_{j-1})$, $u(x_j)$, *and* $u(x_{j+1})$: finding those values is the point of our entire endeavor. Thus we define approximate values

$$u_j \approx u(x_j), \qquad j = 1, \ldots, n-1.$$

and will instead use the polynomial $p_{2,j}$ that interpolates $u_{j-1}$, $u_j$, and $u_{j+1}$, giving

$$p''_{2,j}(x) = \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2}. \tag{1.22}$$

Let us accumulate all our equations for $j = 0, \ldots, n$:

$$\begin{aligned}
u_0 &= 0 \\
u_0 - 2u_1 + u_2 &= -h^2 g(x_1) \\
u_1 - 2u_2 + u_3 &= -h^2 g(x_2) \\
&\vdots \\
u_{n-3} - 2u_{n-2} + u_{n-1} &= -h^2 g(x_{n-2}) \\
u_{n-2} - 2u_{n-1} + u_n &= -h^2 g(x_{n-1}) \\
u_n &= 0.
\end{aligned}$$

Notice that this is a system of $n+1$ linear equations in $n+1$ variables $u_0, \ldots, u_{n+1}$. Thus we can arrange this in matrix form as

$$\begin{bmatrix}
1 & & & & & & & \\
1 & -2 & 1 & & & & & \\
 & 1 & -2 & 1 & & & & \\
 & & \ddots & \ddots & \ddots & & & \\
 & & & 1 & -2 & 1 & & \\
 & & & & 1 & -2 & 1 & \\
 & & & & & & 1
\end{bmatrix}
\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{bmatrix}
=
\begin{bmatrix} 0 \\ -h^2 g(x_1) \\ -h^2 g(x_2) \\ \vdots \\ -h^2 g(x_{n-2}) \\ -h^2 g(x_{n-1}) \\ 0 \end{bmatrix}, \tag{1.23}$$

where the blank entries are zero. Notice that the first and last entries are trivial: $u_0 = u_n = 0$, and so we can trim them off to yield the slightly simpler matrix

$$\begin{bmatrix}
-2 & 1 & & & \\
1 & -2 & 1 & & \\
 & \ddots & \ddots & \ddots & \\
 & & 1 & -2 & 1 \\
 & & & 1 & -2
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}
=
\begin{bmatrix} -h^2 g(x_1) \\ -h^2 g(x_2) \\ \vdots \\ -h^2 g(x_{n-2}) \\ -h^2 g(x_{n-1}) \end{bmatrix}. \tag{1.24}$$

Solve this $(n-1) \times (n-1)$ linear system of equations using Gaussian elimination. One can show that the solution to the differential equation inherits the accuracy of the interpolant: the error $|u(x_j) - u_j|$ behaves like $O(h^2)$ as $h \to 0$.

Ideally, use an efficient version of Gaussian elimination that exploits the banded structure of this matrix to give a solution in $O(n)$ operations.

**Example 1.7** (Mixed boundary conditions)**.** Modify the last example to keep the same differential equation

$$-u''(x) = g(x), \qquad x \in [0, 1]$$

but now use *mixed* boundary conditions,

$$u'(0) = u(1) = 0.$$

The derivative condition on the left is the only modification; we must change the first row of equation (1.23) to encode this condition. One might be tempted to use a simple linear interpolant to approximate the boundary condition on the left side, adapting formula (1.14) to give:

$$\frac{u_1 - u_0}{h} = 0. \tag{1.25}$$

This equation makes intuitive sense: it forces $u_1 = u_0$, so the approximate solution will have zero slope on its left end. This gives the equation

$$\begin{bmatrix} -1 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ -h^2 g(x_1) \\ -h^2 g(x_2) \\ \vdots \\ -h^2 g(x_{n-2}) \\ -h^2 g(x_{n-1}) \end{bmatrix}, \tag{1.26}$$

where we have trimmed off the elements associated with the $u_n = 0$.

The approximation of the second derivative (1.22) is accurate up to $\mathcal{O}(h^2)$, whereas the estimate (1.25) of $u'(0) = 0$ is only $\mathcal{O}(h)$ accurate. Will it matter if we compromise accuracy that little bit, if only in one of the $n$ equations in (1.26)? What if instead we approximate $u'(0) = 0$ to second-order accuracy?

Equations (1.17)–(1.19) provide three formulas that approximate the first derivative to second order. Which one is appropriate in this setting? The *right-looking* formula (1.17) gives the approximation

$$u'(0) \approx \frac{-3u_0 + 4u_1 - u_2}{2h}, \tag{1.27}$$

which involves the variables $u_0$, $u_1$, and $u_2$ that we are already considering. In contrast, the centered formula (1.18) needs an estimate of $u(-h)$, and the left-looking formula (1.19) needs $u(-h)$ and $u(-2h)$. Since these values of $u$ fall outside the domain $[0, 1]$ of $u$, the centered and left-looking formulas would not work.

Combining the right-looking formula (1.27) with the boundary condition $u'(0) = 0$ gives

$$\frac{-3u_0 + 4u_1 - u_2}{2h} = 0,$$

with which we replace the first row of (1.26) to obtain

$$\begin{bmatrix} -3 & 4 & -1 & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ -h^2 g(x_1) \\ -h^2 g(x_2) \\ \vdots \\ -h^2 g(x_{n-2}) \\ -h^2 g(x_{n-1}) \end{bmatrix}. \tag{1.28}$$

Is this $\mathcal{O}(h^2)$ accurate approach at the boundary worth the (rather minimal) extra effort? Let us investigate with an example. Set the right-hand side of the differential equation to

$$g(x) = \cos(\pi x/2),$$

which corresponds to the exact solution

$$u(x) = \frac{4}{\pi^2}\cos(\pi x/2).$$

Verify that $u$ satisfies the boundary conditions $u'(0) = 0$ and $u(1) = 0$.

Figure 1.10 compares the solutions obtained by solving (1.26) and (1.28) with $n = 4$. Clearly, the simple adjustment that gave the $\mathcal{O}(h^2)$ approximation to $u'(0) = 0$ makes quite a difference! This figures shows that the solutions from (1.26) and (1.28) differ, but plots like this are not the best way to understand how the approximations compare as $n \to \infty$. Instead, compute maximum error at the interpolation points,

$$\max_{0 \le j \le n} |u(x_j) - u_j|$$

Indeed, we used this small value of $n$ because it is difficult to see the difference between the exact solution and the approximation from (1.28) for larger $n$.

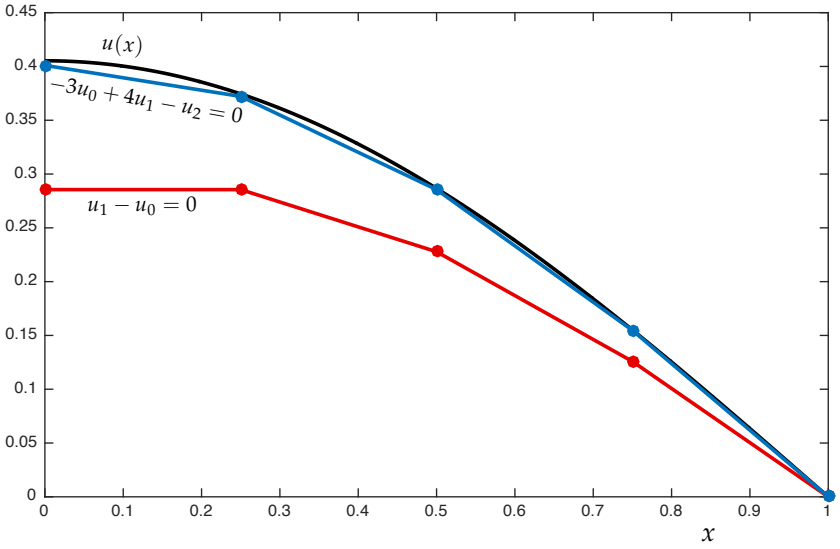



Figure 1.10: Approximate solutions to $-u''(x) = \cos(\pi x/2)$ with $u'(0) = u(1) = 0$. The black curve shows $u(x)$. The red approximation is obtained by solving (1.26), which uses the $\mathcal{O}(h)$ approximation $u'(0) = 0$; the blue approximation is from (1.28) with the $\mathcal{O}(h^2)$ approximation of $u'(0) = 0$. Both approximations use $n = 4$.

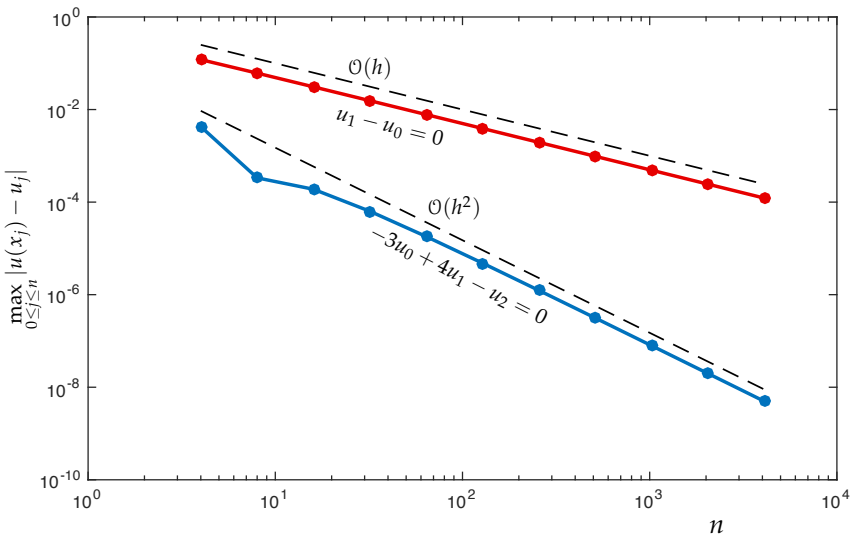



Figure 1.11: Convergence of approximate solutions to $-u''(x) = \cos(\pi x/2)$ with $u'(0) = u(1) = 0$. The red line shows the approximation from (1.26); it converges like $\mathcal{O}(h)$ as $h \to 0$. The blue line shows the approximation from (1.28), which converges like $\mathcal{O}(h^2)$.

for various values of $n$. Figure 1.11 shows the results of such experiments for $n = 2^2, 2^3, \ldots, 2^{12}$. Notice that this figure is a 'log-log' plot; on such a scale, the errors fall on straight lines, and from the slope of these lines one can determine the order of convergence. The slope of the red curve is $-1$, so the accuracy of the approximations from (1.26) is $\mathcal{O}(n^{-1}) = \mathcal{O}(h)$ accurate. The slope of the blue curve is $-2$, so (1.28) gives an $\mathcal{O}(n^{-2}) = \mathcal{O}(h^2)$ accurate approximation.

How large would $n$ need to be, to get the same accuracy from the $\mathcal{O}(h)$ approximation that was produced by the $\mathcal{O}(h^2)$ approximation with $n = 2^{12} = 4096$? Extrapolation of the red curve suggests we would need roughly $n = 10^8$.

This example illustrates a general lesson: when constructing finite difference approximations to differential equations, one must ensure that the approximations to the boundary conditions have the same order of accuracy as the approximation of the differential equation itself. These formulas can be nicely constructed by from derivatives of polynomial interpolants of appropriate degree.

# LECTURE 6: *Interpolating Derivatives*

## *1.8 Hermite Interpolation and Generalizations*

Example 1.1 demonstrated that polynomial interpolants to $\sin(x)$ attain arbitrary accuracy for $x \in [-5, 5]$ as the polynomial degree increases, even if the interpolation points are taken exclusively from $[-1, 1]$. In fact, as $n \to \infty$ interpolants based on data from $[-1, 1]$ will converge to $\sin(x)$ *for all* $x \in \mathbb{R}$. More precisely, for any $x \in \mathbb{R}$ and any $\varepsilon > 0$, there exists some positive integer $N$ such that $|\sin(x) - p_n(x)| < \varepsilon$ for all $n \geq N$, where $p_n$ interpolates $\sin(x)$ at $n + 1$ uniformly-spaced interpolation points in $[-1, 1]$.

In fact, this is not as surprising as it might first appear. The Taylor series expansion uses derivative information at a single point to produce a polynomial approximation of $f$ that is accurate at nearby points. In fact, the interpolation error bound derived in the previous lecture bears close resemblance to the remainder term in the Taylor series. If $f \in C^{(n+1)}[a, b]$, then expanding $f$ at $x_0 \in (a, b)$, we have

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

This is the *Lagrange form* of the error.

for some $\xi \in [x, x_0]$ that depends on $x$. The first sum is simply a degree $n$ polynomial in $x$; from the final term – the Taylor remainder – we obtain the bound

$$\max_{x \in [a,b]} \left| f(x) - \left( \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \right) \right| \leq \left( \max_{\xi \in [a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \right) \left( \max_{x \in [a,b]} |x - x_0|^{n+1} \right),$$

which should certainly remind you of the interpolation error formula in Theorem 1.3.

One can view polynomial interpolation and Taylor series as two extreme approaches to approximating $f$: one uses global information, but only about $f$; the other uses only local information, but requires extensive knowledge of the derivatives of $f$. In this section we shall discuss an alternative based on the best features of each of these ideas: use global information about both $f$ and its derivatives.

### *1.8.1 Hermite interpolation*

In cases where the polynomial interpolants of the previous sections incurred large errors for some $x \in [a, b]$, one typically observes that the slope of the interpolant differs markedly from that of $f$ at some of the interpolation points $\{x_j\}$. (Recall Runge's example in Figure 1.8.) Why not then *force the interpolant to match both $f$ and $f'$ at the interpolation points*?

Often the underlying application provides a motivation for such derivative matching. For example, if the interpolant approximates the position of a particle moving in space, we might wish the interpolant to match not only position, but also velocity. *Hermite interpolation* is the general procedure for constructing such interpolants.

Typically the position of a particle is given in terms of a second-order differential equation (in classical mechanics, arising from Newton's second law, $F = ma$). To solve this second-order ODE, one usually writes it as a system of first-order equations whose numerical solution we will study later in the semester. One component of the system is position, the other is velocity, and so one automatically obtains values for both $f$ (position) and $f'$ (velocity) simultaneously.

Given $f \in C^1[a,b]$ and $n+1$ points $\{x_j\}_{j=0}^n$ satisfying

$$a \le x_0 < x_1 < \cdots < x_n \le b,$$

determine some $h_n \in \mathcal{P}_{2n+1}$ such that

$$h_n(x_j) = f(x_j), \qquad h_n'(x_j) = f'(x_j) \quad \text{for } j = 0, \ldots, n.$$

Note that $h$ must generally be a polynomial of degree $2n+1$ to have sufficiently many degrees of freedom to satisfy the $2n+2$ constraints. We begin by addressing the existence and uniqueness of this interpolant.

Existence is best addressed by explicitly constructing the desired polynomial. We adopt a variation of the *Lagrange approach* used in Section 1.5. We seek degree-$(2n+1)$ polynomials $\{A_k\}_{k=0}^n$ and $\{B_k\}_{k=0}^n$ such that

$$A_k(x_j) = \begin{cases} 0, & j \ne k, \\ 1, & j = k, \end{cases} \qquad A_k'(x_j) = 0 \text{ for } j = 0, \ldots, n;$$

$$B_k(x_j) = 0 \text{ for } j = 0, \ldots, n, \qquad B_k'(x_j) = \begin{cases} 0, & j \ne k \\ 1, & j = k \end{cases}.$$

These polynomials would yield a basis for $\mathcal{P}_{2n+1}$ in which $h_n$ has a simple expansion:

$$h_n(x) = \sum_{k=0}^n f(x_k) A_k(x) + \sum_{k=0}^n f'(x_k) B_k(x). \tag{1.29}$$

To show how we can construct the polynomials $A_k$ and $B_k$, we recall the Lagrange basis polynomials used for the standard interpolation problem,

$$\ell_k(x) = \prod_{j=0, j\ne k}^n \frac{(x - x_j)}{(x_k - x_j)}.$$

Consider the definitions

$$A_k(x) := \big(1 - 2(x - x_k)\ell_k'(x_k)\big)\ell_k^2(x),$$
$$B_k(x) := (x - x_k)\ell_k^2(x).$$

Note that since $\ell_k \in \mathcal{P}_n$, we have $A_k, B_k \in \mathcal{P}_{2n+1}$. Figure 1.12 shows these Hermite basis polynomials and their derivatives for $n = 5$ using

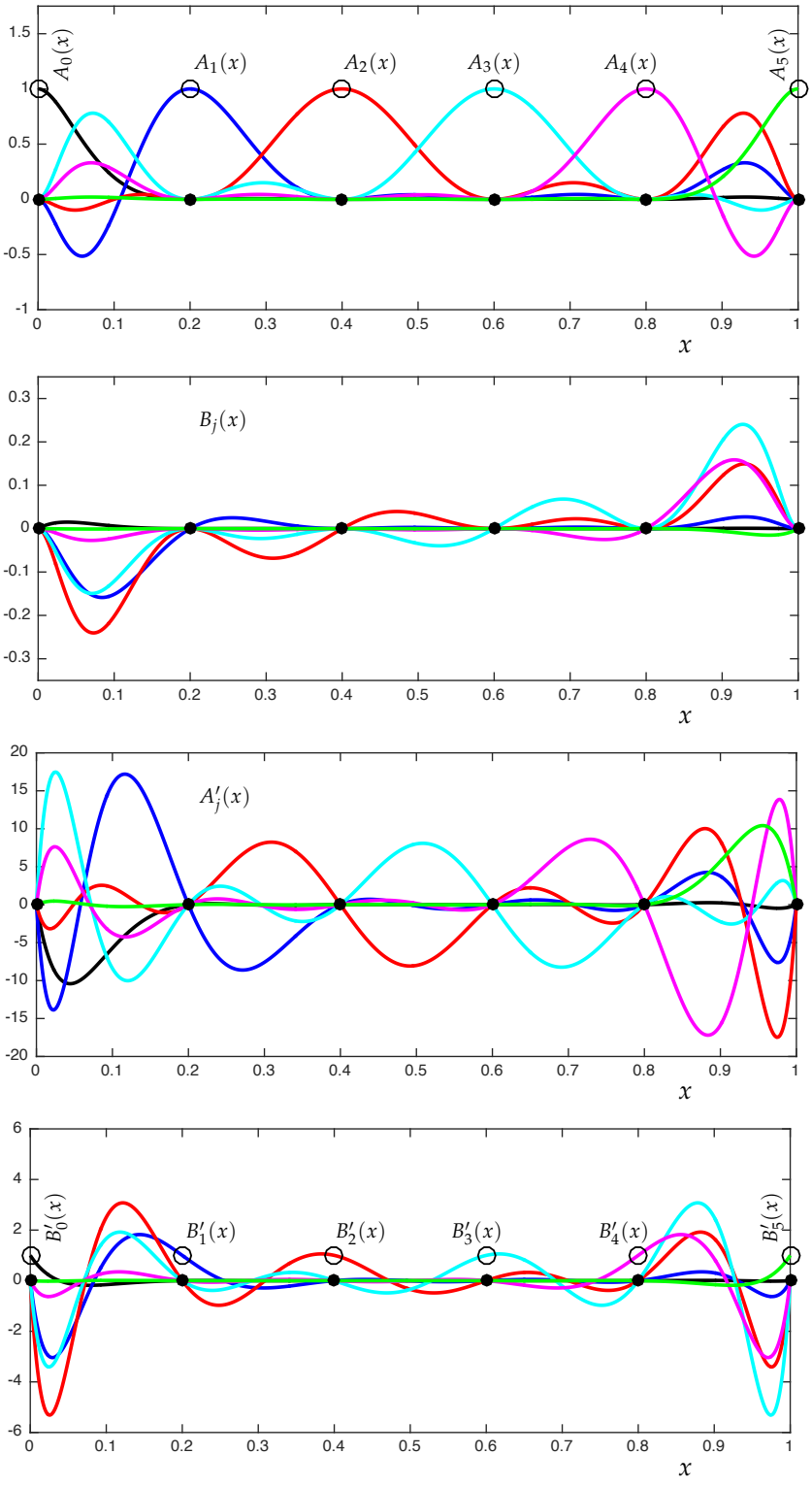



Figure 1.12: The Hermite basis polynomials for $n = 5$ on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (black dots).
- $A_0, \ldots, A_5$: $A_j(x_j) = 1$ (black circles).
- $B_0, \ldots, B_5$: $B_j(x_k) = 0$ for all $j, k$.
- $A'_0, \ldots, A'_5$: $A'_j(x_k) = 0$ for all $j, k$.
- $B'_0, \ldots, B'_5$: $B'_j(x_j) = 1$ (black circles).

uniformly spaced points on $[0, 1]$. It is a straightforward exercise to verify that these $A_k$ and $B_k$, and their first derivatives, obtain the specified values at $\{x_j\}_{j=0}^n$.

These interpolation conditions at the points $\{x_j\}$ ensure that the $2n + 2$ polynomials $\{A_k, B_k\}_{k=0}^n$, each of degree $2n + 1$, form a basis for $\mathcal{P}_{2n+1}$, and thus we can always write $h_n$ via the formula (1.29).

Figure 1.13 compares the standard polynomial interpolant $p_n \in \mathcal{P}_n$ to the Hermite interpolant $h_n \in \mathcal{P}_{2n+1}$ and the standard interpolant of the same degree, $p_{2n+1} \in \mathcal{P}_{2n+1}$ for the example $f(x) = \sin(20x) + e^{5x/2}$ using uniformly spaced points on $[0, 1]$ with $n = 5$. Note the distinction between $h_n$ and $p_{2n+1}$, which are both polynomials of the same degree.

Here are a couple of basic results whose proofs follow the same techniques as the analogous proofs for the standard interpolation problem.

---

**Theorem 1.7.** The Hermite interpolant $h_n \in \mathcal{P}_{2n+1}$ is unique.

The uniqueness result hinges on the fact that we interpolate $f$ and $f'$ both at all interpolation points. If we vary the number of derivatives interpolated at each data point, we open the possibility of non-unique interpolants.

**Theorem 1.8.** Suppose $f \in C^{2n+2}[x_0, x_n]$ and let $h_n \in \mathcal{P}_{2n+1}$ such that $h_n(x_j) = f(x_j)$ and $h_n'(x_j) = f'(x_j)$ for $j = 0, \ldots, n$. Then for any $x \in [x_0, x_n]$, there exists some $\xi \in [x_0, x_n]$ such that

$$f(x) - h_n(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \prod_{j=0}^{n} (x - x_j)^2.$$

---

The proof of this latter result is directly analogous to the standard polynomial interpolation error in Theorem 1.3. Think about how you would prove this result for yourself.

Hint: the proof has some resemblance to our proof of Theorem 1.6. Invoke Rolle's theorem to get $n$ roots of a certain function, then use the derivative interpolation to get another $n + 1$ roots.

### 1.8.2 *Hermite–Birkhoff interpolation*

Of course, one need not stop at interpolating $f$ and $f'$. Perhaps your application has more general requirements, where you want to interpolate higher derivatives, too, or have the number of derivatives interpolated differ at each interpolation point. Such general polynomials are called *Hermite–Birkhoff interpolants*, and you already have the tools at your disposal to compute them. Simply formulate the problem as a linear system and find the desired coefficients, but beware that in some situations, *there may be infinitely many polynomials that satisfy the interpolation conditions*. For these problems, it is generally simplest to work with the monomial basis, though one could design Newton- or Lagrange-inspired bases for particular situations.

For example, suppose you seek an interpolant that is particularly accurate in the vicinity of one of the interpolation points, and so you wish to interpolate higher derivatives at that point: a hybrid between an interpolating polynomial and a Taylor expansion.

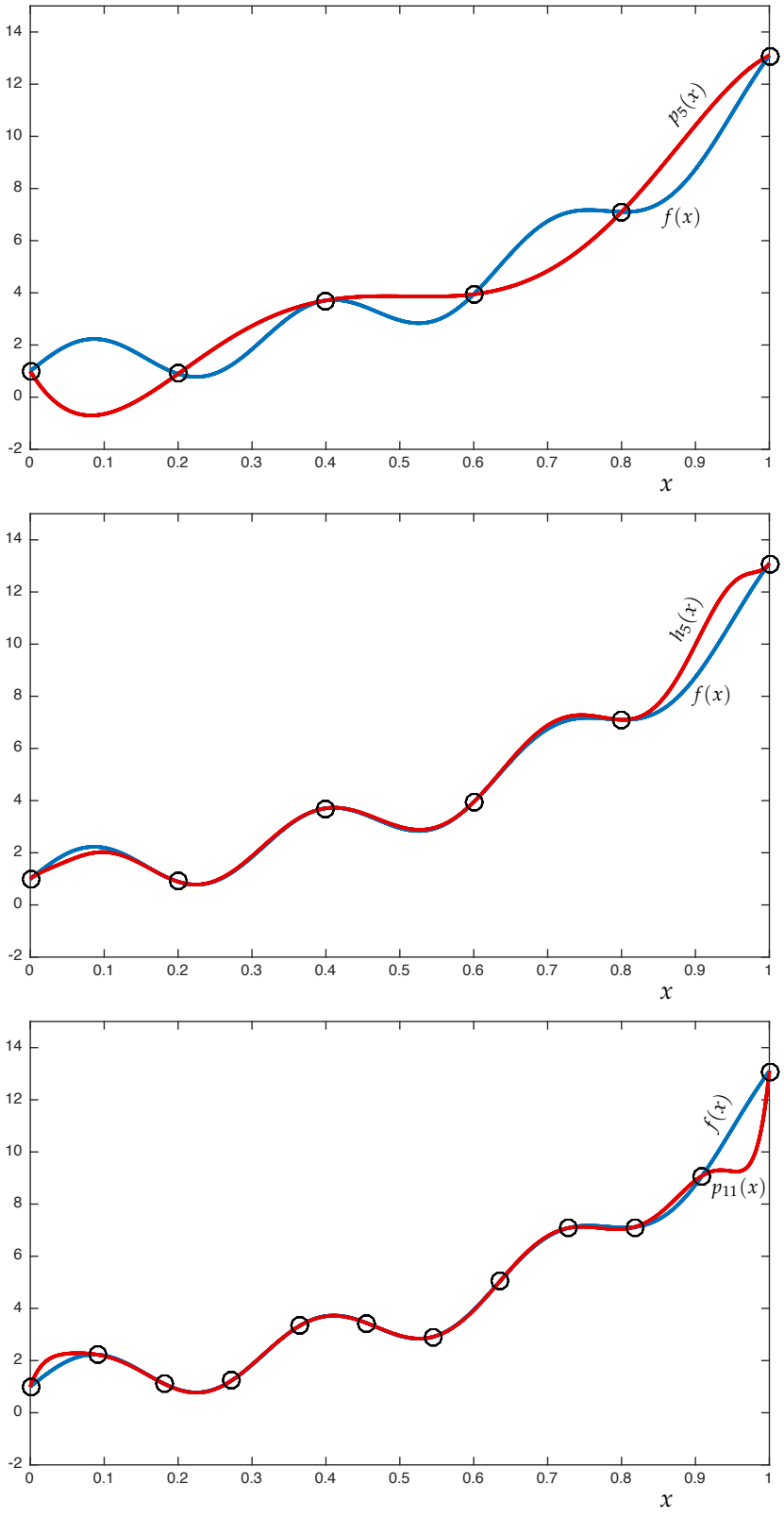



Figure 1.13: Interpolation of $f(x) = \sin(20x) + e^{5x/2}$ at uniformly spaced points for $x \in [0, 1]$. Top plot: the standard polynomial interpolant $p_5 \in \mathcal{P}_5$. Middle plot: the Hermite interpolant $h_5 \in \mathcal{P}_{11}$. Bottom plot: the standard interpolant $p_{11} \in \mathcal{P}_{11}$.

Though the last two plots show polynomials of the same degree, notice how the interpolants differ. (At first glance it appears the Hermite interpolation condition fails at the rightmost point in the middle plot; zoom in to see that the slope of the interpolant indeed matches $f'(1)$.)

### 1.8.3 *Hermite–Fejér interpolation*

Another Hermite-like scheme initially sounds like a bad idea: make the interpolant have *zero slope* at all the interpolation points.

> Given $f \in C^1[a,b]$ and $n+1$ points $\{x_j\}_{j=0}^n$ satisfying
>
> $$a \le x_0 < x_1 < \cdots < x_n \le b,$$
>
> determine some $h_n \in \mathcal{P}_{2n+1}$ such that
>
> $$h_n(x_j) = f(x_j), \qquad h_n'(x_j) = 0 \quad \text{for } j = 0, \dots, n.$$

That is, explicitly construct $h_n$ such that its derivatives in general *do not match those of* $f$. This method, called *Hermite–Fejér interpolation*, turns out to be remarkably effective, even better than standard Hermite interpolation in certain circumstances. In fact, Fejér proved that if we choose the interpolation points $\{x_j\}$ in the right way, $h_n$ is guaranteed to converge to $f$ uniformly as $n \to \infty$.

---

**Theorem 1.9.** For each $n \ge 1$, let $h_n$ be the Hermite–Fejér interpolant of $f \in C[a,b]$ at the Chebyshev interpolation points

$$x_j = \frac{a+b}{2} + \Big(\frac{b-a}{2}\Big)\cos\Big(\frac{(2j+1)\pi}{2n+2}\Big), \quad j = 0, \dots, n.$$

Then $h_n(x)$ converges uniformly to $f$ on $[a,b]$.

---

For a proof of Theorem 1.9, see page 57 of I. P. Natanson, *Constructive Function Theory*, vol. 3 (Ungar, 1965).

# LECTURE 7: *Trigonometric Interpolation*

## *1.9 Trigonometric interpolation for periodic functions*

Thus far all our interpolation schemes have been based on polynomials. However, if the function $f$ is *periodic*, one might naturally prefer to interpolate $f$ with some set of periodic functions.

To be concrete, suppose we have a continuous $2\pi$-periodic function $f$ that we wish to interpolate at the uniformly spaced points $x_k = 2\pi k/n$ for $k = 0, \ldots, n$ with $n = 5$. We shall build an interpolant as a linear combination of the $2\pi$-periodic functions

'$2\pi$-periodic' means that $f$ is continuous throughout $\mathbb{R}$ and $f(x) = f(x + 2\pi)$ for all $x \in \mathbb{R}$.

The choice of period $2\pi$ makes the notation a bit simpler, but the idea can be easily adapted for any period.

$$b_0(x) = 1, \quad b_1(x) = \sin(x), \quad b_2(x) = \cos(x), \quad b_3(x) = \sin(2x), \quad b_4(x) = \cos(2x).$$

Note that we have *six* interpolation conditions at $x_k$ for $k = 0, \ldots, 5$, but only *five* basis functions. This is not a problem: since $f$ is periodic, $f(x_0) = f(x_5)$, and the same will be true of our $2\pi$-periodic interpolant: the last interpolation condition is automatically satisfied.

