waters

The Infancy of Julia: An Inside Look at How Traders and Economists Are Using the Julia Programming Language

Several banks, vendors and a regulator discuss how they are using the Julia programming language to improve their big data analytics capabilities.

Anthony Malakian
@a_malakian

07 Nov 2016

Seven years ago, four individuals came together to create a fast and expressive programming language to compete with the likes of R, Matlab, Python and about three dozen other dynamic tools. Their work gave birth to the Julia programming language. Anthony Malakian takes an inside look at how some finance firms and economists are using Julia, and examines why 2017 could be a year of significant growth for the upstart.

The importance of data and speed to information can't be understated in the capital markets, and while this is nothing new, the demand for bigger data and faster systems increases exponentially with each passing year. Whether it's an algorithmic trader competing with the "flash boys," a commodities trader picking through a sea of unstructured data, a risk manager looking for red flags, a regulator searching out bubbles and anomalies, or a vendor looking to release a real-time analytics solution, speed and the ability to break down large datasets are vital. And this all starts with the programming language.

Back in 2009, Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral Shah set out to create a programming language that combined the best characteristics of C, Matlab, Java, Ruby, Python, Perl and R—easy enough, right? As Shah describes the group's thinking back then, "it kind of sucked" that in order to get speed and ease-of-use for statistical analysis, you'd have to write in one language and then rewrite it in another—the so-called two-language problem.

"Is it a law of nature that you cannot have performance and productivity at the same time, or is it just that compiler-language technology hasn't yet evolved to that point? We believed it was possible to have usability and performance in the same package. We started the project and named it Julia at that point," Shah tells *Waters*. The Julia founders, along with Keno Fischer, have since launched Julia Computing, a consultancy to help firms looking to use the Julia language.

Open to the People

Julia is an open-source programming language that allows users to create numerical algorithms that outperform other similar algos using Just In Time (JIT) interpretive languages. It is easily deployed on cloud containers and is relatively easy to use, as it drastically cuts down on the lines of code

needed. The year 2012 saw the first release of the language —version 0.1—and 0.5 was released in August this year. While the language's community is still small, with a 1.0 release scheduled for next year, there's hope that it will grow rapidly.

"Usually, if you have something in R or Matlab and you want to make it go fast, you have to re-translate it to C++, or some other faster language; with Julia, you don't-it sits right on top." -- Keith Lubell

It is already gaining Wall Street's attention, notes Keith Lubell, CTO at investment bank Berkery Noyes. "What really excites us is that it's interesting that you can write high-level, scientific and numerical computing but without having to re-translate that," he says. "Usually, if you have something in R or Matlab and you want to make it go fast, you have to re-translate it to C++, or some other faster language; with Julia, you don't—it sits right on top."

Julia is still very much in its infancy, and, as such, it will have to grow in many areas in order to take over as the programming language of choice for big data analytics. Most importantly, its community needs to expand to improve the packages and tools available when compared to other, more well-established data analytics languages.

This feature looks at how some firms in finance and in economics are using Julia, where improvements need to be made, and what's ahead. Version 1.0 of the language hasn't been released yet, but as big data analytics continues to take on greater importance in the capital markets, Julia is poised to become one of the leading tools deployed by developers

and programmers at banks, hedge funds, regulators and vendors. If you're looking at machine learning, data mining, distributed and parallel computing, and/or scientific computing, 2017 could be the year that you're introduced to Julia.

From the Ground Up

BestX Ltd. is a London-based software vendor that launched earlier this year. It provides independent trading analytics that allows users to define, achieve and demonstrate best execution in foreign-exchange (FX). It was started by three Morgan Stanley alums: Pete Eggleston, Ollie Jerome, and Aman Thind. BestX unveiled its first product—a tool for post-trade transaction-cost analysis (TCA)—in September and made a conscious decision to use the Julia programming language to build the platform.

Thind used to lead Morgan Stanley's quant solutions innovations group and he found that quants would write their analytics in either Excel VBA, C# or Matlab. When he took those technologies from the quants and ran them in the firm's production environment, he almost always had to rewrite the entire analytics in C or C++ because they were too slow in the original languages and didn't provide efficient interoperability with other platforms.

BestX relies on providing real-time trading analytics with high volumes of tick data. As a result, Thind felt that Julia was the best language on which to build the platform. The BestX platform can run almost all of its analytics in five milliseconds per trade without any special optimizations required, according to Thind, who says he began looking at Julia while at Morgan Stanley.

