Interpolation

1 Preamble

In engineering analysis we must often predict the behavior of a univariate function — a relation between an input (independent variable) and an output (dependent variable) — in order to design or optimize a system; for example, we may wish to understand the effect of temperature on the speed of sound in the design of an acoustical space. Unfortunately, in most cases we are not provided with a closed-form representation of the function, and instead we must base our prediction on limited (experimental) observations or (computational) evaluations. In this nutshell, we answer the following question: if we know output values at some finite number of input values — a "look-up table" — how can we estimate the output for inputs "in between" the input values of our look-up table? We consider here one approach to this approximation problem: *interpolation*. (In a subsequent nutshell we consider an alternative approach, more suitable for noisy data: least-squares projection.)

We introduce in this nutshell the interpolation framework, and we present particular interpolation schemes: piecewise-constant, left-endpoint; piecewise-linear. We then develop the concepts of discretization, convergence, convergence rate or order, and resolution; we provide local and global error bounds based on the Mean Value Theorem and Fundamental Theorem of Calculus; we discuss and demonstrate the role of smoothness; we introduce the definition of operation count and FLOPs (floating point operations); and we establish the connection between accuracy and computational cost. We provide both a global domain formulation and also a more local "point-based" formulation.

Prerequisites: univariate differential and integral calculus.

2 Motivation: Example

Let us consider a concrete example of interpolation. In Table 1, we are given the speed of sound in air for several different values of temperatures ranging from 230K (or −43°C) — cold air at high altitude — to 310K (or 37°C) — hot air at sea level on a summer day. We now wish to approximate the speed of sound for some given outside temperature, and in particular a temperature *not* represented in Table 1. Here, the temperature is the *independent variable* (also denoted *input*)

— what we are given — and the speed of sound is the *dependent variable* (also denoted *output*) — what we wish to know.

How might we estimate the speed of sound for an outside temperature of, say, 283K (10°C)? Can we say anything about the accuracy of our estimate? What is the computational cost associated with the construction of our estimate? The material provided in this nutshell will help you answer these questions, not just for our particular example, but also more generally for any input–output relationship.

temperature (K)	230	240	250	260	270	280	290	300	310
speed of sound (m/s)	304.0	310.5	316.9	323.2	329.4	335.4	341.4	347.2	352.9

Table 1: Variation in the speed of sound in air as a function of the temperature.[1]

3 Piecewise-Constant, Left-Endpoint Interpolation

We first consider arguably the simplest form of interpolation: piecewise-constant, left-endpoint interpolation. The origin of the name will become clear shortly.

Let us first describe the procedure in words for several concrete instances. We wish to approximate the speed of sound at a given temperature of interest, say 283K. We first find the temperature in Table 1 which is closest to 283K but also smaller than (or less than or equal to) 283K: 280K. We next extract from Table 1 the speed of sound associated with the closest-but-smaller temperature, 280K: 335.4m/s. Finally, we now approximate, or estimate, the speed of sound at the temperature of interest, 283K, by the speed of sound associated with the closest-but-smaller temperature, 280K: 335.4m/s. We consider a second example: we are given the temperature of interest, 248K; we find in Table 1 the closest-but-smaller temperature, 240K; we extract from Table 1 the speed of sound associated with the closest-but-smaller temperature, 310.5m/s; we approximate the speed of sound at 248K by 310.5m/s. Note that our estimate depends on both the temperature of interest and the particular set of (temperature, speed of sound) pairs available — the entries of Table 1.

We now formalize the procedure. We shall denote the temperature, our independent variable, as x, and the speed of sound, our dependent variable, as y. We further denote the mapping from the temperature to the speed of sound as $y = f(x)$: f is the function which, given a value of the temperature x, returns the value of the speed of sound, $y = f(x)$. We assume here that f exists but that we do not know the form of f; rather, we only know the values of $y = f(x)$, or equivalently the pairs $(x, f(x))$, for the nine temperatures indicated in Table 1. The methodology we develop in terms of x and $f(x)$ will of course apply to any independent variable (input) and dependent variable (output) — hence the power of abstraction — though you may think of temperature and speed of sound as a concrete application.

Let us say that we are interested in input values x in some interval $[a, b]$. (We recall that $[a, b]$ refers to the *closed interval* — values of the temperature x such that $a \leq x \leq b$. The *open interval* (a, b) refers to x such that $a < x < b$; we may also consider various combinations such as $(a, b]$, which refers to x such that $a < x \leq b$.) We first introduce the discretization depicted in Figure 1. A *discretization*, or "grid," is a decomposition of the interval $[a, b]$ into a number of small pieces for the purposes of construction and analysis of our approximation. In our particular case, we subdivide

[1]The "data" in this Table are generated synthetically from the standard sound-speed relation introduced in **CYAWTP 2**.

the domain $[a, b]$ into $N - 1$ segments, $S_i, 1 \leq i \leq N - 1$, delineated by N points, $x_i, 1 \leq i \leq N$. Note that segment S_i is given by the interval $[x_i, x_{i+1}]$ and is of length $h_i \equiv x_{i+1} - x_i$; for simplicity, we shall assume here that the segments are of equal length, $h_1 \equiv \cdots \equiv h_{N-1} \equiv h \equiv (b-a)/(N-1)$, however in general this need not be the case. In our speed of sound example, for the discretization provided in Table 1, $a \equiv 230$K, $b \equiv 310$K; $N \equiv 9$; $x_1 \equiv 230$K, $x_2 \equiv 240$K, $x_3 \equiv 250$K, $\ldots$; $S_1 \equiv [x_1, x_2] \equiv [230, 240]$K, $S_2 \equiv [x_2, x_3] \equiv [240, 250]$K, $\ldots$; $h \equiv 10$K. Note that the notation "$\equiv$" should be read as "is defined to be" or "is given as".

