Computer System Support for Scientific and Engineering Computation

Lecture 14 - June 16, 1988 (notes revised July 6, 1988)

1 Argument Reduction for Elementary Function Evaluation

It is not unusual for a calculator or computer to give an error message when asked to evaluate $\cos(10^{63})$. The logic of this is as follows. In order to actually compute $\cos(10^{63})$, you need to find an integer n so that $0 \leq 10^{63} - 2n\pi < 2\pi$, and then evaluate $\cos(10^{63} - 2n\pi)$. To evaluate $\cos x$ when $0 \leq x < 2\pi$, first use trigonometric identities to reduce the argument to a still smaller range (of a possibly different trigonometric function) like $[0, \pi/4)$, then use a series approximation, or interpolate from a table to evaluate over this smaller range. In order for $0 \leq 10^{63} - 2n\pi < 2\pi$, $2n\pi$ has to have about as many digits as 10^{63}. On a decimal calculator, the first 63 digits or so will cancel when subtracting. In order for any bits in the difference to be significant, $2n\pi$ has to be accurate to at least 63 digits, which means that π must be computed to at least 63 decimal digits. Since most calculators and computers don't store π to this accuracy, after they compute $10^{63} - 2n\pi$ there are no significant bits left. Thus they signal an error.

1.1 Preserving Identities

There are several problems with this approach. The first is that users normally associate errors with underflow or overflow or singularites. None of these are present when computing $\cos 10^{63}$. Thus signaling an error is misleading. A second problem is that cosine of large arguments is not the only case where evaluating trigonometric functions can be inaccurate. Consider $\cos x$ when $x \approx \frac{\pi}{2}$. Then $\cos x = \sin(\frac{\pi}{2} - x) \approx \frac{\pi}{2} - x$. In a decimal calculator that displays 10 digits, carries 13 internally, and stores π to 13 digits, cancellation will destroy about 7 of the 10 digits in x. Thus most of the digits in the result will be meaningless. And yet certainly a calculator shouldn't display an error message when evaluating cosine near $\frac{\pi}{2}$.

There are at least two alternatives to signaling an error when presented with $\cos(10^{63})$. One is to assume that x is exact, and compute $\cos 10^{63}$ accordingly. This will require having stored more than 63 digits of π. Another approach is to make trigonometric identities hold. Thus even though $\cos 10^{63}$ might not have any correct bits, it will still be true (to within a few ulps) that $\sin^2 10^{63} + \cos^2 10^{63} = 1$, and that $\sin(2 \times 10^{63}) = 2 \sin 10^{63} \cos 10^{63}$, and so on. The argument in favor of this approach is that in real calculations, the arguments to cosine are likely to be even less accurate than the computer's approximation to π, so the extra

time and expense of computing cosine exactly is rarely worth it. For example on a binary computer, 10^{63} can't be represented exactly. Incidentally, you might wonder why anyone would ever want to compute the cosine of large arguments. One example is computing the wave front far from an antenna (that is, many wavelengths away from the antenna). This might require computing the value of a trigonometric function of large argument directly, or indirectly thru approximations like $J_n(x) = \sqrt{\frac{2}{\pi x}}\left(\cos(x - \frac{1}{2}n\pi - \frac{1}{4}\pi) + O(|x|^{-1})\right)$. In either case, what is usually important is not the value of $\cos x$, but rather how it relates to $\cos y$ where $|x - y| \ll |x|$. If trigonometric identities like $\cos x - \cos y = 2\sin\frac{x+y}{2}\sin\frac{x-y}{2}$ are preserved, the calculation is much more likely to be reasonable. It is worth noting that this approach does not say that you can compute cosine inaccurately as long as it preserves some random identity. Rather, cos should be computed as accurately as possible, but the bits which can't be determined from the argument should be chosen so as to preserve all trigonometric identities that do not explicitly reference π.

A method of doing argument reduction that preserves trigonometric identities is as follows. This is the method used in the Intel 8087 and Motorola 68881. Let Π be the value of π as represented in the machine, that is "machine pi". Compute $r = x\mathrm{REM}\frac{\Pi}{2}$, where REM is the IEEE standard remainder function. This is defined by the equation $r = x - n\frac{\Pi}{2}$, where n is the integer nearest the exact value $x/\frac{\Pi}{2}$, and even in case of a tie. Let $\mathrm{trig}(x)$ be one of the trigonometric functions $\sin x$, $\cos x$, $\tan x$ or $\cot x$. Then simply compute $\mathrm{trig}(x)$ as $\pm\mathrm{trig}_1(r)$ where trig_1 is selected using the identities

$$\begin{aligned}
\sin(x - \tfrac{\pi}{2}) &= -\cos x \\
\cos(x - \tfrac{\pi}{2}) &= \sin x \\
\tan(x - \tfrac{\pi}{2}) &= \cot x \\
\cot(x - \tfrac{\pi}{2}) &= -\tan x
\end{aligned}$$

We need to show why this preserves trigonometric identities. First we observe that

Theorem 1 *If trig is one of the functions* sin, cos, tan *or* cot, *then* $trig(x(1+\epsilon)) = (1 + k\epsilon)trig(x)$ *for some modest constant* k, *when* x *is in the range* $|x| \le \frac{\pi}{4}$, *for any integer* n.