"What resonated with me is that you could get the expressiveness of Python while getting the performance of C. That effectively means that you could bridge the gap between

proofs of concept and production deployments without complete rewrites," Thind says. "If we were doing this in Python, we'd really have to think about which routines are going to be our bottlenecks and then either try to write them using C plugins or use some of the specialized libraries or other techniques. With Julia, we don't have to give any such consideration as the base level performance is usually good enough."

Berkery Noyes' Lubell notes similar performance benefits as it's been toying with a Julia proof-of-concept.

The investment bank's first- generation analytics platform, which was deployed in 2008, was based on the Microsoft SQL Server and, as such, all of its data at that point was in SQL. Berkery Noyes, which provides mergers and acquisitions advisory and financial consulting services to middle-market firms, still uses this SQL system today to do all of its market analysis, according to Lubell, but it is slow. So two years ago they started a project to build a "super search engine" that would allow the firm to investigate all the data it collects from a multitude of sources, including marketing data and web data.

"Because of the disparate nature of the data, and the correlations we wanted to make, we realized that a SQL Server-based system was not going to cut it in terms of performance and ease of development," he recalls. "In the past year, we have brought online a NoSQL system that uses Couchbase and Elasticsearch to create an ad-hoc data discovery and analytics dashboard."

Lubell says that at this point, the solution does not do the complex calculations for market analysis that Berkery Noyes employs to determine market size. In addition, the investment bank still has a large semantic topology it uses to determine related business models. "Our old SQL based solution does

not perform well on these two tasks; the topology calculations take hours," he says.

So as a proof of concept, Berkery Noyes has re-engineered its market sizing analysis into a Julia engine.

"In my initial tests, Julia performs 12 times faster than our SQL solution, and since we rerun this model many times, feeding it different slices of the market, it can add up to a substantial performance boost," Lubell says. "Such a performance boost is a big benefit and we could then open up these live analytics to one of our websites, rather than having to do the calculations in batch and store them for fast recall by our web servers."

The Change

Sven Duve, a commodities trader at Zug, Switzerland-based AOT Trading AG, isn't a hardcore programmer by his own admission. Duve says he was a long-time user of the R programming language. He started looking at Julia because of its speed, although its usability played a key role in his making the switch.

As an example, AOT uses swing options, a delivery contract used in the energy markets that allows a client to schedule their offtake of energy for if and when they need it. Duve's model, powered by Julia, allows him to optimize that offtake to provide the most benefit to his clients for scheduling energy uptake. He says they also use Julia to run Monte Carlo simulations and options valuations.

He's able to build all of this as a trader, rather than being a programmer. "As a non-professional coder—I'm a practitioner using the outputs more than I'm creating them—what was useful for me was the easiness of the language," Duve says. "The coding is quite intuitive. It's a functional language, which I like. We have all the freedom to create our own objects in the language."

Like Duve, David Weiss, head of software development for risk solutions at Conning, Inc., recently decided to make a change to Julia.

Conning, which provides analytics solutions to insurance companies, was previously using the K programming language, which was commercialized by Kx Systems. Weiss and his team decided to switch to Julia for its next-generation products because of its speed and ease of use.

"Like K, Julia allows us to achieve extremely high productivity for our software engineers combined with excellent run time performance on today's CPUs," Weiss says. "Julia does this without requiring an array-oriented style of programming, although it incorporates a full suite of array style operations, as well, and does so in a language that is very accessible and familiar."

Need for Speed

Talk to almost anyone who is currently using Julia or is experimenting with the language and they'll likely tell you that speed was the main reason for looking at this nascent offering. According to Julia Computing, when testing seven basic algorithms, Julia is 20 times faster than Python, 100 times faster than R and 93 times faster than Matlab.

Again, these performance improvements come, in part, from decreased lines of code and the fact that rewriting in a different language isn't necessary—the two-language problem. Perhaps the most notable implementation of Julia—and independent proof of its performance—is that of the Federal Reserve Bank of New York (FRBNY) and how it's using the language to transform its estimated model of the US economy.

About two years ago, the founders of consultancy QuantEcon —Thomas Sargent, who won the 2011 Nobel Prize in Economics and who is a professor of economics at New York

University, and John Stachurski, professor of economics at Australian National University in Canberra—approached the New York Fed about translating the institution's dynamic stochastic general equilibrium (DSGE) model to Julia.