We shall construct an interpolant of the form

$$t_5(x) = \sum_{k=0}^{4} c_k b_k(x)$$

such that

$$t_5(x_j) = f(x_j), \quad j = 0, \ldots, 4.$$

To compute the unknown coefficients $c_0, \ldots, c_4$, set up a linear system as usual,

$$\begin{bmatrix} b_0(x_0) & b_1(x_0) & b_2(x_0) & b_3(x_0) & b_4(x_0) \\ b_0(x_1) & b_1(x_1) & b_2(x_1) & b_3(x_1) & b_4(x_1) \\ b_0(x_2) & b_1(x_2) & b_2(x_2) & b_3(x_2) & b_4(x_2) \\ b_0(x_3) & b_1(x_3) & b_2(x_3) & b_3(x_3) & b_4(x_3) \\ b_0(x_4) & b_1(x_4) & b_2(x_4) & b_3(x_4) & b_4(x_4) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix},$$

which can be readily generalized to accommodate more interpolation points. We could solve this system for $c_0, \ldots, c_n$, but we prefer to express the problem in a more convenient basis for the trigonometric functions. Recall Euler's formula,

You would $b_6(x) = \sin(3x)$, $b_7(x) = \cos(3x)$, etc.: one function for each additional interpolation point. Generally you would use an odd value of $n$, to include pairs of sines and cosines.

$$e^{i\theta x} = \cos(\theta x) + i \sin(\theta x),$$

which also implies that

$$e^{-i\theta x} = \cos(\theta x) - i \sin(\theta x).$$

To prove this, write the Taylor expansion of $e^{i\theta x}$, then separate the real and imaginary components to give Taylor expansions for $\cos(\theta x)$ and $\sin(\theta x)$.

From these formulas it follows that

$$\operatorname{span}\{e^{i\theta x}, e^{-i\theta x}\} = \{\cos(\theta x), \sin(\theta x)\}.$$

Note that we can also write $b_0(x) \equiv 1 = e^{i0x}$. Putting these pieces together, we arrive at an alternative basis for the trigonometric interpolation space:

$$\text{span}\{1, \sin(x), \cos(x), \sin(2x), \cos(2x)\} = \text{span}\{e^{-2ix}, e^{-ix}, e^{0ix}, e^{ix}, e^{2ix}\}.$$

The interpolant $t_n$ can thus be expressed in the form

$$t_4(x) = \sum_{k=-2}^{2} \gamma_k e^{ikx} = \sum_{k=-2}^{2} \gamma_k (e^{ix})^k.$$

This last sum is written in a manner that emphasizes that $t_4$ is *a polynomial in the variable* $e^{ix}$, and hence $t_n$ is a *trigonometric polynomial*. In this basis, the interpolation conditions give the linear system

$$\begin{bmatrix} e^{-2ix_0} & e^{-ix_0} & e^{0ix_0} & e^{ix_0} & e^{i2x_0} \\ e^{-2ix_1} & e^{-ix_1} & e^{0ix_1} & e^{ix_1} & e^{i2x_1} \\ e^{-2ix_2} & e^{-ix_2} & e^{0ix_2} & e^{ix_2} & e^{i2x_2} \\ e^{-2ix_3} & e^{-ix_3} & e^{0ix_3} & e^{ix_3} & e^{i2x_3} \\ e^{-2ix_4} & e^{-ix_4} & e^{0ix_4} & e^{ix_4} & e^{i2x_4} \end{bmatrix} \begin{bmatrix} \gamma_{-2} \\ \gamma_{-1} \\ \gamma_0 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix},$$

again with the natural generalization to larger odd integers $n$. At first blush this matrix looks no simpler than the one we first encountered, but a fascinating structure lurks. Notice that a generic entry of this matrix has the form $e^{\ell i x_k}$ for $\ell = -(n-1)/2, \ldots, (n-1)/2$ and $k = 0, \ldots, n-1$. Since $x_k = 2\pi k/n$, rewrite this entry as

$$e^{\ell i x_k} = \left(e^{ix_k}\right)^\ell = \left(e^{2\pi i k/n}\right)^\ell = \left(e^{2\pi i/n}\right)^{k\ell} = \omega^{k\ell},$$

where $\omega = e^{2\pi i/n}$ is an *nth root of unity*. In the $n = 5$ case, the linear system can thus be written as

This name comes from the fact that $\omega^n = 1$.

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^{-2} & \omega^{-1} & \omega^0 & \omega^1 & \omega^2 \\ \omega^{-4} & \omega^{-2} & \omega^0 & \omega^2 & \omega^4 \\ \omega^{-6} & \omega^{-3} & \omega^0 & \omega^3 & \omega^6 \\ \omega^{-8} & \omega^{-4} & \omega^0 & \omega^4 & \omega^8 \end{bmatrix} \begin{bmatrix} \gamma_{-2} \\ \gamma_{-1} \\ \gamma_0 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix}. \tag{1.30}$$

Denote this system by $\mathbf{F}\boldsymbol{\gamma} = \mathbf{f}$. Notice that each column of $\mathbf{F}$ equals some (entrywise) power of the vector

$$\begin{bmatrix} \omega^0 \\ \omega^1 \\ \omega^2 \\ \omega^3 \\ \omega^4 \end{bmatrix}.$$

In other words, *the matrix* $\mathbf{F}$ *has Vandermonde structure*. From our past experience with polynomial fitting addressed in Section 1.2.1, we

might fear that this formulation is ill-suited to numerical computations, i.e., solutions $\gamma$ to the system $\mathbf{F}\gamma = \mathbf{f}$ could be polluted by large numerical errors.

In the language of numerical linear algebra, we might fear that the matrix $\mathbf{F}$ is *ill-conditioned*, i.e., the *condition number* $\|\mathbf{F}\|\,\|\mathbf{F}^{-1}\|$ is large.

Before jumping to this conclusion, examine $\mathbf{F}^*\mathbf{F}$. To form $\mathbf{F}^*$ note that $\overline{\omega^{-\ell}} = \omega^{\ell}$, so

$\mathbf{F}^*$ is the conjugate-transpose of $\mathbf{F}$:

$$\mathbf{F}^* = \overline{\mathbf{F}}^{\mathrm{T}},$$

so $(\mathbf{F}^*)_{j,k} = \overline{\mathbf{F}_{k,j}}$.

$$\mathbf{F}^*\mathbf{F} = \begin{bmatrix} \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 \\ \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \omega^{-4} \\ \omega^0 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \omega^{-8} \end{bmatrix} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^{-2} & \omega^{-1} & \omega^0 & \omega^1 & \omega^2 \\ \omega^{-4} & \omega^{-2} & \omega^0 & \omega^2 & \omega^4 \\ \omega^{-6} & \omega^{-3} & \omega^0 & \omega^3 & \omega^6 \\ \omega^{-8} & \omega^{-4} & \omega^0 & \omega^4 & \omega^8 \end{bmatrix}.$$

The $(\ell, k)$ entry for $\mathbf{F}^*\mathbf{F}$ thus takes the form

$$(\mathbf{F}^*\mathbf{F})_{\ell,k} = \omega^0 + \omega^{(k-\ell)} + \omega^{2(k-\ell)} + \omega^{3(k-\ell)} + \omega^{4(k-\ell)}.$$

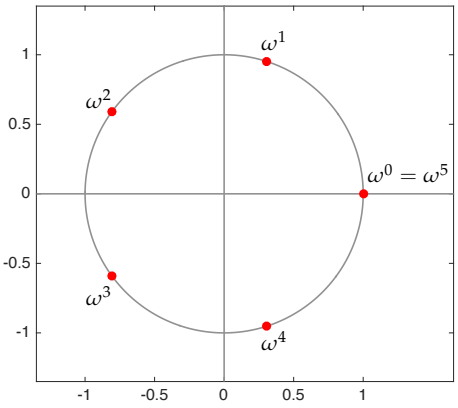
On the diagonal, when $\ell = k$, we simply have

$$(\mathbf{F}^*\mathbf{F})_{k,k} = \omega^0 + \omega^0 + \omega^0 + \omega^0 + \omega^0 = n.$$

On the off-diagonal, use $\omega^n = 1$ to see that all the off diagonal entries simplify to

$$(\mathbf{F}^*\mathbf{F})_{\ell,k} = \omega^0 + \omega^1 + \omega^2 + \omega^3 + \omega^4, \qquad \ell \neq k.$$

You can think of this last entry as $n$ times the average of $\omega^0$, $\omega^1$, $\omega^2$, $\omega^3$, and $\omega^4$, which are uniformly spaced points on the unit circle, shown in the plot to the right.




As these points are uniformly positioned about the unit circle, their mean must be zero, and hence

$$(\mathbf{F}^*\mathbf{F})_{\ell,k} = 0, \qquad \ell \neq k.$$

We thus must conclude that

$$\mathbf{F}^*\mathbf{F} = n\mathbf{I},$$

thus giving a formula for the inverse:

$$\mathbf{F}^{-1} = \frac{1}{n}\mathbf{F}^*.$$

The system $\mathbf{F}\gamma = \mathbf{f}$ can be immediately solved without the need for any factorization of $\mathbf{F}$:

$$\gamma = \frac{1}{n}\mathbf{F}^*\mathbf{f}.$$

The ready formula for $\mathbf{F}^{-1}$ is reminiscent of a *unitary* matrix. In fact, the matrices

$$\frac{1}{\sqrt{n}}\mathbf{F} \quad \text{and} \quad \frac{1}{\sqrt{n}}\mathbf{F}^*$$

$\mathbf{Q} \in \mathbb{C}^{n\times n}$ is unitary if and only if $\mathbf{Q}^{-1} = \mathbf{Q}^*$, or, equivalently, $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$.

are indeed unitary, and hence $\|n^{-1/2}\mathbf{F}\|_2 = \|n^{-1/2}\mathbf{F}^*\|_2 = 1$.

From this we can compute the condition number of $\mathbf{F}$:

$$\|\mathbf{F}\|_2\|\mathbf{F}^{-1}\|_2 = \frac{1}{n}\|\mathbf{F}\|_2\|\mathbf{F}^*\|_2 = \|n^{-1/2}\mathbf{F}\|_2\|n^{-1/2}\mathbf{F}^*\|_2 = 1.$$

*This special Vandermonde matrix is perfectly conditioned!* One can easily solve the system $\mathbf{F}\boldsymbol{\gamma} = \mathbf{f}$ to high precision. The key distinction between this case and standard polynomial interpolation is that now we have a Vandermonde matrix based on *points* $\mathrm{e}^{ix_k}$ *that are equally spaced about the unit circle in the complex plane*, whereas before our points were distributed over an interval on the real line. This distinction makes all the difference between an unstable matrix equation and one that is not only perfectly stable, but also forms the cornerstone of modern signal processing.

In fact, we have just computed the 'Discrete Fourier Transform' (DFT) of the data vector

$$\begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-1}) \end{bmatrix}.$$

The coefficients $\gamma_{-(n-1)/2}, \ldots, \gamma_{(n-1)/2}$ that make up the vector

$$\boldsymbol{\gamma} = \frac{1}{n}\mathbf{F}^*\mathbf{f}$$

are the *discrete Fourier coefficients* of the data in $\mathbf{f}$. From where does this name derive?

### 1.9.1 Connection to Fourier series

In a real analysis course, one learns that a $2\pi$-periodic function $f$ can be written as the Fourier series

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \mathrm{e}^{\mathrm{i}kx},$$

where the *Fourier coefficients* $c_\ell$ are defined via

$$c_k := \frac{1}{2\pi}\int_0^{2\pi} f(x)\mathrm{e}^{-\mathrm{i}kx}.$$

Notice that $\gamma_k = ((1/n)\mathbf{F}^*\mathbf{f})_k$ is an approximation to this $c_k$:

$$\begin{aligned}\gamma_k &= \frac{1}{n}\sum_{\ell=0}^{n-1} f(x_\ell)\omega^{-\ell k} \\ &= \frac{1}{n}\sum_{k=0}^{n-1} f(x_k)\mathrm{e}^{-(2\pi k/n)\mathrm{i}\ell} = \frac{1}{n}\sum_{k=0}^{n-1} f(x_k)\mathrm{e}^{-\mathrm{i}\ell x_k}.\end{aligned}$$

The matrix 2-norm is defined as

$$\|\mathbf{F}\|_2 = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{F}\mathbf{x}\|_2}{\|\mathbf{x}\|_2},$$

where the vector norm on the right hand side is the Euclidean norm

$$\|\mathbf{y}\|_2 = \Big(\sum_k |y_k|^2\Big)^{1/2} = (\mathbf{y}^*\mathbf{y})^{1/2}.$$

The 2-norm of a unitary matrix is one: If $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$, then

$$\|\mathbf{Q}\mathbf{x}\|_2^2 = \mathbf{x}^*\mathbf{Q}^*\mathbf{Q}\mathbf{x} = \mathbf{x}^*\mathbf{x} = \|\mathbf{x}\|^2,$$

so $\|\mathbf{Q}\|_2 = 1$.

To ensure pointwise convergence of this series for all $x \in [0, 2\pi]$, $f$ must be a continuous $2\pi$-periodic function with a continuous first derivative. The functions $e_k(x) = \mathrm{e}^{\mathrm{i}kx}/\sqrt{2\pi}$ form an orthonormal basis for the space $L^2[0, 2\pi]$ with the inner product

$$(f, g) = \int_0^{2\pi} f(x)\overline{g(x)}\,\mathrm{d}x.$$

The Fourier series is simply an expansion of $f$ in this basis: $f = \sum_k (f, e_k)\, e_k$.

Now use the fact that $f(x_0)e^{-i\ell x_0} = f(x_n)e^{-i\ell x_n}$ to view the last sum as a *composite trapezoid rule* approximation of an integral:

The composite trapezoid rule will be discussed in Chapter 3.

$$\begin{aligned} 2\pi\gamma_\ell &= \frac{2\pi}{n}\left(\frac{1}{2}f(x_0)e^{-i\ell x_0} + \sum_{k=1}^{n-1} f(x_k)e^{-i\ell x_k} + \frac{1}{2}f(x_n)e^{-i\ell x_n}\right) \\ &\approx \int_0^{2\pi} f(x)e^{-i\ell x}\,dx \\ &= 2\pi c_\ell. \end{aligned}$$

*The coefficient $\gamma_\ell$ that premultiplies $e^{i\ell x}$ in the trigonometric interpolating polynomial is actually an approximation of the Fourier coefficient $c_\ell$.*

Let us go one step further. Notice that the trigonometric interpolant

$$t_n(x) = \sum_{k=-(n-1)/2}^{(n-1)/2} \gamma_k e^{ikx}$$

is an approximation to the Fourier series

$$f(x) = \sum_{k=-\infty}^{\infty} c_k\, e^{ikx}$$

obtained by (1) truncating the series, and (2) replacing $c_k$ with $\gamma_k$. To assess the quality of the approximation, we need to understand the magnitude of the terms dropped from the sum, as well as the accuracy of the composite trapezoid rule approximation $\gamma_k$ to $c_k$. We will thus postpone discussion of $f(x) - t_n(x)$ until we develop a few more analytical tools in the next two chapters.

### 1.9.2 *Computing the discrete Fourier coefficients*

Normally we would require $O(n^2)$ operations to compute these coefficients using matrix-vector multiplication with $\mathbf{F}^*$, but Cooley and Tukey discovered in 1965 that given the amazing structure in $\mathbf{F}^*$, one can arrange operations so as to compute $\boldsymbol{\gamma} = n^{-1}\mathbf{F}^*\mathbf{f}$ in only $O(n \log n)$ operations: a procedure that we now famously call the *Fast Fourier Transform* (FFT).

Apparently the FFT was discovered earlier by Gauss, but it was forgotten, given its limited utility before the advent of automatic computation. Jack Good (Bletchley Park codebreaker and, later, a Virginia Tech statistician) published a similar idea in 1958. Good recalls: 'John Tukey (December 1956) and Richard L. Garwin (September 1957) visited Cheltenham and I had them round to steaks and fries on separate occasions. I told Tukey briefly about my FFT (with little detail) and, in Cooley and Tukey's well known paper of 1965, my 1958 paper is the only citation.' See D. L. Banks, 'A conversation with I. J. Good,' *Stat. Sci.* 11 (1996) 1–19.

We can summarize this section as follows.

> The FFT of a vector of uniform samples of a $2\pi$-periodic function $f$ gives the coefficients for the trigonometric interpolant to $f$ at those sample points. These coefficients approximate the function's Fourier coefficients.

**Example 1.8** (Trig interpolation of a smooth periodic function)**.** Figure 1.14 shows the degree $n = 5, 7, 9$ and 11 trigonometric interpolants to the $2\pi$-periodic function $f(x) = e^{\cos(x)+\sin(2x)}$. Notice that although all the interpolation points are all drawn from the interval $[0, 2\pi)$ (indicated by the gray region on the plot), the interpolants are just as accurate outside this region. In contrast, a standard polynomial fit through the same points will behave very differently: (non-constant) polynomials must satisfy $|p_n(x)| \to \infty$ as $|x| \to \infty$. Figure 1.15 shows this behavior for $n = 7$: for $x \in [0, 2\pi]$, the polynomial fit to $f$ is about as accurate as the $n = 7$ trigonometric polynomial in Figure 1.14. Outside of $[0, 2\pi]$,the polynomial is much worse.

**Example 1.9** (Trig interpolation of non-smooth function)**.** Figure 1.15 shows that a standard (non-perioidic) polynomial fit to a periodic function can yield a good approximation, at least over the interval from which the interpolation points are drawn. Now turn the tables: how well does a (periodic) trigonometric polynomial fit a smooth but non-periodic function? Simply take $f(x) = x$ on $[0, 2\pi]$, and construct the trigonometric interpolant as described above for $n = 11$. The top plot in Figure 1.16 shows that $t_{11}$ gives a very poor approximation to $f$, constrained by design to be periodic even though $f$ is not. The fact that $f(2\pi) \neq f(0)$ acts like a discontinuity, vastly impairing the quality of the approximation. The bottom plot in Figure 1.16 repeats this exercise for $f(x) = (x - \pi)^2$. Since in this case $f(0) = f(2\pi)$ we might expect better results; indeed, the approximation looks quite reasonable. Note, however, that $f'(0) \neq f'(\pi)$, and this lack of smoothness severely slows convergence of $t_n$ to $f$ as $n \to \infty$. Figure 1.17 contrasts this slow rate of convergence with the much faster convergence observed for $f(x) = e^{\cos(x)+\sin(2x)}$ used in Figure 1.14. Clearly the periodic interpolant is much better suited to smooth $f$.

### *1.9.3 Fast MATLAB implementation*

MATLAB organizes its Fast Fourier Transform is a slightly different fashion than we have described above. To fit with MATLAB, reorder the unknowns in the system (1.30) to obtain

$$\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^{-2} & \omega^{-1} \\ \omega^0 & \omega^2 & \omega^4 & \omega^{-4} & \omega^{-2} \\ \omega^0 & \omega^3 & \omega^6 & \omega^{-6} & \omega^{-3} \\ \omega^0 & \omega^4 & \omega^8 & \omega^{-8} & \omega^{-4} \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_{-2} \\ \gamma_{-1} \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix}, \tag{1.31}$$

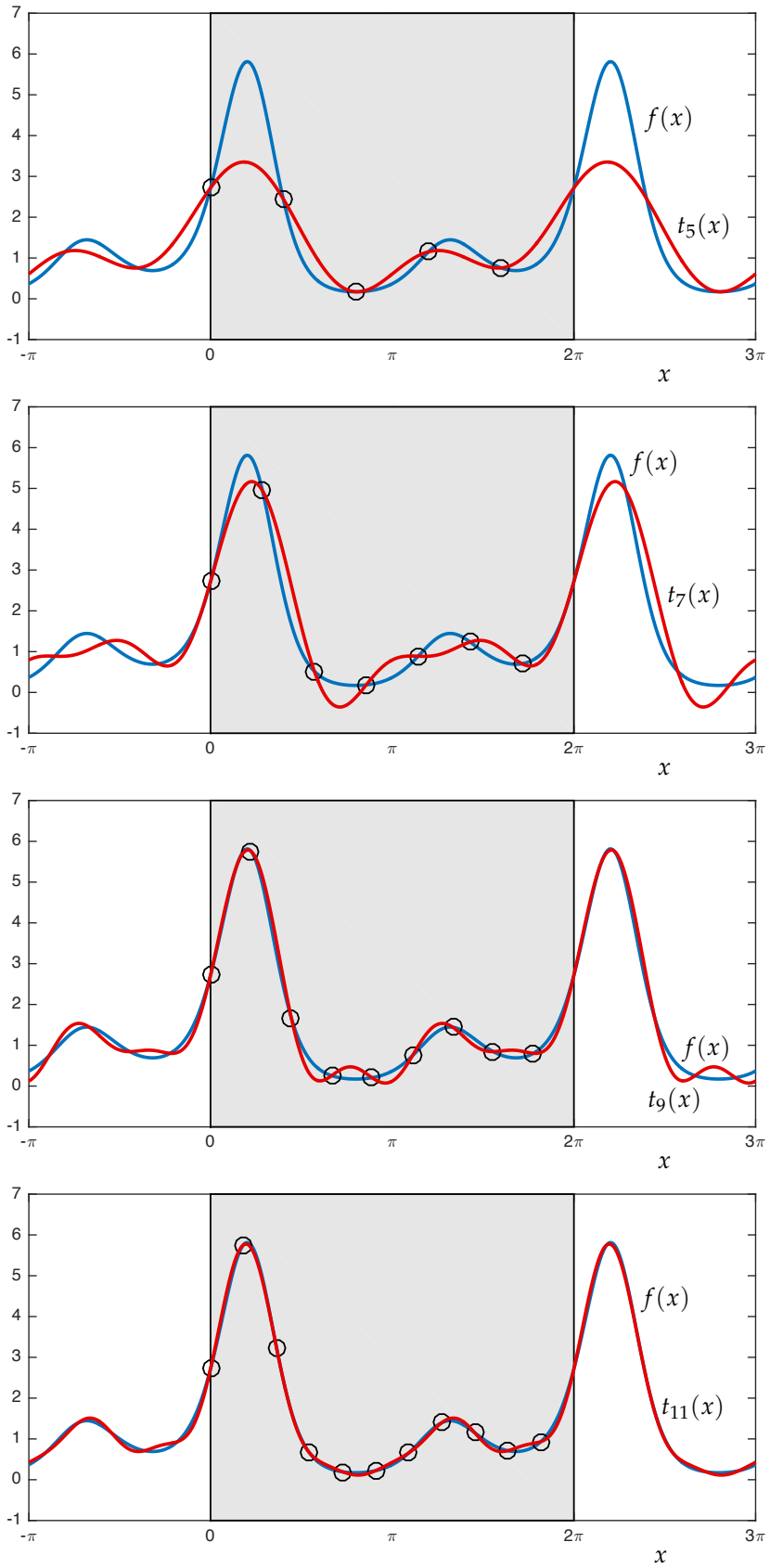



Figure 1.14: Trigonometric interpolant to $2\pi$-periodic function $f(x) = e^{\cos(x)+\sin(2x)}$, using $n = 5, 7, 9$ and 11 points uniformly spaced over $[0, 2\pi)$ ($\{x_k\}_{k=0}^{n}$ for $x_k = 2\pi k/n$). Since both $f$ and the interpolant are periodic, the function fits well throughout $\mathbb{R}$, not just on the interval for which the interpolant was designed.

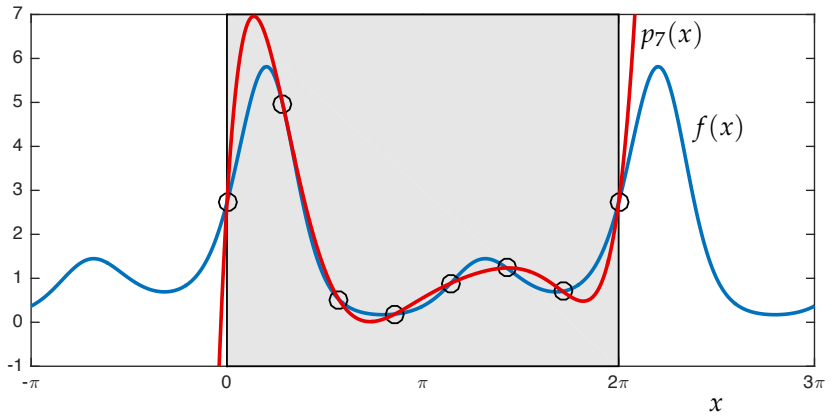



Figure 1.15: Polynomial fit of degree $n = 7$ through uniformly spaced grid points $x_0, \ldots, x_n$ for $x_j = 2\pi j/n$, for the same function $f(x) = e^{\cos(x)+\sin(2x)}$ used in Figure 1.14. In contrast to the trigonometric fits in the earlier figure, the polynomial grows very rapidly outside the interval $[0, 2\pi]$. Moral: if your function is periodic, fit it with a trigonometric polynomial.

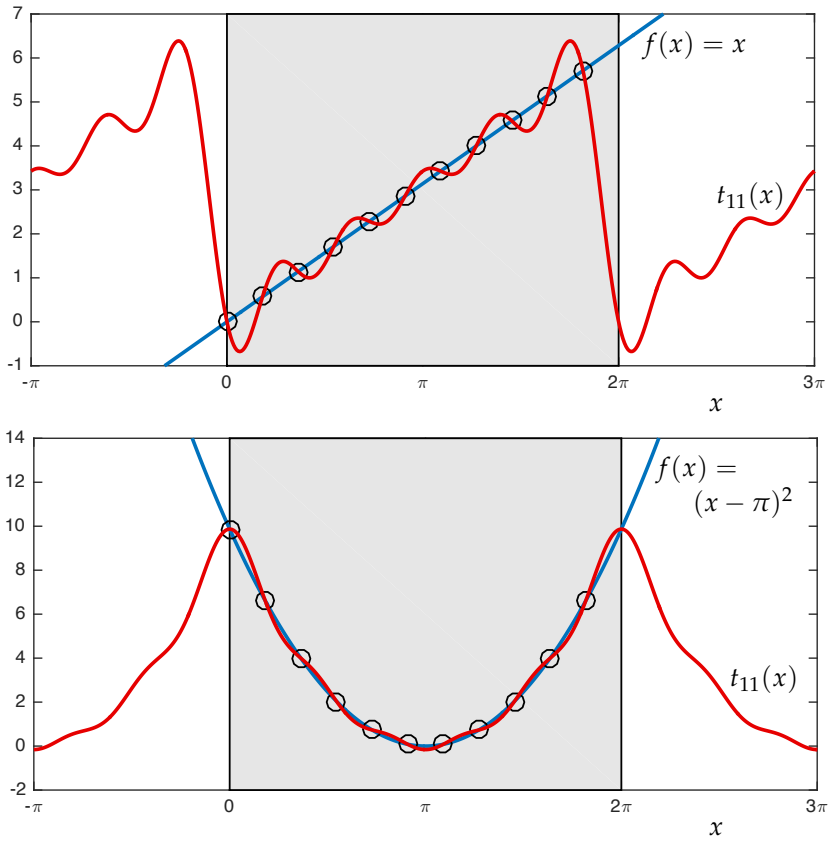



Figure 1.16: Trigonometric polynomial fit of degree $n = 11$ through uniformly spaced grid points $x_0, \ldots, x_n$ for $x_j = 2\pi j/n$, for the *non-periodic* function $f(x) = x$ (top) and for $f(x) = (x-\pi)^2$ (bottom). By restricting the latter function to the domain $[0, 2\pi]$, one can view it as a continuous periodic function with a jump discontinuity in the first derivative. The interpolant $t_{11}$ seems to give a good approximation to $f$, but the discontinuity in the derivative slows the convergence of $t_n$ to $f$ as $n \to \infty$.

which amounts to reordering the *columns* of the matrix in (1.30). You can obtain this matrix by the command `ifft(eye(n))`. For $n = 5$,

$$5 * \texttt{ifft(eye(5))} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^{-2} & \omega^{-1} \\ \omega^0 & \omega^2 & \omega^4 & \omega^{-4} & \omega^{-2} \\ \omega^0 & \omega^3 & \omega^6 & \omega^{-6} & \omega^{-3} \\ \omega^0 & \omega^4 & \omega^8 & \omega^{-8} & \omega^{-4} \end{bmatrix}.$$

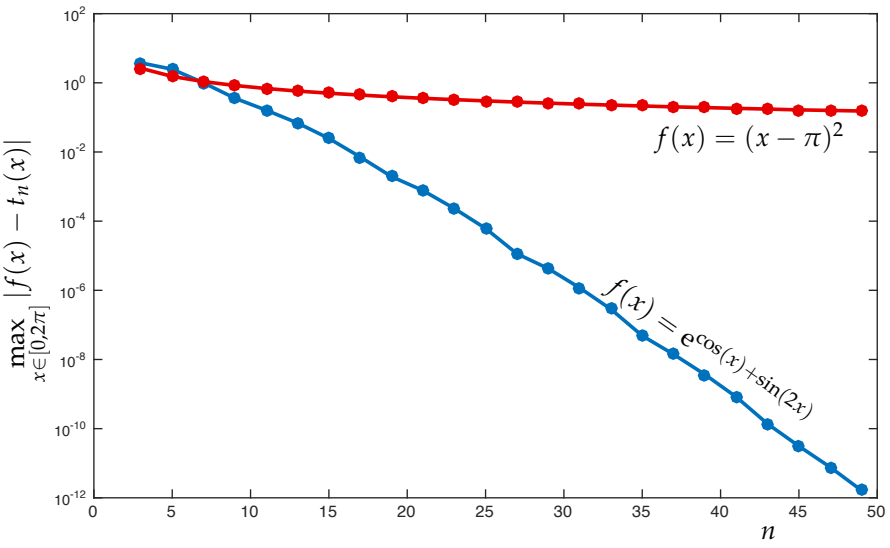



Figure 1.17: Convergence of the trigonometric polynomial interpolants to $f(x) = e^{\cos(x)+\sin(2x)}$ and $f(x) = (x-\pi)^2$. For the first function, convergence is extremely rapid as $n \to \infty$. The second function, restricted to $[0, 2\pi]$, can be viewed as a continuous but not continuously differentiable function. Though the approximation in Figure 1.16 looks good over $[0, 2\pi]$, the convergence of $t_n$ to $f$ is slow as $n \to \infty$.

Similarly, the inverse of this matrix can be computed from `fft(eye(n))` command:

$$\texttt{fft(eye(5))/5} = \frac{1}{5}\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \omega^{-4} \\ \omega^0 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \omega^{-8} \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 \end{bmatrix}.$$

We could construct this matrix and multiply it against $\mathbf{f}$ to obtain $\gamma$, but that would require $\mathcal{O}(n^2)$ operations. Instead, we can compute $\gamma$ directly using the `fft` command:

$$\gamma = \texttt{fft}(\mathbf{f})/\texttt{n}.$$

Recall that this reordered vector gives

$$\gamma = \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_{-2} \\ \gamma_{-1} \end{bmatrix},$$

which must be taken into account when constructing $t_n$.