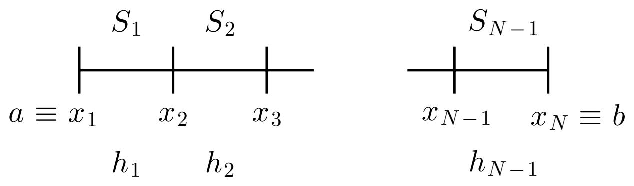

Figure 1: Discretization of the segment $[a, b]$ into $N - 1$ segments defined by N input values.

We now wish to approximate the function $f(x)$ over the continuous interval $[a, b]$ given only the data pairs $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_N, f(x_N))$ associated with our discretization; we may think of these N pairs of data as a "look-up table." We shall pursue a particular kind of approximation known as *interpolation*. In general, an interpolation scheme is characterized by two ingredients: the "what" — the functional form of the interpolant; the "where" — the (interpolation) points at which we shall require $(\mathcal{I}f)(x) = f(x)$. This latter condition is the distinguishing feature of interpolation: our approximation $(\mathcal{I}f)(x)$ must "go through" $f(x)$ — agree with the exact function $f(x)$ — at prescribed values of the independent variable. Note that we must choose an appropriate combination of "what" and "where" such that a unique interpolant can be found.

We now construct the piecewise-constant, left-endpoint interpolant – a particular choice of interpolation scheme. We first focus our attention on interpolation of the data over a single segment S_i. In particular, let us say that the input value of interest — the value of x at which we wish to approximate $y = f(x)$ — resides in interval S_i (for some i). We now choose for the "what" a constant function, and for the "where" the left endpoint of S_i, as depicted in Figure 2; the latter demands that $(\mathcal{I}f)(x_i) = f(x_i)$, but from the former, $\mathcal{I}f$ is constant, and hence $(\mathcal{I}f)(x) = f(x_i)$ for all x in S_i. We thus arrive at a very simple expression for our interpolant over S_i,

$$(\mathcal{I}f)(x) = f(x_i) \quad \text{for all } x \text{ in } S_i \; . \tag{1}$$

Note that, a constant function (the "what") has one degree of freedom, and the agreement between $\mathcal{I}f$ and f at the left-endpoint (the "where") constitutes a single condition, or equation; hence we have one equation in one unknown and we may readily find the unique interpolant.

Although we have focused on a particular segment S_i, our relation (1) is valid for any segment S_i, $i = 1, \ldots, N-1$, and we can thus now estimate $f(x)$ anywhere on the interval $[a, b]$: given any x in $[a, b]$, we first identify the interval in which x resides, S_{i^*} (note that i^* depends on x); we then apply our formula (1) for $x_i \equiv x_{i^*}$ to obtain $(\mathcal{I}f)(x) = f(x_{i^*})$. We depict the interpolant over the entire interval in Figure 3, from which we now understand the genesis of the name "piecewise-constant, left-endpoint": the "piecewise-constant" refers to the "what" (now from a global formulation), and the "left-endpoint" refers to the "where" (in reference to the underlying discretization).

Finally, we may demonstrate that our more mathematical construction here is equivalent to the simple description in words provided at the outset of this section. It suffices to note that, for any

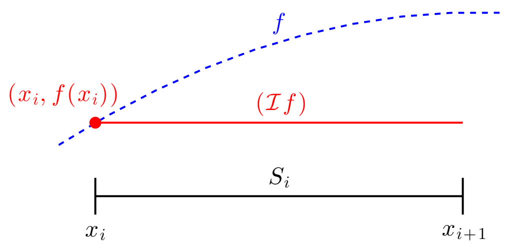

Figure 2: Illustration of piecewise-constant left-endpoint interpolation over the segment S_i.

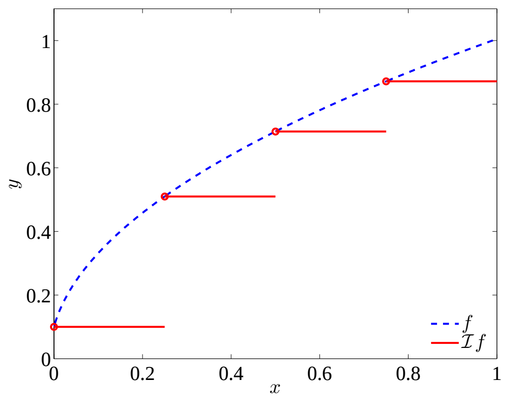

Figure 3: Illustration of the piecewise-constant left-endpoint interpolant over the full interval $[a \equiv 0,\ b \equiv 1]$.

point x in $S_{i^*} \equiv [x_{i^*}, x_{i^*+1}]$, the closest-but-smaller input in our "look-up table" is x_{i^*}, and the output in our look-up table associated to x_{i^*} is $f(x_{i^*})$ — precisely as reproduced by (1).

CYAWTP 1. Evaluate the speed of sound by piecewise-constant, left-endpoint interpolation of the (temperature, speed of sound) data of Table 1 at temperatures of 257K and 291K, respectively.

