Adaptive truncation of infinite series

Applications to Statistics

Luiz Max Carvalho & Guido A. Moreira
lmax.fgv@gmail.com

FGV EMAp

The problem

Suppose you are confronted with computing

$$S := \sum_{n=0}^{\infty} \frac{\mu^n}{(n!)^\nu}. \tag{1}$$

for $\mu, \nu > 0$.

This is known not to have a closed-form solution for most values of μ, ν.

Accurate approximation

For some pre-specified error $\epsilon > 0$, how do you compute an approximation $\hat{S}$ such that $|\hat{S} - S| \le \epsilon$?

Statistical applications

Example applications

- Normalising constants;
- Marginalisation of discrete latent variables;
- Raw and factorial moments;

In general, we can write

$$S = \sum_{n=n_0}^{\infty} p(n)f(n), \tag{2}$$

where p is a (potentially unnormalised) probability mass function (pmf) and f is measurable non-negative[1] function.

[1]This will be assumed for simplicity.

Desiderata

We would like a method that

- requires little to no user input;
- has provable guarantees;
- is numerically robust;
- is easy to implement for practicioners.

Theory: assumptions

We assume that $(a_n)_{n\geq 0}$

- is absolutely convergent;
- is non-negative;
- passes the ratio test, i.e.

$$\lim_{n\to\infty} \frac{a_{n+1}}{a_n} = L < 1.$$

- is decreasing[2].

[2]Notice that this rarely holds for pmfs. Fortunately, if p is unimodal we can always sum up to the mode exactly and only approximate the tail, which is decreasing.

Theory: results

Lemma 1: Bounding a convergent series

Let $S_m := \sum_{n=0}^{m} a_n$. Under the previously mentioned assumptions on $(a_n)_{n\geq 0}$, for every $n < \infty$ the following holds:

$$S_n + a_{n+1}\left(\frac{1}{1-L}\right) < S < S_n + a_{n+1}\left(\frac{1}{1-\frac{a_{n+1}}{a_n}}\right), \tag{3}$$

if $\frac{a_{n+1}}{a_n}$ **decreases** to L and

$$S_n + a_{n+1}\left(\frac{1}{1-\frac{a_{n+1}}{a_n}}\right) < S < S_n + a_{n+1}\left(\frac{1}{1-L}\right), \tag{4}$$

if $\frac{a_{n+1}}{a_n}$ **increases** to L.

Practice: computing an approximation

Sum-to-threshold ("Naive")

For $\epsilon > 0$, just sum until $a_{n^*} \leq \epsilon$.

Guarantee: has error $\leq \epsilon$ when $L \leq 1/2$.

c-folding ("Doubling")

With $c \geq 1$ an integer, compute the partials S_m, S_{cm}; if $S_{cm} - S_m \leq \epsilon$, stop. If not, keep going until M iterations have been reached.

Guarantee: Usually blazing fast, but hard to guarantee anything.

Adaptive truncation

Using the Lemma in the previous slide, we bound the true sum within two functions of the partial sum S_n.

Guarantee: the approximation error is $\leq \epsilon$, **always**.

Practice: numerically robust R implementation

Package **sumR** (`https://github.com/GuidoAMoreira/sumR`)

```r
library(sumR)
approx <- infiniteSum(
  logFunction = "COMP",
  parameters = c(2, 2),
  epsilon = 1E-10,
  maxIter = 1000)
TrueValue <- log(besselI(2*sqrt(2), nu = 0))
exp(TrueValue) - exp(approx$sum)
# [1] 1.303846e-12
```

We also have Stan implementations of these algorithms (`https://github.com/GuidoAMoreira/stan_summer`).

Practice: interfacing with your C/C++ code

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
// [[Rcpp::depends(sumR)]]
#include <sumRAPI.h>
long double some_series(long n, double *p)
{
  long double out = n * log1pl(-p[0]);
  return out;
}
// [[Rcpp::export]]
double sum_series(double param)
{
  ans = infiniteSum(some_series, parameter,
  exp(-35), 100000, log1p(-parameter[0]), 0, &n);
  Rcpp::Rcout << "Summation took " << n << " iterations to converge.\\n";
  return (double)ans;
}
')
```

Take home

Computing infinite sums is not trivial

But it turns out that with some good old infinite series theory we can create algorithms with provable guarantees.

Fast and robust implementation

We provide open-source implementations which are numerically stable and implemented in a low-level language. This will get faster and more robust with time.

Limitations and future work

- You still need to figure out L;
- Currently we cannot handle arbitrary negative values[3].

[3] But **can** handle alternating!

THE
END