Real number calculations in LaTeX: Packages

Joseph Wright

1 Background

TeX does not include any "native" support for floating point calculations, but that has not stopped lots of (LA)TeX users from wanting to do sums (and more complicated things) in their document. As TeX is Turing complete, it's not a surprise that there are several ways to implement calculations. For end users, the differences between these methods are not important: what is key is what to use. Here, I'll give a bit of background, look at the various possibilities, then move on to give a recommendation.

2 History

When Knuth wrote TeX, he had one aim in mind: high-quality typesetting. He also wanted to have sources which were truly portable between different systems. At the time, there was no standard for specifying how floating point operations should be handled at the hardware level: as such, no floating point operations were system-independent.

Thus, Knuth decided that TeX would provide no user access to anything dependent on platform-specific floating-point operations, and not rely on them within algorithms that produce typeset output. That means that the TeX variables and operations that look like numeric floats (in particular dimensions) actually use *integer* arithmetic and convert "at the last minute".

3 Technical considerations

There are two basic approaches to setting up floating point systems in TeX: either using dimensions or doing everything in integer arithmetic.

Using dimensions, the input range is limited and the output has restricted accuracy. But on the other hand, many calculations are quite short and they are fast. On the other hand, if everything is coded in integer arithmetic, the programmer can control the accuracy completely, at the cost of speed.

Although it's not an absolute requirement, ε-TeX does make doing things a bit easier: rather than having to shuffle everything a piece at a time, it is possible to use in-line expressions for quite a lot of the work.

Another key technical aspect is expandability. This is useful for some aspects of TeX work, particularly anywhere that it "expects a number": only expansion is allowed in such places.

Another thing to consider is handling of TeX registers as numbers. Converting for example a length into something that can be used in a floating point calculation is handy, and it matches what ε-TeX does for example in `\numexpr`. But in macro code it has to be programmed in.

The other thing to think about here is functionality: what is and isn't needed in terms of mathematics. Doing straightforward arithmetic is clearly easier than working out trigonometry, logarithms, etc. What exactly you need will depend on the use case, but in principle, more functionality is always better.

4 (Package) options

For simple work using the dimen approach is convenient and fast: it takes only a small amount of work to set up stripping off the `pt` part. I'm writing here for people who don't want to delve into TeX innards, so let's assume a pre-packaged solution is what is required.

There are lots of possible solutions on CTAN which cover some or all of the above. I don't want to get into a "big list", so I'll simply note here that the following are available on CTAN:

- apnum
- calculator
- fltpoint
- pst-fp
- minifp
- realcalc
- xint

Some of these have variable or arbitrary precision, while others work to a pre-determined level. They also vary in terms of functions covered, expandability and so on.

I want to focus in on three possible "contenders": fp, pgf and the LaTeX3 FPU (part of expl3). All of these are well-known and widely-used, offer a full set of functions, and a form of expressions.

The fp package formally uses *fixed* not *floating* point code, but the key for us here is that it allows a wide range of high-precision calculations. It's also been around for a *long* time. However, it's quite slow and doesn't have convenient expression parsing — it requires reverse Polish.

On the flip side, the arithmetic engine in pgf uses dimens internally, so it is (relatively) fast but is limited in accuracy. The range limits also show up in some unexpected places, as a lot of range reduction is needed to make everything work. On the other hand, `\pgfmathparse` does read "normal" arithmetic expressions, so it's pretty easy to use. I'll also come to another aspect below: there is a "swappable" floating point unit to replace the faster dimen-based code.

The LaTeX3 FPU is part of expl3, but is available nowadays as a document-level package xfp. In contrast to both fp and the pgf approach, the LaTeX3 FPU is expandable. Like pgf, using the FPU means we can use expressions, and we also get reasonable performance (Bruno Le Floch worked hard on this aspect). The other thing to note is that the FPU is intended to match behaviour specified in the decimal IEEE 754 standard, and that the team have a set of tests to try to make sure things work as expected.

There's one other option that one must consider: Lua. If you can accept using only LuaTeX, you can happily break out into Lua and use its ability to use the "real" floating point capabilities of a modern PC. The one wrinkle is that without a bit of work, the Lua code *doesn't* know about TeX material: registers and so on need to be pre-processed. It also goes without saying that using Lua means being tied to LuaTeX!

5 Performance

To test the performance of these options, I'm going to use the LaTeX3 benchmarking package l3benchmark. I'm using a basic set up:

```
\usepackage{l3benchmark}
\ExplSyntaxOn
\cs_new_eq:NN
  \benchmark
  \benchmark:n
\ExplSyntaxOff
\newsavebox{\testbox}
\newcommand{\fputest}[1]{%
  \benchmark{\sbox{\testbox}{#1}}%
}
```

(The benchmarking runs perform the same step multiple times, so keeping material in a box helps avoid any overhead for the typesetting step.)

The command `\fputest{...}` was then used with the appropriate input, such as the `\fpeval` command below, to calculate a test expression using a range of packages.

As a test, I'm using the expression

$$\sqrt{2}\sin\left(\frac{40}{180}\pi\right)$$

or the equivalent. Using that, it's immediately apparent that the fp package is by far the slowest approach (Table 1). Unsurprisingly, using Lua is the fastest by an order of magnitude at least. In the middle ground, the standard approach in pgf is fastest, but not by a great deal over using the LaTeX3 or pgf FPUs.

Table 1: Benchmarking results (LuaTeX v1.07, TeX Live 2018, Windows 10, Intel i5-7200)

Package	Time/10^{-4} s
fp	99.4
pgf	2.95
pgf/fpu	5.51
LaTeX3 FPU	6.42
LuaTeX	0.57

6 Recommendation

As you can see above, there are several options. However, for end users wanting to do calculations in documents I think there is a clear best choice: the LaTeX3 FPU.

```
\documentclass{article}
\usepackage{xfp}
\begin{document}
\fpeval{round(sqrt(2) * sind(40),2)}
\end{document}
```

(The test calculation uses angles in degrees, so where provided, a version of the sine function taking degrees as input was used: this is expressed in the LaTeX3 FPU as `sind`. The second argument to `round`, `2`, is the number of places to which to round the result.)

You'd probably expect me to recommend the LaTeX3 package: I am on the LaTeX team. But that's not the reason. Instead, it's that the FPU is almost as fast as using dimens (see pgf benchmarking), but offers the same accuracy as a typical GUI application for maths. It also integrates into normal LaTeX conventions with no user effort. So it offers by far the best balance of features, integration and performance for "typical" users.

◇ Joseph Wright
Northampton, United Kingdom
joseph dot wright (at)
morningstar2.co.uk