We now wish to understand how well our interpolant $\mathcal{I}f$ approximates f. This is an exercise in *error analysis*. We again focus our attention to a segment S_i. We first recall that, from the fundamental theorem of calculus,

$$f(x) - f(x_i) = \int_{\xi = x_i}^{x} f'(\xi) d\xi,$$

presuming that the derivative $f'(\xi)$ exists (in an appropriate sense). We can then readily show that, for any x in S_i,

$$\begin{aligned}
|f(x) - (\mathcal{I}f)(x)| &= |f(x) - f(x_i)| && \text{(by the definition of the interpolant)}\\
&= |\int_{x_i}^{x} f'(\xi) d\xi| && \text{(fundamental theorem of calculus)}\\
&\le \int_{x_i}^{x} |f'(\xi)| d\xi \\
&\le \max_{\xi \in S_i} |f'(\xi)| \int_{x_i}^{x} d\xi \\
&\le h \max_{\xi \in S_i} |f'(\xi)| && \text{(by the definition of } h \equiv \int_{x_i}^{x_{i+1}} d\xi).
\end{aligned}$$

We conclude that the *local interpolation error*, e_i, over the interval S_i satisfies

$$e_i \equiv \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| \le h \max_{x \in S_i} |f'(x)| \ . \tag{2}$$

(We recall that "$z \in Z$" means that the variable z is in the set Z. We also recall that "$\max_{z \in Z} G(z)$" for some variable z and any function $G(z)$ asks us to find the maximum value of $G(z)$ for all values of z in the set Z. For example, $\max_{x \in S_i} |f'(x)|$ is the maximum of the absolute value of f' over all values of x in the segment S_i.) Note that the interpolation error depends on two factors. The first is the discretization resolution parameter h: the shorter the segment, the better the estimate. The second is the first derivative of the underlying function f: the slower the rate of the change of the function, the better the estimate. These results are consistent with our intuition suggested by a Taylor series expansion. The error bound (2) is also *sharp*: there exists a function for which the bound statement holds with equality.

We now study the behavior of the *(global) interpolation error* over the entire interval $[a, b]$. We appeal to the local interpolation error bound to obtain

$$\begin{aligned}
e_{\max} \equiv \max_{x \in [a,b]} |f(x) - (\mathcal{I}f)(x)| &= \max_{i \in \{1,\ldots,N-1\}} \max_{x \in S_i} |f(x) - (\mathcal{I}f)(x)| \le \max_{i \in \{1,\ldots,N-1\}} e_i \\
&= \max_{i \in \{1,\ldots,N-1\}} h \max_{x \in S_i} |f'(x)| \le h \max_{x \in [a,b]} |f'(x)|.
\end{aligned}$$

(We recall that $\{1, \ldots, N-1\}$ refers to the set of integers from 1 to $N-1$.) More concisely,

$$e_{\max} \le Ch \qquad \text{for} \qquad C = \max_{x \in [a,b]} |f'(x)|. \tag{3}$$

The global error bound consists again of two contributions: the discretization resolution parameter h; a constant C which characterizes the underlying function f — in the case of piecewise-constant, left-endpoint interpolation, C depends on the first derivative of f. Note that C is independent of h.

Given some x, we may find i^* such that x resides in S_{i^*}. We may then bound $|f(x) - (\mathcal{I}f)(x)|$ by e_{i^*} of (2) or $e_{\max}$ of (3). In general, the local bound of e_{i^*} will be sharper — closer to the true error — than the global bound, since the local bound reflects the derivative in the vicinity of x, and not the worst case over the entire interval $[a, b]$. These error bounds can serve several purposes, often quite pragmatic, as we will describe shortly.

CYAWTP 2. Laplace first derived — correcting Newton — that (for air) at temperature x the speed of sound can be very accurately predicted by $f(x) = \sqrt{\gamma R x}$; here $\gamma = 1.4$ is the ratio of specific heats, $R = 287\text{J}/(\text{kg}\cdot\text{K})$ is the specific gas constant of air, and x is the temperature in Kelvins. It follows that $f'(x) = \sqrt{\gamma R}/(2\sqrt{x})$. On the basis of this derivative information (admittedly not often available in practice, since f is typically not known), evaluate the local error bound (2) for temperatures 257K and 291K, respectively, and the global error bound (3), associated with piecewise-constant, left-endpoint interpolation of the data of Table 1 over the interval $[230, 310]$.

CYAWTP 3. Find a function f (preferably *not* the constant function) such that the maximum difference between $f(x)$ and $(\mathcal{I}f)(x)$ over an interval $[0, 1]$ is given precisely by the maximum of $|f'(x)|$ over $[0, 1]$ — as predicted by our *bound* of (3). (Recall that because such a function exists, we say the bound (3) is sharp.)

Numerical Experiment 4. Invoke the interpolation GUI for the function $f(x) = \sin(x)$ and piecewise-constant, left-endpoint interpolation: visualize the geometric interpretation of the local interpolation error; confirm empirically the decrease of the global interpolation error as h decreases.

Numerical Experiment 5. Invoke the interpolation GUI for the function obtained in **CYAWTP 3** and confirm empirically that the error bound holds with equality.

We now generalize the result of the error analysis. Very often in the analysis of numerical approximations we encounter error bounds of the form

$$e_{\max} \le C h^p \quad \text{for } C \text{ independent of } h.$$

We say a scheme is *convergent* if $e_{\max} \to 0$ as $h \to 0$; for the form above, *convergence* is achieved for any $p > 0$. *Convergence* is an attractive property because it tells us that, as $h \to 0$, the scheme can *eventually* replicate the exact solution to any desired accuracy. (Actually, not quite true: we discuss the issue of finite-precision arithmetic and round-off errors in a subsequent nutshell.) The *convergence rate*, which tells us how *fast* the scheme converges in the limit of $h \to 0$, is reflected in the order p. If $p = 1$, we say the scheme is *first-order* accurate; for h sufficiently small, reduction of h by a factor of two will reduce the error by a factor of *two*. The piecewise-constant, left-endpoint interpolation is an example of a first-order scheme. If $p = 2$, we say the scheme is *second-order* accurate; for h sufficiently small, reduction of h by a factor of two will reduce the error by a factor of *four*. We shall shortly introduce an example of a second-order scheme.