FRBNY's DSGE model—which is used for both forecasting and policy analysis—was run on Matlab. The idea of the project was to see if a relatively large model like DSGE could be easily translated to Julia and whether it could run significantly faster. FRBNY jumped at the chance to work with QuantEcon, says Marco Del Negro, assistant vice president of macroeconomic and monetary studies function for the New York Fed. "That sounded like a great idea to me because we were looking to move away from Matlab-both for cost and for reasons of speed," Del Negro tells *Waters*. "We wanted to develop our models further and explore new things."

In the first stage of the process FRBNY's project team—which included Del Negro, Marc Giannoni, Pearl Li, Erica Moszkowski and Micah Smith—found that Julia reduced the model's running time to 1/10th to 3/4th that of the Matlab code. According to FRBNY, the Metropolis-Hastings sampling—a Markov chain Monte Carlo method for obtaining a sequence of random samples from a probability distribution—is the most time-consuming step; DSGE.jl ran approximately 10 times faster than the Matlab code. DSGE.jl also cut the lines of code needed by almost 50 percent compared to Matlab.

It should be noted, however, that these improvements could stem from the fact that they were building the model from the ground up using new technology. Even the New York Fed noted that "our Matlab code suffers from many inefficiencies due to its long, cumulative development, and support for a plethora of models and features," and those issues were known and targeted when designing the Julia-based model. Still, measurement algorithms gensys and kalman_filter functions—which have "relatively little redesign and

optimization" needed when compared to the Matlab code, and are thus similar whether using Julia or Matlab—saw a reduction of 1/5th to 3/4th in computing time in favor of Julia. (See chart.)

JULIA VS. MATLAB

TEST	MATLAB (14a)	JULIA (0.4.0)
gensys	1.00	0.17
solve	1.00	0.09
kalman_filter	1.00	0.75
posterior	1.00	0.26
csminwel	1.00	0.33
hessian	1.00	0.23
metropolis_hastings	1.00	0.11

Source: Federal Reserve Bank of New York

The Federal Reserve Bank of New York published its initial findings from the first phase last year. The Julia project took a backseat for a little while so that the team could focus on other projects, but the Fed has begun to push forward with the second stage of the project, which will address the forecasting capabilities of the model. Del Marco says the institution wants to "complicate" the DSEG model going forward. As they add more data and levels to it, the Fed thinks that Julia is better positioned to keep up with those complexities using a run time of a day or two, while using Matlab it could take a month to spit out the desired information, although that is yet to be proven.

Community Builder

The greatest knock against Julia is its "community" of developers and programmers. R is almost 25 years old. Python is exactly 25 years old. Matlab is over 30 years old. As such, they have robust communities that are constantly adding new packages, library support and tools. And there is

a worry that one of these languages could simply evolve and improve past Julia.

"The Julia user community is also pretty small at the moment, and there is a fear that a language with a larger user base like Python could potentially improve its performance, and then leave Julia in the dust," says Berkery Noyes' Lubell. However, he adds, as the investment banking industry takes on increased compliance demands and competition from financial technology organizations, firms will have to at least give Julia a look.

In September this year, the Julia language project released Julia 0.5, which included more than 1,100 packages, an increase of 57 percent since version 0.4, which was released in the fall of 2015.

Julia Computing's Shah estimates that there are about 150,000 users of Julia worldwide today. He says what needs to happen next is to move that user base to 1.5 million. The release of version 1.0, planned for JuliaCon 2017 at the University of California, Berkeley, nine months from now, will help in that push; version 0.6 will come in the next few months, Shah says, and that will be the final release before v1.0.

"That will be a signal from the community that this software is now ready for exponential growth," Shah says. For current iterations, support lasts for six to 12 months; 1.0 will be supported for five years, he adds.

To help build the community, AOT's Duve also says he is hopeful that Julia will have something similar to RStudio, which develops free and open tools for R and enterprise-ready professional products for teams to scale and share work, according to the provider. Duve also notes that he thinks Julia is being built and released in a way that will encourage more enterprise customers to take a look at it,

because it is user-friendly and was built in a structured way that's easy for finance and economics professionals to use.

"To compare it to R, the way that they've developed the whole language—the way they've approached it—is that it's much more structured," he says. "In the early stage of Julia, it seems much more structured with the approach that the community has taken on Julia than it ever was for R."

In addition to its community, Julia will have to further develop its automatic memory management; while it is fast, it is still not fast enough for most high-frequency traders. Shah says that when v1.0 is released, users who want to record "every single millisecond of everything that comes in, which Julia is not fully ready to do yet. But it should be able to give you that level of granularity, as well," with 1.0.

