RangeEnclosures.jl: A framework to bound function ranges

Luca Ferranti[1], Marcelo Forets[2], and Christian Schilling[3]

[1]University of Vaasa, Finland
[2]Universidad de la República, Uruguay
[3]Aalborg University, Denmark

ABSTRACT

Computing the range of a function is needed in several application domains. During the past decades, several algorithms to compute or approximate the range have been developed, each with its own merits and limitations. Motivated by this, we introduce `RangeEnclosures.jl`, a unified framework to bound the range of univariate and multivariate functions. In addition to its own algorithms, the package allows to easily integrate third-party algorithms, offering a unified interface that can be used across different domains and allows to easily benchmark different approaches.

Keywords

range enclosure, rigorous computing, reachability analysis, interval methods

1. Introduction

Given a function $f : \mathcal{D} \to \mathbb{R}$ over a domain $\mathcal{D} \subseteq \mathbb{R}$, the *range* (or image) is the set $\mathcal{R} = \{y \in \mathbb{R} \mid \exists x \in \mathcal{D} : f(x) = y\}$. In practical applications, we are interested in determining the *interval range* of f, i.e., the smallest interval containing $\mathcal{R}$. Unfortunately, computing the interval range of a multivariate function is NP-hard [9]. For this reason, we practically seek an *enclosure* $\mathcal{E} \supseteq \mathcal{R}$ of the interval range. A standard method to obtain an enclosure is to evaluate the function with interval arithmetic [10], which however often produces a wide overestimation due to issues such as the dependency problem [5] and the wrapping effect [11]. For this reason, different algorithms have been developed over the past decades [13], but each comes with its own strengths and weaknesses. This is visualized in Fig. 1, where the enclosure obtained with plain interval arithmetic, called *natural enclosure*, is compared to the result of a branch-and-bound algorithm. In general, when choosing an algorithm to compute a function enclosure, a trade-off between accuracy and computational efficiency has to be made.

We present `RangeEnclosures.jl`, a Julia [4] package offering a unified framework to bound the range of univariate and multivariate functions. The package comes with built-in solvers but also seamlessly integrates solvers defined in third-party libraries. This allows to easily compare different approaches.

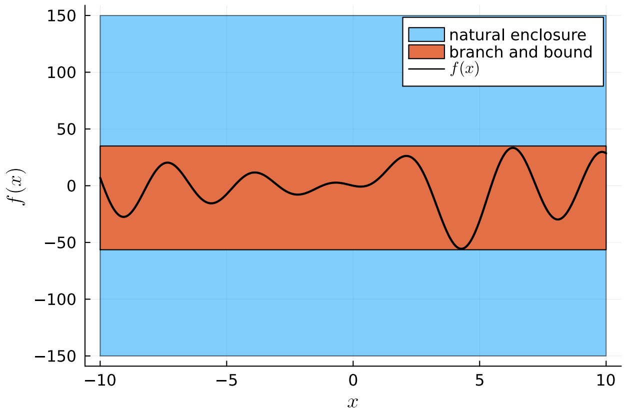

Fig. 1. Two enclosures of $f(x) = -\sum_{k=1}^{5} kx\sin(\frac{k(x-3)}{3})$.

2. A tour through RangeEnclosures

In this section we give a quick overview of the API to bound function ranges. The package offers several solvers for this purpose, such as natural (interval) enclosure, mean-value form [10], Moore-Skelboe algorithm [7], branch-and-bound [8] and Taylor models [3], or polynomial optimization [12]. The full list of implemented solvers can be found in the package documentation[1].

2.1 The enclose API

The *RangeEnclosures* API works through the function `enclose`. The basic usage is via `enclose(f, D, solver; kwargs...)`, where `f` is the function whose range we want to bound, `D` is the domain over which we want to compute the range, `solver` is the solver used to compute the range (if no solver is specified, the package will default to the `NaturalEnclosure` solver), and `kwargs` are possible keyword arguments used by the solver.

In *RangeEnclosures*, the `solver` is an instance of a struct that must be a subtype of `AbstractEnclosureAlgorithm`. If a user wants to add a new solver, they just have to add a new struct, say, `MyEnclosure` and extend the method `enclose`, as the following code snippet demonstrates.

```
import RangeEnclosures: enclose
using IntervalArithmetic: Interval
function enclose(f::Function,
                 D::Union{Interval,IntervalBox},
                 solver::MyEnclosure; kwargs...)
    # solver-specific implementation
end
```

Note that `D` can be of type `Interval` for univariate ($n = 1$) functions or of type `IntervalBox` for multivariate ($n > 1$) functions.

