Optimization: Julia and JuMP

Software for Analytics, Day 2

Instructor: Emma Gibson

August 28, 2019

Agenda for today

- Introduction to Julia
- Julia practical
- Introduction to optimization
- Linear optimization practical
- Mixed-integer optimization practical
- Project work

Software

Essential installations for today:

- Julia
- Gurobi
- Jupyter
- Julia packages

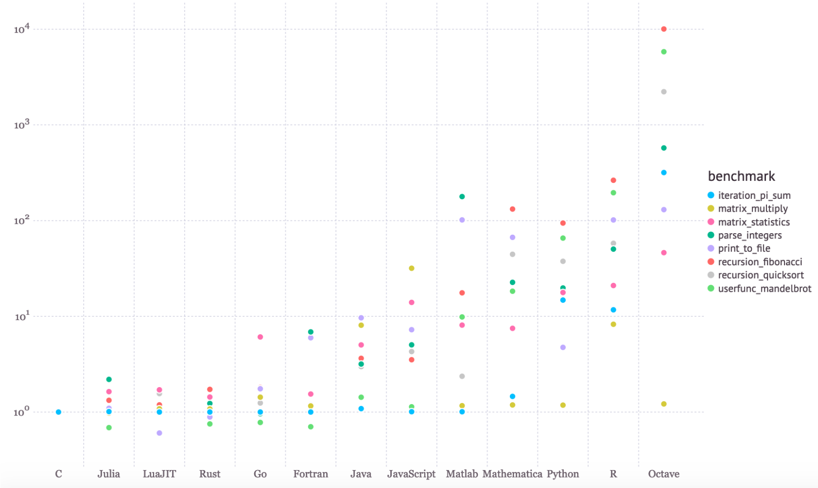
If you made it through the pre-assignment, well done!

Source: `xkcd.com`

Julia

- High-level, high-performance, open-source dynamic language for technical computing
- Developed at MIT over the last decade
- Gives users the convenience of a dynamic language without compromising on efficiency

Julia benchmarks



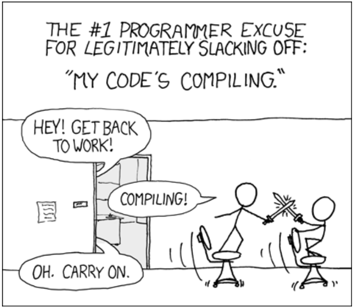
Source: `https://julialang.org/benchmarks/`

Dynamic vs. compiled languages

Compiled languages like C are optimized for speed, but they are not very flexible (no interactive coding).

Dynamic languages like Python are a more popular choice for data scientists, but they are slower.

Julia has the best of both worlds: code is compiled just before execution for speed and flexibility.



Source: `xkcd.com`

Learning Julia

The best way to learn a new coding language is to practice!

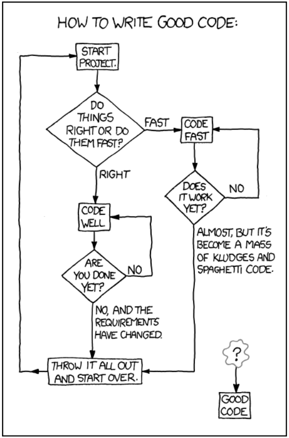
Official documentation
Julia is a new language that is actively maintained and developed. Help files and documentation for the latest version are available at `https://docs.julialang.org/en/v1/index.html`

Other sources of information
Sites like `stackoverflow.com` can be great resources, but may be out-of-date when a new version is released.

Learning Julia

Things to remember:

- Comments are your friend
- Save regularly
- Use some kind of version control
- Check license restrictions before trying something new
- Ask questions / be kind



Source: `xkcd.com`

Why optimization?

