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

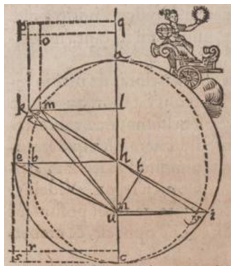

Image from Johannes Kepler's *Astronomia nova*, 1609, (ETH Bibliothek). In this text Kepler derives his famous equation that solves two-body orbital motion,

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

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 *Basic 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 $\{x_j\}_{j=0}^n$, $a \le x_0 < x_1 < \cdots < x_n \le b$, 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.

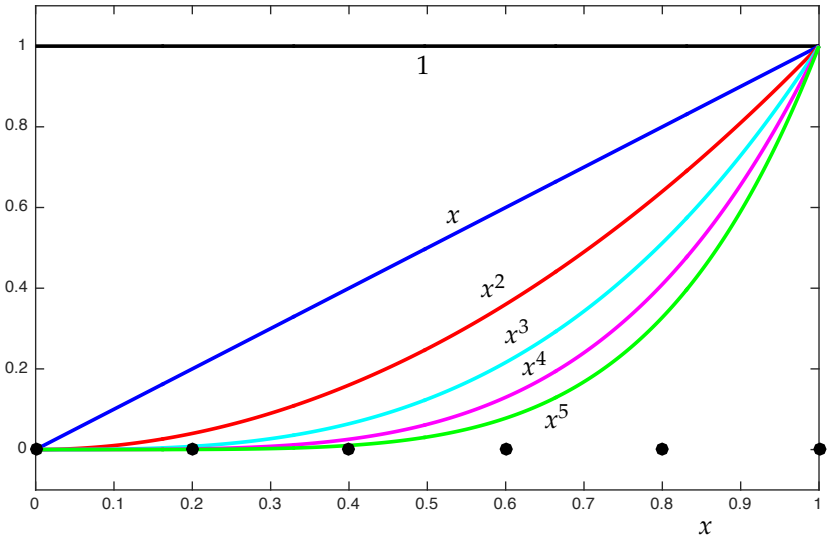

Figure 1.1: The six monomial basis vectors for $\mathcal{P}_5$, based on the interval $[a, b] = [0, 1]$ with $x_j = j/5$ (red circles). Note that the basis vectors increasingly align as the power increases: this basis becomes *ill-conditioned* as the degree of the interpolant grows.

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

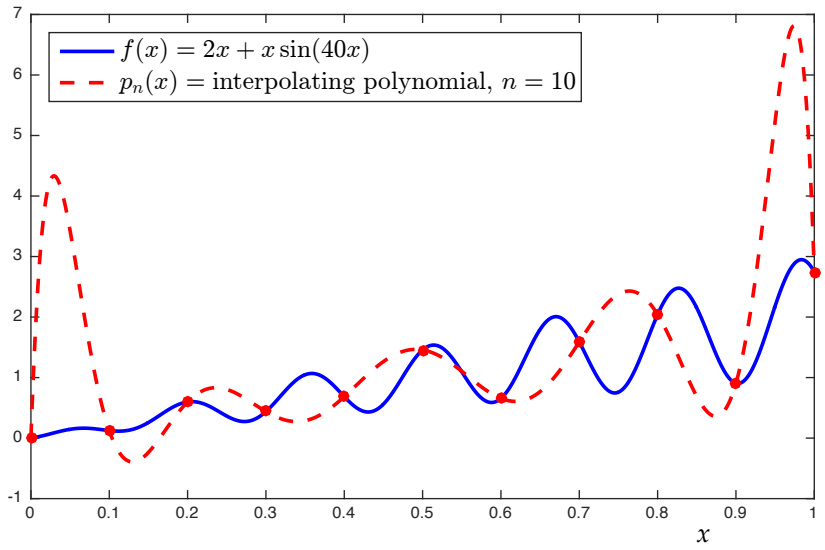

Figure 1.2: Degree $n = 10$ interpolant $p_{10}(x)$ to $f(x) = 2x + x\sin(40x)$ at the uniformly spaced points $x_0, \ldots, x_{10}$ for $x_j = j/10$ over $[a, b] = [0, 1]$. Even though $p_{10}(x_j) = f(x_j)$ at the eleven points $x_0, \ldots, x_n$ (red circles), the interpolant gives a poor approximation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