To prove this, use the Taylor approximation $\mathrm{trig}(x(1+\epsilon)) = \mathrm{trig}(x + x\epsilon) \approx \mathrm{trig}(x) + x\epsilon\mathrm{trig}'(x)$. Thus showing $\mathrm{trig}(x(1+\epsilon)) = (1+k\epsilon)\mathrm{trig}(x)$ is the same as showing $|x\mathrm{trig}'(x)| \le k|\mathrm{trig}(x)|$. In the following table

$xf'(x)$	$f(x)$
$x\cos x$	$\sin x$
$x\sin x$	$\cos x$
$x\sec^2 x$	$\tan x$
$x\csc^2 x$	$\cot x$

it is easy to check that when $|x| \le \frac{\pi}{4}$, each entry in the left column is always bounded by a small constant times the corresponding entry in the right column. This completes the proof.

Let trig be the true value and TRIG the computed value of a trigonometric function. For an arbitrary argument x, using the range reduction method involving REM, we have

$$\mathrm{TRIG}(x) = \pm\mathrm{TRIG}_1(x - m\frac{\Pi}{2}) = \pm(1 \pm \epsilon)\mathrm{trig}_1(x - m\frac{\Pi}{2}),$$

since in the reduced range we will assume an algorithm for TRIG_1 satisfying $\mathrm{TRIG}_1(x) = (1+\epsilon)\mathrm{trig}_1(x)$. Also

$$\mathrm{trig}_1(x - m\frac{\Pi}{2}) = \mathrm{trig}_1\left((x\frac{\pi}{\Pi} - m\frac{\pi}{2})\frac{\Pi}{\pi}\right) = (1\pm\epsilon)\mathrm{trig}_1(x\frac{\pi}{\Pi} - m\frac{\pi}{2}) = (1\pm\epsilon)\mathrm{trig}(x\frac{\pi}{\Pi})$$

using theorem 1, since $\pi/\Pi = 1 \pm \epsilon$. Thus

$$\mathrm{TRIG}(x) = (1\pm 2\epsilon)\mathrm{trig}(x\frac{\pi}{\Pi}).$$

What this means is that identities like $\sin(2x) = 2\sin x \cos x$ that remain true when x is replaced by αx will still hold true after argument reduction (to within a few ulps), using $\alpha = \pi/\Pi$, which differs from 1 by a fraction of an ulp.

The effect of computing $\mathrm{trig}(x(\pi/\Pi))$ when $\mathrm{trig}(x)$ is requested is to horizontally stretch or shrink the graph of trig by a factor Π/π. This causes a *phase shift* in scientific or engineering calculations at large radian arguments, but it is otherwise no more harmful than, say

- changing the dielectric constant of air by π/Π.
- changing the wavelenth by a factor Π/π.
- changing the distance from an antenna systematically by a factor of π/Π.

In computation of complex exponentials $e^{x+iy} = e^x(\cos y + i\sin y)$, the effect of Π is a change of phase without any effect upon magnitudes (of electric fields and gradients, ...). Consequently, the use of Π instead of π might be judged harmless to all scientific and engineering computations that do not explicitly involve the knowledge of more digits of π than Π has.

The main cost of this method is that computing the REM function requires doing long division and takes about 1 step for each bit of the quotient, and so can be quite time consuming when n is large. One other point that deserves mention has to do with the special values of the trigonometric functions. Suppose that $x = \frac{\Pi}{2}$ exactly. Then is $\cos x = 0$? And if so, is it $+0$ or -0, which are distinct in the IEEE standard? The simplest way around this is to store $\frac{\pi}{2}$ to a few more bits of precision than the user has available, so that $x = \frac{\Pi}{2}$ can never happen. A relevant mathematical fact is that a trigonometric function evaluated at a rational number of radians is always irrational except possibly when the argument is 0[1].