We may also empirically characterize the convergence behavior of schemes. For instance, Figure 4 shows the convergence behavior of the piecewise-constant, left-endpoint interpolant for the function $f(x) = \sin(10\pi x)$ over $x \in [a \equiv 0, b \equiv 1]$. We present two results, both plotted on a log-log scale. The first result is the *logarithmic convergence curve*, which is the (non-straight) curve of the logarithm of the error, $\log_{10}(e_{\max})$, plotted against the logarithm of the number of segments, $\log_{10}(1/h)$. The second result is the *logarithmic convergence asymptote*, which is the (straight-line) asymptote of the logarithmic convergence curve for h tends to zero (hence $1/h \equiv N - 1$ tends to infinity). Recall that we say a function $A(z)$ asymptotes to a function $B(z)$ as $z \to s$ if the ratio of the two functions approaches unity: $\lim_{z\to s} \frac{A(z)}{B(z)} \to 1$; in our context, s is typically either 0 or ∞. We write $A(z)$ asymptotes to $B(z)$ as $z \to s$ more succinctly as $A(z) \sim B(z)$ as $z \to s$. Our logarithmic convergence asymptote is of the form

$$\log_{10}(e_{\max}) \sim \log_{10}(C_{\text{asymp}}) - p \times \log_{10}(1/h) \quad \text{as} \quad h \to 0;$$

here, $\log_{10}(C_{\text{asymp}})$ is the intercept of the logarithmic convergence asymptote (typically $C_{\text{asymp}} = C$ of our bound), p is the slope of the logarithmic convergence asymptote (and the order of the scheme, under certain smoothness assumptions), and $\log_{10}(1/h)$ is the logarithm of the number of segments.

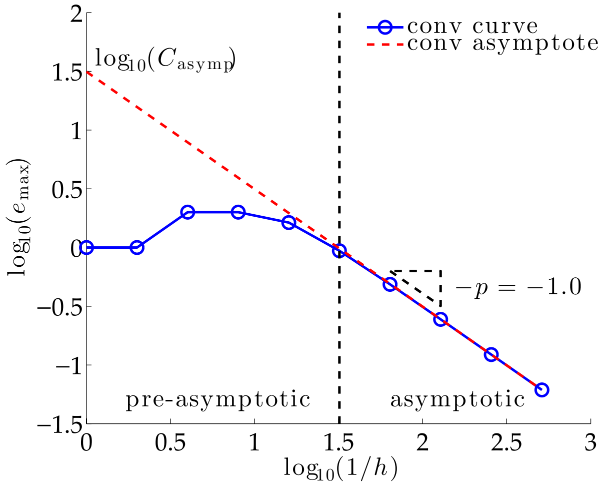

Figure 4: Convergence of the piecewise-constant, left-endpoint interpolation for $f(x) = \sin(10\pi x)$ over the interval $x \in [0, 1]$.

CYAWTP 6. Consider functions

$$\begin{aligned}
g_1(z) &= z^3 + z^2 + \sqrt{z}\\
g_2(z) &= 3z^2 + 3\\
g_3(z) &= \cos(3z) + \sin(z)\\
g_4(z) &= \sqrt{z^4 + 3z^3 + z + 1}.
\end{aligned}$$

Which functions are asymptotic to z^2 as $z \to \infty$?

The length of the segments, h, is known as the discretization parameter: from our assumption of equispaced points, the value of h $(= (b - a)/N$, for some integer $N)$ suffices to specify the segment endpoints and the segments — in short, the complete discretization. We say, when we decrease h, that we *refine* the discretization (or grid). In practice — to obtain the desired accuracy, or to construct our logarithmic convergence curve — we consider a particular *refinement strategy*. For example, the Interpolation GUI implements a "doubling" uniform refinement strategy (for the interval $[a, b] = [0, 1]$): we choose the discretization parameters as $h = 1$, $h = 1/2$, $h = (1/2)^2$, $h = (1/2)^3, \ldots$ such that at each successive refinement we double the number of segments. A sequence of discretizations in which h is decreased only slightly from level to level, $h = 1, h = 1/2, h = 1/3, h = 1/4, h = 1/5, \ldots$, is not of much interest: the decrease in the error from level to level will be very modest.

We present in Figure 4 the logarithmic convergence curve and logarithmic convergence asymptote for piecewise-constant, left-endpoint interpolation of $f(x) = \sin(10\pi x)$. (Although we draw a suggestive curve, we only calculate $e_{\max}$ for values of the discretization parameter h corresponding to a doubling uniform refinement strategy: $h = 1$, $h = 1/2$, $h = (1/2)^2, \ldots$.) Note that the intercept of the logarithmic convergence asymptote is $\log_{10}(C_{\text{asymp}}) = \log_{10}(C) = \log_{10}(\max_{x\in[0,1]} |f'(x)|) = \log_{10}(10\pi)$. We observe that there are two qualitatively different regimes in the convergence behavior: the pre-asymptotic regime in which the error does not behave in a predictable manner with h, and indeed does not decrease in any systematic fashion; the asymptotic regime in which the convergence curve closely tracks the convergence asymptote — and realizes the predicted convergence rate of $p = 1$ (as expected, since $\sin(10\pi x)$ is smooth, even if wiggly relative to the length of the interval $[a, b]$). The *resolution requirement* — the value of the discretization parameter h required to enter the asymptotic regime of convergence — is problem dependent.