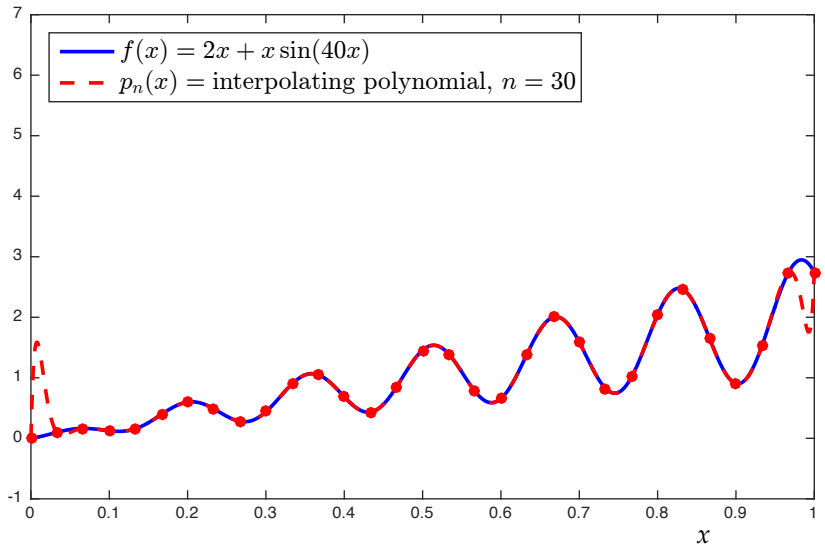

Figure 1.3: Repetition of Figure 1.2, but now with the degree $n = 30$ interpolant at uniformly spaced points $x_j = j/30$ on $[0, 1]$. The polynomial still overshoots f near $x = 0$ and $x = 1$, though by less than for $n = 10$; for this example, the overshoot goes away as n is increased further.

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.

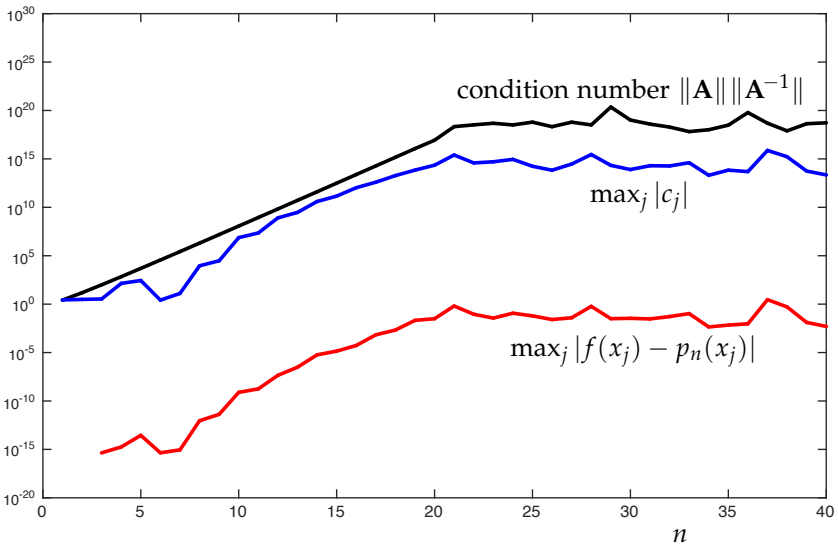

Figure 1.4: Illustration of some pitfalls of working with interpolants in the monomial basis for large n: (a) the condition number of $\mathbf{A}$ grows large with n; (b) as a result, some coefficients c_j are large in magnitude (blue line) and inaccurately computed; (c) consequently, the computed 'interpolant'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

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.