Designing and building efficient look-up tables

Wagner L. Truppel

July 12, 2020

Introduction

Suppose we have a smooth (*i.e.*, differentiable) function $f(x)$, defined in some interval $x \in [a, b]$, which we need to invoke many many times during the execution of a program and assume that any such invocation is expensive. Examples, to name just a few, might include a trigonometric function, such as $\sin(x)$, or a hyperbolic one, such as $\cosh(x)$, or the error function

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \,.$$

Even though modern computer platforms generally provide very fast floating point evaluations, it may still be the case that storing a look-up table for such functions provides a significant performance boost in performance-critical applications.

Of course, there's a trade-off, namely, *memory*: we're gaining a boost in performance at the expense of storing in memory a potentially large number of floating point numbers, and the size of that storage is larger the more precision is required of the look-up tables.

It stands to reason, then, that look-up tables need to be *designed*. In this essay, I'll use $\sin(x)$ as an example, when appropriate.

Dynamic Sampling

Naively, one might imagine that there's not much to creating a look-up table. We create an array of N pairs of values $(x_k, f(x_k))$, where $k = 0, 1, 2, \ldots, N-1$ and the x_k values are equally separated in the range $[a, b]$, as follows:

$$x_0 = a \qquad x_{N-1} = b \qquad dx = \frac{(b-a)}{(N-1)}$$

$$x_k = x_0 + k\, dx \qquad f_k = f(x_k)\,, \qquad \text{with } k = 0, 1, 2, \ldots, (N-1)$$

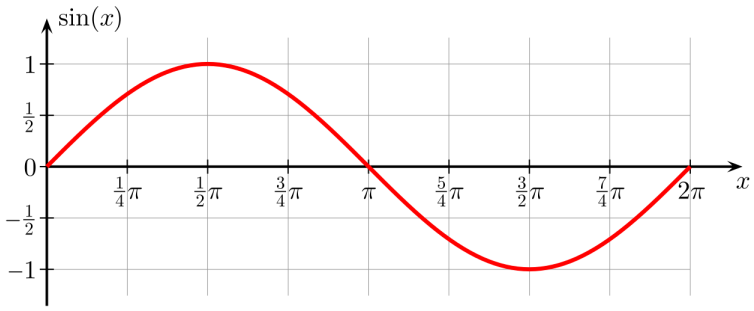
The choice of N is determined by the precision required: the more precision we demand of the table, the larger N must be. Once built, the table can quickly return or compute the value of $f(x)$ for *any* value of x in the range $[a, b]$, even for values that aren't actually stored in the table, by linearly interpolating between bracketing values that *are* found in the table:

$$f(x) \approx f_k + \frac{(x - x_k)}{(x_{k+1} - x_k)} (f_{k+1} - f_k),$$

where (x_k, f_k) and (x_{k+1}, f_{k+1}) are two *successive* pairs found in the table such that x_k and x_{k+1} *bracket* x, that is, such that $x_k \leq x \leq x_{k+1}$.

One problem with the above approach is that, in general, a uniform distribution of x values does not result in a uniform distribution of $f(x)$ values: regions of the x-domain for which $f(x)$ changes dramatically result in poor interpolated values.

Take, for instance, the case of $\sin(x)$. Near $x = 0$, $\sin(x)$ changes fairly quickly but, near $x = \pi/2$, it changes rather slowly:[1]



No problem, right? We can simply choose a larger value of N to guarantee a good precision in those regions of fast change. The problem with that, of course, is that we're also storing a potentially large number of pairs of values $(x_k, f(x_k))$ in regions where $f(x)$ changes slowly, wasting memory.

We'd like to dynamically determine the value of dx, depending on the current value of x for which we want to compute $f(x)$, so that regions where $f(x)$ changes quickly have more pairs of points (*i.e.*, smaller values of dx) while regions where $f(x)$ changes slowly have fewer pairs of points (*i.e.*, larger values of dx):

$$x_0 = a \qquad\qquad x_{N-1} = b$$

$$x_{k+1} = x_k + dx_k \qquad\qquad f_k = f(x_k), \qquad \text{with } k = 0, 1, 2, \ldots, (N-1)$$

This is called *Dynamic Sampling*. But how do we determine dx_k?