CYAWTP 7. Sketch the logarithmic convergence curve, $\log_{10}(e_{\max})$ vs $\log_{10}(1/h)$, and the corresponding logarithmic convergence asymptote associated with piecewise-constant, left-endpoint interpolation of (a) the function $f_1(x) = \sin(\pi x)$, and (b) the function $f_2(x) = \sin(10\pi x)$, in both cases over the interval $[0, 1]$. How are the slopes of the two logarithmic convergence asymptotes related? How are the intercepts $\log_{10}(C_{\text{asymp}})$ of the two logarithmic convergence asymptotes related?

Numerical Experiment 8. Invoke the Interpolation GUI to confirm your sketches of **CYAWTP 7**.

We now consider the cost associated with piecewise-constant left-endpoint interpolation. Towards this end, we introduce the notion of *floating point operations*. A floating point operation is an arithmetic operation: an addition, subtraction, multiplication, or division. We abbreviate the term floating point operation as *FLOP* (<u>FL</u>oating-point <u>OP</u>eration) and in plural as *FLOPs* (<u>FL</u>oating-point <u>OP</u>eration<u>s</u>). (Note the lowercase s; with a capital final letter, FLOPS refers to the speed of a computer as measured by the number of FLOPs which may be performed in a second.) We characterize the cost of a particular computational task by the *operation count*: the number of floating point operations — FLOPs — required to perform the necessary operations.[2] As a very simple example, the computation of $2 + 3 \times 4$ requires 2 FLOPs: a multiplication and an addition.

Most often we are much less concerned by factors of two or three and much more concerned with the dependence of the operation count on our discretization parameter, h, or equivalently the number of degrees of freedom, N. For that reason, operation counts are often provided in big-O notation. We say "Task A requires $\mathcal{O}(g(K))$" operations if the operation count associated with Task A, as measured in FLOPs, is bounded by $cg(K)$ as $K \to \infty$ for some finite c independent of K. By this construction, we obtain, for example, $\mathcal{O}(K^2 + K) = \mathcal{O}(K^2)$, $\mathcal{O}(100K^2 + 0.01K^3) = \mathcal{O}(K^3)$, $\mathcal{O}(10K^5 + K!) = \mathcal{O}(K!)$ (where ! denotes factorial). The big-O notation is convenient for the characterization of the predominant cost associated with a particular task as the problem size — measured by K (in our case, N) — grows. Note big-O is not unique, in that $10K^2$ is $\mathcal{O}(10K^2)$ but also $\mathcal{O}(K^2)$ and even $\mathcal{O}(5K^2)$; the convention is to choose unity as the multiplicative factor.

CYAWTP 9. Given a set of K numbers $\{x_1, \ldots, x_K\}$, consider the computation of $\sum_{k=1}^{K}(x_k^2 + x_k)$. What is the precise operation count, measured in FLOPs, as a function of K? What is the operation count expressed in big-O notation as $K \to \infty$?

[2]In practice, the number of floating point operations will not directly predict "wall clock" time — the time to perform the computations. There are various pieces of overhead, and in particular the process by which data (numbers) are brought into the arithmetic units — access to storage — plays a large role. Modern computer architectures mitigate many of the impediments to rapid computation, for example through a hierarchy of memory.

In the context of interpolation, we are interested in understanding how the cost varies with the number of data points N. We consider two different types of cost. First is the *Offline cost*, which is associated with the construction of our data $(x_1, f(x_1)), \ldots, (x_N, f(x_N))$. To construct this look-up table we must evaluate the function f for N different input values, $x_1, \ldots, x_N$; each evaluation — which might entail collecting experimental data or performing a large numerical simulation — can be very expensive. This Offline cost is $\mathcal{O}(N)$.

Second is the *Online cost*, which is associated with the construction of the interpolation estimate based on the look-up table constructed in the Offline stage. Given x, we must first locate the closest-but-smaller input value in our table, x_{i^*}. If the values of the input are equispaced, $h_i = h$, $i = 1, \ldots, N-1$, then $i^* = \text{floor}((x-a)/h) + 1$, a calculation which we can perform in $\mathcal{O}(1)$ operations; here floor() rounds down the argument to the nearest integer. If the data is not equispaced, then we can find i^* by binary search in $\mathcal{O}(\log(N))$ operations (or, by more brute force approaches, in $\mathcal{O}(N)$). Once we identify x_{i^*}, the look-up of $f(x_{i^*})$ takes $\mathcal{O}(1)$ operations. Thus the overall Online cost for piecewise-constant, left-endpoint interpolation is $\mathcal{O}(1)$ for equispaced data and $\mathcal{O}(\log(N))$ (or $\mathcal{O}(N)$, brute force) for variably spaced data.

We close this section with two comments, one philosophical, the other pragmatic. From the philosophical side, we note the crucial role that discretization plays in the success of our interpolation approach: the computational cost is small because once we identify i^* the function is very simple — *constant* — over S_{i^*}; but we achieve reasonable accuracy despite this simple representation precisely thanks to discretization — we enlist a *different* constant over each segment. The beneficial effect of the latter is reflected in the convergence of our approximation as h tends to zero. From the pragmatic side, we observe that the error is inversely proportional to the number of entries in our look-up table, N. In order to reduce N we might consider different "whats," in particular higher-order schemes, over each segment; an example is discussed in the next section. But we can also, for the same piecewise constant "what," consider segments of different lengths — smaller h_i for the segments S_i over which $f'(x)$ is larger, as suggested by the local interpolation error bound (2). Of course, to effect this strategy, we must be able to estimate $f'(x)$; numerical differentiation is the topic of a subsequent nutshell.

