Open-source Contributions to Julia, a Scientific Programming Language, for Mathematical Optimisation

13th April 2016

In the world of programming languages, those that have good runtime performance (like C or Fortran) rarely have nice dynamic features, whereas languages that offer a better programmer productivity (like MATLAB or Python) lack performance. Julia is a newcomer in the field of scientific and technical computations that has a productivity-oriented syntax, but closely matches the performance of lower-level languages like C.

Julia also has a strong community for mathematical optimisation, named JuliaOpt. It has developed a series of packages to propose a state-of-the-art environment for mathematical programming, mainly several interfaces to existing solvers (MathProgBase.jl), and two modelling layers (JuMP.jl for general cases, Convex.jl for convex programming). Another JuliaOpt package, Optim.jl, provides full Julia implementation of optimisation algorithms, which can better handle Julia's peculiarities than external solvers (in particular, it can exploit properties in linear algebra not exposed through standard libraries, which could improve performance and scaleability).

This environment can be further developed with other contributions, such as:

- Implementation of missing optimisation algorithms in Optim.jl, especially for constrained optimisation or convex programming, such as interior point or dual methods, or sequential techniques (SQP, SLP), or less-known algorithms. The performance of the implementation should be compared to existing solutions in other environments.

- Generic modelling for optimisation under uncertainty, as an extension to JuMP.jl. Currently, the two main paradigms to add uncertainty into mathematical programs are implemented in Julia (stochastic programming with StochJuMP.jl, robust programming with JuMPeR.jl); however, practitioners have a hard time to switch from one paradigm to another and see which one is the best suited to their needs. The goal would be to see how far both paradigms can share a modelling interface.

- DAE modelling as an extension to JuMP.jl. Many problems in engineering use differential equations, including when optimising some parameters. However, there is no modelling layer for these differential equations to date which is integrated in Julia packages. Other modelling packages such as Pyomo have such functionalities (with an implementation based on the discretisation of the equations), which could be ported to JuMP.jl.

Any of those three subjects is a candidate for a master's thesis.

The proposed master's theses would take the form of open-source contributions to existing Julia packages or to new open-source libraries. Previous knowledge of the language is appreciated, but is not mandatory.

Feel free to propose other kinds of contributions to the Julia environment!