**Example 1.10** (MATLAB code for trigonometric interpolation)**.** We close with a sample of MATLAB code one could use to construct the interpolant $t_n$ for the function $f(x) = e^{\cos(x)+\sin(2x)}$. First we present a generic code that will work for any (real- or complex-

valued) $2\pi$-periodic $f$. Take special note of the simple one line command to find the coefficient vector $\gamma$.

```
f = @(x) exp(cos(x)+sin(2*x));                % define the function
n = 5;                                        % # terms in trig polynomial (must be odd)
xk = [0:n-1]'*2*pi/n;                         % interpolation points
xx = linspace(0,2*pi,500)';                   % fine grid on which to plot f, t_n
tn = zeros(size(xx));                         % initialize t_n

gamma = fft(f(xk))/n;                         % solve for coefficients, gamma

for k=1:(n+1)/2
   tn = tn + gamma(k)*exp(1i*(k-1)*xx);       % gamma_0, gamma_1, ... gamma_{(n-1)/2} terms
end
for k=(n+1)/2+1:n
   tn = tn + gamma(k)*exp(1i*(-n+k-1)*xx);    % gamma_{-(n-1)/2}, ..., gamma_{-1} terms
end
plot(xx,f(xx),'b-'), hold on                  % plot f
plot(xx, tn,'r-')                             % plot t_n
```

In the case that $f$ is real-valued (as with all the examples shown in this section), one can further show that

$$\gamma_{-k} = \overline{\gamma_k},$$

indicating that the imaginary terms will not make any contribution to $t_n$. Since for $k = 1, \ldots, (n-1)/2$,

$$\gamma_{-k} e^{-1ikx} + \gamma_k e^{1ikx} = 2\Big(\mathrm{Re}(\gamma_k)\cos(kx) - \mathrm{Im}(\gamma_k)\sin(kx)\Big),$$

the code can be simplified slightly to construct $t_n$ as follows.

```
gamma = fft(f(xk))/n;                         % solve for coefficients, gamma
tn = gamma(1)*exp(1i*0*xx);                   % initialize t_n(x) = gamma_0
for k=2:(n+1)/2
   tn = tn + 2*real(gamma(k))*cos((k-1)*xx) ...
           - 2*imag(gamma(k))*sin((k-1)*xx);  % exploit gamma_{-k} = conj(gamma_k)
end
```

# LECTURE 8: *Piecewise interpolation*

## *1.10 Piecewise polynomial interpolation*

WE HAVE SEEN, through Runge's example, that high degree polynomial interpolation can lead to large errors when the $(n+1)$st derivative of $f$ is large in magnitude. In other cases, the interpolant converges to $f$, but the polynomial degree must be fairly high to deliver an approximation of acceptable accuracy throughout $[a, b]$. Beyond theoretical convergence questions, high-degree polynomials can be delicate to work with, even when using a stable implementation (the Lagrange basis, in its barycentric form). Many practical approximation problems are better solved by a simpler 'piecewise' alternative: instead of approximating $f$ with one high-degree interpolating polynomial over a large interval $[a, b]$, patch together many low-degree polynomials that each interpolate $f$ on some subinterval of $[a, b]$.

### *1.10.1 Piecewise linear interpolation*

The simplest piecewise polynomial interpolation uses linear polynomials to interpolate between adjacent data points. Informally, the idea is to 'connect the dots.' Given $n+1$ data points $\{(x_j, f_j)\}_{j=0}^n$, we need to construct $n$ linear polynomials $\{s_j\}_{j=1}^n$ such that

$$s_j(x_{j-1}) = f_{j-1}, \qquad \text{and} \qquad s_j(x_j) = f_j$$

for each $j = 1, \ldots, n$. It is simple to write down a formula for these polynomials,

$$s_j(x) = f_j - \frac{(x_j - x)}{(x_j - x_{j-1})}(f_j - f_{j-1}).$$

Note that all the $s_j$'s are *linear* polynomials. Unlike our earlier notation, the subscript $j$ *does not* reflect the polynomial degree.

Each $s_j$ is valid on $x \in [x_{j-1}, x_j]$, and the interpolant $S(x)$ is defined as $S(x) = s_j(x)$ for $x \in [x_{j-1}, x_j]$.

To analyze the error, we can apply the interpolation bound developed in the last lecture. If we let $\Delta$ denote the largest space between interpolation points,

$$\Delta := \max_{j=1,\ldots,n} |x_j - x_{j-1}|,$$

then the standard interpolation error bound gives

$$\max_{x \in [x_0, x_n]} |f(x) - S(x)| \leq \max_{x \in [x_0, x_n]} \frac{|f''(x)|}{2} \Delta^2.$$

In particular, this proves convergence as $\Delta \to 0$ provided $f \in C^2[x_0, x_n]$.

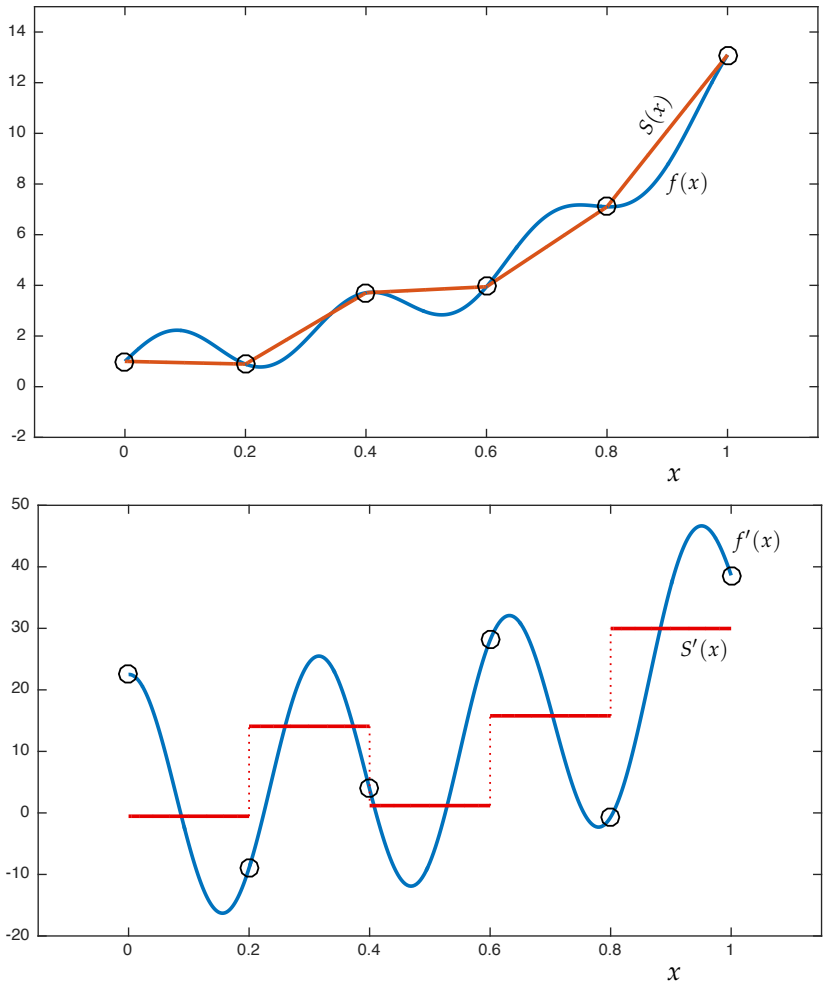



Figure 1.18: Piecewise linear interpolant to $f(x) = \sin(20x) + e^{5x/2}$ at $n = 5$ uniformly spaced points (top), and the derivative of this interpolant (bottom). Notice that the interpolant is continuous, but its derivative has jump discontinuities.

What could go wrong with this simple approach? The primary difficulty is that the interpolant is *continuous*, but generally not *continuously differentiable*. Still, these functions are easy to construct and cheap to evaluate, and can be very useful despite their simplicity.

### 1.10.2 *Piecewise cubic Hermite interpolation*

To remove the discontinuities in the first derivative of the piecewise linear interpolant, we begin by modeling our data with cubic polynomials over each interval $[x_j, x_{j+1}]$. Each such cubic has four free parameters (since $\mathcal{P}_3$ is a vector space of dimension 4); we require

these polynomials to interpolate both $f$ and its first derivative:

$$\begin{aligned}
s_j(x_{j-1}) &= f(x_{j-1}), && j = 1, \ldots, n; \\
s_j(x_j) &= f(x_j), && j = 1, \ldots, n; \\
s_j'(x_{j-1}) &= f'(x_{j-1}), && j = 1, \ldots, n; \\
s_j'(x_j) &= f'(x_j), && j = 1, \ldots, n.
\end{aligned}$$

To satisfy these conditions, take $s_j$ to be the Hermite interpolant to the data $(x_{j-1}, f(x_{j-1}), f'(x_{j-1}))$ and $(x_j, f(x_j), f'(x_j))$. The resulting function, $S(x) := s_j(x)$ for $x \in [x_{j-1}, x_j]$, will always have a continuous derivative, $S \in C^1[x_0, x_n]$, but generally $S \notin C^2[x_0, x_n]$ due to discontinuities in the second derivative at the interpolation points.

In many applications, we lack specific values for $S'(x_j) = f'(x_j)$; we simply want the function $S(x)$ to be as *smooth* as possible. That motivates our next topic: splines.

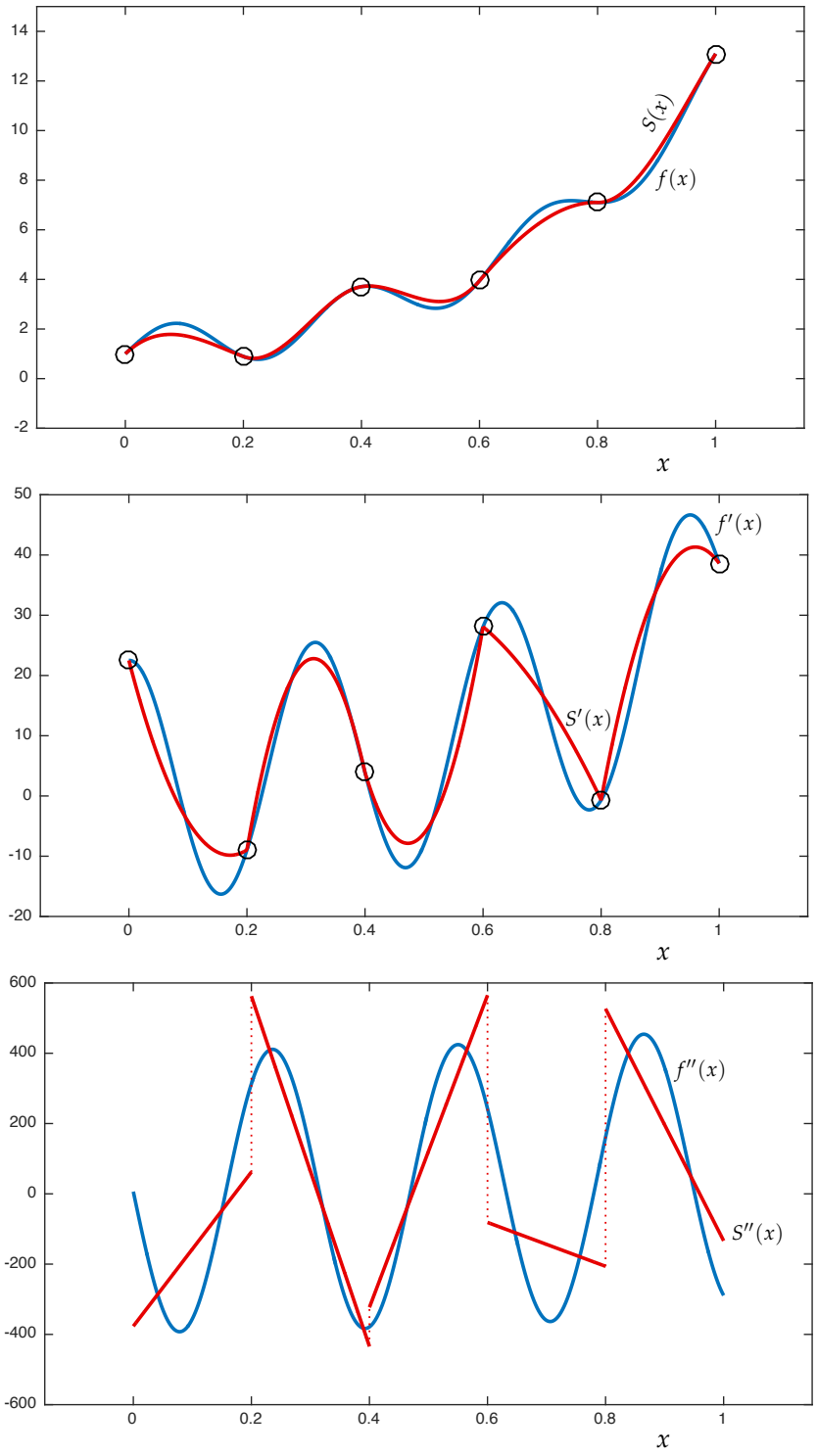



Figure 1.19: Piecewise cubic Hermite interpolant to $f(x) = \sin(20x) + e^{5x/2}$ at $n = 5$ uniformly spaced points (top), and the derivative of this interpolant (middle). Now both the interpolant and its derivative are continuous, and the derivative interpolates $f'$. However, the second derivative of the interpolant now has jump discontinuities (bottom).

# LECTURE 9: *Introduction to Splines*

## *1.11 Splines*

Spline fitting, our next topic in interpolation theory, is an essential tool for engineering design. As in the last lecture, we strive to interpolate data using low-degree polynomials between consecutive grid points. The piecewise linear functions of Section 1.10 were simple, but suffered from unsightly kinks at each interpolation point, reflecting a discontinuity in the first derivative. By increasing the degree of the polynomial used to model $f$ on each subinterval, we can obtain smoother functions.

Long before numerical analysts got their hands on them, 'splines' were used in the woodworking, shipbuilding, and aircraft industries. In such work 'splines' refer to thin pieces of wood that are bent between physical constraints called *ducks* (apparently these were also called *dogs* and *rats* in some settings; modern versions are sometimes called *whales* because of their shape). The spline, a thin beam, bends gracefully between the ducks to give a graceful curve. For some discussion of this history, see the brief 'History of Splines' note by James Epperson in the 19 July 1998 NA Digest, linked from the class website. For a beautiful derivation of cubic splines from Euler's beam equation—that is, from the original physical situation, see Gilbert Strang's *Introduction to Applied Mathematics*, Wellesley Cambridge Press, 1986.

### *1.11.1 Cubic splines: first approach*

Rather than setting $S'(x_j)$ to a particular value, suppose we simply require $S'$ to be continuous throughout $[x_0, x_n]$. This added freedom allows us to impose a further condition: require $S''$ to be continuous on $[x_0, x_n]$, too. The polynomials we construct are called *cubic splines*. In spline parlance, the interpolation points $\{x_j\}_{j=0}^n$ are called *knots*.

These cubic spine requirements can be written as:

$$\begin{aligned}
s_j(x_{j-1}) &= f(x_{j-1}), && j = 1, \ldots, n;\\
s_j(x_j) &= f(x_j), && j = 1, \ldots, n;\\
s_j'(x_j) &= s_{j+1}'(x_j), && j = 1, \ldots, n-1;\\
s_j''(x_j) &= s_{j+1}''(x_j), && j = 1, \ldots, n-1.
\end{aligned}$$

Compare these requirements to those imposed by piecewise cubic Hermite interpolation. Add up all these new requirements:

$$n + n + (n-1) + (n-1) = 4n - 2 \text{ constraints}$$

and compare to the total free variables available:

$$(n \text{ cubic polynomials}) \times (4 \text{ variables per cubic}) = 4n \text{ variables}.$$

So far, we thus have an *underdetermined system*, and there will be infinitely many choices for the function $S(x)$ that satisfy the constraints.

There are several canonical ways to add two extra constraints that uniquely define $S$:

- *natural* splines require $S''(x_0) = S''(x_n) = 0$;
- *complete* splines specify values for $S'(x_0)$ and $S'(x_n)$;
- *not-a-knot* splines require $S'''$ to be continuous at $x_1$ and $x_{n-1}$.

Since the third derivative of a cubic is a constant, the *not-a-knot* requirement forces $s_1 = s_2$ and $s_{n-1} = s_n$. Hence, while $S(x)$ interpolates the data at $x_2$ and $x_{n-1}$, the derivative continuity requirements are automatic at those knots; hence the name "not-a-knot".

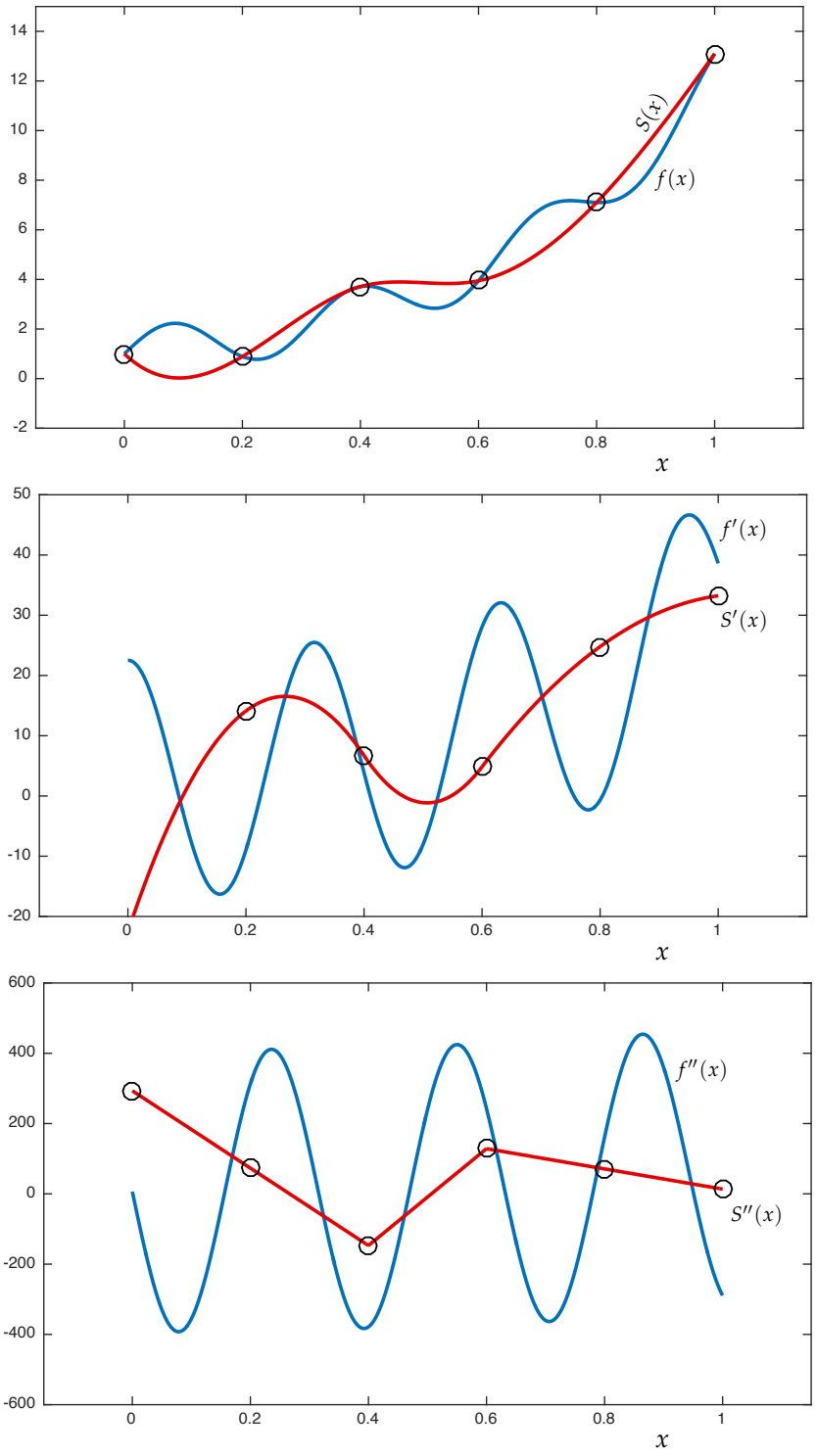



Figure 1.20: Not-a-knot cubic spline interpolant to $f(x) = \sin(20x) + e^{5x/2}$ at $n = 5$ uniformly spaced knots (top), along with its first (middle) and second (bottom) derivative. Note that $S$, $S'$, and $S''$ are all continuous. Look closely at the plot of $S''$: clearly this function will have jump discontinuities at the interior nodes $x_2$ and $x_3$, but the *not-a-knot* condition forces $S'''$ to be continuous at the knots $x_1$ and $x_4 = x_{n-1}$.

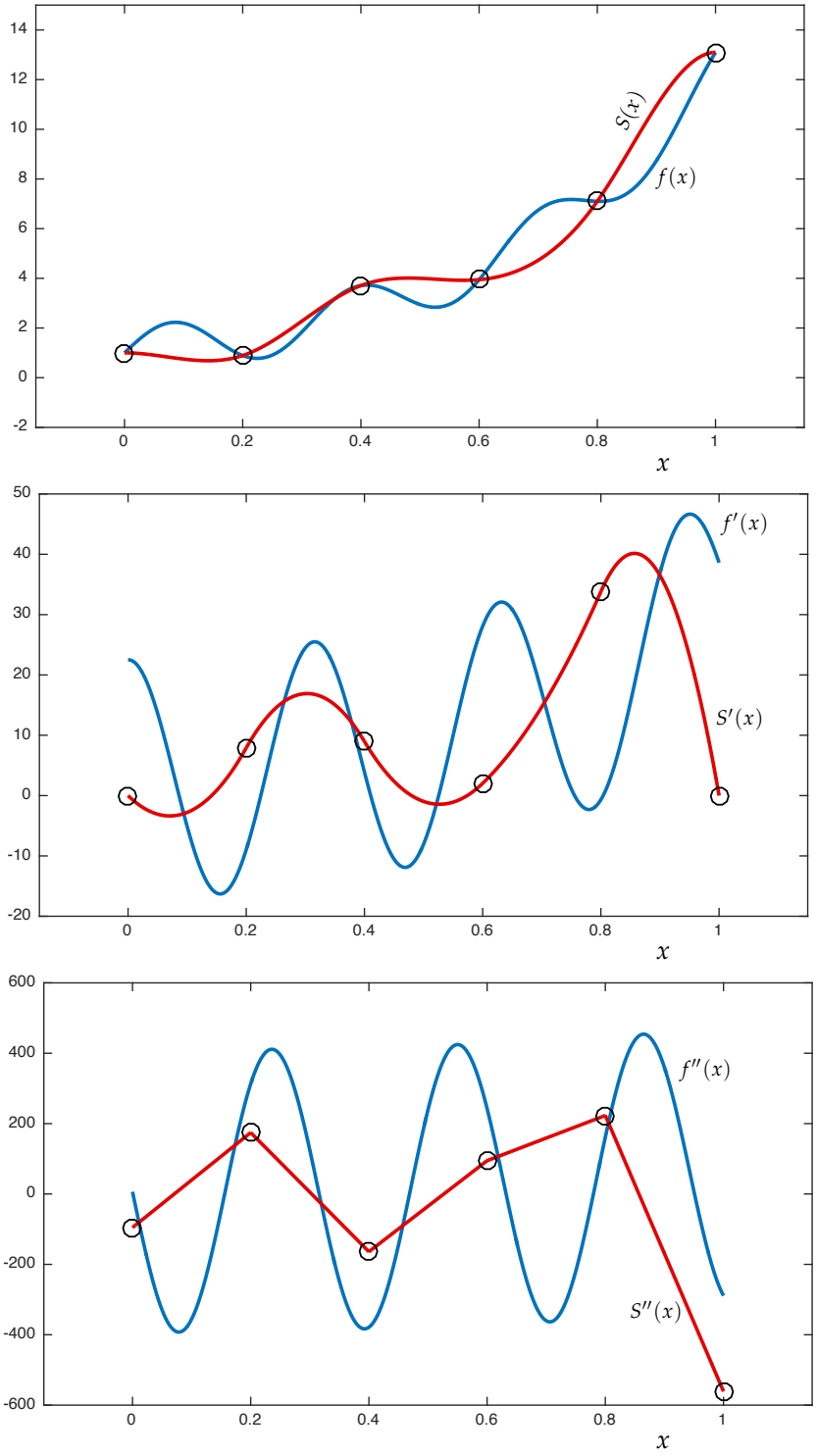



Figure 1.21: Complete cubic spline interpolant to $f(x) = \sin(20x) + e^{5x/2}$ at $n = 5$ uniformly spaced knots (top), along with its first (middle) and second (bottom) derivative. Note that $S$, $S'$, and $S''$ are all continuous. For a *complete* cubic spline, one specifies the value of $S'(x_0)$ and $S'(x_n)$; in this case we have set $S'(x_0) = S'(x_n) = 0$, as you can confirm in the middle plot. In the bottom plot, see that $S'''(x)$ will have jump discontinuities at all the interior knots $x_1, \ldots, x_{n-1}$, in contrast to the not-a-knot spline shown in Figure 1.20.

Natural cubic splines are a popular choice for they can be shown, in a precise sense, to minimize curvature over all the other possible splines. They also model the physical origin of splines, where beams of wood extend straight (i.e., zero second derivative) beyond the first and final 'ducks.'

Continuing with the example from the last section, Figure 1.20 shows a not-a-knot spline, where $S'''$ is continuous at $x_1$ and $x_{n-1}$. The cubic polynomials $s_1$ for $[x_0, x_1]$ and $s_2$ for $[x_1, x_2]$ must satisfy

$$\begin{aligned} s_1(x_1) &= s_2(x_1) \\ s_1'(x_1) &= s_2'(x_1) \\ s_1''(x_1) &= s_2''(x_1) \\ s_1'''(x_1) &= s_2'''(x_1) \end{aligned}$$

Two cubics that match these four conditions must be the same: $s_1(x) = s_2(x)$, and hence $x_1$ is 'not a knot.' (The same goes for $x_{n-1}$.) Notice this behavior in Figure 1.20. In contrast, Figure 1.21 shows the complete cubic spline, where $S'(x_0) = S'(x_n) = 0$.

However we assign the two additional conditions, we get a system of $4n$ equations (the various constraints) in $4n$ unknowns (the cubic polynomial coefficients). These equations can be set up as a system involving a banded coefficient matrix (zero everywhere except for a limited number of diagonals on either side of the main diagonal). We could derive this linear system by directly enforcing the continuity conditions on the cubic polynomial that we have just described. Instead, we will develop a more general approach that expresses the spline function $S(x)$ as the linear combination of special basis functions, which themselves are splines.

One can arrange Gaussian elimination to solve an $n \times n$ tridiagonal system in $\mathcal{O}(n)$ operations.

Try constructing this matrix!

# LECTURE 10: *B-Splines*

### 1.11.2 *B-Splines: a basis for splines*

Throughout our discussion of standard polynomial interpolation, we viewed $\mathcal{P}_n$ as a linear space of dimension $n+1$, and then expressed the unique interpolating polynomial in several different bases (monomial, Newton, Lagrange). The most elegant way to develop spline functions uses the same approach. A set of *basis splines*, depending only on the location of the knots and the degree of the approximating piecewise polynomials can be developed in a convenient, numerically stable manner. (Cubic splines are the most prominent special case.)

For example, each cubic basis spline, or *B-spline*, is a continuous piecewise-cubic function with continuous first and second derivatives. Thus any linear combination of such B-splines will inherit the same continuity properties. The coefficients in the linear combination are chosen to obey the specified interpolation conditions.

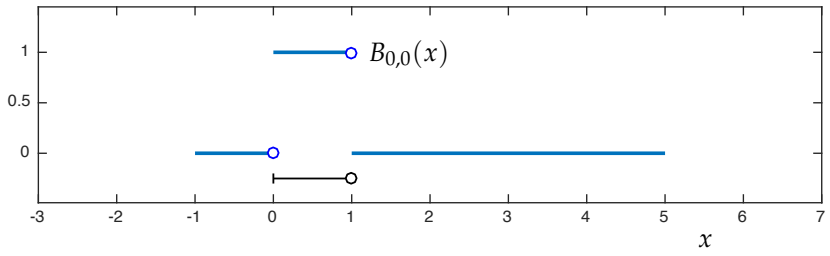
B-splines are built up recursively from constant B-splines. Though we are interpolating data at $n+1$ knots $x_0, \ldots, x_n$, to derive B-splines we need extra nodes outside $[x_0, x_n]$ as scaffolding upon which to construct the basis. Thus, add knots on either end of $x_0$ and $x_n$:

$$\cdots < x_{-2} < x_{-1} < x_0 < x_1 < \cdots < x_n < x_{n+1} < \cdots .$$

Given these knots, define the constant (zeroth-degree) B-splines:

$$B_{j,0}(x) = \begin{cases} 1 & x \in [x_j, x_{j+1}); \\ 0 & \text{otherwise.} \end{cases}$$

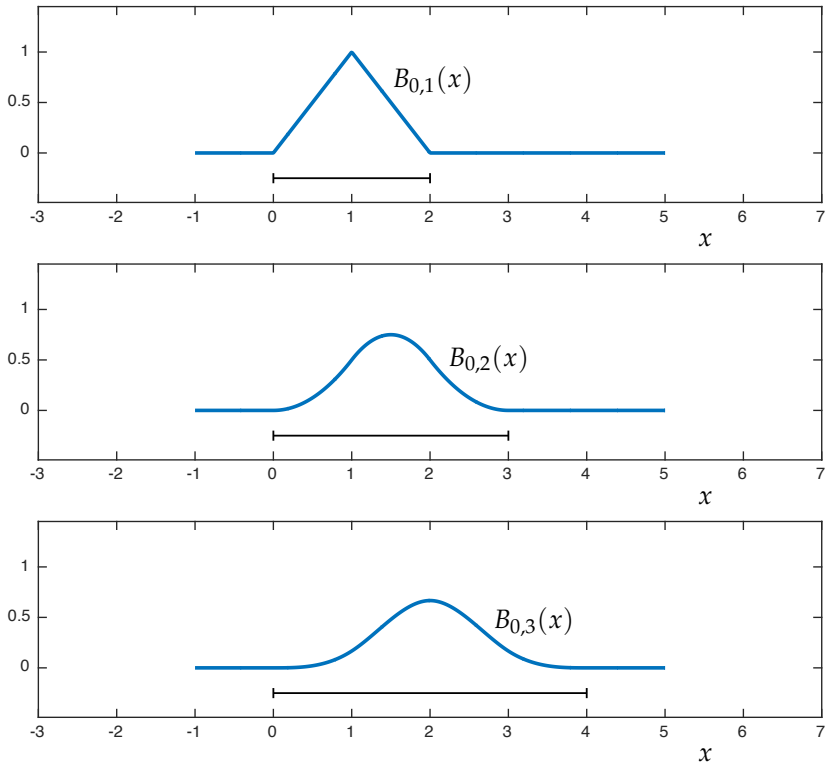
The following plot shows the basis function $B_{0,0}$ for the knots $x_j = j$. Note, in particular, that $B_{j,0}(x_{j+1}) = 0$. The line drawn beneath the spline marks the *support* of the spline, that is, the values of $x$ for which $B_{0,0}(x) \neq 0$.




