Interval Computations in Julia programming language

Evgeniya Vorontsova

Interval Computations in Julia programming language

Evgeniya Vorontsova[1, 2]

[1]Far Eastern Federal University, Vladivostok, Russia
[2]Grenoble Alpes University, Grenoble, France

Keywords: Numerical Computing; Interval Arithmetic; Julia Programming Language; JuliaIntervals; Octave Interval Package

Introduction

The general-purpose Julia programming language [5] was designed for speed, efficiency, and high performance. It is a flexible, optionally-typed, and dynamic computing language for scientific, numerical and technical computing applications. Julia is open source language with all sources free available on GitHub. The language was developed and incubated at MIT [6]. Currently, after Julia 1.0 was officially released to the public in August 2018, the language is becoming increasingly popular. Julia has been downloaded more than 8.4 million times, as of May 2019 [2], and is used at more than 1,500 universities.

So, it is very important for researchers, working in a field of interval analysis, to have fast, efficient and robust publicly available software packages for performing computations with interval arithmetic written in Julia.

IntervalArithmetic.jl

In this paper, we review and compare a recently developed Julia package for performing Validated Numerics, i.e. rigorous computations with finite-precision floating-point arithmetic, IntervalArithmetic.jl [4], with performance of GNU Octave interval package for real-valued interval arithmetic [1]. This Octave toolbox was chosen for comparison because of several important reasons. First of all, it is a free, open-source software, unlike INTLAB [9], a Matlab/Octave toolbox for Reliable Computing. The other fundamental difference between INTLAB and GNU Octave interval package is non-conformance of INTLAB to IEEE 1788-2015 — IEEE Standard for Interval Arithmetic [3]. On the other hand GNU Octave interval package's main goal is to be compliant with the Standard. Likewise, authors of IntervalArithmetic.jl wrote [4] that they were working towards having the package be conforming with the Standard. So, all calculations in these packages are performed using interval arithmetic: all quantities are treated as intervals. The final result is also an interval contained the correct answer.

In next section we would like to show some practical examples with interval arithmetic in Julia.

Examples

Getting Started

The basic object in the IntervalArithmetic.jl package is the parameterized type `Interval`. By default, `Interval` objects contain `Float64` s. Intervals are created using the `@interval` macro:

```
using IntervalArithmetic
a = @interval(1, 2)
b = @interval(3, 4)
print(a + b, a - b, a * b, a / b)
```

The output of this code is

```
[4, 6] [-3, -1] [3, 8]
[0.25, 0.666667]
```

As you may have noticed, the package permits to write quite clear and intuitive code for interval computations.

Matrix Multiplication

In this section we present the results of experiments comparing the IntervalArithmetic.jl library with the GNU Octave interval package. In summary, we show that Julia interval library is significantly faster than the Octave library.

In our first experiment we measured the time to evaluate the interval matrix multiplication. The Julia code is:

```
function MultMatr(A, B)
    return A*B
end
n = 10
M1 = 10*rand(n, n)
M2 = 10*rand(n, n)
iM1 = map(Interval, M1)
iM2 = map(Interval, M2)
A = iM1 .± 5
B = iM2 .± 5
@benchmark MultMatr(A,B)
```

Here we use BenchmarkTools package by Jarrett Revels [8], a framework for writing and running groups of benchmarks.

And Octave code for MultMatr function is:

```
pkg load interval
function [t] = MultMatr(n)
  A = infsupdec(rand(n),
    10*rand(n) + 1);
  B = infsupdec(rand(n),
    10*rand(n) + 1);
  tic
  C = A*B;
  t = toc;
end
```

Table 1: Time for interval matrix multiplications

Matrix size, rows	Julia, ms	Octave, ms
10	0.095	13.317
100	111.91	849.61
1000	125870	863340

For Octave we create 10 random interval matrix pairs and calculate the mean experimental time over all multiplications. The results of the first setting are summarized in Table 1. This experiment shows that performance of Julia interval package for that problem is significantly better.

Elementary functions

In our second experiment we compared the times for evaluation of the elementary functions (exp, sin, cos, etc.) for random interval arguments. The design of the experiment is taken from [7].