1.2 Exact Argument Reduction

Hopefully the last section convinced you that it is not necessary to compute trigonometric functions to within a few ulps when the arguments are extremely large. However, there is a reasonably efficient technique for doing this that was discovered independently by Bob Corbett when he was a student at Berkeley, and by Mary Payne at DEC. If you use the naive method of argument reduction, computing $\cos 2^n$ would require using at least an n bit approximation of π when reducing 2^n to range. Their method stores as many bits of $\frac{2}{\pi}$ as are needed, but uses only the few that really matter.

[1]This follows from the Gelfand-Schneider theorem. See *Transcendental Number Theory* by Alan Baker or *Topics in Number Theory* Vol 2 by Leveque.

Argument reduction requires finding an integer n so that $x + n\frac{\pi}{2} = f\frac{\pi}{2}$, where $|f| \leq \frac{1}{2}$. To compute trig(x), compute $\pm\text{trig}_1(f\frac{\pi}{2})$ instead. Since all the trigonometric functions have a period of 2π or less, n only needs to be determined mod 4 in order to figure out which trig_1 to choose. The equation above is equivalent to $\frac{2}{\pi}x + n = f$, so we only need to know the low order 2 bits of the integer part of $\frac{2}{\pi}x$, and the p most significant bits of the fractional part of $\frac{2}{\pi}x$, where p is the number of bits in the precision we are working in. Consider the following diagram of the multiplication of $\frac{2}{\pi}$ ($= .101000_2\cdots$) with x.

```
 .101000 ...   xxvvvvvv vvv ...
×xxxx00 ...   00000000.
 ---------------------------
  xxxxx ...   xxxxxxuu.uuu ...
   xxxx ...   xxxxxxuu.uuu ...
    xxx ...   xxxxxxuu.uuu ...
     xx ...   xxxxxxuu.uuu ...
 ---------------------------
 xxxxxx ...   xxxxxxuu.uuu ...
```

To keep the picture simple, we assumed that the precision p was 4. Since we only are interested in the low order 2 bits of the integer part, we only care about the bits labeled u. These are generated from the bits marked v in $\frac{2}{\pi}$, which extend $p+2$ bits to the left of the binary point of x. In particular, when we carry out the multiplication, instead of using all the bits of $\frac{2}{\pi}$, we only need to use some of them. Of course, we must store many bits of $\frac{2}{\pi}$, even though we only use a few of them in any particular argument reduction.

Suppose we have hardware that can multiply two p precision floating point numbers exactly, that is, produce a product with $2p$ fraction bits. Then the details of exact argument reduction go something like this. If $x = f2^e$ with $\frac{1}{2} \leq f < 1$ and p is the precision, then we skip the first $e-2-p$ bits of $\frac{2}{\pi}$, that is replace $\frac{2}{\pi}$ with $z = \{\frac{2}{\pi}2^{e-2-p}\}2^{-(e-2-p)}$, where $\{t\} = t - \lfloor t \rfloor$ represents the fractional part of t. Then we break z into p bit chunks $z = z_0 + z_1 2^{-p} + z_2 2^{-2p} + \cdots$. Next we compute $x_0 = x \cdot z_0$, and $x_1 = x \cdot z_1 \cdot 2^{-p}$ exactly as $2p$ precision numbers. Finally, we would like to add x_0 and x_1. If we add them using $2p$ precision arithmetic, the low order p bits of x_1 will be lost. So first replace $x_0 = n + f$ with $x_0' = n \bmod 4 + f$ and then compute $y = x_0' + x_1$. The integer part of y gives the integer part $\frac{2}{\pi}x$ modulo 4, which is all that matters. And the fractional part of y has almost p correct bits of the fractional part of $\frac{2}{\pi}x$.

The Vax™ does not have an instruction for exact multiplication. However, it does have an EMOD instruction, that takes a floating point number with p bits of precision and multiplies it by a floating point number with $p+k$ bits of precision to return a result whose integer part is stored as a 32 bit integer, and whose fractional part is stored as an ordinary p precision floating point number. The adjustment factor k is equal to the number of bits in the exponent field. This instruction can be used in place of an exact multiply when carrying out the argument reduction.

One problem with this method is that the bits in the product immediately to the right of the binary point might be zero, which would require additional multiplications of x by more digits of $\frac{2}{\pi}$ in order to get p normalized bits of significand. In IEEE double precision, if you started out with an x that was near the largest representable number, there would be about 2^{11} bits of $\frac{2}{\pi}$ in the top line of the diagram. If sufficiently many zeros kept appearing, you might need to keep computing until the factional part of the diffference underflowed. That would require another 2^{11} bits of $\frac{2}{\pi}$ to be stored. It can be shown that there is a limit to how many zeros can appear to the right of the binary point, and consequently why slightly more than 2^{11} bits need be stored.