From these degree-0 B-splines, manufacture B-splines of higher degree via the recurrence

$$B_{j,k}(x) = \left(\frac{x - x_j}{x_{j+k} - x_j}\right) B_{j,k-1}(x) + \left(\frac{x_{j+k+1} - x}{x_{j+k+1} - x_{j+1}}\right) B_{j+1,k-1}(x). \tag{1.32}$$

While not immediately obvious from the formula, this construction ensures that $B_{j,k}$ has one more continuous derivative than does $B_{j,k-1}$. Thus, while $B_{j,0}$ is discontinuous (see previous plot), $B_{j,1}$ is continuous, $B_{j,2} \in C^1(\mathbb{R})$, and $B_{j,3} \in C^2(\mathbb{R})$. One can see this in the three plots below, where again $x_j = j$. As the degree increases, the B-spline $B_{j,k}$ becomes increasingly smooth. Smooth is good, but it has a consequence: the *support* of $B_{j,k}$ gets larger as we increase $k$. This, as we will see, has implications on the number of nonzero entries in the linear system we must ultimately solve to find the expansion of the desired spline in the B-spline basis.




From these plots and the recurrence defining $B_{j,k}$, one can deduce several important properties:

- $B_{j,k} \in C^{k-1}(\mathbb{R})$ (continuity);
- $B_{j,k}(x) = 0$ if $x \notin (x_j, x_{j+k+1})$ (compact support);
- $B_{j,k}(x) > 0$ for $x \in (x_j, x_{j+k+1})$ (positivity).

Finally, we are prepared to write down a formula for the spline that interpolates $\{(x_j, f_j)\}_{j=0}^n$ as a linear combination of the basis splines we have just constructed. Let $S_k(x)$ denote the spline consisting of piecewise polynomials in $\mathcal{P}_k$. In particular, $S_k$ must obey the following properties:

- $S_k(x_j) = f_j$ for $j = 0, \ldots, n$;
- $S_k \in C^{k-1}[x_0, x_n]$ for $k \geq 1$.

The beauty of B-splines is that the second of these properties is automatically inherited from the B-splines themselves. (Any linear combination of $C^{k-1}(\mathbb{R})$ functions must itself be a $C^{k-1}(\mathbb{R})$ function.) The interpolation conditions give $n+1$ equations that constrain the unknown coefficients $c_{j,k}$ in the expansion of $S_k$:

$$S_k(x) = \sum_j c_{j,k} B_{j,k}(x). \tag{1.33}$$

What limits should $j$ have in this sum? For the greatest flexibility, let $j$ range over all values for which

$$B_{j,k}(x) \neq 0 \quad \text{for some } x \in [x_0, x_n].$$

If $B_{j,k}(x) = 0$ for all $x \in [x_0, x_n]$, it cannot contribute to the interpolation requirement $S_k(x_j) = f_j$, $j = 0, \ldots, n$.

Figure 1.22 shows the B-splines of degree $k = 1, 2, 3$ that overlap the interval $[x_0, x_4]$ for $x_j = j$. For $k \geq 1$, $B_{j,k}(x)$ is supported on $(x_j, x_{j+k+1})$, and hence the limits on the sum in (1.33) take the form

$$S_k(x) = \sum_{j=-k}^{n-1} c_{j,k} B_{j,k}(x), \qquad k \geq 1. \tag{1.34}$$

The sum involves $n + k$ coefficients $c_{j,k}$, which must be determined to

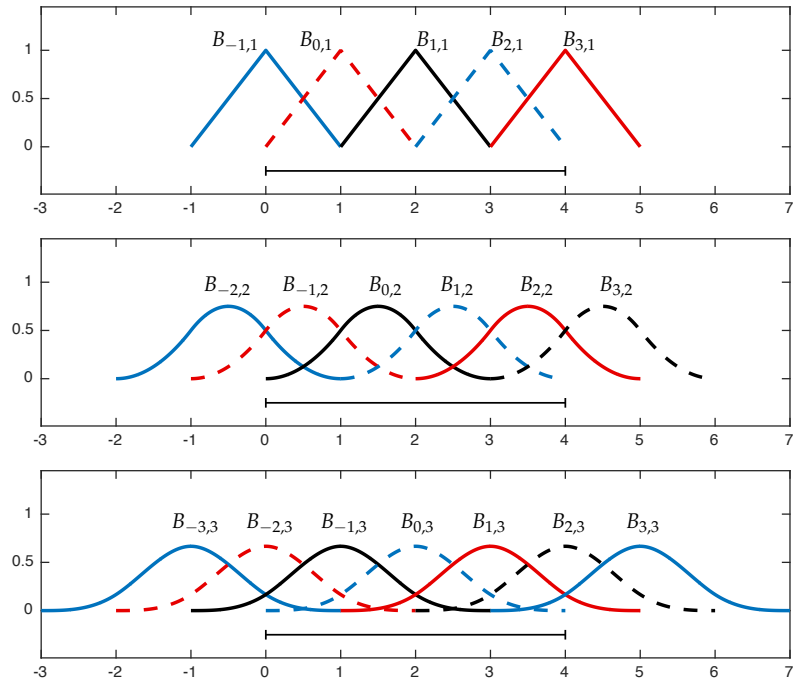



Figure 1.22: B-splines of degree $k = 1$ (top), $k = 2$ (middle), and $k = 3$ (bottom) that are supported on the interval $[x_0, x_n]$ for $x_j = j$ with $n = 4$. Note that $n + k$ B-splines are supported on $[x_0, x_n]$.

satisfy the $n+1$ interpolation conditions

$$f_\ell = S_k(x_\ell) = \sum_{j=-k}^{n-1} c_{j,k} B_{j,k}(x_\ell), \qquad \ell = 0, \ldots, n.$$

Before addressing general $k \geq 1$, we pause to handle the special case of $k = 0$, i.e., constant splines.

### *1.11.3 Constant splines, $k = 0$*

The constant $B$-splines give $B_{n,0}(x_n) = 1$ and so, unlike the general $k \geq 1$ case, the $j = n$ B-spline must be included in the spline sum:

$$S_0(x) = \sum_{j=0}^{n} c_{j,0} B_{j,0}(x).$$

The interpolation conditions give, for $\ell = 0, \ldots, n$,

$$\begin{aligned} f_\ell = S_0(x_\ell) &= \sum_{j=0}^{n} c_{j,0} B_{j,0}(x_\ell) \\ &= c_{\ell,0} B_{\ell,0}(x_\ell) = c_{\ell,0}, \end{aligned}$$

since $B_{j,0}(x_\ell) = 0$ if $j \neq \ell$, and $B_{\ell,0}(x_\ell) = 1$ (recall the plot of $B_{0,0}(x)$ shown earlier). Thus $c_{\ell,0} = f_\ell$, and the degree $k = 0$ spline interpolant is simply

$$S_0(x) = \sum_{j=0}^{n} f_j B_{j,0}(x).$$

# LECTURE 11: *Matrix Determination of Splines; Energy Minimization*

### 1.11.4 *General case, $k \geq 1$*

Now consider the general spline interpolant of degree $k \geq 1$,

$$S_k(x) = \sum_{j=-k}^{n-1} c_{j,k} B_{j,k}(x),$$

with constants $c_{-k,k}, \ldots, c_{n-1,k}$ determined to satisfy the interpolation conditions $S_k(\ell) = f_\ell$, i.e.,

$$\sum_{j=-k}^{n-1} c_{j,k} B_{j,k}(x_\ell) = f_\ell, \quad \ell = 0, \ldots, n.$$

By now we are accustomed to transforming constraints like this into matrix equations. Each value $\ell = 0, \ldots, n$ gives a row of the equation

$$\begin{bmatrix} B_{-k,k}(x_0) & B_{-k+1,k}(x_0) & \cdots & B_{n-1,k}(x_0) \\ B_{-k,k}(x_1) & B_{-k+1,k}(x_1) & \cdots & B_{n-1,k}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_{-k,k}(x_n) & B_{-k+1,k}(x_n) & \cdots & B_{n-1,k}(x_n) \end{bmatrix} \begin{bmatrix} c_{-k,k} \\ c_{-k+1,k} \\ \vdots \\ c_{n-1,k} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}. \tag{1.35}$$

Let us consider the matrix in this equation. The matrix will have $n+1$ rows and $n+k$ columns, so when $k > 1$ the system of equations will be *underdetermined*. Since B-splines have 'small support' (i.e., $B_{j,k}(x) = 0$ for most $x \in [x_0, x_n]$), this matrix will be *sparse*: most entries will be zero.

One could obtain an $(n+1) \times (n+1)$ matrix by arbitrarily setting $k-1$ certain values of $c_{j,k}$ to zero, but this would miss a great opportunity: we can constructively include all $n+k$ B-splines and impose $k$ extra properties on $S_k$ to pick out a unique spline interpolant from the infinitely many options that satisfy the interpolation conditions.

The following subsections will describe the particular form the system (1.35) takes for $k = 1, 2, 3$. In each case we will illustrate the resulting spline interpolant through the following data set.

| $j$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x_j$ | 0 | 1 | 2 | 3 | 4 |
| $f_j$ | 1 | 3 | 2 | $-1$ | 1 |

(1.36)

### 1.11.5 *Linear splines, $k = 1$*

Linear splines are simple to construct: in this case $n + k = n + 1$, so the matrix in (1.35) is square. Let us evaluate it: since

$$B_{j,1}(x_\ell) = \begin{cases} 1, & \ell = j+1; \\ 0, & \ell \neq j+1, \end{cases}$$

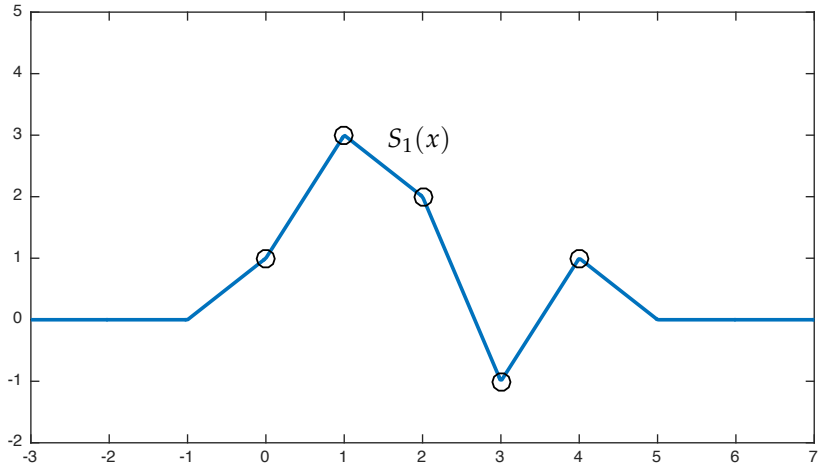



Figure 1.23: Linear spline $S_1$ interpolating 5 data points $\{(x_j, f_j)\}_{j=0}^4$.

the matrix is simply

$$\begin{bmatrix} B_{-1,1}(x_0) & B_{0,1}(x_0) & \cdots & B_{n-1,1}(x_0) \\ B_{-1,1}(x_1) & B_{0,1}(x_1) & \cdots & B_{n-1,1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_{-1,1}(x_n) & B_{0,1}(x_n) & \cdots & B_{n-1,1}(x_n) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \mathbf{I}.$$

The system (1.35) is thus trivial to solve, reducing to

$$\begin{bmatrix} c_{-1,1} \\ c_{-0,k} \\ \vdots \\ c_{n-1,k} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix},$$

which gives the unique linear spline

$$S_1(x) = \sum_{j=-1}^{n-1} f_{j+1} B_{j,1}(x).$$

This above discussion is a pedantic way to arrive at an obvious solution: Since the $j$th 'hat function' B-spline equals one at $x_{j+1}$ and zero at all other knots, just write the unique formula for the interpolant immediately.

Figure 1.23 shows the unique piecewise linear spline interpolant to the data in (1.36), which is a linear combination of the five linear splines shown in Figure 1.22. Explicitly,

$$\begin{aligned} S_1(x) &= f_0 B_{-1,1}(x) + f_1 B_{0,1}(x) + f_2 B_{1,1}(x) + f_3 B_{2,1}(x) + f_4 B_{3,1}(x) \\ &= B_{-1,1}(x) + 3\,B_{0,1}(x) + 2\,B_{1,1}(x) - B_{2,1}(x) + B_{3,1}(x). \end{aligned}$$

Note that linear splines are simply $C^0$ functions that interpolate a given data set—between the knots, they are identical to the piecewise linear functions constructed in Section 1.10.1. Note that $S_1(x)$ is supported on $(x_{-1}, x_{n+1})$ with $S_1(x) = 0$ for all $x \notin (x_{-1}, x_{n+1})$. This is a general feature of splines: Outside the range of interpolation, $S_k(x)$ goes to zero as quickly as possible for a given set of knots while still maintaining the specified continuity.

### *1.11.6 Quadratic splines, $k = 2$*

The construction of quadratic B-splines from the linear splines via the recurrence (1.32) forces the functions $B_{j,2}$ to have a continuous derivative, and also to be supported over three intervals per spline, as seen in the middle plot in Figure 1.22. The interpolant takes the form

$$S_2(x) = \sum_{j=-2}^{n-1} c_{j,2} B_{j,2}(x),$$

with coefficients specified by $n+1$ equations in $n+2$ unknowns:

$$\begin{bmatrix} B_{-2,2}(x_0) & B_{-1,2}(x_0) & \cdots & B_{n-1,2}(x_0) \\ B_{-2,2}(x_1) & B_{-1,2}(x_1) & \cdots & B_{n-1,2}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_{-2,2}(x_n) & B_{-1,2}(x_n) & \cdots & B_{n-1,2}(x_n) \end{bmatrix} \begin{bmatrix} c_{-2,2} \\ c_{-1,2} \\ \vdots \\ c_{n-1,2} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}. \tag{1.37}$$

Since there are more variables than constraints, we expect infinitely many quadratic splines that interpolate the data.

Evaluate the entries of the matrix in (1.37). First note that

$$B_{j,2}(x_\ell) = 0, \quad \ell \notin \{j+1, j+2\},$$

so the matrix is zero in all entries except the main diagonal ($B_{j,2}(x_{j+2})$) and the first superdiagonal ($B_{j,2}(x_{j+1})$). To evaluate these nonzero entries, recall that the recursion (1.32) for B-splines gives

$$B_{j,2}(x) = \Big(\frac{x - x_j}{x_{j+2} - x_j}\Big) B_{j,1}(x) + \Big(\frac{x_{j+3} - x}{x_{j+3} - x_{j+1}}\Big) B_{j+1,1}(x).$$

Evaluate this function at $x_{j+1}$ and $x_{j+2}$, using our knowledge of the

$B_{j,1}$ linear B-splines ('hat functions'):

$$B_{j,2}(x_{j+1}) = \Big(\frac{x_{j+1}-x_j}{x_{j+2}-x_j}\Big)B_{j,1}(x_{j+1}) + \Big(\frac{x_{j+3}-x_{j+1}}{x_{j+3}-x_{j+1}}\Big)B_{j+1,1}(x_{j+1})$$

$$= \Big(\frac{x_{j+1}-x_j}{x_{j+2}-x_j}\Big)\cdot 1 + \Big(\frac{x_{j+3}-x_{j+1}}{x_{j+3}-x_{j+1}}\Big)\cdot 0 = \frac{x_{j+1}-x_j}{x_{j+2}-x_j};$$

$$B_{j,2}(x_{j+2}) = \Big(\frac{x_{j+2}-x_j}{x_{j+2}-x_j}\Big)B_{j,1}(x_{j+2}) + \Big(\frac{x_{j+3}-x_{j+2}}{x_{j+3}-x_{j+1}}\Big)B_{j+1,1}(x_{j+2})$$

$$= \Big(\frac{x_{j+2}-x_j}{x_{j+2}-x_j}\Big)\cdot 0 + \Big(\frac{x_{j+3}-x_{j+2}}{x_{j+3}-x_{j+1}}\Big)\cdot 1 = \frac{x_{j+3}-x_{j+2}}{x_{j+3}-x_{j+1}}.$$

Use these formulas to populate the superdiagonal and subdiagonal of the matrix in (1.37). In the (important) special case of uniformly spaced knots

$$x_j = x_0 + jh, \quad \text{for fixed } h > 0,$$

gives the particularly simple formulas

$$B_{j,2}(x_{j+1}) = B_{j,2}(x_{j+2}) = \frac{1}{2},$$

hence the system (1.37) becomes

$$\begin{bmatrix} 1/2 & 1/2 & & & \\ & 1/2 & 1/2 & & \\ & & \ddots & \ddots & \\ & & & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} c_{-2,2} \\ c_{-1,2} \\ c_{0,2} \\ \vdots \\ c_{n-1,2} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix},$$

where the blank entries are zero. This $(n+1)\times(n+2)$ system will have infinitely many solutions, i.e., infinitely many splines that satisfy the interpolation conditions. How to choose among them? Impose *one* extra condition, such as $S_2'(x_0) = 0$ or $S_2'(x_n) = 0$.

As an example, let us work through the condition $S_2'(x_0) = 0$; it raises an interesting issue. Refer to the middle plot in Figure 1.22. Due to the small support of the quadratic B-splines, $B_{j,2}'(x_0) = 0$ for $j > 0$, so

$$S_2'(x_0) = c_{-2,2}B_{-2,2}'(x_0) + c_{-1,2}B_{-1,2}'(x_0) + c_{0,2}B_{0,2}'(x_0). \tag{1.38}$$

The derivatives of the B-splines at knots are tricky to compute. Differentiating the recurrence (1.32) with $k = 2$, we can formally write

$$B_{j,2}'(x) = \Big(\frac{1}{x_{j+2}-x_j}\Big)B_{j,1}(x) + \Big(\frac{x-x_j}{x_{j+2}-x_j}\Big)B_{j,1}'(x) - \Big(\frac{1}{x_{j+3}-x_{j+1}}\Big)B_{j+1,1}(x) + \Big(\frac{x_{j+3}-x}{x_{j+3}-x_{j+1}}\Big)B_{j+1,1}'(x).$$

Try to evaluate this expression at $x_j$, $x_{j+1}$, or $x_{j+2}$: you must face that fact that the linear B-splines $B_{j,1}$ and $B_{j+1,1}$ *are not differentiable at the knots*! You must instead check that the one-sided derivatives match, e.g.,

$$\lim_{\substack{h\to 0\\ h<0}} \frac{B_{j,2}(x_{j+1}+h) - B_{j,2}(x_{j+1})}{h} = \lim_{\substack{h\to 0\\ h>0}} \frac{B_{j,2}(x_{j+1}+h) - B_{j,2}(x_{j+1})}{h}.$$

A mildly tedious calculation verifies that indeed these one-sided first derivatives do match, and that is the point of splines: each time you increase the degree $k$, you increase the smoothness, so $B_{j,2} \in C^1(\mathbb{R})$.

Now regarding formula (1.38), one can compute

$$B'_{-2,2}(x_0) = -\frac{2}{x_1 - x_{-1}}, \quad B'_{-1,2}(x_0) = \frac{2}{x_1 - x_{-1}}, \quad B'_{0,2}(x_0) = 0,$$

and so, in the special case of a uniformly spaced grid ($x_j = x_0 + jh$), the condition $S'(x_0) = 0$ becomes

$$-\frac{1}{h}c_{-2,2} + \frac{1}{h}c_{-1,2} = 0.$$

Insert this equation as the first row in the linear system for the coefficients,

$$\begin{bmatrix} -1/h & 1/h & & & & \\ 1/2 & 1/2 & & & & \\ & 1/2 & 1/2 & & & \\ & & \ddots & \ddots & & \\ & & & & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} c_{-2,2} \\ c_{-1,2} \\ c_{0,2} \\ \vdots \\ c_{n-1,2} \end{bmatrix} = \begin{bmatrix} 0 \\ f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix},$$

and solve this for $c_{-2,2}, \ldots, c_{n-1,2}$ to determine the unique interpolating quadratic spline with $S_2'(x_0) = 0$.

Figure 1.24 shows quadratic spline interpolants to the data in (1.36). One spline is determined with the extra condition $S_2'(x_0) = 0$ described above; the other satisfies $S_2'(x_n) = 0$. In any case, the quadratic spline $S_2$ is supported on $(x_{-2}, x_{n+2})$.

### *1.11.7 Cubic splines, $k = 3$*

Cubic splines are the most famous of all splines. We began this section by discussing cubic splines as an alternative to piecewise cubic Hermite interpolation. Now we will show how to derive the same cubic splines from the cubic B-splines.

Begin by reviewing the bottom plot in Figure 1.22. The cubic B-splines $B_{-3,3}, \ldots, B_{n-1,3}$ take nonzero values on the interval $[x_0, x_n]$, and hence we write the cubic spline as

$$S_3(x) = \sum_{j=-3}^{n-1} c_{j,3} B_{j,3}(x). \tag{1.39}$$

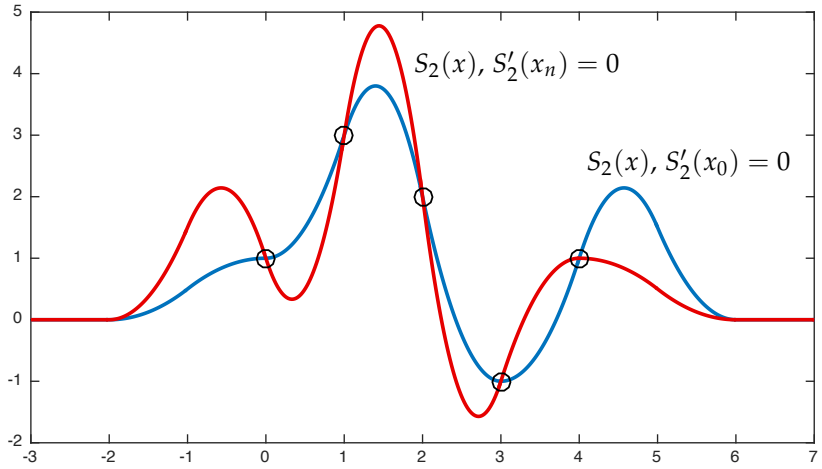



Figure 1.24: Two choices for the quadratic spline $S_2$ that interpolates the 5 data points $\{(x_j, f_j)\}_{j=0}^4$ in (1.36). The blue spline satisfies the extra condition that $S_2'(x_0) = 0$, while the red spline satisfies $S_2'(x_n) = 0$. Check to see that these conditions are consistent with the splines in the plot.

The linear system (1.35) now involves $n+1$ equations in $n+3$ unknowns:

$$\begin{bmatrix} B_{-3,3}(x_0) & B_{-2,3}(x_0) & \cdots & B_{n-1,3}(x_0) \\ B_{-3,3}(x_1) & B_{-2,3}(x_1) & \cdots & B_{n-1,3}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_{-3,3}(x_n) & B_{-2,3}(x_n) & \cdots & B_{n-1,3}(x_n) \end{bmatrix} \begin{bmatrix} c_{-3,3} \\ c_{-2,3} \\ \vdots \\ c_{n-1,3} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}. \tag{1.40}$$

Given the support of cubic splines, note that

$$B_{j,3}(x_\ell) = 0, \quad \ell \notin \{j+1, j+2, j+3\},$$

which implies that only three diagonals of the matrix in (1.40) will be nonzero. We shall only work out the nonzero entries in the case of uniformly spaced knots, $x_j = x_0 + jh$ for fixed $h > 0$. In this case,

$$\begin{aligned} B_{j,3}(x_{j+1}) &= \left(\frac{x_{j+1} - x_j}{x_{j+3} - x_j}\right) B_{j,2}(x_{j+1}) + \left(\frac{x_{j+4} - x_{j+1}}{x_{j+4} - x_{j+1}}\right) B_{j+1,2}(x_{j+1}) = \left(\frac{h}{3h}\right) \cdot \frac{1}{2} + \left(\frac{3h}{3h}\right) \cdot 0 = \frac{1}{6} \\ B_{j,3}(x_{j+2}) &= \left(\frac{x_{j+2} - x_j}{x_{j+3} - x_j}\right) B_{j,2}(x_{j+2}) + \left(\frac{x_{j+4} - x_{j+2}}{x_{j+4} - x_{j+1}}\right) B_{j+1,2}(x_{j+2}) = \left(\frac{2h}{3h}\right) \cdot \frac{1}{2} + \left(\frac{2h}{3h}\right) \cdot \frac{1}{2} = \frac{2}{3} \\ B_{j,3}(x_{j+3}) &= \left(\frac{x_{j+3} - x_j}{x_{j+3} - x_j}\right) B_{j,2}(x_{j+3}) + \left(\frac{x_{j+4} - x_{j+3}}{x_{j+4} - x_{j+1}}\right) B_{j+1,2}(x_{j+3}) = \left(\frac{3h}{3h}\right) \cdot 0 + \left(\frac{h}{3h}\right) \cdot \frac{1}{2} = \frac{1}{6}, \end{aligned}$$

where we have used the fact that $B_{j,2}(x_{j+1}) = B_{j,2}(x_{j+2}) = 1/2$ and

$B_{j,2}(x_j) = B_{j,2}(x_{j+3}) = 0$. Substituting these values into (1.40) gives

$$
(1.41)\qquad
\begin{bmatrix}
1/6 & 2/3 & 1/6 & & & \\
 & 1/6 & 2/3 & 1/6 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & 1/6 & 2/3 & 1/6
\end{bmatrix}
\begin{bmatrix} c_{-3,3} \\ c_{-2,3} \\ \vdots \\ c_{n-1,3} \end{bmatrix}
=
\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix}
$$

involving a matrix with $n+1$ rows and $n+3$ columns. Again, infinitely many cubic splines satisfy these interpolation conditions; two independent requirements must be imposed to determine a unique spline. Recall the three alternatives discussed in Section 1.11.1: complete splines (specify a value for $S_3'$ at $x_0$ and $x_n$), natural splines (force $S_3''(x_0) = S_3''(x_n) = 0$), or not-a-knot splines. One can show that imposing natural spline conditions on $S_3$ requires

$$
(x_2 - x_{-1})c_{-3,3} - (x_2 + x_1 - x_{-1} - x_{-2})c_{-2,3} + (x_1 - x_{-2})c_{-1,3} = 0
$$

$$
(x_{n+2} - x_{n-1})c_{n-3,3} - (x_{n+2} + x_{n+1} - x_{n-1} - x_{n-2})c_{n-2,3} + (x_{n+1} - x_{n-2})c_{n-1,3} = 0,
$$

which for equally spaced knots ($x_j = x_0 + jh$) simplify to

$$
3hc_{-3,3} - 6hc_{-2,3} + 3hc_{-1,3} = 0
$$

$$
3hc_{n-3,3} - 6hc_{n-2,3} + 3hc_{n-1,3} = 0.
$$

It is convenient to add these conditions (dividing out the $h$) as the first and last row of (1.40) to give

$$
(1.42)\qquad
\begin{bmatrix}
3 & -6 & 3 & & & \\
1/6 & 2/3 & 1/6 & & & \\
 & 1/6 & 2/3 & 1/6 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & 1/6 & 2/3 & 1/6 \\
 & & & 3 & -6 & 3
\end{bmatrix}
\begin{bmatrix} c_{-3,3} \\ c_{-2,3} \\ c_{-1,3} \\ \vdots \\ c_{n-2,3} \\ c_{n-1,3} \end{bmatrix}
=
\begin{bmatrix} 0 \\ f_0 \\ f_1 \\ \vdots \\ f_n \\ 0 \end{bmatrix}.
$$

This system of $n+3$ equations in $n+3$ variables has a unique solution, the natural cubic spline interpolant.

It is a useful exercise to work out the extra rows you would add to (1.40) to impose *complete* or *not-a-knot* boundary conditions.

Figure 1.25 shows the natural cubic spline interpolant to the data (1.36). Clearly this spline satisfies the interpolation conditions, but now there seems to be an artificial peak near $x = 5$ that you might not have anticipated from the data values. This is a feature

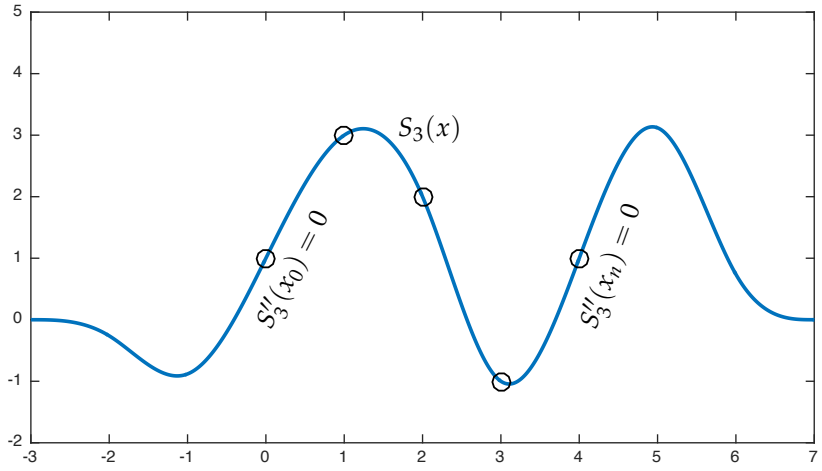



Figure 1.25: Cubic spline $S_3$ interpolant to 5 data points $\{(x_j, f_j)\}_{j=0}^4$, imposing the two extra *natural spline* conditions $S_3''(x_0) = S_3''(x_n) = 0$ to give a unique spline.

of the natural boundary conditions: by forcing $S_3''$ to be zero at $x_0$ and $x_n$, we ensure that the spline $S_3$ has constant slope at $x_0$ and $x_n$. Eventually this slope must be reversed, as our B-splines force $S_3(x)$ to be zero outside $(x_{-3}, x_{n+3})$, the support of the B-splines that contribute to the sum (1.39).

Of course, one can implement splines of higher degree, $k > 3$, if if greater continuity is required at the knots, or if there are more than two boundary conditions to impose (e.g., if one wants both first and second derivatives to be zero at the boundary). The procedure in that case follows the pattern detailed above: work out the entries in the matrix (1.35) and add in rows to encode the additional $k - 1$ constraints needed to specify a unique degree-$k$ spline interpolant.

### *1.11.8 Optimality properties of splines*

Splines often enjoy a beautiful property: among all sufficiently smooth interpolants , certain splines minimize 'energy', quantified for a function $g \in C^2[x_0, x_n]$ as

$$\int_{x_0}^{x_n} g''(x)^2 \, \mathrm{d}x.$$

To give a flavor for such results, we present one example.

For a similar result involving complete cubic splines, see Theorem 2.3.1 of Gautschi's *Numerical Analysis* (2nd ed., Birkhäuser, 2012). The proof here is an easy adaptation of Gautchsi's.