Optimization is a powerful tool for solving real-world problems. You've probably already encountered optimization several times today:

- Logistics and transportation
- Supply chain and manufacturing
- Revenue management
- Advertising
- Machine learning and artificial intelligence

Optimization structure

Optimization problems have three main components:

- Decision variables (the solution we want to calculate)
- An objective function (to measure how good a solution is)
- Constraints (restrictions on the type of solution we want)

Example:

$$
\begin{aligned}
&\min_{\mathbf{x}} f(\mathbf{x}) \\
&\text{subject to:} \\
&\quad a_1(\mathbf{x}) \leq b_1 \\
&\quad a_2(\mathbf{x}) \geq b_2 \\
&\quad a_3(\mathbf{x}) = b_3 \\
&\quad \mathbf{x} \geq 0
\end{aligned}
$$

Optimization terminology

- Minimize or maximize – do we want f(x) to be high or low?
- Feasible solution: a solution that does not break any constraints
- Infeasible: there is no solution that satisfies all the constraints (e.g. $x \leq 0$ and $x = 2$)
- Optimal solution: the best feasible solution
- Unbounded: best objective value is infinite (e.g. $\max x$ s.t. $x \geq 0$)

Types of problems

Optimization problems can be classified into a few different categories based on the type of functions and variables used in the model:

- Linear: everything is linear/affine
- Quadratic: quadratic objective and constraints
- Conic: variables and constraints lie in cones
- Convex: constraints and objective are convex
- Nonlinear: everything else
- (Mixed) Integer: (some) variables restricted to integer values

Linear optimization

In a linear model, the objective and constraints are all linear functions of the decision variables:

$$\begin{aligned}
&\min_{\mathbf{x}} c_1x_1 + c_2x_2 + c_3x_3 \\
&\text{subject to:} \\
&\quad a_1x_1 + a_2x_2 + a_3x_3 \leq b \\
&\quad \mathbf{x} \geq 0
\end{aligned}$$

Linear optimization

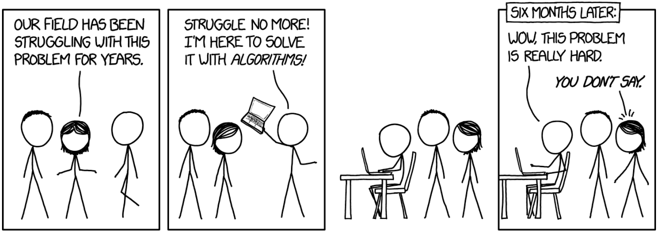
Linear models are simple but powerful. Over the years, many researchers have developed efficient algorithms for solving linear problems.

- Simplex algorithm (1947)
- Ellipsoid method (1970)
- Interior point method (1984)
- ...

However, solving the model is just one part of the process!

Optimization steps

1. Identify a problem (human)
2. Create a mathematical model (human)
3. Collect data (human)
4. Optimize the model (algorithm)
5. Validate, improve (human)



Source: xkcd.com

Optimization solvers

It takes a great deal of work to translate real-world problems into something that a computer can understand. Luckily, you don't need to be an expert mathematician or programmer to solve optimization models!
There are many *solvers* available to find solutions to your models.

Optimization solvers

We don't need to understand the algorithms that optimization solvers use, but it helps to know a little bit about each solver so that we can select the best one for our problem.

Solver criteria:

1. Type of problem (linear, quadratic, integer etc.)
2. Software constraints (operating system, dependencies)
3. Hardware constraints
4. Cost and licensing
5. Documentation, language, interface

Interacting with solvers

Solvers need us to write out our problem in a way that matches their inputs. In reality, we don't want to write a new version of our problem for every solver. Instead, we'll use an Algebraic Modeling Language (AML).

- AMLs let us write human-readable code to describe our variables and constraints.
- Constraints are translated into the correct inputs required by the solver.
- The solver outputs are translated into a familiar format that we can use to access the results.

JuMP

Today we'll learn to use JuMP, an AML that runs in Julia.

Why JuMP?

- JuMP is free and open-source, with performance comparable to commercial options
- Integrates with many commercial and free solvers
- It's embedded in Julia, so we don't need a separate language for optimization problems
- Developed by ORC students and often used for ORC research and classes