[1] Figure credit: https://en.wikipedia.org/wiki/Sine#/media/File:Sine_one_period.svg

Consider what we're trying to compute, namely, the value of $f(x)$ for some value of x, say, at x_{k+1}. Using a Taylor series expansion, we have:

$$f(x_{k+1}) = f(x_k) + (x_{k+1} - x_k)\, f'(x_k) + \frac{1}{2}\,(x_{k+1} - x_k)^2\, f''(x_k) + \cdots$$

where $f'(x_k)$ and $f''(x_k)$ are the first and second derivatives of $f(x)$, evaluated at $x = x_k$. By choosing $(x_{k+1} - x_k)$ sufficiently small, and if $f(x)$ is sufficiently smooth, we may neglect the terms in the series beyond the one involving the first derivative, and write

$$f(x_{k+1}) \approx f(x_k) + (x_{k+1} - x_k)\, f'(x_k) = f(x_k) + dx_k\, f'(x_k)\,.$$

We can now *demand* that the sampled values of $f(x)$ be distributed uniformly in the table, that is, we impose

$$df \equiv |f(x_{k+1}) - f(x_k)|$$

to be *independent* of k. df is determined by the precision required of the table. Then, from the expression just prior,

$$df = |dx_k|\,|f'(x_k)| \qquad \Rightarrow \qquad |dx_k| = \frac{df}{|f'(x_k)|}\,.$$

Since the sign of dx_k is determined by whether $b \geq a$ or $a \geq b$ and we're assuming that $b \geq a$, it follows that $dx_k \geq 0$ and we can drop the absolute value bars for dx_k:

$$\begin{aligned} x_0 &= a & x_{N-1} &= b & dx_k &= \frac{df}{|f'(x_k)|} \\ x_{k+1} &= x_k + dx_k & f_k &= f(x_k)\,, & &\text{with } k = 0, 1, 2, \ldots, (N-1) \end{aligned}$$

Note that N, the size of the look-up table, is now some finite but undetermined integer. The result above makes sense: where the derivative is large (in magnitude), $f(x)$ is changing quickly, so dx_k needs to be small to maintain precision. Conversely, where the magnitude of the derivative is small, dx_k can be larger because $f(x)$ is changing slowly.

There is something to be careful about the result above, however. What happens when the derivative at x_k is so small that dx_k becomes larger than the interval $(b - a)$ or, even worse, the derivative at x_k is exactly zero? We can deal with both situations by altering the definition of dx_k slightly:

$$dx_k = \min\Big(\frac{df}{|f'(x_k)|}, dx_{\max}\Big)\,,$$

where $dx_{\max}$ is some arbitrarily chosen *maximum* value. We may also define a *minimum* value $dx_{\min}$, to prevent having too large a value of N,

$$dx_k = \min\Big(\max\Big(dx_{\min}, \frac{df}{|f'(x_k)|}\Big), dx_{\max}\Big)\,,$$

so that $dx_{\min} \leq dx_k \leq dx_{\max}$.

Estimating N

The next question is: can we determine or, at least estimate, the resulting number of values in the table, N?

We can, by using the idea that the area of the region under the curve $f(x)$, bounded by the x-axis and the vertical lines $x = a$ and $x = b$ is the integral of $f(x)$ between a and b, as well as approximately the sum of the areas of rectangles of width dx_k and height f_k:

$$\sum_{k=0}^{N-1} f_k \, dx_k \approx \int_a^b f(x) \, dx \, .$$

But,

$$\sum_{k=0}^{N-1} f_k \, dx_k = \sum_{k=0}^{N-1} f_k \, \frac{df}{|f'_k|} = df \, \sum_{k=0}^{N-1} \frac{f_k}{|f'_k|} \, .$$

Suppose we can estimate the average value of $f_k/|f'_k|$, call it $\bar{s}$,

$$\bar{s} = \frac{1}{N} \sum_{k=0}^{N-1} \frac{f_k}{|f'_k|} \, .$$

Then,

$$N \bar{s} \, df \approx \int_a^b f(x) \, dx \qquad \Rightarrow \qquad N \approx \frac{1}{\bar{s} \, df} \int_a^b f(x) \, dx \, .$$

Take the case of $f(x) = \sin(x)$ again. Its derivative is $f'(x) = \cos(x)$ so, taking the interval $[a, b] = [0, \pi/2]$,

$$N \approx \frac{1}{\bar{s} \, df} \, ,$$

since $\displaystyle\int_0^{\pi/2} \sin(x) \, dx = 1$. We now need to estimate $\bar{s}$. In the interval $[0, \pi/2]$, $f'(x) = cos(x)$ is positive so $|f'_k| = f'_k$ and

$$\bar{s} = \frac{1}{N} \sum_{k=0}^{N-1} \frac{f_k}{|f'_k|} = \frac{1}{N} \sum_{k=0}^{N-1} \frac{\sin(x_k)}{\cos(x_k)} = \frac{1}{N} \sum_{k=0}^{N-1} \tan(x_k) = \overline{\tan(x)} = \frac{1}{\pi/2} \int_0^{\pi/2} \tan(x) \, dx \, .$$

The integral of $\tan(x)$ between 0 and $\pi/2$ does not converge. However, we know that there will be relatively few pairs of values in the table near $x = \pi/2$, since $\sin(x)$ is changing

slowly there. Thus, we can cut the integral off at some value of x below $\pi/2$, without much of a loss in the estimate of N. Say we choose a cut-off value of $x = 1.5 \approx 86°$. Then,

$$\bar{s} \approx \frac{2}{3}\int_0^{3/2} \tan(x)\,dx \approx 1.77 \qquad \Rightarrow \qquad N \approx \frac{1}{1.77\,df} \approx \frac{0.57}{df}\,.$$

So, for $f(x) = \sin(x)$, N is of the same order of magnitude as the inverse of df.