4 Piecewise-Linear Interpolation

We now consider arguably the most ubiquitous interpolation scheme: piecewise-linear interpolation. As the name suggests, this interpolant is linear over each segment and is hence piecewise-linear over the entire interval $[a, b]$.

To construct the interpolant, we again start with the $(N-1)$-segment discretization as shown in Figure 1. We first focus our attention on the interpolation of data over a single segment S_i delineated by the endpoints x_i and x_{i+1}, as shown in Figure 5. We construct our approximation in terms of $(x_i, f(x_i))$ and $f(x_{i+1}, f(x_{i+1}))$ — the available data over S_i — which furthermore satisfies the following two conditions: 1) "what": the approximation is linear over the segment S_i, and 2) "where": the approximation matches the exact function at the two endpoints,

$$(\mathcal{I}f)(x_i) = f(x_i) \quad \text{and} \quad (\mathcal{I}f)(x_{i+1}) = f(x_{i+1}) \ . \tag{4}$$

These "what" and "where" conditions define a unique interpolant: two points determine a straight line; we depict the resulting interpolant in Figure 5. We can express the interpolant in functional form as

$$(\mathcal{I}f)(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) \quad \text{for } x \text{ in } S_i = [x_i, x_{i+1}] \; ; \tag{5}$$

clearly $(\mathcal{I}f)(x)$ is linear over the segment S_i, and furthermore $\mathcal{I}f$ satisfies (4). We recognize that $f(x_i)$ is the left-endpoint of the interpolant and $(f(x_{i+1}) - f(x_i))/(x_{i+1} - x_i)$ is the slope of the interpolant.

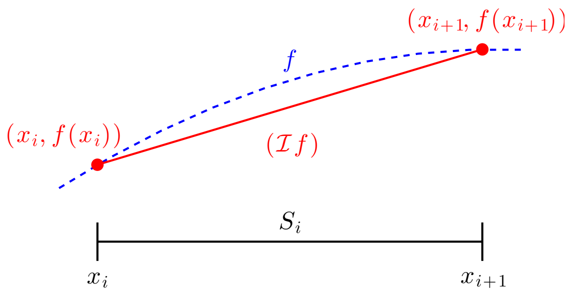

Figure 5: Illustration of piecewise-linear interpolation over the segment S_i.

We can now directly extend our result to the full interval: Given any x in $[a, b]$, we first identify the interval in which x resides, S_{i^*} (as before, i^* of course depends on x); we then apply our formula (5) for $x_i \equiv x_{i^*}$ and $x_{i+1} \equiv x_{i^*+1}$. We depict the interpolant over the entire interval in Figure 6. As a concrete example, we apply piecewise-linear interpolation to predict the speed of sound as a function of temperature from the look-up table of Table 1; we consider $x = 248$K as the temperature at which we wish to estimate the speed of sound. We identify $i^* = 2$ since $x_{i^*} \equiv 240 \le x < x_{i^*+1} \equiv 250$; we then apply (5) for $i = i^*$ to obtain $(\mathcal{I}f)(248) = f(240) + (f(250) - f(240))/(250 - 240) \cdot (248 - 240) = 310.5 + (316.9 - 310.5)/(250 - 240) \cdot (248 - 240) = 315.6$ (m/s).

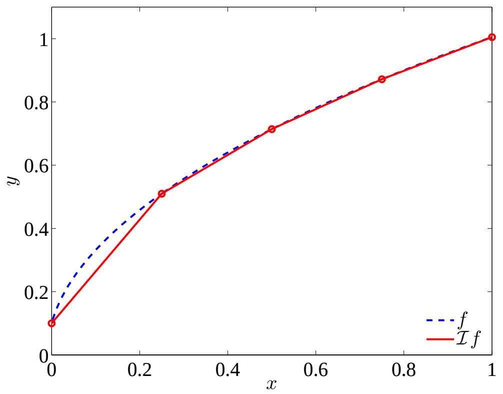

Figure 6: Illustration of the piecewise-linear interpolant over the full interval $[a \equiv 0, b \equiv 1]$.

CYAWTP 10. Evaluate the speed of sound by piecewise-linear interpolation of the (temperature, speed of sound) data of Table 1 at temperatures of 257K and 291K, respectively.

We may analyze the error associated with our piecewise-linear approximation $\mathcal{I}f$ of f. In particular, we may obtain bounds (see Appendix A) for the local interpolation error,

$$e_i \leq \frac{h^2}{8} \max_{x \in S_i} |f''(x)| \; , \tag{6}$$

as well as for the global interpolation error,

$$e_{\max} \leq \frac{h^2}{8} \max_{x \in [a,b]} |f''(x)| \; ; \tag{7}$$

theses bounds are sharp. We make two observations. First, the convergence rate for the piecewise-linear interpolation is $p = 2$ — the method is second-order accurate. Thus, we expect the piecewise-linear interpolant to converge more rapidly with h than the piecewise-constant, left-endpoint interpolant for sufficiently small h. Second, the error bound depends on the *second derivative*, f''; in contrast, for piecewise-constant, left-endpoint interpolation the error bound depends on the *first derivative*, f'. If f'' does not exist, the bound (7) may not be valid.

Note that interpolation only "works" because of smoothness: it is the property that the value of a function at one input point is related to the value of a function at a neighboring input point which permits us to accurately estimate the behavior of the function over the entire interval $[a, b]$ based only on limited data — $(x_i, f(x_i))$, $i = 1, \ldots, N$. Furthermore, the more smoothness available, the more we can predict from just a small snippet of genetic material, either function values on some small interval, or function values at a few select points. This argument also correctly suggests that interpolation is not a good strategy in the presence of noisy data — $f(x)$ contaminated by measurement error, say; we shall consider (non-interpolatory) approximation schemes for noisy data in a subsequent nutshell.