[1]`https://juliareach.github.io/RangeEnclosures.jl/`

2.2 How to use the package

Below we show Julia code to specify the motivating example from above as well as to compute a range enclosure. Here we use the solvers `NaturalEnclosure` and `BranchAndBoundEnclosure`.

```
julia> f(x) = -sum(k*x*sin(k*(x-3)/3) for k in 1:5);
julia> D = -10..10;
julia> enclose(f, D, NaturalEnclosure())
[-150, 150]
julia> enclose(f, D, BranchAndBoundEnclosure())
[-56.4232, 34.9988]
```

Combining different solvers. Sometimes there is no "best" solver, as one solver might give a tighter estimate of the range's upper bound and another solver might give a tighter estimate of the lower bound. In this case, the results can be combined. Consider the function $g(x) = x^2 - 2x + 1$ over the domain $D_g = [0, 4]$. We use the solvers `NaturalEnclosure` and the `MeanValueEnclosure`:

```
julia> g(x) = x^2 - 2*x + 1;
julia> Dg = 0..4;
julia> enclose(g, Dg, NaturalEnclosure())
[-7, 17]
julia> enclose(g, Dg, MeanValueEnclosure())
[-11, 13]
```

A better enclosure could be obtained by taking the intersection of the two results. This can be easily done in one command by passing a vector of solvers to `enclose`:

```
julia> enclose(g, Dg, [NaturalEnclosure(),
                       MeanValueEnclosure()])
[-7, 13]
```

Using solvers based on external libraries. Some of the available solvers are implemented in external libraries. To keep the start-up time of *RangeEnclosures* low, these libraries are not imported by default. To use the corresponding solver, the library needs to be manually loaded. For instance, the Moore-Skelboe algorithm is available upon loading the package `IntervalOptimisation.jl`.

```
julia> import IntervalOptimisation
julia> enclose(g, Dg, MooreSkelboeEnclosure())
[-0.00191952, 9.00109]
```

Multivariate functions. The techniques generalize to multivariate functions. Note that the domain becomes an `IntervalBox` (instead of an `Interval`). For example, consider the bivariate function $h(x_1, x_2) = \sin(x_1) - \cos(x_2) - \sin(x_1)\cos(x_1)$ over the domain $D_h = [-5, 5] \times [-5, 5]$. Fig. 2 visualizes the result.

```
julia> h(x) = sin(x[1]) - cos(x[2]) - sin(x[1]) *
cos(x[1]);
julia> Dh = IntervalBox(-5..5, -5..5);
julia> enclose(h, Dh, BranchAndBoundEnclosure())
[-2.71068, 2.71313]
```

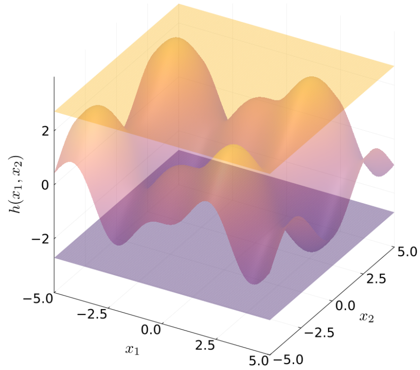

Fig. 2. An enclosure of the bivariate function h.

3. Future Applications

We envision applying the package to the domain of reachability analysis [1, 2]. *RangeEnclosures* currently only supports functions with univariate range. To represent multivariate ranges as convex and non-convex sets, we plan to use `LazySets.jl` [6].

Acknowledgments

This research was partly supported by DIREC - Digital Research Centre Denmark and the Villum Investigator Grant S4OS.

4. References

[1] Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1), 2021. doi:10.1146/annurev-control-071420-081941.

[2] Matthias Althoff, Dmitry Grebenyuk, and Niklas Kochdumper. Implementation of Taylor models in CORA 2018. In *ARCH*, pages 145–173, 2018. doi:10.29007/zzc7.

[3] Luis Benet, Marcelo Forets, David P. Sanders, and Christian Schilling. TaylorModels.jl: Taylor models in Julia and its application to validated solutions of ODEs. In *SWIM*, 2019.

[4] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Rev.*, 59(1):65–98, 2017. doi:10.1137/141000671.

[5] Luiz Henrique de Figueiredo and Jorge Stolfi. Affine arithmetic: Concepts and applications. *Numer. Algorithms*, 37(1-4):147–158, 2004. doi:10.1023/B:NUMA.0000049462.70970.b6.

[6] Marcelo Forets and Christian Schilling. LazySets.jl: Scalable symbolic-numeric set computations. *Proceedings of the JuliaCon Conferences*, 1(1):11, 2021. doi:10.21105/jcon.00097.

[7] Eldon Hansen and G William Walster. *Global optimization using interval analysis: revised and expanded*, volume 264. CRC Press, 2003.

[8] Sam Karhbet and Ralph Baker Kearfott. Range bounds of functions over simplices, for branch and bound algorithms. *Reliab. Comput.*, 25(7):53–73, 2017.

[9] Vladik Kreinovich. Range estimation is NP-hard for ε^2 accuracy and feasible for $\varepsilon^{2-\delta}$. *Reliab. Comput.*, 8(6):481–491, 2002. doi:10.1023/A:1021368627321.

[10] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. doi:10.1137/1.9780898717716.

[11] Arnold Neumaier. *The wrapping effect, ellipsoid arithmetic, stability and confidence regions*. Springer, 1993. doi:10.1007/978-3-7091-6918-6_14.

[12] Tillmann Weisser, Benoît Legat, Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. Polynomial and moment optimization in Julia and JuMP. In *JuliaCon*, 2019.

[13] Julius Žilinskas and Ian David Lockhart Bogle. A survey of methods for the estimation ranges of functions using interval arithmetic. In *Models and Algorithms for Global Optimization: Essays Dedicated to Antanas Žilinskas on the Occasion of His 60th Birthday*, pages 97–108. Springer, 2007. doi:10.1007/978-0-387-36721-7_6.