---

**Theorem 1.10** (Natural cubic splines minimize energy)**.** Suppose $S_3$ is the natural cubic spline interpolant to $\{(x_j, f_j)\}_{j=0}^n$, and $g$ is any $C^2[x_0, x_n]$ function that also interpolates the same data. Then

$$\int_{x_0}^{x_n} S_3''(x)^2 \, \mathrm{d}x \leq \int_{x_0}^{x_n} g''(x)^2 \, \mathrm{d}x.$$

---

*Proof.* The proof will actually quantify how much larger $g''$ is than $S_3''$ by showing that

$$\int_{x_0}^{x_n} g''(x)^2\,\mathrm{d}x = \int_{x_0}^{x_n} S_3''(x)^2\,\mathrm{d}x + \int_{x_0}^{x_n} \big(g''(x) - S_3''(x)\big)^2\,\mathrm{d}x. \tag{1.43}$$

Expanding the right-hand side, see that this claim is equivalent to

$$\int_{x_0}^{x_n} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = 0. \tag{1.44}$$

To prove this claim, break the integral on the left into segments $[x_j, x_{j+1}]$ between the knots. Write

$$\int_{x_0}^{x_n} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = \sum_{j=1}^{n} \int_{x_{j+1}}^{x_j} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x.$$

This decomposition of $[x_0, x_n]$ will allow us to exploit the fact that $S_3$ is a standard cubic polynomial, and hence infinitely differentiable, on these subintervals.

On each subinterval, integrate by parts to obtain

$$\int_{x_{j+1}}^{x_j} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = \Big[\big(g'(x) - S_3'(x)\big)S_3''(x)\Big]_{x=x_{j-1}}^{x_j} - \int_{x_{j+1}}^{x_j} \big(g'(x) - S_3'(x)\big)S_3'''(x)\,\mathrm{d}x. \tag{1.45}$$

Focus now on the integral on the right-hand side; we can show it is *zero* by integrating it by parts to get

$$\int_{x_{j+1}}^{x_j} \big(g'(x) - S_3'(x)\big)S_3'''(x)\,\mathrm{d}x = \Big[\big(g(x) - S_3(x)\big)S_3'''(x)\Big]_{x=x_{j-1}}^{x_j} - \int_{x_{j+1}}^{x_j} \big(g(x) - S_3(x)\big)S_3''''(x)\,\mathrm{d}x. \tag{1.46}$$

The boundary term on the right is zero, since $g(x_\ell) - S_3(x_\ell) = 0$ for $\ell = 0, \ldots, n$ (both $g$ and $S_3$ must interpolate the data). The integral on the right is also zero: since $S_3$ is a cubic polynomial on $[x_{j-1}, x_j]$, $S_3''''(x) = 0$. Thus (1.45) reduces to

$$\int_{x_{j+1}}^{x_j} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = \Big[\big(g'(x) - S_3'(x)\big)S_3''(x)\Big]_{x=x_{j-1}}^{x_j}$$

Adding up these contributions over all the subintervals,

$$\int_{x_0}^{x_n} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = \sum_{j=1}^{n} \Big[\big(g'(x) - S_3'(x)\big)S_3''(x)\Big]_{x=x_{j-1}}^{x_j}.$$

Most of the boundary terms on the right cancel one another out, leaving only

$$\int_{x_0}^{x_n} \big(g''(x) - S_3''(x)\big)S_3''(x)\,\mathrm{d}x = \Big(\big(g'(x_n) - S_3'(x_n)\big)S_3''(x_n)\Big) - \Big(\big(g'(x_0) - S_3'(x_0)\big)S_3''(x_0)\Big).$$

Each of the terms on the right is zero by virtue of the *natural* cubic spline condition $S_3''(x_0) = S_3''(x_n) = 0$. This confirms the formula (1.44), and hence the equivalent (1.43) that quantifies how much larger $g''$ can be than $S_3''$. ∎

### 1.11.9 *Some omissions*

The great utility of B-splines in engineering has led to the development of the subject far beyond these basic notes. Among the omissions are: interpolation imposed at points distinct from the knots, convergence of splines to the function they are approximating as the number of knots increases, integration and differentiation of splines, tension splines, etc. Splines in higher dimensions ('thin-plate splines') are used, for example, to design the panels of a car body.

## 1.12 *Handling Polynomials in MATLAB*

To close this discussion of interpolating polynomials, we mention a few notes about polynomials in MATLAB.

### 1.12.1 *MATLAB's Polynomial Format*

By convention, MATLAB represents polynomials by their coefficients, *listed by decreasing powers of* $x$. Thus $c_0 + c_1x + c_2x^2 + c_3x^3$ is represented by the vector

$$[c_3 \;\; c_2 \;\; c_1 \;\; c_0],$$

while $7 + 3x + 5x^3 - 2x^4$ would be represented by

$$[-2 \;\; 5 \;\; 0 \;\; 3 \;\; 7]$$

In this last example note the `0` corresponding to the $x^2$ term: all lower powers of $x$ must be accounted for in coefficient vector.

Given a polynomial in a vector, say `p = [ -2 5 0 3 7]`, one can evaluate $p(x)$ using the command `polyval`, e.g.

```
>> polyval(p,x)
```

This command works if `x` is a scalar or a vector. Thus, for example, to plot $p(x)$ for $x \in [0, 1]$, one could use

```
>> x = linspace(0,1,500);    % 500 uniform points between 0 and 1
>> plot(x,polyval(p,x))      % plot p(x) with x from 0 to 1
```

One can also compute the roots of polynomials very easily with the command

```
>> roots(p)                  % compute roots of p(x)=0
```

though one should be cautious of numerical errors when the degree of the polynomial is large. One can construct a polynomial directly from its roots, using the `poly` command. For example,

```
>> poly([1:4])
ans =
     1   -10    35   -50    24
```

Type `type roots` to see MATLAB's code for the `roots` command. Scan to the bottom to see the crucial lines. From the coefficients MATLABconstructs a companion matrix, then computes its eigenvalues using the `eig` command. For some (larger degree) polynomials, these eigenvalues are very sensitive to perturbations, and the roots can be very inaccurate. For a famous example due to Wilkinson, try `roots(poly[1:24]))`, should return the roots $1, \ldots, 24$.

`poly` returns the coefficients of the monic polynomial with roots $1, 2, 3, 4$:

$$24 - 50x + 35x^2 - 10x^3 + x^4 = (x-1)(x-2)(x-3)(x-4).$$

This gives a slick way to construct the Lagrange basis function

$$\ell_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}$$

given the vector $\texttt{xx} = [x_0 \ \cdots \ x_n]$ of interpolation points:

```
>> ell_j = poly(xx([1:j j+2:end]));         % specify roots of ell_j
>> ell_j = ell_j/polyval(ell_j,xx(j+1));    % scale so ell_j(xx(j+1)) = 1
```

Note that the indices of `xx` account for the fact that $x_j = \texttt{xx(j+1)}$.

### *1.12.2 Constructing Polynomial Interpolants*

MATLABhas a built-in code for constructing polynomial interpolants. In fact, it is a special case of the polynomial approximation code `polyfit`. When you request that `polyfit` produce a degree-$n$ polynomial through $n+1$ pairs of data, you obtain an interpolant. For example, the following code will interpolate $f(x) = sqrt(x)$ at $x_j = j/4$ for $j = 0, \ldots, 4$:

Beware! The numerical implementation of `polyfit` is not ideal for polynomial interpolation: the code uses the Vandermonde basis. Thus, restrict your use of `polyfit` to low degree polynomials. The command `type polyfit` will show you MATLAB's code.

```
>> f  = @(x) sqrt(x);          % define f
>> xx = [0:4]/4;               % define interpolation points
>> p = polyfit(xx,f(xx),4);    % quartic polynomial interpolant
>> polyval(p,xx)               % evaluate p at interpolation points
ans =
    0.0000    0.5000    0.7071    0.8660    1.0000
>> f(xx)                       % compare to f at interpolation points
ans =
         0    0.5000    0.7071    0.8660    1.0000
```

### *1.12.3 Piecewise Polynomial Interpolants and Splines*

MATLAB also includes a general-purpose `interp1` command that constructs various piecewise polynomial interpolants. For example, the `'linear'` option constructs piecewise linear interpolants.

```
>> f  = @(x) sin(3*pi*x);               % define f
>> xx = [0:10]/10;                      % define "knots"
>> x  = linspace(0,1,500);              % evaluation points
>> plot(x,interp1(xx,f(xx),x,'linear'))  % plot piecewise linear interpolant
```

Alternatively, the `'spline'` option constructs the not-a-knot cubic spline approximation.

```
>> plot(x,interp1(xx,f(xx),x,'spline'))  % plot cubic spline interpolant
```

The `spline` command (which `interp1` uses to construct the spline) will return MATLAB's data structure that stores the cubic spline interpolant. `>> S = spline(xx,f(xx))`

```
S =
      form: 'pp'
    breaks: [1x11 double]
     coefs: [10x4 double]
    pieces: 10
     order: 4
       dim: 1
```

For example, `S.breaks` contains the list of knots. One can also pass arguments to `spline` to specify *complete* boundary conditions. However, there is no easy way to impose natural boundary conditions. For more sophisticated data fitting operations, MATLAB offers a Curve Fitting Toolbox (which fits both curves and surfaces).

Another option to `interp1` has a misleading name: `'pchip'` constructs a particular spline-like interpolant designed to be quite smooth: it cannot match any derivative information about $f$, as no derivative information is even passed to the function.

### *1.12.4 Chebfun*

Chebfun is a free package of MATLAB routines developed by Nick Trefethen and colleagues at Oxford University. Using sophisticated techniques from polynomial approximation theory, Chebfun automatically represents an arbitrary (piecewise smooth) function $f(x)$ to machine precision, and allows all manner of operations on this function, overloading every conceivable MATLAB matrix/vector operation. There is no way to do this beautiful and powerful system justice in a few lines of text here. Go to `chebfun.org`, download the software, and start exploring. Suffice to say, Chebfun significantly enrich one's study and practice of numerical analysis.

In fact, it was used to generate a number of the plots in these notes.

# 2
# *Approximation Theory*

## LECTURE 12: *Introduction to Approximation Theory*

INTERPOLATION IS AN INVALUABLE TOOL in numerical analysis: it provides an easy way to replace a complicated function by a polynomial (or piecewise polynomial), and, *at least as importantly*, it provides a mechanism for developing numerical algorithms for more sophisticated problems. Interpolation is not the only way to approximate a function, though: and indeed, we have seen that the quality of the approximation can depend perilously on the choice of interpolation points.

We saw one example in Section 1.7: finite difference formulas for approximating derivatives and solving differential equation boundary value problems. Several other applications will follow later in the semester.

If approximation is our goal, interpolation is only one means to that end. In this chapter we investigate alternative approaches that directly optimize the quality of the approximation. How do we measure this quality? That depends on the application. Perhaps the most natural means is to *minimize the maximum error* of the approximation.

> Given $f \in C[a,b]$, find $p_* \in \mathcal{P}_n$ such that
>
> $$\max_{x\in[a,b]} |f(x) - p_*(x)| = \min_{p\in\mathcal{P}_n} \max_{x\in[a,b]} |f(x) - p(x)|.$$

This is called the *minimax approximation problem*.

Norms clarify the notation. For any $g \in C[a,b]$, define

$$\|g\|_\infty := \max_{x\in[a,b]} |g(x)|,$$

the 'infinity norm of $g$'. One can show that $\|\cdot\|_\infty$ satisfies the basic norm axioms on the vector space $C[a,b]$ of continuous functions. Thus the minimax approximation problem seeks $p_* \in \mathcal{P}_n$ such that

$$\|f - p_*\|_\infty = \min_{p\in\mathcal{P}_n} \|f - p\|_\infty.$$

$\|g\|_\infty \geq 0$ for all $g \in C[a,b]$
$\|g\|_\infty = 0 \iff g(x) = 0$ for all $x \in [a,b]$.
$\|\alpha g\|_\infty = |\alpha|\|g\|_\infty$ for all $\alpha \in \mathbb{C}$, $g \in C[a,b]$.
$\|g+h\|_\infty \leq \|g\|_\infty + \|h\|_\infty$, for all $g, h \in C[a,b]$.

Notice that, for better or worse, this approximation will be heavily influenced by extreme values of $f(x)$, even if they occur over only a small range of $x \in [a,b]$.

Some applications call instead for an approximation that balances the size of the errors against the range of $x$ values over which they are attained. In such cases it is most common to minimize the integral of the square of the error, the *least squares approximation problem*.

> Given $f \in C[a,b]$, find $p_* \in \mathcal{P}_n$ such that
>
> $$\Big( \int_a^b (f(x) - p_*(x))^2 \, \mathrm{d}x \Big)^{1/2} = \min_{p \in \mathcal{P}_n} \Big( \int_a^b (f(x) - p(x))^2 \, \mathrm{d}x \Big)^{1/2}.$$

This problem is often associated with *energy minimization* in mechanics, giving one motivation for its widespread appeal. As before, we express this more compactly by introducing the *two-norm* of $g \in [a,b]$:

$$\|g\|_2 = \Big( \int_a^b |g(x)|^2 \, \mathrm{d}x \Big)^{1/2},$$

so the least squares problem becomes

$$\|f - p_*\|_2 = \min_{p \in \mathcal{P}_n} \|f - p\|_2.$$

This chapter focuses on these two problems. Before attacking them we mention one other possibility, minimizing the absolute value of the integral of the error: the *least absolute deviations* problem.

> Given $f \in C[a,b]$, find $p_* \in \mathcal{P}_n$ such that
>
> $$\int_a^b |f(x) - p_*(x)| \, \mathrm{d}x = \min_{p \in \mathcal{P}_n} \int_a^b |f(x) - p(x)| \, \mathrm{d}x.$$

This problem has become quite important in recent years. In particular, the analogous problem resulting when $f$ is replaced by its vector discretization $\mathbf{f} \in \mathbb{C}^n$ plays a pivotal role in *compressive sensing*.

With this problem we associate the *one-norm* of $g \in C[a,b]$,

$$\|g\|_1 = \int_a^b |g(x)| \, \mathrm{d}x,$$

giving the least absolute deviations problem as

$$\|f - p_*\|_1 = \min_{p \in \mathcal{P}_n} \|f - p\|_1.$$

## 2.1 *Minimax Approximation: General Theory*

The goal of minimizing the maximum error of a polynomial $p$ from the function $f \in C[a,b]$ is called *minimax* (or *uniform*, or $L^\infty$) approximation: Find $p_* \in \mathcal{P}_n$ such that

$$\|f - p_*\|_\infty = \min_{p \in \mathcal{P}_n} \|f - p\|_\infty.$$

Let us begin by connecting this problem to polynomial interpolation. On Problem Set 2 you were asked to prove that

$$\|f - \mathbf{\Pi}_n f\|_\infty \le \big(1 + \|\mathbf{\Pi}_n\|_\infty\big)\|f - p_*\|_\infty, \tag{2.1}$$

where $\mathbf{\Pi}_n$ is the linear interpolation operator for

That is, $p = \mathbf{\Pi}_n f \in \mathcal{P}_n$ is the polynomial that interpolates $f$ at $x_0, \ldots, x_n$.

$$x_0 < x_1 < \cdots < x_n$$

with $x_0, \ldots, x_n \in [a,b]$. Here $\|\mathbf{\Pi}_n\|_\infty$ is the *operator norm* of $\mathbf{\Pi}_n$:

$$\|\mathbf{\Pi}_n\|_\infty = \max_{f \in C[a,b]} \frac{\|\mathbf{\Pi}_n f\|_\infty}{\|f\|_\infty}$$

You further show that

$$\|\mathbf{\Pi}_n\| = \max_{x \in [a,b]} \sum_{j=0}^{n} |\ell_j(x)|,$$

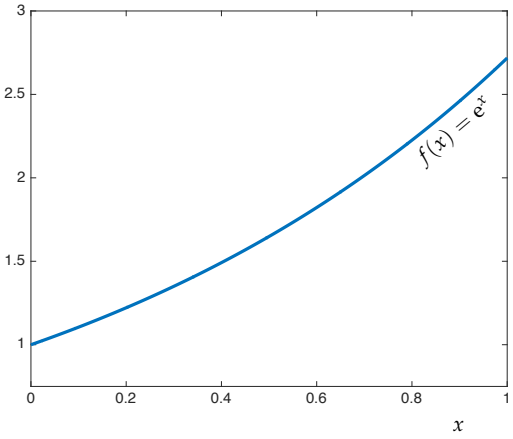
where $\ell_j$ denotes the $j$th Lagrange interpolation basis function

$$\ell_j(x) = \prod_{\substack{k=0 \\ k \ne j}}^{n} \frac{x - x_k}{x_j - x_k}.$$

Now appreciate the utility of bound (2.1): the linear interpolant $\mathbf{\Pi}_n f$ (*which is easy to compute*) is within a factor of $1 + \|\mathbf{\Pi}_n\|_\infty$ of the optimal approximation $p_*$. Note that $\|\mathbf{\Pi}_n\|_\infty \ge 1$: how much larger than one depends on the distribution of the interpolation points.

In the following sections we shall characterize and compute $p_*$ (indeed more difficult than computing the interpolant), then use the theory of minimax approximation to find an excellent set of *almost fail-safe* interpolation points.

We begin by working out a simple example by hand.



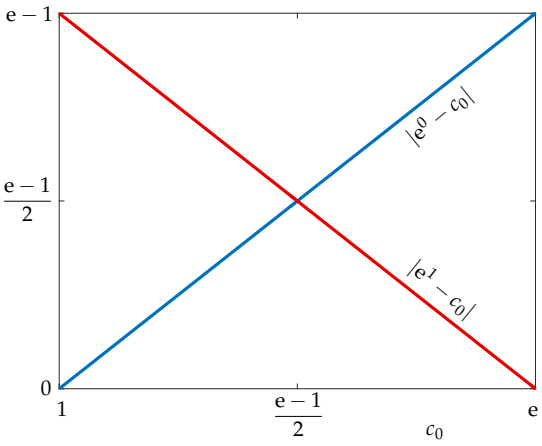

**Example 2.1.** Suppose we seek the constant that best approximates $f(x) = e^x$ over the interval $[0,1]$, shown in the margin. Before going on, sketch out a constant function (degree-0 polynomial) that approximates $f$ in a manner that *minimizes the maximum error*.

Since $f(x)$ increases monotonically for $x \in [0,1]$, the optimal constant approximation $p_* = c_0$ must fall between $f(0) = 1$ and

$f(1) = \mathrm{e}$, i.e., $1 \le c_0 \le \mathrm{e}$. Moreover, since $f$ is monotonic and $p_*$ is a constant, the function $f - p_*$ is also monotonic, so the maximum error $\max_{x\in[a,b]} |f(x) - p_*(x)|$ must be attained at one of the end points, $x = 0$ or $x = 1$. Thus,

$$\|f - p_*\|_\infty = \max\{|\mathrm{e}^0 - c_0|, |\mathrm{e}^1 - c_0|\}.$$

The picture to the right shows $|\mathrm{e}^0 - c_0|$ (blue) and $|\mathrm{e}^1 - c_0|$ (red) for $c_0 \in [1, \mathrm{e}]$. The optimal value for $c_0$ will be the point at which *the larger of these two lines is minimal*. The figure clearly reveals that this happens when the errors are equal, at $c_0 = (1 + \mathrm{e})/2$. We conclude that the optimal minimax constant polynomial approximation to $\mathrm{e}^x$ on $x \in [0, 1]$ is $p_*(x) = c_0 = (1 + \mathrm{e})/2$.




The plots in Figure 2.1 compare $f$ to the optimal polynomial $p_*$ (top), and show the error $f - p_*$ (bottom). We picked $c_0$ so that the error $f - p_*$ was equal in magnitude at the end points $x = 0$ and $x = 1$; in fact, it is equal in magnitude, but opposite in sign,

$$\mathrm{e}^0 - c_0 = -(\mathrm{e}^1 - c_0).$$

This property—maximal error being attained with alternating sign—is a key feature of minimax approximation.

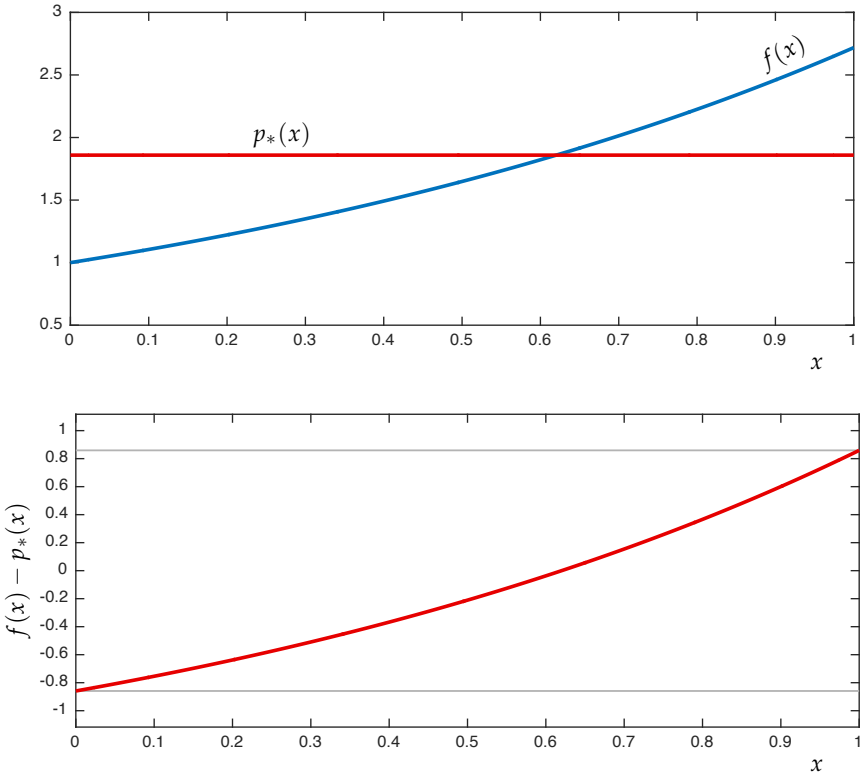



Figure 2.1: Minimax approximation of degree $k = 0$ to $f(x) = \mathrm{e}^x$ on $x \in [0, 1]$. The top plot compares $f$ to $p_*$; the bottom plot shows the error $f - p_*$, whose extreme magnitude is *attained, with opposite sign, at two values of* $x \in [0, 1]$.

# LECTURE 13: *Equioscillation, Part 1*

## 2.2 *Oscillation Theorem*

The previous example hints that the points at which the error $f - p_*$ attains its maximum magnitude play a central role in the theory of minimax approximation. The Theorem of de la Vallée Poussin is a first step toward such a result. We include its proof to give a flavor of how such results are established.

The proof is adapted from Section 8.3 of Süli and Mayers, *An Introduction to Numerical Analysis* (Cambridge, 2003).

---

**Theorem 2.1** (de la Vallée Poussin's Theorem)**.**
Let $f \in C[a,b]$ and suppose $r \in \mathcal{P}_n$ is some polynomial for which there exist $n+2$ points $\{x_j\}_{j=0}^{n+1}$ with $a \le x_0 < x_1 < \cdots < x_{n+1} \le b$ at which the error $f(x) - r(x)$ oscillates signs, i.e.,

$$\operatorname{sgn}(f(x_j) - r(x_j)) = -\operatorname{sgn}(f(x_{j+1}) - r(x_{j+1}))$$

$$\operatorname{sgn}(x) = \begin{cases} 1, & x > 0; \\ 0, & x = 0; \\ -1, & x < 0. \end{cases}$$

for $j = 0, \ldots, n$. Then

$$\min_{p \in \mathcal{P}_n} \|f - p\|_\infty \ge \min_{0 \le j \le n+1} |f(x_j) - r(x_j)|. \tag{2.2}$$

---

Before proving this result, look at Figure 2.2 for an illustration of the theorem. Suppose we wish to approximate $f(x) = \mathrm{e}^x$ with some quintic polynomial, $r \in \mathcal{P}_5$ (i.e., $n = 5$). *This polynomial is not necessarily the minimax approximation to $f$ over the interval* $[0,1]$. However, in the figure it is clear that for this $r$, we can find $n + 2 = 7$ points at which the sign of the error $f(x) - r(x)$ oscillates. The red curve shows the error for the optimal minimax polynomial $p_*$ (whose computation is discussed below). This is the point of de la Vallée Poussin's theorem: *Since the error $f(x) - r(x)$ oscillates sign $n+2$ times, the minimax error $\pm\|f - p_*\|_\infty$ exceeds $|f(x_j) - r(x_j)|$ at one of the points $x_j$ that give the oscillating sign.* In other words, de la Vallée Poussin's theorem gives a nice mechanism for developing *lower bounds* on $\|f - p_*\|_\infty$.

These $n+2$ points are by no means unique: we have a continuum of choices available. However, taking the extrema of $f - r$ will give the the best bounds in the theorem.

*Proof.* Suppose we have $n+2$ ordered points, $\{x_j\}_{j=0}^{n+1} \subset [a,b]$, such that $f(x_j) - r(x_j)$ alternates sign at consecutive points, and let $p_*$ denote the minimax polynomial,

$$\|f - p_*\|_\infty = \min_{p \in \mathcal{P}_n} \|f - p\|_\infty.$$

We will prove the result by contradiction. Thus suppose

$$\|f - p_*\|_\infty < |f(x_j) - r(x_j)|, \qquad \text{for all } j = 0, \ldots, n+1. \tag{2.3}$$

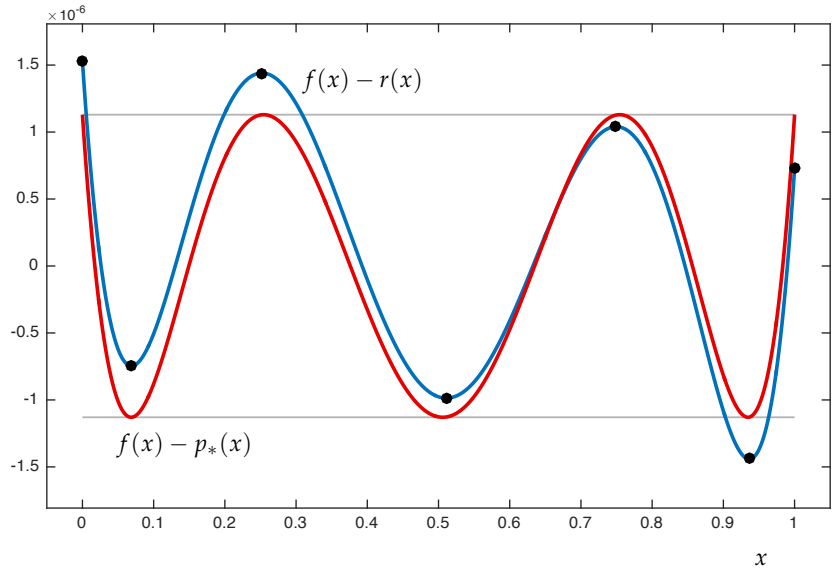



Figure 2.2: Illustration of de la Vallée Poussin's theorem for $f(x) = e^x$ and $n = 5$. Some polynomial $r \in \mathcal{P}_5$ gives an error $f - r$ for which we can identify $n + 2 = 7$ points $x_j$, $j = 0, \ldots, n + 1$ (black dots) at which $f(x_j) - r(x_j)$ oscillates sign. The minimum value of $|f(x_j) - r(x_j)|$ gives a *lower bound* the maximum error $\|f - p_*\|_\infty$ of the optimal approximation $p_* \in \mathcal{P}_5$.

As the left hand side is the maximum difference of $f - p_*$ over all $x \in [a, b]$, that difference can be no larger at $x_j \in [a, b]$, and so:

$$|f(x_j) - p_*(x_j)| < |f(x_j) - r(x_j)|, \qquad \text{for all } j = 0, \ldots, n + 1. \tag{2.4}$$

Now consider

$$p_*(x) - r(x) = (f(x) - r(x)) - (f(x) - p_*(x)),$$

which is a degree $n$ polynomial, since $p_*, r \in \mathcal{P}_n$. Equation (2.4) states that $f(x_j) - r(x_j)$ always has larger magnitude than $f(x_j) - p_*(x_j)$. Thus, regardless of the sign of $f(x_j) - p_*(x_j)$, the magnitude $|f(x_j) - p_*(x_j)|$ will never be large enough to overcome $|f(x_j) - r(x_j)|$, and hence

$$\operatorname{sgn}(p_*(x_j) - r(x_j)) = \operatorname{sgn}(f(x_j) - r(x_j)).$$

We know from the hypothesis that $f(x) - r(x)$ must change sign at least $n + 1$ times (at least once in each interval $(x_j, x_{j+1})$ for $j = 0, \ldots, n$), and thus the degree-$n$ polynomial $p_* - r$ must do the same. But $n + 1$ sign changes implies $n + 1$ roots; the only degree-$n$ polynomial with $n + 1$ roots is the zero polynomial, i.e., $p_* = r$. However, this contradicts the strict inequality in equation (2.3). Hence, there must be at least one $j$ for which

$$\|f - p_*\|_\infty \geq |f(x_j) - r(x_j)|,$$

thus yielding the theorem. ∎

Now suppose we can find some degree-$n$ polynomial, call it $\widetilde{r} \in \mathcal{P}_n$, and $n + 2$ points $x_0 < \cdots < x_{n+1}$ in $[a, b]$ such that not only does

the sign of $f - \widetilde{r}$ oscillate, but the error takes its extremal values at these points. That is,

$$|f(x_j) - \widetilde{r}(x_j)| = \|f - \widetilde{r}\|_\infty, \quad j = 0, \ldots, n+1,$$

and

$$f(x_j) - \widetilde{r}(x_j) = -\big(f(x_{j+1}) - \widetilde{r}(x_{j+1})\big), \quad j = 0, \ldots, n.$$

The red curve in Figure 2.2 shows an error function that satisfies these requirements.

Now apply de la Valée Poussin's theorem to this special polynomial $\widetilde{r}$. Equation (2.2) gives

$$\min_{p \in \mathcal{P}_n} \|f - p\| \geq \min_{0 \leq j \leq n+1} |f(x_j) - \widetilde{r}(x_j)|.$$

On the other hand, we have presumed that

$$|f(x_j) - \widetilde{r}(x_j)| = \|f - \widetilde{r}\|_\infty$$

for all $j = 0, \ldots, n+1$. Thus, by de la Vallée Poussin's theorem,

$$\min_{p \in \mathcal{P}_n} \|f - p\| \geq \min_{0 \leq j \leq n+1} |f(x_j) - \widetilde{r}(x_j)| = \|f - \widetilde{r}\|_\infty.$$

Since $\widetilde{r} \in \mathcal{P}_n$, it follows that

$$\min_{p \in \mathcal{P}_n} \|f - p\| = \|f - \widetilde{r}\|_\infty,$$

and this *equioscillating* $\widetilde{r}$ must be an optimal approximation to $f$.

The question remains: Does such a polynomial with equioscillating error always exist? The following theorem ensures it does.

For a direct proof, see Section 8.3 of Süli and Mayers. Another excellent resource is G. W. Stewart, *Afternotes Goes to Graduate School*, SIAM, 1998; see Stewart's Lecture 3.