CYAWTP 11. Reconsider **CYAWTP 2** but now for the case of piecewise-linear interpolation: evaluate the local error bound (6) for temperatures 257K and 291K, respectively, and the global error bound (7) associated with piecewise-linear interpolation of the data of Table 1 over the interval $[230, 310]$.

CYAWTP 12. Which of the following functions $f(x)$ can be exactly represented by a piecewise-linear interpolant (such that $\mathcal{I}f(x) - f(x) = 0$ for all $x \in [0, 1]$): $f(x) =$ constant; $f(x)$ a linear polynomial in x; $f(x)$ a quadratic polynomial in x; $f(x)$ a sinusoidal function of x?

CYAWTP 13. Sketch the logarithmic convergence curve and the logarithmic convergence asymptote associated with interpolation of $f(x) = \sin(10\pi x)$ over the interval $[0, 1]$ for (a) piecewise-constant, left-endpoint interpolation, and (b) piecewise-linear interpolation. How are the slopes of the two logarithmic convergence asymptotes related? How are the intercepts $\log_{10}(C_{\text{asymp}})$ of the two logarithmic convergence asymptotes related?

Numerical Experiment 14. Invoke the Interpolation GUI for $f(x) = \sin(10\pi x)$ to confirm your predictions of **CYAWTP 13**.

CYAWTP 15. Sketch the logarithmic convergence curve and logarithmic convergence asymptote associated with interpolation of the discontinuous function

$$f(x) = \begin{cases} 0, & 0 \leq x < 1/3 \\ 1, & 1/3 \leq x \leq 1 \; , \end{cases} \tag{8}$$

for (a) piecewise-constant, left-endpoint interpolation, and (b) piecewise-linear interpolation, respectively. Consider a sequence of discretizations corresponding to a "doubling" uniform refinement strategy $h = 1$, $h = 1/2$, $h = (1/2)^2$, $h = (1/2)^3$, $\ldots$. Note that the jump in the function at $x = 1/3$ will always reside *inside* a segment and not at a segment endpoint. Indicate the convergence rate — the negative of the slope of the logarithmic convergence asymptote — for each case.

Numerical Experiment 16. Invoke the interpolation GUI for the discontinuous function (8) and confirm empirically your predictions of **CYAWTP 15**. Recall that the GUI considers a "doubling" uniform refinement strategy for $h = 1$, $h = 1/2$, $h = (1/2)^2$, $h = (1/2)^3$, $\ldots$, such that the jump in the function at $x = 1/3$ will always reside *inside* a segment and not at a segment endpoint.

We briefly comment on the cost of piecewise-linear interpolation. In the Offline stage, the interpolation requires preparation of N data pairs $(x_1, f(x_1)), \ldots, (x_N, f(x_N))$; the cost is $\mathcal{O}(N)$. In the Online stage, we first identify the segment $S_{i^*} = [x_{i^*}, x_{i^*+1}]$ to which x belongs; as before, this requires $\mathcal{O}(1)$ operations for a constant h and $O(\log N)$ (or more simply, $\mathcal{O}(N)$) operations for variable h. We then look up $r = f(x_{i^*})$ and $s = f(x_{i^*+1})$ in $\mathcal{O}(1)$ operations and compute

$$(\mathcal{I}f)(x) = r + \frac{s - r}{x_{i^*+1} - x_{i^*}}(x - x_{i^*})$$

in 6 FLOPs.

We note that the cost of piecewise-linear interpolation is comparable to the cost of piecewise-constant, left-endpoint interpolation. The Offline cost for both methods is $\mathcal{O}(N)$; the Online cost for both methods is $\mathcal{O}(1)$ for a constant h and $\mathcal{O}(\log(N))$ for variable h. However, for the same h (and hence N), the piecewise-linear interpolation is much more accurate than the piecewise-constant, left-endpoint interpolation, at least in the asymptotic regime: the former is second-order accurate, whereas the latter is first-order accurate. We can thus conclude that the piecewise-linear interpolation is more efficient than piecewise-constant, left-endpoint interpolation in terms of accuracy obtained for prescribed computational cost. It is for this reason — no pain, but gain — that piecewise-linear interpolation is perhaps the most commonly invoked data-interpolation scheme.

5 A Point-Based Formulation

In the previous sections, we have viewed interpolation as a global approximation problem over a finite interval $[a, b]$ which we decompose into segments S_i, $i = 1, \ldots, N-1$, the extent h_i of each of which decreases as we refine our discretization. Alternatively, we may view interpolation as a local approximation problem, in which given some few pairs $(x, f(x))$, we interpolate $f(x)$ over $[x_{\min}, x_{\max}]$; here $x_{\min}$ is the smallest value of x for which $f(x)$ is provided, and $x_{\max}$ is the largest value of x for which $f(x)$ is provided.

As a first example, let us say we are given $\bar{x}_1$ and $\bar{x}_2$ for $\bar{x}_1 < \bar{x}_2$ and $h = \bar{x}_2 - \bar{x}_1$. We can then construct, say, a piecewise-constant, left-endpoint interpolant over $[x_{\min} \equiv \bar{x}_1, x_{\max} \equiv \bar{x}_2]$ as

$$(\mathcal{I}f)(x) = f(\bar{x}_1), \quad x \in [x_{\min}, x_{\max}].$$

Similarly, we can construct a piecewise-linear interpolant over $[x_{\min} \equiv \bar{x}_1, x_{\max} \equiv \bar{x}_2]$ as

$$(\mathcal{I}f)(x) = f(\bar{x}_1) + \frac{f(\bar{x}_2) - f(\bar{x}_1)}{h}(x - \bar{x}_1), \quad x \in [x_{\min}, x_{\max}].$$

But we can now proceed further.