Interpolation

As mentioned before, we can linearly interpolate between two successive pairs of values $(x, f(x))$ in the table to compute the value of $f(x)$ for a value of x that is bracketed by the two x-values from the table.

However, linear interpolation is generally not as accurate as higher-order interpolations, for the same number N of points, and does not preserve the continuity of the function's first derivative. If that's important, we can select a different interpolation strategy, one that guarantees that both the interpolated values and their derivatives are continuous.

Suppose we want to interpolate between (x_k, f_k) and (x_{k+1}, f_{k+1}) to find the value of $f(x)$ for some value x such that $x_k \le x \le x_{k+1}$, and we want to preserve the continuity of $f(x)$'s first derivative.

We can do this by using a cubic polynomial,

$$f(x) = A + Bx + Cx^2 + Dx^3\,.$$

There are 4 constants to determine and we have 4 conditions, namely, the polynomial must match the values of both the function and its derivative at both ends of the interpolating interval.

It's much much easier to find these constants, however, if we work with a slightly different, but equivalent, polynomial. We know that $f(x_k) = f_k$, which means that $f(x) - f_k$ must have a root at $x = x_k$. Thus, we can choose this polynomial, instead:

$$f(x) = f_k + A(x - x_k) + B(x - x_k)^2 + C(x - x_k)^3$$

(with different constants from the previous one). Imposing the remaining conditions, we find:

$$\begin{aligned} A(x_{k+1} - x_k) + B(x_{k+1} - x_k)^2 + C(x_{k+1} - x_k)^3 &= f_{k+1} - f_k \\ A &= f'_k \\ A + 2B(x_{k+1} - x_k) + 3C(x_{k+1} - x_k)^2 &= f'_{k+1} \end{aligned}$$

which we can rewrite as

$$\begin{aligned} A &= f'_k \\ B(x_{k+1}-x_k)^2 + C(x_{k+1}-x_k)^3 &= (f_{k+1}-f_k) - f'_k(x_{k+1}-x_k) \\ 2B(x_{k+1}-x_k) + 3C(x_{k+1}-x_k)^2 &= (f'_{k+1}-f'_k) \end{aligned}$$

Solving for B and C is easy, and we find

$$\begin{aligned} A &= f'_k \\ B &= \frac{3(f_{k+1}-f_k) - (f'_{k+1}+2f'_k)(x_{k+1}-x_k)}{(x_{k+1}-x_k)^2} \\ C &= \frac{(f'_{k+1}+f'_k)(x_{k+1}-x_k) - 2(f_{k+1}-f_k)}{(x_{k+1}-x_k)^3}\,. \end{aligned}$$

Thus, the interpolating polynomial we seek is

$$\begin{aligned} f(x) &= f_k + f'_k(x-x_k) \\ &+ \Big(3(f_{k+1}-f_k) - (f'_{k+1}+2f'_k)(x_{k+1}-x_k)\Big)\Big(\frac{x-x_k}{x_{k+1}-x_k}\Big)^2 \\ &+ \Big((f'_{k+1}+f'_k)(x_{k+1}-x_k) - 2(f_{k+1}-f_k)\Big)\Big(\frac{x-x_k}{x_{k+1}-x_k}\Big)^3. \end{aligned}$$

Bracketing

Regardless of the interpolation method used, how do we find the two pairs of values (x_k, f_k) and (x_{k+1}, f_{k+1}) in the table whose x-values bracket x? If we had a uniform dx, it would very simple, since

$$x_k = x_0 + k\,dx \qquad \text{and, so,} \qquad k = \lfloor \frac{x-x_0}{dx} \rfloor\,,$$

and then we wouldn't even have to store the x values but only have an array of $f(x)$ values, indexed by k.

But we're using dynamic sampling and dx is not uniform. The way to efficiently find x_k is to recall that the table is sorted in ascending order of x-values. Therefore, we can use *binary search* to find the bracket that contains x. This is especially efficient since binary search runs in $O(\log N)$ time rather than $O(N)$.

How do we know it's worth, though?

We've seen how to build an efficient look-up table for expensive numeric function evaluations. It involves a pre-processing step of building the table, which is usually not an issue, but the table itself may require a lot of memory. Moreover, each function evaluation requires a binary search, followed by either a linear interpolation or a cubic interpolation. Is it possible that this effort is not worth the trouble? More specifically, is it possible that, in the end, the built-in function evaluations end up being faster than the table look-ups? Of course it's possible and the only way to be sure is actually to *measure* the time it takes to perform the computations required, in both cases, and compare them.

■