**Theorem 2.2** (Oscillation Theorem)**.** Suppose $f \in C[a, b]$. Then $p_* \in \mathcal{P}_n$ is a minimax approximation to $f$ from $\mathcal{P}_n$ on $[a, b]$ *if and only if* there exist $n+2$ points $x_0 < x_1 < \cdots < x_{n+1}$ such that

$$|f(x_j) - p_*(x_j)| = \|f - p_*\|_\infty, \quad j = 0, \ldots, n+1$$

and the sign of the error oscillates at these points:

$$f(x_j) - p_*(x_j) = -\big(f(x_{j+1}) - p_*(x_{j+1})\big), \quad j = 0, \ldots, n.$$

Note that this result is *if and only if*: the oscillation property exactly *characterizes* the minimax approximation. We have proved one direction already by appeal to de la Vallée Poussin's theorem. The proof of the other direction is rather more involved.

## LECTURE 14: *Equioscillation, Part 2*

A direct proof that an optimal minimax approximation $p_* \in \mathcal{P}_n$ must give an equioscillating error is rather tedious, requiring one the chase down the oscillation points one at a time. The following approach is a bit more appealing. We begin with a technical result from which the main theorem will readily follow.

---

**Lemma 2.1.** Let $p_* \in \mathcal{P}_n$ be a minimax approximation of $f \in C[a,b]$,

$$\|f - p_*\|_\infty = \min_{p \in \mathcal{P}_n} \|f - p\|_\infty,$$

and let $\mathcal{X}$ denote the set of all points $x \in [a,b]$ for which

$$|f(x) - p_*(x)| = \|f - p_*\|_\infty.$$

Then for all $q \in \mathcal{P}_n$,

$$\max_{x \in \mathcal{X}} \big(f(x) - p_*(x)\big) q(x) \geq 0. \tag{2.5}$$

This 'lemma' is a diluted version of *Kolmolgorov's Theorem*, which is (a) an 'if and only if' version of this lemma that (b) appeals to approximation with much more general classes of functions, not just polynomials, and (c) handles complex-valued functions. The proof here is adapted from that more general setting given in Theorem 2.1 of DeVore and Lorentz, *Constructive Approximation* (Springer, 1993).

---

*Proof.* We will prove the lemma by contradiction. Suppose $p_* \in \mathcal{P}_n$ is a minimax approximation, but that (2.5) fails to hold, i.e., there exists some $\widetilde{q} \in \mathcal{P}_n$ and $\varepsilon > 0$ such that

$$\max_{x \in \mathcal{X}} \big(f(x) - p_*(x)\big) \widetilde{q}(x) < -2\varepsilon.$$

We first note that $\|\widetilde{q}\|_\infty > 0$. Since $\big(f(x) - p_*(x)\big)q(x)$ is a continuous function on $[a,b]$, it must remain negative on some sufficiently small neighborhood of $\mathcal{X}$. More concretely, we can find $\delta > 0$ such that

$$\max_{x \in \widetilde{\mathcal{X}}} \big(f(x) - p_*(x)\big) \widetilde{q}(x) < -\varepsilon, \tag{2.6}$$

where

$$\widetilde{\mathcal{X}} := \{\xi \in [a,b] : \min_{x \in \mathcal{X}} |\xi - x| < \delta\}.$$

To arrive at a contradiction, we will design a function $\widetilde{p}$ that better approximates $f$ than $p_*$, i.e., $\|f - \widetilde{p}\|_\infty < \|f - p_*\|_\infty$. This function will take the form

$$\widetilde{p}(x) = p_*(x) - \lambda \widetilde{q}(x)$$

for (small) constant $\lambda$ we shall soon determine. Let $E := \|f - p_*\|_\infty$ and pick $M$ such that $|\widetilde{q}(x)| \leq M$ for all $x \in \widetilde{\mathcal{X}}$. Then for all $x \in \widetilde{\mathcal{X}}$,

$$\begin{aligned} |f(x) - \widetilde{p}(x)|^2 &= \big(f(x) - p_*(x)\big)^2 + 2\lambda \big(f(x) - p_*(x)\big)\widetilde{q}(x) + \lambda^2 \widetilde{q}(x)^2 \\ &= E^2 + 2\lambda \big(f(x) - p_*(x)\big)\widetilde{q}(x) + \lambda^2 \widetilde{q}(x)^2 \\ &< E^2 - 2\lambda\varepsilon + \lambda^2 M^2, \end{aligned} \tag{2.7}$$

where this inequality follows from (2.6). To show that $\widetilde{p}$ is a better approximation to $f$ than $p_*$, it will suffice to show that the right-hand side of (2.7) is smaller than $E^2$: note that for any $\lambda \in (0, 2\varepsilon/M^2)$, then

$$|f(x) - \widetilde{p}(x)|^2 < E^2 - 2\lambda\varepsilon + \lambda^2 M^2 < E^2 - \frac{4\varepsilon^2}{M^2} + \frac{4\varepsilon^2}{M^2} = E^2 = \|f - p_*\|^2 \tag{2.8}$$

for all $x \in \widetilde{\mathcal{X}}$. Thus $\widetilde{p}$ beats $p_*$ on $\widetilde{\mathcal{X}}$. Now since $\mathcal{X}$ comprises the points where $|f(x) - p_*(x)|$ attains its maximum, away from $\widetilde{\mathcal{X}}$ this error must be bounded away from its maximum, i.e., there exists some $\eta > 0$ such that

$$\max_{\substack{x \in [a,b] \\ x \notin \widetilde{\mathcal{X}}}} |f(x) - p_*(x)| \leq E - \eta.$$

Now we want to show that $|f(x) - \widetilde{p}(x)| < E$ for these $x \notin \widetilde{\mathcal{X}}$ as well. In particular, for such $x$

$$\begin{aligned} |f(x) - \widetilde{p}(x)| &= |f(x) - p_*(x) + \lambda\widetilde{q}(x)| \\ &\leq |f(x) - p_*(x)| + \lambda|\widetilde{q}(x)| \\ &\leq E - \eta + \lambda\|\widetilde{q}\|_\infty, \end{aligned}$$

and so if $\lambda \in (0, \eta/\|\widetilde{q}\|_\infty)$,

$$|f(x) - \widetilde{p}(x)| < E - \eta + \frac{\eta}{\|\widetilde{q}\|_\infty}\|\widetilde{q}\|_\infty = E.$$

In conclusion, if

$$\lambda \in \Big(0, \min(2\varepsilon/M^2, \eta/\|\widetilde{q}\|_\infty)\Big),$$

then we constructed $\widetilde{p}(x) := p_*(x) - \lambda\widetilde{q}(x)$ such that

$$|f(x) - \widetilde{p}(x)| < E \quad \text{for all } x \in [a,b],$$

i.e., $\|f - \widetilde{p}\|_\infty < \|f - p_*\|$, contradicting the optimality of $p_*$. ∎

With this lemma, we can readily complete the proof of the Oscillation Theorem.

*Completion of the Proof of the Oscillation Theorem.* We must show that if $p_*$ is a minimax approximation to $f$, then there exist $n+2$ points in $[a,b]$ on which the error $f - p_*$ changes sign. If $p_* = f$, the result holds trivially. Suppose then that $\|f - p_*\|_\infty > 0$. In the language of Lemma 2.1, we need to show that (a) the set $\mathcal{X}$ contains (at least) $n+2$ points and (b) the error oscillates sign at these points. Suppose this is not the case, i.e., we cannot identify $n+2$ consecutive points in $\mathcal{X}$ at which the error oscillates in sign. Suppose we can only identify

Recall that $\mathcal{X}$ contains all the points $x \in [a,b]$ for which the maximum error is attained: $|f(x) - p_*(x)| = \|f - p_*\|_\infty$.

$m$ such points, $1 \le m < n+2$, which we label $x_0 < \cdots < x_{m-1}$. We will show how to construct a $q$ that violates Lemma 2.1.

If $m = 1$, $f(x) - p_*(x)$ has the same sign for all $x \in \mathcal{X}$. Set $q(x) = -\text{sgn}(f(x_0) - p_*(x_0))$ (a constant, hence in $\mathcal{P}_n$), so that $(f(x) - p_*(x))q(x) < 0$ for all $x \in \mathcal{X}$, contradicting Lemma 2.1.

If $m > 1$, the between each consecutive pair of these $m$ points one can then identify $\widetilde{x}_1, \ldots, \widetilde{x}_{m-1}$ where the error changes sign. (See the sketch in the margin.) Then define

$$q(x) = \pm(x - \widetilde{x}_1)(x - \widetilde{x}_2)\cdots(x - \widetilde{x}_{m-1}).$$

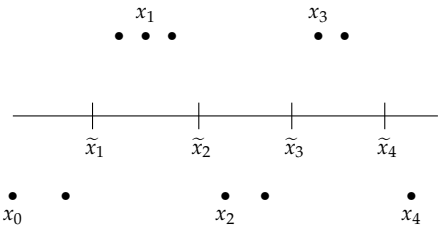
Since $m < n+2$ by assumption, $m - 1 \le n$, i.e., $q \in \mathcal{P}_n$, so Lemma 2.1 should hold with this choice of $q$. Since the sign of $q(x)$ does not change between its roots, it does not change within the intervals

$$(a, \widetilde{x}_1), \quad (\widetilde{x}_1, \widetilde{x}_2), \quad \cdots, \quad (\widetilde{x}_{m-2}, \widetilde{x}_{m-1}), \quad (\widetilde{x}_{m-1}, b),$$

and the sign of $q$ flips between each of these intervals. Thus the sign of $\big(f(x) - p_*(x)\big)q(x)$ is the same for all $x \in \mathcal{X}$. Pick the $\pm$ sign in the definition of $q$ such that

$$\big(f(x) - p_*(x)\big)q(x) < 0 \quad \text{for all } x \in \mathcal{X},$$

thus contradicting Lemma 2.1. Hence, there must be (at least) $n+2$ consecutive points in $\mathcal{X}$ at which the error flips sign. ∎




*sketch for $m = 5$*
$\bullet = f(x) - p_*(x)$ for $x \in \mathcal{X}$

Thus far we have been careful to only speak of *a* minimax approximation, rather than *the* minimax approximation. In fact, the later terminology is more precise, for the minimax approximant is unique.

---

**Theorem 2.3** (Uniqueness of minimax approximant)**.**
The minimax approximant $p_* \in \mathcal{P}_n$ of $f \in C[a,b]$ over the interval $[a,b]$ is unique.

---

The proof is a straightforward application of the Oscillation Theorem. Suppose $p_1$ and $p_2$ are both minimax approximations from $\mathcal{P}_n$ to $f$ on $[a,b]$. Then one can show that $(p_1 + p_2)/2$ is also a minimax approximation. Apply the Oscillation Theorem to obtain $n+2$ points at which the error for $(p_1 + p_2)/2$ oscillates sign. One can show that these points must also be oscillation points for $p_1$ and $p_2$, and that $p_1$ and $p_2$ agree at these $n+2$ points. Polynomials of degree $n$ that agree at $n+2$ points must be the same.

For the full details of this proof, see Theorem 8.5 in Süli and Mayers.

This oscillation property forms the basis of algorithms that find the minimax approximation: iteratively adjust an approximating polynomial until it satisfies the oscillation property. The most famous algorithm for computing the minimax approximation is called the

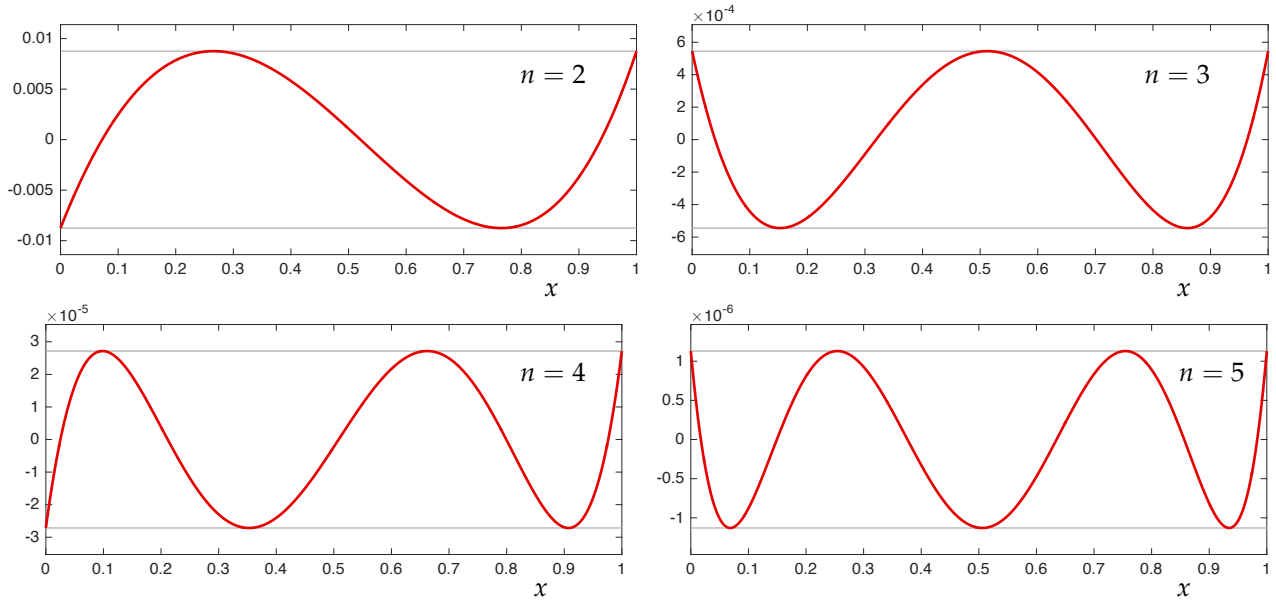



Figure 2.3: Illustration of the equioscillating minimax error $f - p_*$ for approximations of degree $n = 2, 3, 4$, and 5 with $f(x) = e^x$ for $x \in [a, b]$. In each case, the error attains its maximum with alternating sign at $n + 2$ points.

*Remez exchange algorithm*, essentially a specialized linear programming procedure. In exact arithmetic, this algorithm is guaranteed to terminate with the correct answer in finitely many operations.

The oscillation property is demonstrated in the Example 2.1, where we approximated $f(x) = e^x$ with a constant. Indeed, the maximum error is attained at two points (that is, $n + 2$, since $n = 0$), and the error differs in sign at those points. Figure 2.3 shows the errors $f(x) - p_*(x)$ for minimax approximations $p_*$ of increasing degree. The oscillation property becomes increasingly apparent as the polynomial degree increases. In each case, there are $n + 2$ extreme points of the error, where $n$ is the degree of the approximating polynomial.

These examples were computed in MATLAB using the Chebfun package's `remez` algorithm. For details, see `www.chebfun.org`.

**Example 2.2 ($e^x$ revisited).** Now we shall use the Oscillation Theorem to compute the optimal linear minimax approximation to $f(x) = e^x$ on $[0, 1]$. Assume that the minimax polynomial $p_* \in \mathcal{P}_1$ has the form $p_*(x) = \alpha + \beta x$. Since $f$ is convex, a quick sketch of the situation suggests the maximal error will be attained at the end points of the interval, $x_0 = 0$ and $x_2 = 1$. We assume this to be true, and seek some third point $x_1 \in (0, 1)$ that attains the same maximal error, $\delta$, but with opposite sign. If we can find such a point, then the Oscillation Theorem guarantees that the resulting polynomial is optimal, confirming our assumption that the maximal error was attained at the ends of the interval.

This scenario suggests the following three equations:

$$\begin{aligned} f(x_0) - p_*(x_0) &= \delta \\ f(x_1) - p_*(x_1) &= -\delta \\ f(x_2) - p_*(x_2) &= \delta. \end{aligned}$$

Substituting the values $x_0 = a$, $x_2 = b$, and $p_*(x) = \alpha + \beta x$, these equations become

$$\begin{aligned} 1 - \alpha &= \delta \\ \mathrm{e}^{x_1} - \alpha - \beta x_1 &= -\delta \\ \mathrm{e} - \alpha - \beta &= \delta. \end{aligned}$$

The first and third equation together imply $\beta = \mathrm{e} - 1$. We also deduce that $2\alpha = \mathrm{e}^{x_1} - x_1(\mathrm{e} - 1) + 1$. A variety of choices for $x_1$ will satisfy these conditions, but in those cases $\delta$ will not be the *maximal error*. We must ensure that

$$|\delta| = \max_{x \in [a,b]} |f(x) - p_*(x)|.$$

To make this happen, require that *the derivative of error* be zero at $x_1$, reflecting that the error $f - p_*$ attains a local minimum/maximum at $x_1$. (The plots in Figure 2.3 confirm that this is reasonable.) Imposing the condition that $f'(x_1) - p_*'(x_1) = 0$ yields

This requirement need not hold at the points $x_0$ and $x_2$, since these points are on the ends of the interval $[a, b]$; it is only required at the interior points where the extreme error is attained, $x_j \in (a, b)$.

$$\mathrm{e}^{x_1} - \beta = 0.$$

Now we can explicitly solve the equations to obtain

$$\begin{aligned} \alpha &= \tfrac{1}{2}\big(\mathrm{e} - (\mathrm{e} - 1)\log(\mathrm{e} - 1)\big) = 0.89406\ldots \\ \beta &= \mathrm{e} - 1 = 1.71828\ldots \\ x_1 &= \log(\mathrm{e} - 1) = 0.54132\ldots \\ \delta &= \tfrac{1}{2}\big(2 - \mathrm{e} + (\mathrm{e} - 1)\log(\mathrm{e} - 1)\big) = 0.10593\ldots. \end{aligned}$$

Notice that we have a system of four *nonlinear* equations in four unknowns, due to the $\mathrm{e}^{x_1}$ term. Generally such nonlinear systems might not have a solution; in this case we can compute one.

Figure 2.4 shows the optimal approximation, along with the error $f(x) - p_*(x) = \mathrm{e}^x - (\alpha + \beta x)$. In particular, notice the size of the maximum error ($\delta = 0.10593\ldots$) and the point $x_1 = 0.54132\ldots$ at which this error is attained.

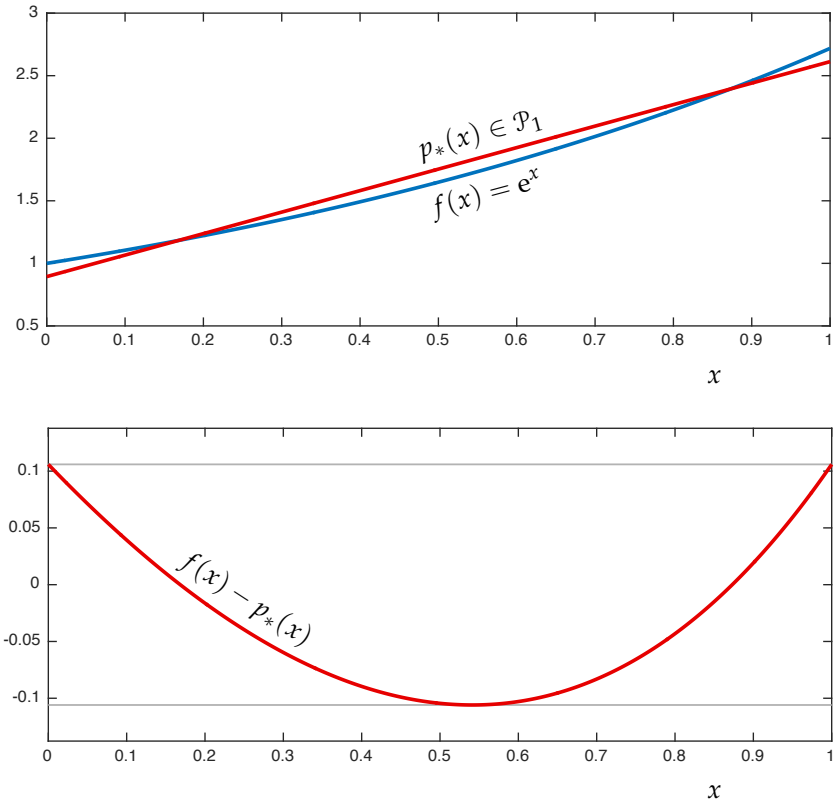



Figure 2.4: The top plot shows the minimax approximation $p_*$ of degree $n = 1$ (red) to $f(x) = e^x$ (blue); the bottom plot shows the error $f(x) - p_*(x)$, equioscillating at $n + 1 = 3$ points.

# LECTURE 15: *Chebyshev Polynomials for Optimal Interpolation*

## 2.3 *Optimal Interpolation Points via Chebyshev Polynomials*

As an application of the minimax approximation procedure, we consider how best to choose interpolation points $\{x_j\}_{j=0}^n$ to minimize

$$\|f - p_n\|_\infty,$$

where $p_n \in \mathcal{P}_n$ is the interpolant to $f$ at the specified points.

Recall the interpolation error bound developed in Section 1.6: If $f \in C^{n+1}[a,b]$, then for any $x \in [a,b]$ there exists some $\xi \in [a,b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n}(x - x_j).$$

Taking absolute values and maximizing over $[a,b]$ yields the bound

$$\|f - p_n\|_\infty = \max_{\xi\in[a,b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \max_{x\in[a,b]} \Big|\prod_{j=0}^{n}(x - x_j)\Big|.$$

For Runge's example, $f(x) = 1/(1+x^2)$ for $x \in [-5,5]$, we observed that $\|f - p_n\|_\infty \to \infty$ as $n \to \infty$ if the interpolation points $\{x_j\}$ are uniformly spaced over $[-5,5]$. However, Marcinkiewicz's theorem (Section 1.6) guarantees there is always some scheme for assigning the interpolation points such that $\|f - p_n\|_\infty \to 0$ as $n \to \infty$. While there is no fail-safe *a priori* system for picking interpolations points that will yield uniform convergence for *all* $f \in C[a,b]$, there is a distinguished choice that works exceptionally well for just about every function you will encounter in practice. We determine this set of interpolation points by choosing those $\{x_j\}_{j=0}^n$ that *minimize the error bound* (which is distinct from – but hopefully akin to – minimizing the error itself, $\|f - p_n\|_\infty$). That is, we want to solve

$$\min_{x_0,\ldots,x_n} \max_{x\in[a,b]} \Big|\prod_{j=0}^{n}(x - x_j)\Big|. \tag{2.9}$$

Notice that

$$\begin{aligned}\prod_{j=0}^{n}(x - x_j) &= x^{n+1} - x^n \sum_{j=0}^{n} x_j + x^{n-1} \sum_{j=0}^{n}\sum_{k=0}^{n} x_j x_k - \cdots + (-1)^{n+1} \prod_{j=0}^{n} x_j \\ &= x^{n+1} - r(x),\end{aligned}$$

where $r \in \mathcal{P}_n$ is a degree-$n$ polynomial depending on the interpolation nodes $\{x_j\}_{j=0}^n$.

For example, when $n = 1$,

$$(x - x_0)(x - x_1) = x^2 - \Big((x_0 + x_1)x - x_0 x_1\Big) = x^2 - r_1(x),$$

where $r_1(x) = (x_0 + x_1)x - x_0x_1$. By varying $x_0$ and $x_1$, we can obtain make $r_1$ any function in $\mathcal{P}_1$.

To find the optimal interpolation points according to (2.9), we should solve

$$\min_{r\in\mathcal{P}_n} \max_{x\in[a,b]} |x^{n+1} - r(x)| \;=\; \min_{r\in\mathcal{P}_n} \|x^{n+1} - r(x)\|_\infty.$$

Here the goal is to approximate an $(n+1)$-degree polynomial, $x^{n+1}$, with an $n$-degree polynomial. The method of solution is somewhat indirect: we will produce a class of polynomials of the form $x^{n+1} - r(x)$ that satisfy the requirements of the Oscillation Theorem, and thus $r(x)$ must be the minimax polynomial approximation to $x^{n+1}$. As we shall see, the roots of the resulting polynomial $x^{n+1} - r(x)$ will fall in the interval $[a, b]$, and can thus be regarded as 'optimal' interpolation points. For simplicity, we shall focus on the interval $[a, b] = [-1, 1]$.

**Definition 2.1.** The degree-$n$ *Chebyshev polynomial* is defined for $x \in [-1, 1]$ by the formula

$$T_n(x) = \cos(n \cos^{-1} x).$$

At first glance, this formula may not appear to define a polynomial at all, since it involves trigonometric functions. But computing the first few examples, we find

$$n = 0: \quad T_0(x) = \cos(0 \cos^{-1} x) = \cos(0) = 1$$
$$n = 1: \quad T_1(x) = \cos(\cos^{-1} x) = x$$
$$n = 2: \quad T_2(x) = \cos(2 \cos^{-1} x) = 2\cos^2(\cos^{-1} x) - 1 = 2x^2 - 1.$$

For $n = 2$, we employed the identity $\cos 2\theta = 2\cos^2\theta - 1$, substituting $\theta = \cos^{-1} x$. More generally, use the cosine addition formula

$$\cos\alpha + \cos\beta = 2\cos\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right)$$

to get the identity

$$\cos\big((n+1)\theta\big) = 2\cos\theta\cos n\theta - \cos\big((n-1)\theta\big).$$

This formula implies, for $n \geq 2$,

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

a formula related to the three term recurrence used to construct orthogonal polynomials.

Chebyshev polynomials exhibit a wealth of interesting properties, of which we mention just three.

Furthermore, it doesn't apply if $|x| > 1$. For such $x$ one can define the Chebyshev polynomials using hyperbolic trigonometric functions, $T_n(x) = \cosh(n \cosh^{-1} x)$. Indeed, using hyperbolic trigonometric identities, one can show that this expression generates for $x \notin [-1, 1]$ the same polynomials we get for $x \in [-1, 1]$ from the standard trigonometric identities. We discuss this point in more detail at the end of the section.

In fact, Chebyshev polynomials are orthogonal polynomials on $[-1, 1]$ with respect to the inner product

$$\langle f, g\rangle = \int_a^b \frac{f(x)g(x)}{\sqrt{1-x^2}}\,\mathrm{d}x,$$

a fact we will use when studying Gaussian quadrature later in the semester.

**Theorem 2.4.** Let $T_n$ be the degree-$n$ Chebyshev polynomial

$$T_n(x) = \cos(n \cos^{-1} x)$$

for $x \in [-1, 1]$.

- $|T_n(x)| \leq 1$ for $x \in [-1, 1]$.
- The roots of $T_n$ are the $n$ points $\xi_j = \cos \frac{(2j-1)\pi}{2n}$, $j = 1, \ldots, n$.
- For $n \geq 1$, $|T_n(x)|$ is maximized on $[-1, 1]$ at the $n + 1$ points $\eta_j = \cos(j\pi/n)$, $j = 0, \ldots, n$:

$$T_n(\eta_j) = (-1)^j.$$

*Proof.* These results follow from direct calculations. For $x \in [-1, 1]$, $T_n(x) = \cos(n \cos^{-1}(x))$ cannot exceed one in magnitude because cosine cannot exceed one in magnitude. To verify the formula for the roots, compute

$$T_n(\xi_j) = \cos\left(n \cos^{-1} \cos\left(\frac{(2j-1)\pi}{2n}\right)\right) = \cos\left(\frac{(2j-1)\pi}{2}\right) = 0,$$

since cosine is zero at half-integer multiples of $\pi$. Similarly,

$$T_n(\eta_j) = \cos\left(n \cos^{-1} \cos\left(\frac{j\pi}{n}\right)\right) = \cos(j\pi) = (-1)^j.$$

Since $T_n(\eta_j)$ is a nonzero degree-$n$ polynomial, it cannot attain more than $n + 1$ extrema on $[-1, 1]$, including the endpoint: we have thus characterized all the maxima of $|T_n|$ on $[-1, 1]$. ∎

Figure 2.5 shows Chebyshev polynomials $T_n$ for nine different values of $n$.

### 2.3.1 *Interpolation at Chebyshev Points*

Finally, we are ready to solve the key minimax problem that will reveal optimal interpolation points. Looking at the above plots of Chebyshev polynomials, with their striking equioscillation properties, perhaps you have already guessed the solution yourself.

We defined the Chebyshev polynomials so that

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

with $T_0(x) = 1$ and $T_1(x) = x$. Thus $T_{n+1}$ has the leading coefficient $2^n$ for $n \geq 0$. Define

$$\widehat{T}_{n+1} = 2^{-n} T_{n+1}$$

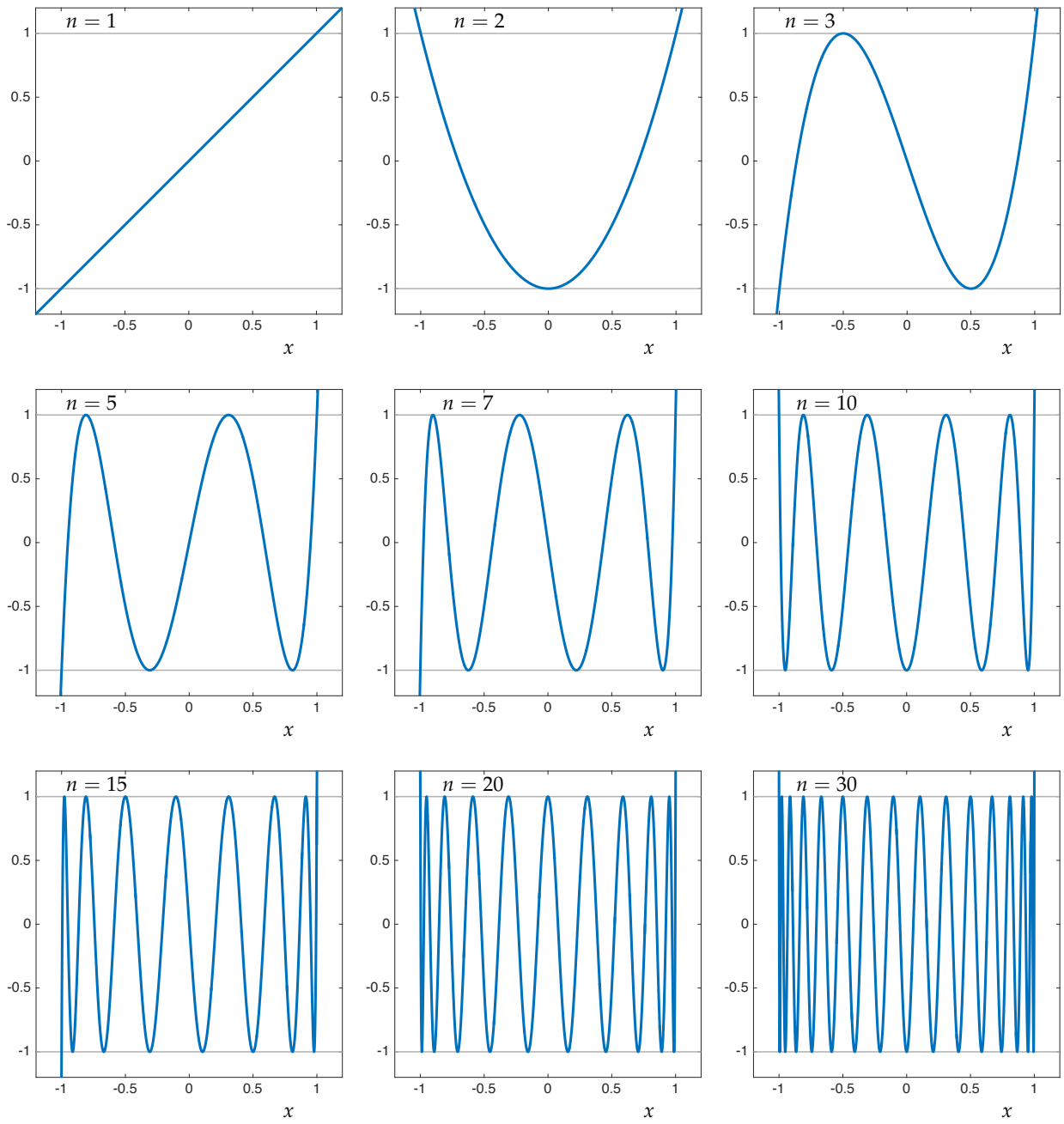



Figure 2.5: Chebyshev polynomials $T_n$ of degree $n = 1, 2, 3$ (top), $n = 5, 7, 10$ (middle), and $n = 15, 20, 30$ (bottom). Note how rapidly these polynomials grow outside the interval $[-1, 1]$.

for $n \geq 0$, with $\widehat{T}_0(x) = 1$. These *normalized* Chebyshev polynomials are *monic*, i.e., the leading term in $\widehat{T}_{n+1}(x)$ is $x^{n+1}$, rather than $2^n x^{n+1}$ as for $T_{n+1}(x)$. Thus, we can write

$$\widehat{T}_{n+1}(x) = x^{n+1} - r_n(x)$$

for some polynomial $r_n(x) = x^{n+1} - \widehat{T}_{n+1}(x) \in \mathcal{P}_n$. We do not especially care about the particular coefficients of this $r_n$; our quarry will be the *roots* of $\widehat{T}_{n+1}$, the optimal interpolation points.