Let us consider three points $\bar{x}_1$, $\bar{x}_2$, and $\bar{x}_3$ where $\bar{x}_1 < \bar{x}_2 < \bar{x}_3$; for simplicity, we assume the points are equispaced such that $h \equiv \bar{x}_2 - \bar{x}_1 = \bar{x}_3 - \bar{x}_2$. We can now construct a *quadratic* interpolant over $[x_{\min} \equiv \bar{x}_1, x_{\max} \equiv \bar{x}_3]$ as

$$\begin{aligned}(\mathcal{I}f)(x) = \frac{1}{2h^2} f(\bar{x}_1)(x - \bar{x}_2)(x - \bar{x}_3) - \frac{1}{h^2} f(\bar{x}_2)(x - \bar{x}_1)(x - \bar{x}_3) \\ + \frac{1}{2h^2} f(\bar{x}_3)(x - \bar{x}_1)(x - \bar{x}_2), \quad x \in [x_{\min}, x_{\max}].\end{aligned}$$

Note we retain the two ingredients of interpolation: the "what" is a quadratic polynomial; the "where" is the three equally spaced points, $\bar{x}_1, \bar{x}_2, \bar{x}_3$ — it is easy to verify that $(\mathcal{I}f)(\bar{x}_i) = f(\bar{x}_i)$, $i = 1, 2, 3$. Our interpolant is unique since a quadratic polynomial can be expressed in terms of three degrees of freedom and the interpolation condition $(\mathcal{I}f)(\bar{x}_i) = f(\bar{x}_i)$, $i = 1, 2, 3$, imposes three constraints: three (independent) linear equations in three unknowns.

There are many applications of the point-based formulation, in particular because there is no need for a underlying global discretization over the full domain. In a later nutshell, we will take advantage of this point-based formulation of interpolation to develop numerical differentiation formulas.

6 Perspectives

We have only here provided a first look at the topic of numerical interpolation. A more in-depth study may be found in *Math, Numerics, and Programming (for Mechanical Engineers)*, M Yano, JD Penn, G Konidaris, and AT Patera, available on MIT OpenCourseWare, which adopts similar notation to these nutshells and hence can serve as a companion reference. For an even more comprehensive view from both the computational and theoretical perspectives we recommend *Numerical Mathematics*, A Quarteroni, R Sacco, F Saleri, Springer, 2000.

Of the many further topics of interest, perhaps the most important is the treatment of interpolation in higher dimensions. In this nutshell we consider interpolation of functions over an interval — functions of a single independent variable defined over a one-dimensional domain. How might we construct an interpolant if the domain is not an interval, but rather some region in two dimensions (two independent variables) or even three dimensions (three independent variables)? For instance, suppose we are given a table of the speed of sound of a dense gas for several values of the gas temperature *and* the gas pressure; how might we approximate the speed of sound for any given temperature and pressure? In fact, the interpolation methods we present here do extend to higher dimensions, however, the presentation and analysis is somewhat more involved, and more importantly convergence can be slow in very high dimensions. The latter is known as the curse of dimensionality.

Appendix A Derivation of Error Bound for Piecewise-Linear Interpolation

We first recall the mean-value theorem: if g is continuously differentiable on $[c, d]$, then there exists a point $x^* \in [c, d]$ such that $g'(x^*) = (g(d) - g(c))/(d - c)$. We then appeal to the fundamental theorem of calculus and the mean-value theorem:

$$\begin{aligned}
f(x) - (\mathcal{I}f)(x) &= \int_{x_i}^{x} (f - \mathcal{I}f)'(t)\, dt && \text{(fundamental theorem of calculus)} \\
&= \int_{x_i}^{x} \int_{x^*}^{t} (f - \mathcal{I}f)''(s)\, ds\, dt && \text{(mean-value theorem)} \\
&= \int_{x_i}^{x} \int_{x^*}^{t} f''(s)\, ds\, dt && \text{($(\mathcal{I}f)'' = 0$ since $\mathcal{I}f$ is linear)} \\
&\le \max_{x \in S_i} |f''(x)| \int_{x_i}^{x} \int_{x^*}^{t} ds\, dt && \\
&\le \frac{h^2}{2} \cdot \max_{x \in S_i} |f''(x)| && \text{(integration over the s-t triangle);}
\end{aligned}$$

here, the second step follows from first invoking the mean-value theorem — there exists a point $x^* \in [x_i, x_{i+1}]$ such that $f'(x^*) = (f(x_{i+1}) - f(x_i))/(x_{i+1} - x_i) = (\mathcal{I}f)'(x^*)$ and hence $(f - \mathcal{I}f)'(x^*) = 0$ — and then invoking the fundamental theorem of calculus with the limits x^* and t. We conclude that the local interpolation error is bounded from above by $e_i \le (h^2/2) \max_{x \in S_i} |f''(x)|$.

We can in fact obtain a tighter bound through a more careful analysis (see *Math, Numerics, and Programming (for Mechanical Engineers)*, M Yano, JD Penn, G Konidaris, and AT Patera). We state here the result for the local interpolation error,

$$e_i \le \frac{h^2}{8} \max_{x \in S_i} |f''(x)| \ ,$$

as well as for the global interpolation error,

$$e_{\max} \le \frac{h^2}{8} \max_{x \in [a,b]} |f''(x)| \ ;$$

theses bounds are sharp.