JInterval Library: Principles, Development, and Perspectives

Dmitry Nadezhin[1] and Sergei Zhilin[2]

[1] Oracle Labs, Zelenograd, Russia
[2] Altai State University, 61, Lenin ave., 6560049, Barnaul, Russia
dmitry.nadezhin@oracle.com, sergei@asu.ru

Keywords: interval computations, Java, library

JInterval [1] was started in 2008 as a research project to develop a Java library for interval computations. The library is intended mainly for developers who create Java-based applied software. The design of the JInterval library was guided by the following requirements ordered by descending priority:

1. *The library must be clear and easy to use.* No matter how wonderful a software tool is, it will be hardly accepted by developers if it is not transparent and easy to use.

2. *The library should provide flexibility in the choice of interval arithmetic for computations.* The user must be able to choose interval arithmetic (classical, Kaucher, complex rectangular, complex circular, etc.) and to switch one arithmetic to another if they are compatible. Syntactic differences between the use of this or that arithmetic should be minimized.

3. *The library should provide flexibility in extending its functionality.* The library must be layered functionally. Four layers should be defined: interval arithmetic operators, elementary interval functions, interval vector and matrix operations, and, finally, high-level interval methods, such as solvers of equations, optimization procedures, etc. Architecture of the library must allow for extensions at every layer, starting from the bottom one.

4. *The library should provide flexibility in choosing precision of interval endpoints and associated rounding policies.* The choice of interval endpoints representation and the rounding mode could allow the user to tune accuracy and speed of computation depending on the problem he solves.

5. *The library must be portable.* Cross-platform portability of the library is one of its major strengths, being a key distinction over its closest competitors. To a large extent, this requirement is ensured by the choice of the Java technology built on the principle "write once, run anywhere". However, the design must adhere to certain restrictions on practical implementation of this requirement.

6. *The library should provide high performance.* In the era of multicore and multiprocessor systems, a prerequisite for high performance is the ability to use the library safely in a multithreaded environment.

Achieving the required flexibility leads to widening the scope of the library, which results in a vast and obscure design, contrary to the simplicity requirement. To avoid this contradiction, and to preserve clarity of the library, the overall architecture needs to be transparent and consistent. This is done due to approproate design decisions. Methods for interval classes, regardless of interval arithmetic and of the internal representatiuon of intervals are unified. Intervals are considered as immutable objects. The user is provided with a simple interface to manage rounding policy and interval endpointd representation.

At the moment, `JInterval` provides a user with several interval arithmetics (classical real, extended Kaucher, complex rectangular, complex circular, complex sector, complex ring), interval elementary functions, interval vector and matrix operations, as well as a few methods for inner and outer estimation of the solution sets to interval linear systems.

A number of applications have been built using `JInterval library`. A collection of plugins is developed for the data mining platform KNIME. The collection include interval regression builder, outlier detector, ILS solver, etc. Another example is mobile applications, where `JInterval` is used for position uncertainty modeling in hybrid navigation.

The experience of `JInterval` implementation and usage taught us several lessons, and further development of `JInterval` will be governed by the following principles:

1. Java language has a lot of advantages, but its syntax is not expressive enough for computational programming. Scala language (fully compilant with JVM) is considered as a basic language for a new `JInterval` implementation.

2. Presently, `JInterval` is not compilant with the project of interval arithmetic standard IEEE P1788. A new implementation will be adjusted for P1788.

3. To achieve high performance, `JInterval` will be equipped (using Java Native Interface) with optional plugins for machine-dependent implementation of high precision arithmetic and interval linear algebra algorithms.

4. For applied software developers, a rich content of the fourth layer of the library (high-level interval analysis methods) is one of the most valuable issues. Therefore the replenishment of `JInterval` with solvers of algebraic and differential equations, interval optimizers, etc., remains the foreground task.

References:

[1] Java Library for Interval Computations, `http://jinterval.kenai.com`.