For $n \geq 0$, the polynomials $\widehat{T}_{n+1}(x)$ oscillate between $\pm 2^{-n}$ for $x \in [-1, 1]$, with the maximal values attained at

$$\eta_j = \cos\left(\frac{j\pi}{n+1}\right)$$

for $j = 0, \ldots, n+1$. In particular,

$$\widehat{T}_{n+1}(\eta_j) = (\eta_j)^{n+1} - r_n(\eta_j) = (-1)^j 2^{-n}.$$

Thus, we have found a polynomial $r_n \in \mathcal{P}_n$, together with $n+2$ distinct points, $\eta_j \in [-1, 1]$ where the maximum error

$$\max_{x \in [-1,1]} |x^{n+1} - r_n(x)| = 2^{-n}$$

is attained with alternating sign. Thus, by the oscillation theorem, we have found the minimax approximation to $x^{n+1}$.

---

**Theorem 2.5** (Optimal approximation of $x^{n+1}$)**.**

The optimal approximation to $x^{n+1}$ from $\mathcal{P}_n$ for $x \in [-1, 1]$ is

$$r_n(x) = x^{n+1} - \widehat{T}_{n+1}(x) = x^{n+1} - 2^{-n} T_{n+1}(x) \in \mathcal{P}_n.$$

---

Thus, the optimal interpolation points are those $n+1$ roots of $x^{n+1} - r_n(x)$, that is, the roots of the degree-$(n+1)$ Chebyshev polynomial:

$$\xi_j = \cos\left(\frac{(2j+1)\pi}{2n+2}\right), \quad j = 0, \ldots, n.$$

For generic intervals $[a, b]$, a change of variable demonstrates that the same points, appropriately shifted and scaled, will be optimal.

Similar properties hold if interpolation is performed at the $n+1$ points

$$\eta_j = \cos\left(\frac{j\pi}{n}\right), \quad j = 0, \ldots, n,$$

which are also called Chebyshev points and are perhaps more popular due to their slightly simpler formula. (We used these points to successfully interpolate Runge's function, scaled to the interval $[-5, 5]$.) While these points differ from the roots of the Chebyshev polynomial, they *have the same distribution* as $n \to \infty$. That is the key.

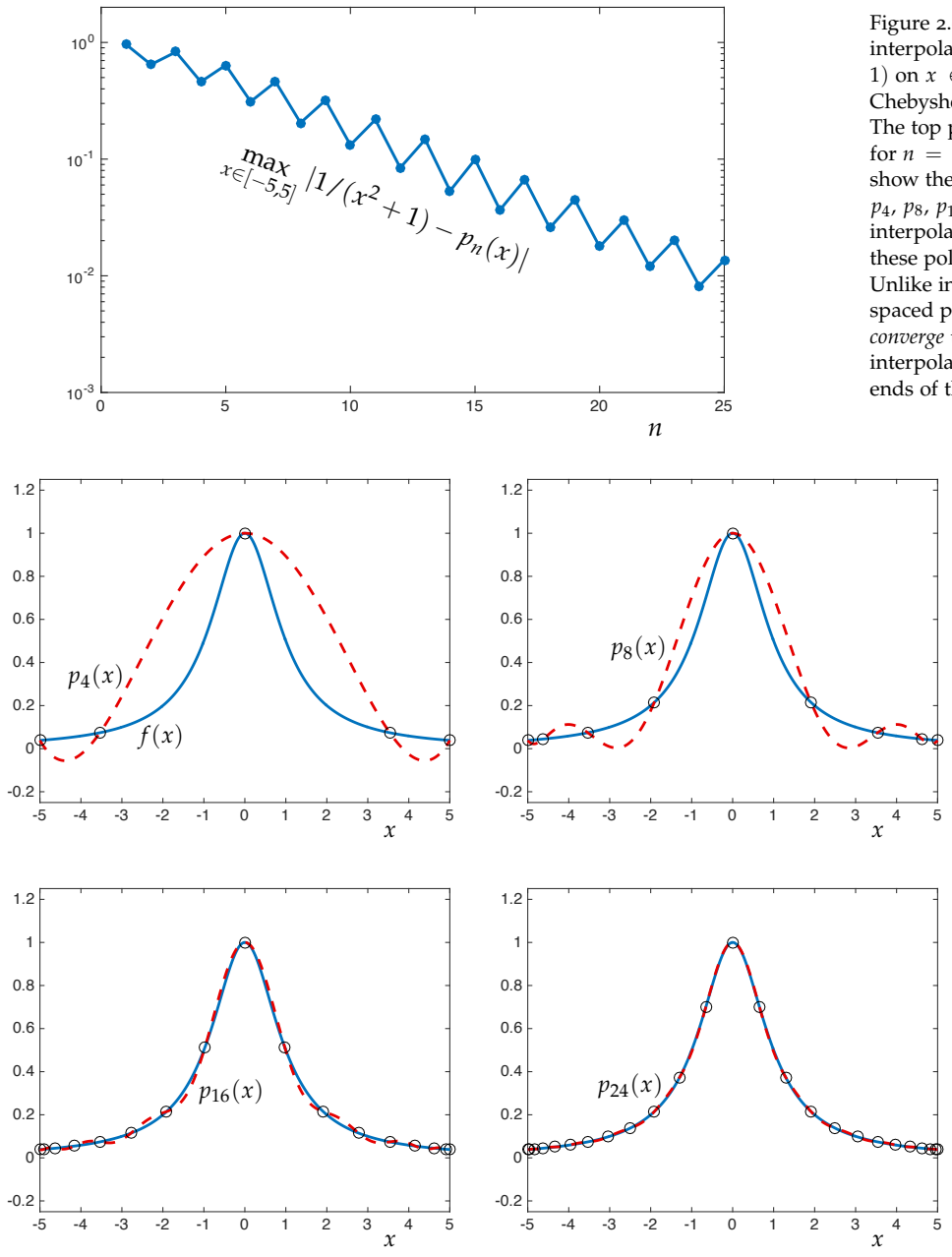



Figure 2.6: Repetition of Figure 1.8, interpolating Runge's function $1/(x^2+1)$ on $x \in [-5,5]$, but now using Chebyshev points $x_j = 5\cos(j\pi/n)$. The top plot shows this convergence for $n = 0,\ldots,25$; the bottom plots show the interpolating polynomials $p_4$, $p_8$, $p_{16}$, and $p_{24}$, along with the interpolation points that determine these polynomials (black circles). Unlike interpolation at uniformly spaced points, these interpolants *do converge* to $f$ as $n \to \infty$. Notice how the interpolation points cluster toward the ends of the domain $[-5,5]$.

We emphasize the utility of interpolation at Chebyshev points by quoting the following result from Trefethen's excellent *Approximation Theory and Approximation Practice* (SIAM, 2013). Trefethen emphasizes that worst-case results like Faber's theorem (Theorem 1.4) give misleadingly pessimistic concerns about interpolation. If the function $f \in C[a,b]$ has just a bit of smoothness (i.e., bounded derivatives), interpolation in Chebyshev points is 'bulletproof'. The following theorem consolidates aspects of Theorem 7.2 and 8.2 in Trefethen's book.

The results are stated for $[a,b] = [-1,1]$ but can be adapted to any real interval.

---

**Theorem 2.6** (Convergence of Interpolants at Chebyshev Points)**.**
For any $n > 0$, let $p_n$ denote the interpolant to $f \in C[-1,1]$ at the Chebyshev points

$$x_j = \cos\left(\frac{j\pi}{n}\right), \quad j = 0, \ldots, n.$$

- Suppose $f \in C^\nu[-1,1]$ for some $\nu \geq 1$, with $f^{(\nu)}$ having variation $V(\nu)$, i.e.,

$$V(\nu) := \max_{x\in[-1,1]} f^{(\nu)}(x) - \min_{x\in[-1,1]} f^{(\nu)}(x).$$

  Then for any $n > \nu$,

$$\|f - p_n\|_\infty \leq \frac{4V(\nu)}{\pi(\nu(n-\nu)^\nu)}.$$

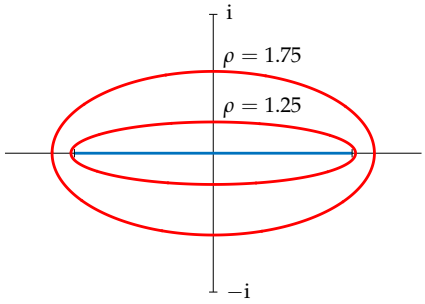
- Suppose $f$ is *analytic* on $[-1,1]$ and can be analytically continued (into the complex plane) onto the region bounded by the ellipse

$$E_\rho := \left\{\frac{\rho e^{i\theta} + e^{-i\theta}/\rho}{2} : \theta \in [0, 2\pi)\right\}.$$

  Suppose further that $|f(z)| \leq M$ on and inside $E_\rho$. Then

$$\|f - p_n\|_\infty \leq \frac{2M\rho^{-n}}{\rho - 1}.$$

---




Interval $[-1,1]$ (blue), with two ellipses $E_\rho$ for $\rho = 1.25$ and $\rho = 1.75$.

For example, the first part of this theorem implies that if $f'$ exists and is bounded, then $\|f - p_n\|_\infty$ must converge at least as fast as $1/n$ as $n \to \infty$. While that is not such a fast rate, it does indeed show convergence of the interpolant. The second part of the theorem ensures that if $f$ is well behaved in the region of the complex plane around $[-1,1]$, the convergence will be extremely fast: the larger the area of $\mathbb{C}$ in which $f$ is well behaved, the faster the convergence.

### 2.3.2 *Chebyshev polynomials beyond* $[-1,1]$

Another way of interpreting the equioscillating property of Chebyshev polynomials is that $T_n$ solves the approximation problem

$$\|T_n\|_\infty = \min_{\substack{p\in\mathcal{P}_n \\ p \text{ monic}}} \|p\|_\infty,$$

over the interval $[-1,1]$, where a polynomial is *monic* if it has the form $x^n + q(x)$ for $q \in \mathcal{P}_{n-1}$.

In some applications, such as the analysis of iterative methods for solving large-scale systems of linear equations, one needs to bound the size of the Chebyshev polynomial *outside the interval* $[-1,1]$. Figure 2.5 shows that $T_n$ grows very quickly outside $[-1,1]$, even for modest values of $n$. How fast?

To describe Chebyshev polynomials outside $[-1,1]$, we must replace the trigonometric functions in the definition $T_n(x) = \cos(n\cos^{-1}x)$ with hyperbolic trigonometric functions:

$$T_n(x) = \cosh(n\cosh^{-1}x), \qquad x \notin (-1,1). \tag{2.10}$$

Is this definition is consistent with

$$T_n(x) = \cos(n\cos^{-1}x), \qquad x \in [-1,1]$$

used previously? Trivially one can see that the new definition also gives $T_0(x) = 1$ and $T_1(x) = x$. Like standard trigonometric functions, the hyperbolic functions also satisfy the addition formula

$$\cosh\alpha + \cosh\beta = 2\cosh\left(\frac{\alpha+\beta}{2}\right)\cosh\left(\frac{\alpha-\beta}{2}\right),$$

and so

$$\cosh\big((n+1)\theta\big) = 2\cosh\theta\cosh n\theta - \cosh\big((n-1)\theta\big),$$

leading to the same three-term recurrence as before:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Thus, the definitions are consistent.

We would like a more concrete formula for $T_n(x)$ for $x \notin [-1,1]$ than we could obtain from the formula (**??**). Thankfully Chebyshev polynomials have infinitely many interesting properties to lean on. Consider the change of variables

$$x = \frac{w + w^{-1}}{2},$$

which allows us to write

$$x = \frac{e^{\log w} + e^{-\log w}}{2} = \cosh(\log w).$$

Thus work from the definition to obtain

$$\begin{aligned} T_n(x) &= \cosh(n\cosh^{-1}(x)) \\ &= \cosh(n\log w) \\ &= \cosh(\log w^n) = \frac{e^{\log(w^n)} + e^{-\log(w^n)}}{2} = \frac{w^n + w^{-n}}{2}. \end{aligned}$$

We emphasize this last important formula:

$$T_n(x) = \frac{w^n + w^{-n}}{2}, \qquad x = \frac{w + w^{-1}}{2} \notin (-1,1). \tag{2.11}$$

We have thus shown that $|T_n(x)|$ *will grow exponentially in* $n$ *for any* $x \notin (-1,1)$ *for which* $|w| \neq 1$. When does $|w| = 1$? Only when $x = \pm 1$. Hence,

$|T_n(x)|$ grows exponentially in $n$ for all $x \notin [-1,1]$.

**Example 2.3.** We want to evaluate $T_n(2)$ as a function of $n$. First, find $w$ such that $2 = (w + w^{-1})/2$, i.e.,

$$w^2 - 4w + 1 = 0.$$

Solve this quadratic for

$$w_\pm = 2 \pm \sqrt{3}.$$

We take $w = 2 + \sqrt{3} = 3.7320\ldots$. Thus by (2.11)

$$T_n(2) = \frac{(2+\sqrt{3})^n + (2-\sqrt{3})^n}{2} \approx \frac{(2+\sqrt{3})^n}{2}$$

as $n \to \infty$, since $(2-\sqrt{3})^n = (0.2679\ldots)^n \to 0$.

Take a moment to reflect on this: We have a beautifully concrete way to write down $|T_n(x)|$ that does not involve any hyperbolic trigonometric formulas, or require use of the Chebyshev recurrence relation. Formulas of this type can be very helpful for analysis in various settings. You will see one such example on Problem Set 3.

Which $\pm$ choice should you make? It does not matter. Notice that $(2-\sqrt{3})^{-1} = 2+\sqrt{3}$, and this happens in general: $w_\pm = 1/w_\mp$.

# LECTURE 16: *Introduction to Least Squares Approximation*

## 2.4 Least squares approximation

The minimax criterion is an intuitive objective for approximating a function. However, in many cases it is more appealing (for both computation and for the given application) to find an approximation to $f$ that *minimizes the integral of the square of the error*.

Given $f \in C[a,b]$, find $P_* \in \mathcal{P}_n$ such that

$$\Big(\int_a^b (f(x)-P_*(x))^2\,\mathrm{d}x\Big)^{1/2} = \min_{p\in\mathcal{P}_n}\Big(\int_a^b (f(x)-p(x))^2\,\mathrm{d}x\Big)^{1/2}. \tag{2.12}$$

This is an example of a *least squares problem*.

### 2.4.1 Inner products for function spaces

To facilitate the development of least squares approximation theory, we introduce a formal structure for $C[a,b]$. First, recognize that $C[a,b]$ is a *linear space*: any linear combination of continuous functions on $[a,b]$ must itself be continuous on $[a,b]$.

**Definition 2.2.** The *inner product* of the functions $f, g \in C[a,b]$ is

$$\langle f, g\rangle = \int_a^b f(x)g(x)\,\mathrm{d}x.$$

The inner product satisfies the following basic axioms:

- $\langle \alpha f + g, h\rangle = \alpha\langle f, h\rangle + \langle g, h\rangle$ for all $f, g, h \in C[a,b]$ and all $\alpha \in \mathbb{R}$;
- $\langle f, g\rangle = \langle g, f\rangle$ for all $f, g \in C[a,b]$;
- $\langle f, f\rangle \geq 0$ for all $f \in C[a,b]$.

With this inner product we associate the norm

$$\|f\|_2 := \langle f, f\rangle^{1/2} = \Big(\int_a^b f(x)^2\,\mathrm{d}x\Big)^{1/2}.$$

This is often called the '$L^2$ norm,' where the superscript '2' in $L^2$ refers to the fact that the integrand involves the *square* of the function $f$; the $L$ stands for *Lebesgue*, coming from the fact that this inner product can be generalized from $C[a,b]$ to the set of all functions that are *square-integrable*, in the sense of Lebesgue integration. By restricting our attention to continuous functions, we dodge the measure-theoretic complexities.

For simplicity we are assuming that $f$ and $g$ are real-valued. To handle complex-valued functions, one generalizes the inner product to

$$\langle f, g\rangle = \int_a^b f(x)\overline{g(x)}\,\mathrm{d}x,$$

which then gives $\langle f, g\rangle = \overline{\langle g, f\rangle}$.

The Lebesgue theory gives a more robust definition of the integral than the conventional Riemann approach. With such notions one can extend least squares approximation beyond $C[a,b]$, to more exotic function spaces.

### 2.4.2 *Least squares minimization via calculus*

We are now ready to solve the least squares problem. We shall call the optimal polynomial $P_* \in \mathcal{P}_n$, i.e.,

$$\|f - P_*\|_2 = \min_{p \in \mathcal{P}_n} \|f - p\|_2.$$

We can solve this minimization problem using basic calculus. Consider this example for $n = 1$, where we optimize the error over polynomials of the form $p(x) = c_0 + c_1 x$. The polynomial that minimizes $\|f - p\|_2$ will also minimize its square, $\|f - p\|_2^2$. For any given $p \in \mathcal{P}_1$, define the error function

$$\begin{aligned} E(c_0, c_1) := \|f(x) - (c_0 + c_1 x)\|_{L^2}^2 &= \int_a^b (f(x) - c_0 - c_1 x)^2 \, \mathrm{d}x \\ &= \int_a^b \Big( f(x)^2 - 2f(x)(c_0 + c_1 x) + (c_0^2 + 2c_0 c_1 x + c_1^2 x^2) \Big) \, \mathrm{d}x \\ &= \int_a^b f(x)^2 \, \mathrm{d}x - 2c_0 \int_a^b f(x) \, \mathrm{d}x - 2c_1 \int_a^b x f(x) \, \mathrm{d}x \\ &\qquad + c_0^2 (b - a) + c_0 c_1 (b^2 - a^2) + \tfrac{1}{3} c_1^2 (b^3 - a^3). \end{aligned}$$

To find the optimal polynomial, $P_*$, optimize $E$ over $c_0$ and $c_1$, i.e., find the values of $c_0$ and $c_1$ for which

$$\frac{\partial E}{\partial c_0} = \frac{\partial E}{\partial c_1} = 0.$$

First, compute

$$\begin{aligned} \frac{\partial E}{\partial c_0} &= -2 \int_a^b f(x) \, \mathrm{d}x + 2c_0 (b - a) + c_1 (b^2 - a^2) \\ \frac{\partial E}{\partial c_1} &= -2 \int_a^b x f(x) \, \mathrm{d}x + c_0 (b^2 - a^2) + c_1 \tfrac{2}{3} (b^3 - a^3). \end{aligned}$$

Setting these partial derivatives equal to zero yields

$$\begin{aligned} 2c_0 (b - a) + c_1 (b^2 - a^2) &= 2 \int_a^b f(x) \, \mathrm{d}x \\ c_0 (b^2 - a^2) + c_1 \tfrac{2}{3} (b^3 - a^3) &= 2 \int_a^b x f(x) \, \mathrm{d}x. \end{aligned}$$

These equations, linear in the unknowns $c_0$ and $c_1$, can be written in the matrix form

$$\begin{bmatrix} 2(b - a) & b^2 - a^2 \\ b^2 - a^2 & \frac{2}{3}(b^3 - a^3) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 2 \int_a^b f(x) \, \mathrm{d}x \\ 2 \int_a^b x f(x) \, \mathrm{d}x \end{bmatrix}.$$

When $b \neq a$ this system always has a unique solution. The resulting $c_0$ and $c_1$ are the coefficients for the monomial-basis expansion of the least squares approximation $P_* \in \mathcal{P}_1$ to $f$ on $[a, b]$.

**Example 2.4** ($f(x) = e^x$)**.** Apply this result to $f(x) = e^x$ for $x \in [0,1]$. Since

$$\int_0^1 e^x \, dx = e - 1, \qquad \int_0^1 x e^x \, dx = \left[ e^x (x-1) \right]_{x=0}^1 = 1,$$

we must solve the system

$$\begin{bmatrix} 2 & 1 \\ 1 & \frac{2}{3} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 2e - 2 \\ 2 \end{bmatrix}.$$

The desired solution is

$$c_0 = 4e - 10, \qquad c_1 = 18 - 6e.$$

Figure 2.7 compares $f$ to this least squares approximation $P_*$ and the minimax approximation $p_*$ computed earlier.

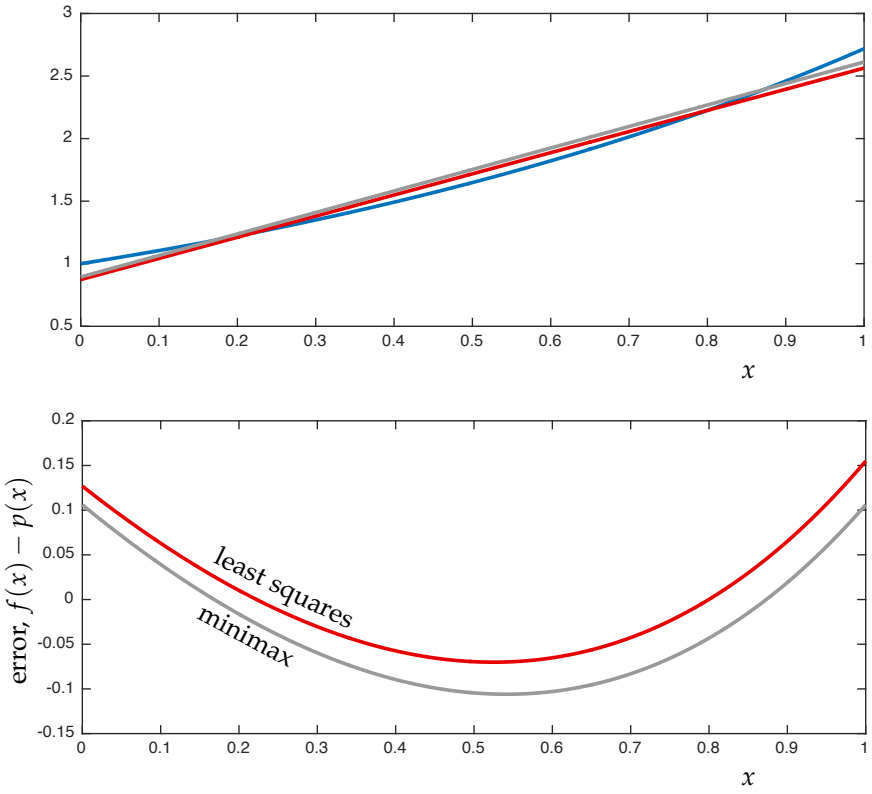



Figure 2.7: Top: Approximation of $f(x) = e^x$ (blue) over $x \in [0,1]$ via least squares ($P_*$, shown in red) and minimax ($p_*$, shown as a gray line).

Bottom: Error curves for least squares, $f - P_*$ (red), and minimax, $f - p_*$ (gray) approximation. While the curves have similar shape, note that the red curve does not attain its maximum deviation from $f$ at $n + 2 = 3$ points, while the gray one does.

We can see from the plots in Figure 2.7 that the approximation looks decent to the eye, but the error is not terribly small. We can decrease that error by increasing the degree of the approximating polynomial. Just as we used a 2-by-2 linear system to find the best linear approximation, a general $(n+1)$-by-$(n+1)$ linear system can be constructed to yield the degree-$n$ least squares approximation.

In fact, $\|f - P_*\|_2 = 0.06277\ldots$. This is indeed smaller than the 2-norm error of the minimax approximation $p_*$: $\|f - p_*\|_2 = 0.07228\ldots$.

### 2.4.3 General polynomial bases

Note that we performed the above minimization in the monomial basis: $p(x) = c_0 + c_1 x$ is a linear combination of 1 and $x$. Our experience with interpolation suggests that different choices for the basis may yield approximation algorithms with superior numerical properties. Thus, we develop the form of the approximating polynomial in an arbitrary basis.

Suppose $\{\phi_k\}_{k=0}^n$ is a basis for $\mathcal{P}_n$. Any $p \in \mathcal{P}_n$ can be written as

$$p(x) = \sum_{k=0}^{n} c_k \phi_k(x).$$

The error expression takes the form

$$E(c_0, \ldots, c_n) := \|f(x) - p(x)\|_{L^2}^2 = \int_a^b \Big(f(x) - \sum_{k=0}^{n} c_k \phi_k(x)\Big)^2 \, \mathrm{d}x$$

$$= \langle f, f\rangle - 2\sum_{k=0}^{n} c_k \langle f, \phi_k\rangle + \sum_{k=0}^{n}\sum_{\ell=0}^{n} c_k c_\ell \langle \phi_k, \phi_\ell\rangle.$$

To minimize $E$, we seek critical values of $\mathbf{c} = [c_0, \ldots, c_{n+1}]^T \in \mathbb{R}^{n+1}$, i.e., we want coefficients where the gradient of $E$ with respect to $\mathbf{c}$ is zero: $\nabla_{\mathbf{c}} E = \mathbf{0}$. To compute this gradient, evaluate $\partial E/\partial c_j$ for $j = 0, \ldots, n$:

$$\frac{\partial E}{\partial c_j} = \frac{\partial}{\partial c_j}\langle f, f\rangle - \frac{\partial}{\partial c_j}\Big(2\sum_{k=0}^{n} c_k \langle f, \phi_k\rangle\Big) + \frac{\partial}{\partial c_j}\Big(\sum_{k=0}^{n}\sum_{\ell=0}^{n} c_k c_\ell \langle \phi_k, \phi_\ell\rangle\Big)$$

$$= 0 - 2\langle f, \phi_j\rangle + \frac{\partial}{\partial c_j}\Big(c_j^2\langle \phi_j, \phi_j\rangle + \sum_{\substack{k=0\\k\neq j}}^{n} c_k c_j \langle \phi_k, \phi_j\rangle + \sum_{\substack{\ell=0\\\ell\neq j}}^{n} c_j c_\ell \langle \phi_j, \phi_\ell\rangle + \sum_{\substack{k=0\\k\neq j}}^{n}\sum_{\substack{\ell=0\\\ell\neq j}}^{n} c_k c_\ell \langle \phi_k, \phi_\ell\rangle\Big)$$

In this last line, we have broken the double sum on the previous line into four parts: one that contains $c_j^2$, two that contain $c_j$ ($c_k c_j$ for $k \neq j$; $c_j c_\ell$ for $\ell \neq j$), and one (the double sum) that does not involve $c_j$ at all. This decomposition makes it easier to compute the derivative:

$$\frac{\partial}{\partial c_j}\Big(c_j^2\langle \phi_j, \phi_j\rangle \;+ \sum_{\substack{k=0\\k\neq j}}^{n} c_k c_j \langle \phi_k, \phi_j\rangle + \sum_{\substack{\ell=0\\\ell\neq j}}^{n} c_j c_\ell \langle \phi_j, \phi_\ell\rangle + \sum_{\substack{k=0\\k\neq j}}^{n}\sum_{\substack{\ell=0\\\ell\neq j}}^{n} c_k c_\ell \langle \phi_k, \phi_\ell\rangle\Big)$$

$$= 2c_j\langle \phi_j, \phi_j\rangle + \sum_{\substack{k=0\\k\neq j}}^{n} c_k \langle \phi_k, \phi_j\rangle + \sum_{\substack{\ell=0\\\ell\neq j}}^{n} c_\ell \langle \phi_j, \phi_\ell\rangle + 0$$

$$= 2c_j\langle \phi_j, \phi_j\rangle + 2\sum_{\substack{k=0\\k\neq j}}^{n} c_k \langle \phi_k, \phi_j\rangle.$$

These terms contribute to $\partial E/\partial c_j$ to give

$$\frac{\partial E}{\partial c_j} = -2\langle f, \phi_j \rangle + 2 \sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle. \tag{2.13}$$

To minimize $E$, set $\partial E/\partial c_j = 0$ for $j = 0, \ldots, n$, which gives the $n+1$ equations

$$\sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle = \langle f, \phi_j \rangle, \quad j = 0, \ldots, n, \tag{2.14}$$

in the $n+1$ unknowns $c_0, \ldots, c_n$. Since these equations are linear in the unknowns, write them in matrix form:

$$\begin{bmatrix} \langle \phi_0, \phi_0 \rangle & \langle \phi_0, \phi_1 \rangle & \cdots & \langle \phi_0, \phi_n \rangle \\ \langle \phi_1, \phi_0 \rangle & \langle \phi_1, \phi_1 \rangle & \cdots & \langle \phi_1, \phi_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi_n, \phi_0 \rangle & \langle \phi_n, \phi_1 \rangle & \cdots & \langle \phi_n, \phi_n \rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f, \phi_0 \rangle \\ \langle f, \phi_1 \rangle \\ \vdots \\ \langle f, \phi_n \rangle \end{bmatrix}, \tag{2.15}$$

which we denote $\mathbf{G}\mathbf{c} = \mathbf{b}$. The matrix $\mathbf{G}$ is called the *Gram matrix*. Using this matrix-vector notation, we can accumulate the partial derivatives formulas (2.13) for $E$ into the gradient

$$\nabla_{\mathbf{c}} E = 2\big(\mathbf{G}\mathbf{c} - \mathbf{b}\big).$$

Since $\mathbf{c}$ is a critical point if and only if $\nabla_{\mathbf{c}} E(\mathbf{c}) = \mathbf{0}$, we must ask:

- How many critical points are there? Equivalently, how many $\mathbf{c}$ solve $\mathbf{G}\mathbf{c} = \mathbf{b}$?
- If $\mathbf{c}$ is a critical point, is it a (local or even global) minimum?

