How to draw 10e9 samples from a judgmental estimation model

Jorge Sierra, Nuño Sempere

March 28, 2024

Background

Forecasters in the tradition of Tetlock sometimes create judgmental estimation models. By this we mean models that, instead of fitting an underlying set of samples, instead express the subjective credences of their creators.

One way to do this is to use a set of idioms that grew from the foretold.io experimental forecasting platform into Squiggle, a programming language for intuitive estimation. This original iteration of squiggle has also been ported to Python (Squiggle.py) and to C (Squiggle.c).

However, those idioms rely on Monte Carlo estimation, which sometimes has difficulty modelling long-tail behaviour.

To explore those limits, in this paper we describe how to take a reasonably complex judgmental estimation model, and draw 10e9 samples from it—an american trillion.

1. Use a fast language

Our time to botec repository compares how fast drawing 10e6 samples from a simple back-of-the-envelope calculation runs in different languages. The comparison depends on our familiarity with the different languages: as we are more familiar with a language, we grow capable of writing better & faster code in it. Nonetheless, speeds are as follows:

Language	Time
C	6.20ms
squiggle.c	7.20ms
go	21.20ms
Nim	41.10ms
Lua (LuaJIT)	68.80ms
Python (numpy)	118ms

Language	Time
OCaml (flambda)	185ms
Squiggle (bun)	384ms
Javascript (node)	395ms
SquigglePy (v0.27)	1,542ms
R (3.6.1)	4,494ms
Python 3.9	11,909ms
Gavin Howard's bc	16,170ms

Readers might have expected this list to include languages such as Rust or zig. FORTRAN, Lisp, Haskell, Java, C#, C++, or Julia are also missing. This is because we are not very familiar with those languages, or because in our attempts to use them we found them too "clunky".

2. Use some straightforward tricks

Pass pointers, not values, inline functions. Avoid the overhead of calling a function by inlining.

Define your own primitives, as opposed to calling a library. Reference how to do this here & limits of rng.

Compile to the native architecture. Profile. Compiler flags.

Perhaps concession was defining a mixture One of the few concessions was defining a function to mix several other distributions. This was worse than doing something like:

```
double p = sample_uniform(0,1, seed);
if(p < p1){
  return sample_dist_1(seed);
} else if (p<p1+p2){
  return sample_dist_2(seed);
} else {
  return sample_dist_3(seed);
}
```

but much more convenient

3. Introduce parallelism with OpenMP

Introducing parallelism with OpenMP was fairly straightforward.

alignment

5. Run the code on a supercomputer with MPI

Talk about

6. Conclusion