RangeEnclosures.jl: A framework to bound function ranges

Luca Ferranti\textsuperscript{1}, Marcelo Forets\textsuperscript{2}, and Christian Schilling\textsuperscript{3}

\textsuperscript{1}University of Vaasa, Finland
\textsuperscript{2}Universidad de la República, Uruguay
\textsuperscript{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 \( f : D \to \mathbb{R} \) over a domain \( D \subseteq \mathbb{R} \), the range (or image) is the set \( \mathcal{R} = \{ y \in \mathbb{R} | \exists x \in D : f(x) = y \} \). 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 \cite{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 \cite{10}, which however often produces a wide overestimation due to issues such as the dependency problem \cite{5} and the wrapping effect \cite{11}. For this reason, different algorithms have been developed over the past decades \cite{13}, but each comes with its own strengths and weaknesses.

We present RangeEnclosures.jl, a Julia 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.

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 \cite{10}, Moore-Skelboe algorithm \cite{7}, branch-and-bound \cite{8} and Taylor models \cite{3}, or polynomial optimization \cite{12}. The full list of implemented solvers can be found in the package documentation\footnote{https://juliareach.github.io/RangeEnclosures.jl/}.

2.1 The enclose API

The RangeEnclosures API works through the function \texttt{enclose}. The basic usage is via \texttt{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, \( \texttt{solver} \) is the solver used to compute the range (if no solver is specified, the package will default to the \texttt{NaturalEnclosure} solver), and \( \texttt{kwargs} \) are possible keyword arguments used by the solver.

Note that \( D \) can be of type \texttt{Interval} for univariate \((n = 1)\) functions or of type \texttt{IntervalBox} for multivariate \((n > 1)\) functions.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\caption{Two enclosures of \( f(x) = - \sum_{k=1}^{5} kx \sin\left(\frac{k(x-3)}{3}\right) \).}
\end{figure}
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
julia> f(x) = -sin(x[1]*x[1]+sin(x[2]*(x[2]-3)/3)) for x in 1:5;
julia> D = [-10.10];
julia> enclosure(f, D, NaturalEnclosure())
[-5.64232, 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
julia> g(x) = x^2 - 2x + 1;
julia> Dg = 0.4;
julia> enclosure(g, Dg, NaturalEnclosure())
[-7.17]
```

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 `enclosure`:

```julia
julia> enclosure(g, Dg, [NaturalEnclosure(), MeanValueEnclosure()])
[-11.13]
```

Using solvers based on external libraries. Some of the available solvers are implemented in external libraries. To keep the startup 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
julia> import IntervalOptimisation
julia> enclosure(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\( h(x_1, x_2) = \sin(x_1) - \cos(x_2) - \sin(x_1) \cos(x_2) \) over the domain \( D_h = [-5, 5] \times [-5, 5] \). Fig. 2 visualizes the result.

```julia
julia> h(x) = sin(x[1]) - cos(x[2]) - sin(x[1]) * cos(x[2]);
julia> Dh = IntervalBox(-5.5, -5.5);
julia> enclosure(h, Dh, BranchAndBoundEnclosure())
[-2.71068, 2.71313]
```

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`\([4]\).

Acknowledgments

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

4. References