We will answer the first question by showing that $\mathbf{G}$ is invertible, and hence $E$ has a unique critical point. To answer the second question, we must inspect the Hessian

$$\nabla_{\mathbf{c}}^2 E = \nabla_{\mathbf{c}}(\nabla_{\mathbf{c}} E) = 2\mathbf{G}.$$

The critical point $\mathbf{c}$ is local minimum if and only if the Hessian is *symmetric positive definite*.

A matrix $\mathbf{G}$ is *positive definite* provided $\mathbf{z}^*\mathbf{G}\mathbf{z} > 0$ for all $\mathbf{z} \neq \mathbf{0}$.

The symmetry of the inner product implies $\langle \phi_j, \phi_k \rangle = \langle \phi_k, \phi_j \rangle$, and hence $\mathbf{G}$ is symmetric. (In this case, symmetry also follows from the equivalence of mixed partial deritivates.) The following theorem confirms that $\mathbf{G}$ is indeed positive definite.

# LECTURE 17: *Fundamentals of Least Squares Approximation, Part I*
# LECTURE 18: *Fundamentals of Least Squares Approximation, Part II*

**Theorem 2.7.** If $\phi_0, \ldots, \phi_n$ are linearly independent, the Gram matrix $\mathbf{G}$ is positive definite.

This proof is very general: we are thinking of $\phi_0, \ldots, \phi_n$ being a basis for $\mathcal{P}_n$ (and hence linearly independent), but the same proof applies to any linearly independent set of vectors in a general inner product space.

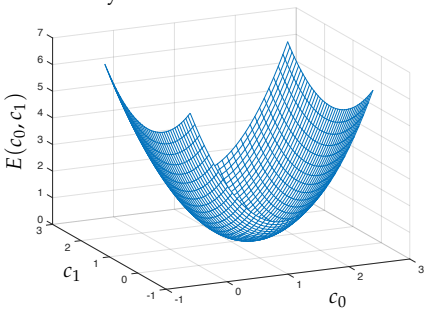
*Proof.* For a generic $\mathbf{z} \in \mathbb{R}^{n+1}$, consider the product

$$\mathbf{z}^*\mathbf{G}\mathbf{z} = \begin{bmatrix} z_0 & z_1 & \cdots & z_n \end{bmatrix} \begin{bmatrix} \langle \phi_0, \phi_0 \rangle & \langle \phi_0, \phi_1 \rangle & \cdots & \langle \phi_0, \phi_n \rangle \\ \langle \phi_1, \phi_0 \rangle & \langle \phi_1, \phi_1 \rangle & \cdots & \langle \phi_1, \phi_n \rangle \\ \vdots & & \ddots & \vdots \\ \langle \phi_n, \phi_0 \rangle & \langle \phi_n, \phi_1 \rangle & \cdots & \langle \phi_n, \phi_n \rangle \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix}$$

$$= \begin{bmatrix} z_0 & z_1 & \cdots & z_n \end{bmatrix} \begin{bmatrix} \sum_{k=0}^n z_j \langle \phi_0, \phi_k \rangle \\ \sum_{k=0}^n z_j \langle \phi_1, \phi_k \rangle \\ \vdots \\ \sum_{k=0}^n z_j \langle \phi_n, \phi_k \rangle \end{bmatrix} = \sum_{j=0}^n \sum_{k=0}^n z_j z_k \langle \phi_j, \phi_k \rangle.$$

*Eigenvalues illuminate.* The surfaces below visualize $E(c_0, c_1)$ for best approximation of $f(x) = e^x$ from $\mathcal{P}_1$ over $x \in [-1, 1]$ (top) and $x \in [0, 1]$.

For $[-1, 1]$, the eigenvalues of $\mathbf{G}$ are relatively large, and the error surface looks very bowl-like.




Eigenvalues of $\mathbf{G}$: $\lambda_1 = 2$, $\lambda_2 = 2/3$

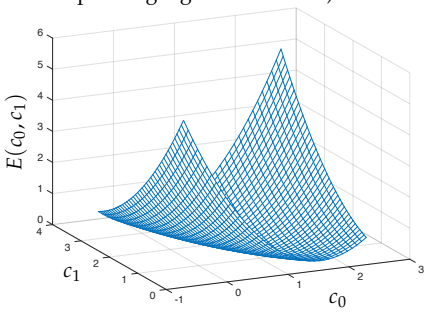
Now use linearity of the inner product to write

$$\mathbf{z}^*\mathbf{G}\mathbf{z} = \sum_{j=0}^n \sum_{k=0}^n z_j z_k \langle \phi_j, \phi_k \rangle = \left\langle \sum_{j=0}^n z_j \phi_j, \sum_{k=0}^n z_k \phi_k \right\rangle = \left\| \sum_{j=0}^n z_j \phi_j \right\|^2.$$

Thus, by nonnegativity of the norm, $\mathbf{z}^*\mathbf{G}\mathbf{z} \geq 0$. This is enough to show that $\mathbf{G}$ is *positive semidefinite*. To show that $\mathbf{G}$ is *positive definite*, we must show that $\mathbf{z}^*\mathbf{G}\mathbf{z} > 0$ if $\mathbf{z} \neq \mathbf{0}$. Now since $\phi_0, \ldots, \phi_n$ are linearly independent, $\sum_{j=0}^n c_j \phi_j = 0$ if and only if $c_0 = \cdots = c_n = 0$, i.e., if and only if $\mathbf{z} = \mathbf{0}$. Thus, if $\mathbf{z} \neq \mathbf{0}$, $\mathbf{z}^*\mathbf{G}\mathbf{z} > 0$. ∎

This answers the second question posed above, and also makes the answer to the first trivial.

For $[0, 1]$, $\mathbf{G}$ has a small eigenvalue: the error surface is much more 'shallow' in one direction. (The orientation of the trough can be found from the corresponding eigenvector of $\mathbf{G}$.)




Eigenvalues of $\mathbf{G}$: $\lambda_1 = (4 + \sqrt{13})/6 = 1.26759\ldots$
$\lambda_2 = (4 - \sqrt{13})/6 = 0.06574\ldots$

**Corollary 2.1.** If $\phi_0, \ldots, \phi_n$ are linearly independent, the Gram matrix $\mathbf{G}$ is invertible.

*Proof.* The matrix $\mathbf{G}$ is invertible if $\mathbf{G}\mathbf{z} = \mathbf{0}$ implies $\mathbf{z} = \mathbf{0}$, i.e., $\mathbf{G}$ has a trivial null space. If $\mathbf{G}\mathbf{z} = \mathbf{0}$, then $\mathbf{z}^*\mathbf{G}\mathbf{z} = 0$. Theorem 2.7 ensures that $\mathbf{G}$ is positive definite, so $\mathbf{z}^*\mathbf{G}\mathbf{z} = 0$ implies $\mathbf{z} = \mathbf{0}$. Hence, $\mathbf{G}$ has a trivial null space, and is thus invertible. ∎

We can summarize our findings as follows.

Given any basis $\phi_0, \ldots, \phi_n$, the least squares approximation $P_*$ to $f \in C[a,b]$ is unique and can be expressed as

$$P_* = \sum_{j=0}^{n} c_j \phi_j,$$

where the coefficients $\mathbf{c}$ are computed as the unique solution of $\mathbf{Gc} = \mathbf{b}$.

As with the interpolation problem studied earlier, different choices of basis will give different linear algebra problems, but ultimately result in the same overall approximation $P_*$. We shall study several choices for the basis in Sections 2.4.5 and 2.4.6. Before doing so, we establish a fundamental property of least squares approximation.

### 2.4.4 *Orthogonality of the error*

This is the fundamental theorem of linear least squares problems:

The error $f - P_*$ is orthogonal to the approximating subspace $\mathcal{P}_n$.

Having worked hard to characterize the optimal approximation, the formula $\mathbf{Gc} = \mathbf{b}$ makes the proof of this result trivial.

---

**Theorem 2.8.** The function $P_* \in \mathcal{P}_n$ is the least squares approximation to $f \in C[a,b]$ if and only if the error $f - P_*$ is orthogonal to the subspace $\mathcal{P}_n$ from which the approximation was drawn:

$$\langle f - P_*, q \rangle = 0, \qquad \text{for all } q \in \mathcal{P}_n.$$

---

*Proof.* First suppose that $P_*$ is the least squares approximation. Thus given any basis $\phi_0, \ldots, \phi_n$ for $\mathcal{P}_n$, we can express $P_* = c_0\phi_0 + \cdots + c_n\phi_n$, where the coefficients solve $\mathbf{Gc} = \mathbf{b}$. Now for any basis function $\phi_j$, use the linearity of the inner product to compute

$$\begin{aligned}\langle f - P_*, \phi_j \rangle &= \langle f, \phi_j \rangle - \langle P_*, \phi_j \rangle \\ &= \langle f, \phi_j \rangle - \Big\langle \sum_{k=0}^{n} c_k \phi_k, \phi_j \Big\rangle = \langle f, \phi_j \rangle - \sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle.\end{aligned}$$

Recall that the $j$th row of the equation $\mathbf{Gc} = \mathbf{b}$ (see (2.14) is precisely

$$\sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle = \langle f, \phi_j \rangle,$$

so since the least squares approximation must satisfy $\mathbf{Gc} = \mathbf{b}$, conclude that $\langle f - P_*, \phi_j \rangle = 0$. Since this orthogonality holds for

$j = 0, \ldots, n$, the error $f - P_*$ is orthogonal to the entire basis for $\mathcal{P}_n$, and hence it is orthogonal to any vector $q = \sum_{j=0}^{n} d_j \phi_j \in \mathcal{P}_n$, since

$$\langle f - P_*, q \rangle = \Big\langle f - P_*, \sum_{j=0}^{n} d_j \phi_j \Big\rangle = \sum_{k=0}^{n} d_j \langle f - P_*, \phi_j \rangle = \sum_{k=0}^{n} d_j \cdot 0 = 0.$$

Thus, the least squares error $f - P_*$ is orthogonal to all $q \in \mathcal{P}_n$.

On the other hand suppose that $p \in \mathcal{P}_n$ gives an error $f - p$ that is orthogonal to all $q \in \mathcal{P}_n$, i.e.,

$$\langle f - p, q \rangle = 0, \qquad \text{for all } q \in \mathcal{P}_n. \tag{2.16}$$

Let $\phi_0, \ldots, \phi_n$ be a basis for $\mathcal{P}_n$. Then we can find $c_0, \ldots, c_n$ so that $p = c_0\phi_0 + \cdots + c_n\phi_n$. The orthogonality of $f - p$ to $\mathcal{P}_n$ in (**??**) implies in particular that $\langle f - p, \phi_j \rangle = 0$ for all $j = 0, \ldots, n$, i.e., using linearity of the inner product,

$$0 = \Big\langle f - \sum_{k=0}^{n} c_k \phi_k, \phi_j \Big\rangle = \langle f, \phi_j \rangle - \sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle$$

for $j = 0, \ldots, n$. Thus orthogonality of the error implies that

$$\sum_{k=0}^{n} c_k \langle \phi_k, \phi_j \rangle = \langle f, \phi_j \rangle, \qquad j = 0, \ldots, n,$$

and these $n + 1$ equations precisely give $\mathbf{G}\mathbf{c} = \mathbf{b}$: since the coefficients of $p$ satisfy the linear system that characterizes the (unique) least squares approximation, $p$ must be that least squares approximation, $p = P_*$. Thus, orthogonality of the error $f - p$ with all $q \in \mathcal{P}_n$ implies that $p$ is the least squares approximation. ∎

### 2.4.5 *Monomial basis*

Suppose we apply this method on the interval $[a, b] = [0, 1]$ with the monomial basis, $\phi_k(x) = x^k$. In that case,

$$\langle \phi_k, \phi_j \rangle = \langle x^k, x^j \rangle = \int_0^1 x^{j+k} \, \mathrm{d}x = \frac{1}{j + k + 1},$$

and the coefficient matrix has an elementary structure. In fact, this is a form of the notorious *Hilbert matrix*. It is exceptionally difficult to obtain accurate solutions with this matrix in floating point arithmetic, reflecting the fact that the monomials are a poor basis for $\mathcal{P}_n$ on $[0, 1]$. Let $\mathbf{G}$ denote the $n + 1$-dimensional Hilbert matrix, and suppose $\mathbf{b}$ is constructed so that the exact solution to the system $\mathbf{G}\mathbf{c} = \mathbf{b}$ is $\mathbf{c} = [1, 1, \ldots, 1]^T$. Let $\widehat{\mathbf{c}}$ denote computed solution to the system in MATLAB. Ideally the forward error $\|\mathbf{c} - \widehat{\mathbf{c}}\|_2$ will be nearly zero (if the rounding errors incurred while constructing $\mathbf{b}$ and solving the

See M.-D. Choi, 'Tricks or treats with the Hilbert matrix,' *American Math. Monthly* 90 (1983) 301–312.

system are small). Unfortunately, this is not the case – the condition number of $\mathbf{G}$ grows exponentially in the dimension $n$, and the accuracy of the computed solution to the linear system quickly degrades as $n$ increases.

| $n$ | $\|\mathbf{G}\|\|\mathbf{G}^{-1}\|$ | $\|\mathbf{c} - \widehat{\mathbf{c}}\|$ |
|---|---|---|
| 5 | $1.495 \times 10^{7}$ | $7.548 \times 10^{-11}$ |
| 10 | $1.603 \times 10^{14}$ | 0.01288 |
| 15 | $4.380 \times 10^{17}$ | 12.61 |
| 20 | $1.251 \times 10^{18}$ | 46.9 |

The last few *condition numbers* $\|\mathbf{G}\|\|\mathbf{G}^{-1}\|$ are in fact *smaller* than they ought to be: MATLAB computes the condition number based as the ratio of the largest to smallest singular values of $\mathbf{G}$; the smallest singular value can only be determined accurately if it is larger than about $\|\mathbf{G}\|\varepsilon_{\text{mach}}$, where $\varepsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$. Thus, if the true condition number is larger than about $1/\varepsilon_{\text{mach}}$, we should not expect MATLAB to compute it accurately.

Clearly these errors are not acceptable!

In summary: *The monomial basis forms an ill-conditioned basis for $\mathcal{P}_n$ over the real interval $[a, b]$.*

### 2.4.6 Orthogonal basis

In the search for a basis for $\mathcal{P}_n$ that will avoid the numerical difficulties, let the structure of the equation $\mathbf{Gc} = \mathbf{b}$ be our guide. What choice of basis would make the matrix $\mathbf{G}$, written out in (2.15), as simple as possible? If the basis vectors are *orthogonal*, i.e.,

Section 2.5 will derive a procedure for computing an orthogonal basis for $\mathcal{P}_n$.

$$\langle \phi_j, \phi_k \rangle \begin{cases} \neq 0, & j = k; \\ = 0, & j \neq k, \end{cases}$$

then $\mathbf{G}$ only has nonzeros on the main *diagonal*, giving the system

$$\begin{bmatrix} \langle \phi_0, \phi_0 \rangle & 0 & \cdots & 0 \\ 0 & \langle \phi_1, \phi_1 \rangle & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \langle \phi_n, \phi_n \rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f, \phi_0 \rangle \\ \langle f, \phi_1 \rangle \\ \vdots \\ \langle f, \phi_n \rangle \end{bmatrix}.$$

This system decouples into $n + 1$ scalar equations $\langle \phi_j, \phi_j \rangle c_j = \langle f, \phi_j \rangle$ for $j = 0, \ldots, n$. Solve these scalar equations to get

$$c_j = \frac{\langle \phi_j, \phi_j \rangle}{\langle f, \phi_j \rangle}, \qquad j = 0, \ldots, n.$$

Thus, with respect to the orthogonal basis the least squares approximation to $f$ is given by

$$P_*(x) = \sum_{j=0}^{n} c_j \phi_j(x) = \sum_{j=0}^{n} \frac{\langle f, \phi_j \rangle}{\langle \phi_j, \phi_j \rangle} \phi_j(x). \tag{2.17}$$

The formula (2.17) has an outstanding property: if we wish to extend approximation from $\mathcal{P}_n$ one degree higher to $\mathcal{P}_{n+1}$, we simply

add in one more term. If we momentarily use the notation $P_{*,k}$ for the least squares approximation from $\mathcal{P}_k$, then

$$P_{*,n+1}(x) = P_{*,n}(x) + \frac{\langle f, \phi_{n+1}\rangle}{\langle \phi_{n+1}, \phi_{n+1}\rangle}\phi_{n+1}(x).$$

In contrast, to increase the degree of the least squares approximation in the monomial basis, one would need to extend the $\mathbf{G}$ matrix by one row and column, and re-solve form $\mathbf{Gc} = \mathbf{b}$: *increasing the degree changes all the old coefficients in the monomial basis.*

An orthogonal basis also permits a beautifully simple formula for the norm of the error, $\|f - P_*\|_2$.

**Theorem 2.9.** Let $\phi_0, \ldots, \phi_n$ denote an orthogonal basis for $\mathcal{P}_n$. Then for any $f \in \mathbb{C}[a,b]$, the norm of the error $f - P_*$ of the least squares approximation $P_* \in \mathcal{P}_n$ is

$$\|f - P_*\|_2 = \sqrt{\|f\|_2^2 - \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle}}. \tag{2.18}$$

This result is closely related to *Parseval's identity*, which essentially says that if $\phi_0, \phi_1, \ldots$ forms an orthogonal basis for the (possibly infinite dimensional) vector space $V$, then for any $f \in V$,

$$\|f\|^2 = \sum_j \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle}.$$

To put the utility of the formula (2.18) in context, think about minimax approximation. We have various bounds, like de la Vallée Poussin's theorem, on the minimax error, but no easy formula exists to give you that error directly.

*Proof.* First, use the formula (2.17) for $P_*$ to compute

$$\begin{aligned}\|P_*\|_2^2 &= \left\langle \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\phi_j, \sum_{k=0}^{n} \frac{\langle f, \phi_k\rangle}{\langle \phi_k, \phi_k\rangle}\phi_k \right\rangle \\ &= \sum_{j=0}^{n}\sum_{k=0}^{n} \frac{\langle f, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\frac{\langle f, \phi_k\rangle}{\langle \phi_k, \phi_k\rangle}\langle \phi_j, \phi_k\rangle,\end{aligned}$$

using linearity of the inner product. Since the basis polynomials are orthogonal, $\langle \phi_j, \phi_k\rangle = 0$ for $j \neq k$, which reduces the double sum to

$$\|P_*\|_2^2 = \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\frac{\langle f, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\langle \phi_j, \phi_j\rangle = \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle}.$$

This calculation simplifies our primary concern:

$$\begin{aligned}\|f - P_*\|_2^2 = \langle f - P_*, f - P_*\rangle &= \langle f, f\rangle - \langle f, P_*\rangle - \langle P_*, f\rangle + \langle P_*, P_*\rangle \\ &= \langle f, f\rangle - 2\langle f, P_*\rangle + \langle P_*, P_*\rangle \\ &= \langle f, f\rangle - 2\left\langle f, \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle}{\langle \phi_j, \phi_j\rangle}\phi_j \right\rangle + \langle P_*, P_*\rangle \\ &= \langle f, f\rangle - 2\sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle} + \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle} \\ &= \|f\|_2^2 - \sum_{j=0}^{n} \frac{\langle f, \phi_j\rangle^2}{\langle \phi_j, \phi_j\rangle},\end{aligned}$$

as required. ∎

### 2.4.7 Coda: Connection to discrete least squares (Optional Section)

Studies of numerical linear algebra inevitably address the *discrete least squares* problem: Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ with $m \geq n$, solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2, \tag{2.19}$$

using the Euclidean norm $\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}^*\mathbf{v}}$. One can show that the minimizing $\mathbf{x}$ solves the linear system

$$\mathbf{A}^*\mathbf{A}\mathbf{x} = \mathbf{A}^*\mathbf{b}, \tag{2.20}$$

which are called the *normal equations*. If $\text{rank}(\mathbf{A}) = n$ (i.e., the columns of $\mathbf{A}$ are linearly independent), then $\mathbf{A}^*\mathbf{A} \in \mathbb{R}^{n\times n}$ is invertible, and

$$\mathbf{x} = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*\mathbf{b}. \tag{2.21}$$

One learns that, for purposes of numerical stability, it is preferable to compute the *QR factorization*

$$\mathbf{A} = \mathbf{Q}\mathbf{R},$$

where the columns of $\mathbf{Q} \in \mathbb{R}^{m\times n}$ are orthonormal, $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$, and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is upper triangular ($r_{j,k} = 0$ if $j > k$) and invertible if the columns of $\mathbf{A}$ are linearly independent. Substituting $\mathbf{QR}$ for $\mathbf{A}$ reduces the solution formula (2.21) to

$$\mathbf{x} = \mathbf{R}^{-1}\mathbf{Q}^*\mathbf{b}. \tag{2.22}$$

How does this "least squares problem" relate to the polynomial approximation problem in this section? We consider two perspectives.

### 2.4.8 Discrete least squares as subspace approximation

Notice that the he problem (2.19) can be viewed as

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 = \min_{\mathbf{v}\in\text{Ran}(\mathbf{A})} \|\mathbf{b} - \mathbf{v}\|_2, \tag{2.23}$$

i.e., the discrete least squares problem seeks to approximate $\mathbf{b}$ with some vector $\mathbf{v} = \mathbf{A}\mathbf{x}$ from the subspace $\text{Ran}(\mathbf{A}) \subset \mathbb{R}^m$. Writing

$\text{Ran}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$ is the range (column space) of $\mathbf{A}$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix}$$

for $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$, we seek

$$\mathbf{v} = \mathbf{A}\mathbf{x} = x_1\mathbf{a}_1 + \cdots x_n\mathbf{a}_n \in \mathbb{R}^m$$

to approximate $\mathbf{b} \in \mathbb{R}^m$.

Viewing $\mathbf{a}_1, \ldots, \mathbf{a}_n$ as a basis for the approximating subspace $\mathrm{Ran}(\mathbf{A})$, one can develop the least squares theory precisely as we have earlier in this section, using the inner product

$$\langle \mathbf{a}_j, \mathbf{a}_k \rangle = \mathbf{a}_k^* \mathbf{a}_j.$$

Minimizing the error function

$$E(x_1, \ldots, x_n) = \|\mathbf{b} - (x_1\mathbf{a}_1 + \cdots + x_n\mathbf{a}_n)\|_2^2$$

with respect to $x_1, \ldots, x_n$ just as in the previous development leads to the Gram matrix problem

$$\begin{bmatrix} \langle \mathbf{a}_1, \mathbf{a}_1 \rangle & \langle \mathbf{a}_1, \mathbf{a}_2 \rangle & \cdots & \langle \mathbf{a}_1, \mathbf{a}_n \rangle \\ \langle \mathbf{a}_2, \mathbf{a}_1 \rangle & \langle \mathbf{a}_2, \mathbf{a}_2 \rangle & \cdots & \langle \mathbf{a}_2, \mathbf{a}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{a}_n, \mathbf{a}_1 \rangle & \langle \mathbf{a}_n, \mathbf{a}_2 \rangle & \cdots & \langle \mathbf{a}_n, \mathbf{a}_n \rangle \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \langle \mathbf{b}, \mathbf{a}_1 \rangle \\ \langle \mathbf{b}, \mathbf{a}_2 \rangle \\ \vdots \\ \langle \mathbf{b}, \mathbf{a}_n \rangle \end{bmatrix}, \tag{2.24}$$

which is a perfect analogue of (2.15). In fact, notice that (2.24) is nothing other than

$$\mathbf{A}^*\mathbf{A}\mathbf{x} = \mathbf{A}^*\mathbf{b},$$

the familiar normal equations! What role does the QR factorization play? The columns of $\mathbf{Q}$ form an orthonormal basis for $\mathrm{Ran}(\mathbf{A})$:

$$\mathrm{Ran}(\mathbf{A}) = \mathrm{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_n\} = \mathrm{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_n\} = \mathrm{Ran}(\mathbf{Q}).$$

So the approximation problem (2.23) is equivalent to

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 &= \min_{\mathbf{v} \in \mathrm{Ran}(\mathbf{Q})} \|\mathbf{b} - \mathbf{v}\|_2 \\ &= \min_{c_1, \ldots, c_n} \|\mathbf{b} - (c_1\mathbf{q}_1 + \cdots + c_n\mathbf{q}_n)\|_2. \end{aligned}$$

The Gram matrix system with respect to this basis is

$$\begin{bmatrix} \langle \mathbf{q}_1, \mathbf{q}_1 \rangle & \langle \mathbf{q}_1, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{q}_1, \mathbf{q}_n \rangle \\ \langle \mathbf{q}_2, \mathbf{q}_1 \rangle & \langle \mathbf{q}_2, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{q}_2, \mathbf{q}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{q}_n, \mathbf{q}_1 \rangle & \langle \mathbf{q}_n, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{q}_n, \mathbf{q}_n \rangle \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle \mathbf{b}, \mathbf{q}_1 \rangle \\ \langle \mathbf{b}, \mathbf{q}_2 \rangle \\ \vdots \\ \langle \mathbf{b}, \mathbf{q}_n \rangle \end{bmatrix}. \tag{2.25}$$

The orthonormality of the vectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$ means that $\mathbf{q}_j^*\mathbf{q}_k = 0$ if $j \neq k$ and $\mathbf{q}_j^*\mathbf{q}_j = \|\mathbf{q}_j\|_2^2 = 1$, and so the matrix in (2.25) is the identity. Hence

$$c_j = \langle \mathbf{b}, \mathbf{q}_j \rangle,$$

so we can write

$$\mathbf{c} = \mathbf{Q}^*\mathbf{b}.$$

The approximation $\mathbf{v}$ to $\mathbf{b}$ is then

$$\mathbf{v} = c_1\mathbf{q}_1 + \cdots + c_n\mathbf{q}_n = \mathbf{Q}\mathbf{c} = \mathbf{Q}\mathbf{Q}^*\mathbf{b}.$$

Now using the fact that $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$,

$$\begin{aligned}\mathbf{Q}\mathbf{Q}^* &= (\mathbf{A}\mathbf{R}^{-1})(\mathbf{A}\mathbf{R}^{-1})^* \\ &= \mathbf{A}(\mathbf{R}^*\mathbf{R})^{-1}\mathbf{A}^* = \mathbf{A}(\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*.\end{aligned}$$

Thus the least squares approximation to $\mathbf{b}$ is

$$\mathbf{v} = \mathbf{Q}\mathbf{Q}^*\mathbf{b} = \mathbf{A}\big((\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*\mathbf{b}\big) = \mathbf{A}\mathbf{x},$$

where $\mathbf{x}$ solves the original least squares problem (2.19).

Thus, the orthogonal basis for the approximating space $\mathrm{Ran}(\mathbf{A})$ leads to an easy formula for the approximation, in just the same fashion that orthogonal polynomials made quick work of the polynomial least squares problem in (2.17).

### 2.4.9 *Discrete least squares for polynomial approximation*

Now we turn the tables for another view of the connection between the the polynomial approximation problem and the matrix least squares problem (2.19).

Suppose we only know how to solve discrete least squares problems like (2.19), and want to use that technology to construct some polynomial $p \in P_n$ that approximates $f \in C[a,b]$ over $x \in [a,b]$.

We could sample $f$ at, say, $m+1$ discrete points $x_0, \ldots, x_m$ uniformly distributed over $[a,b]$: set $h_m := (b-a)/m$ and let

$$x_k = a + kh_m.$$

We then want to solve

$$\min_{p\in\mathcal{P}_n} \sum_{j=0}^{m} |f(x_j) - p(x_j)|^2. \tag{2.26}$$

This least squares error, when scaled by $h_m$, takes the form of a Riemann sum that, in the $m \to \infty$ limit, approximates an integral:

$$\lim_{m\to\infty}\ h_m \sum_{k=0}^{m} (f(x_k) - p(x_k))^2 = \int_a^b (f(x) - p(x))^2\,\mathrm{d}x.$$

That is, as we take more and more approximation points, the error (2.26) that we are minimizing better and better approximates the integral error formulation (2.12).

To solve (2.26), represent $p \in \mathcal{P}_n$ using the monomial basis,

$$p(x) = c_0 + c_1 x + \cdots + c_n x^n.$$

Then write (2.26) as

$$\min_{c_0,\ldots,c_n} \|\mathbf{f} - \mathbf{A}\mathbf{c}\|_2^2,$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix}.$$

This discrete problem can be solved via the *normal equations*, i.e., find $\mathbf{c} \in \mathbb{R}^{n+1}$ to solve the matrix equation

$$\mathbf{A}^*\mathbf{A}\mathbf{c} = \mathbf{A}^*\mathbf{f}.$$

Compute the right-hand side as

$$\mathbf{A}^*\mathbf{f} = \begin{bmatrix} \sum_{k=0}^n f(x_k) \\ \sum_{k=0}^n x_k f(x_k) \\ \sum_{k=0}^n x_k^2 f(x_k) \\ \vdots \\ \sum_{k=0}^n x_k^n f(x_k) \end{bmatrix} \in \mathbb{R}^{n+1}.$$

Notice that if $m+1$ approximation points are uniformly spaced over $[a,b]$, $x_k = a + kh_m$ for $h_m = (b-a)/m$, then

$$\lim_{m\to\infty} h_m \mathbf{A}^*\mathbf{f} = \begin{bmatrix} \int_a^b f(x)\,\mathrm{d}x \\ \int_a^b x f(x)\,\mathrm{d}x \\ \int_a^b x^2 f(x)\,\mathrm{d}x \\ \vdots \\ \int_a^b x^n f(x)\,\mathrm{d}x \end{bmatrix} = \begin{bmatrix} \langle f, 1\rangle \\ \langle f, x\rangle \\ \langle f, x^2\rangle \\ \vdots \\ \langle f, x^n\rangle \end{bmatrix},$$

which is precisely the right hand side vector $\mathbf{b} \in \mathbb{R}^{n+1}$ obtained for the original least squares problem at the beginning of this section in (2.15). Similarly, the $(j+1, k+1)$ entry of $\mathbf{A}^*\mathbf{A} \in \mathbb{R}^{(n+1)\times(n+1)}$ for the discrete problem can be formed as

$$(\mathbf{A}^*\mathbf{A})_{j+1,k+1} = \sum_{\ell=0}^m x_\ell^j x_\ell^k = \sum_{\ell=0}^m x_\ell^{j+k},$$

and thus for uniformly spaced approximation points,

$$\lim_{m\to\infty} h_m (\mathbf{A}^*\mathbf{A})_{j+1,k+1} = \int_a^b x^{j+k}\,\mathrm{d}x = \langle x^j, x^k\rangle.$$

Thus in aggregate we have

$$\lim_{m\to\infty} h_m \mathbf{A}^*\mathbf{A} = \mathbf{G},$$

where $\mathbf{G}$ is the same Gram matrix in (2.15).

We arrive at the following beautiful conclusion: The normal equations $\mathbf{A}^*\mathbf{A}\mathbf{c} = \mathbf{A}^*\mathbf{f}$ formed for polynomial approximation by *discrete least squares* converges to *exactly the same* $(n+1)\times(n+1)$ *system* $\mathbf{G}\mathbf{c} = \mathbf{b}$ that we independently derived for the polynomial approximation problem (2.12) with the integral form of the error.