Table 2: Time for 10^5 evaluations of the elementary functions

Function	Julia, s	Octave, s
exp	0.49	102.7
sin	0.749	147.85
cos	0.638	230.2
tan	0.49	126.13
arcsin	0.858	119.01
arccos	1.132	169.02
arctan	1.318	127.01

The results of the second setting are summarized in Table 2. We may see that these calculations in Julia are almost two orders of magnitude faster.

Plotting

In this section, we will illustrate how to visualize the interval extension of a given function over an interval. The process of splitting the interval into many smaller adjacent pieces for range evaluations of the given function is called *mincing*.

Figures 1- 2 show visualization of mincing process for one function (Julia code was adapted from [10]). For implementation The `IntervalBox` type constructed from an array of `Interval` was used.

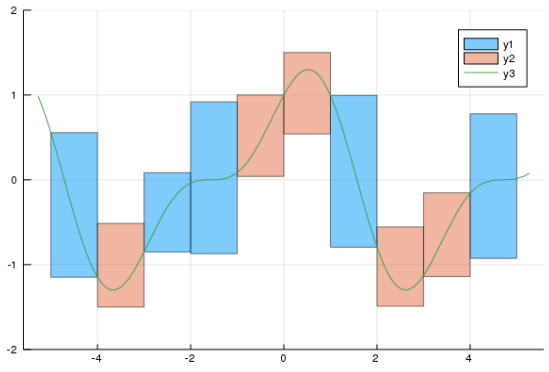

Figure 1: Function $\cos(x) + 0.5\sin(2x)$, 10 sub-intervals.

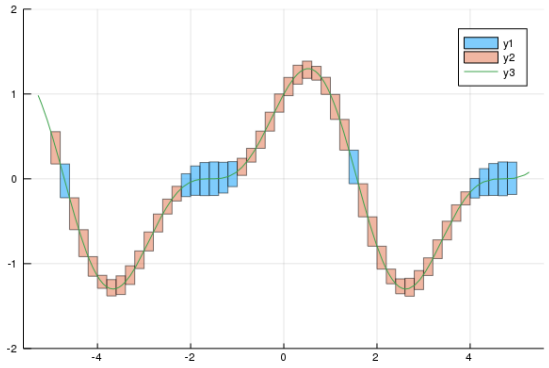

Figure 2: Function $\cos(x) + 0.5\sin(2x)$, 50 sub-intervals.

Acknowledgement

The work was supported by the Russian Foundation for Basic Research, project no. 18-29-03071 mk.

Conclusion

Public available Julia package for interval arithmetic has been investigated. Experimental comparison of Octave and Julia packages for interval arithmetic shows that Julia IntervalArithmetic.jl package is significantly faster then Octave interval package. In addition, the implementation process of interval arithmetic computations in this Julia package is easy and convenient, due to intuitive syntax of the language and the package.

References

[1] O. Heimlich GNU Octave interval package, version 3.2.0, 2015–2018, `https://octave.sourceforge.io/interval/`.

[2] Julia computing newsletter, May 2019. `https://juliacomputing.com/blog/2019/05/03/may-newsletter.html`, 2019.

[3] IEEE Std 1788-2015 — IEEE Standard for Interval Arithmetic. IEEE: Institute of Electrical and Electronic Engineers, IEEE Computer Society, New York, June 2015, `https://ieeexplore.ieee.org/document/7140721`.

[4] L. Benet and D. Sanders. JuliaIntervals.jl Package — Rigorous numerics with interval arithmetic & applications. `https://github.com/JuliaIntervals/IntervalArithmetic.jl`.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A fresh approach to numerical computing. SIAM Review, 59:1, 65–98, 2017.

[6] J. Bezanson, S. Karpinski, V. Shah, and A. Edelman. Why we created julia. `https://julialang.org/blog/2012/02/why-we-created-julia`, 2012.

[7] W. Mascarenhas. Moore: Interval Arithmetic in Modern C++, `https://arxiv.org/pdf/1611.09567.pdf`.

[8] J. Revels. BenchmarkTools Julia package, `https://github.com/JuliaCI/BenchmarkTools.jl`.

[9] S. Rump. Intlab — interval laboratory. In Tibor Csendes, editor, Developments in Reliable Computing, 77–104, Kluwer Academic Publishers, Dordrecht, 1999.

[10] D. Sanders. Métodos numéricos para sistemas dinámicos, `https://github.com/dpsanders/dinamica_nacional`.