Also, while the language is easily deployed on cloud containers—a huge selling point for the language—interfacing with the cloud has its perils, says BestX's Thind. "On the cloud services, it's very easy to deploy multiple processes across your entire cluster to do something like MapReduce, if you wanted to; that's very easy and efficient to do in Julia," he says. "But if you wanted to interface with, say, some specific services like DynamoDB or RedShift, the application programming interfaces (APIs) in Python are much more mature and robust than native Julia libraries."

Potential users looking to take the plunge with Julia will have to carefully consider what they are using the language for and what else they will need to build alongside it.

"Any firm that is trying to do this needs to be very aware of what are the different functionalities that you'll want to use this language for and whether it provides native support for at least 50 percent of those," Thind says. "You could perform the rest through its C and Python APIs, which are completely seamless. However, if a bulk of your implementation ends up

needing other languages to compensate, you should either wait before taking the plunge or engage Julia Computing for dedicated support for missing functionalities."

Long Road Ahead

In RedMonk's rankings of programming languages, Julia has yet to crack the top 50, slotting in at 52 in the most recent rankings in June. IEEE Spectrum ranks it 33rd; the TIOBE Index for October 2016 places it just outside the top 50. Julia did not crack any of Stack Overflow's top results in this year's developer survey. Obviously, Julia has a long way to go before it can be viewed as a serious competitor to overtake the likes of R, Python, Matlab and other data-analysis languages.

Some will play a game of wait and see. Louis Lovas, director of solutions for OneMarketData, which provides solutions for time-series data management, real-time event processing and analytics, says that the vendor started a preliminary investigation of Julia over a year ago. He says the jury is still out on Julia in quant finance, and as a result the vendor will not invest further time until it sees usage pick up. Lovas adds that currently, the de facto standard numerical programming language when talking to banks and other finance firms is Python/NumPy.

However, others are going to start their exploration in 2017. Corbin Kidd, CTO at DV Trading, which employs high-frequency trading strategies, is just starting to look at the language. "Julia is an exciting language that's coming up right now, but it's in its infancy," he says. "We're very excited to see how that will mature and help us. Our firm is quite reliant on R, so it will potentially be a big improvement for us. That's a place that we'd like to find talent to help us improve."

And Conning's Weiss says the year ahead will be a big one for Julia. "As Julia marches forward to its release 1.0 during

2017, the community is looking forward to leading-edge support for multi-threading and also a state-of-the-art integrated development environment (IDE). With those done right, you'll see Julia take off in a big way. It's going to be an exciting year," he says.

Viral Shah acknowledges that Julia isn't a "unicorn" in that it can't solve everything, and adoption will take time. Finance and economics are the biggest sectors poised to pounce on the language, but there are still plenty of CTOs and quants who haven't heard of Julia yet or who have only heard about it in passing or by reading about it in blogs. Julia has a lot of promise, but it still has plenty of challenges.

The growth of this language will depend on the community around it. After all, it takes a village to raise a child.

What's in a Name

You could say that the Julia programming language came to fruition at least in part thanks to the game of Ultimate Frisbee. While at the University of California, Santa Barbara (UCSB), Viral Shah and Stefan Karpinski both ran their computer science department's Ultimate Frisbee team for several years. But in addition to their love of the game, they also started collaborating on machine learning algorithms.

At the same time, Alan Edelman was a professor of applied mathematics at MIT—where he still serves today—and he was on Shah's thesis committee, since his PhD project on parallel computing was in collaboration with MIT.

After Shah completed his PhD at UCSB in 2007, he became a senior scientist at Interactive Supercomputing, which was founded by Edelman, and it's where Jeff Bezanson was working as a senior software engineer. It was at this point that the four future Julia creators were in the same orbit. So, sure, it's a bit of an exaggeration to say that Ultimate 'Bee created Julia, but it makes for a fun story.

In September 2009, Microsoft bought Interactive Supercomputing. It was at this point that Edelman, Bezanson, and Shah began discussing the idea of creating a new language for numerical computing. Karpinski and Shah were having the same discussion.

Shah had to move back to Bangalore, India, after the acquisition and Karpinski moved to New York. Edelman and Bezanson were at MIT in Cambridge, Massachusetts. But in the midst of all this, an email thread was created to connect the four and "the sparks flew," recalls Shah. In just a few days, Karpinski had the first lines of code written. It was October 2009.

"Jeff, Stefan, and I were all in different geographies and hence the whole language design was carried out as email conversations," Shah says. "In fact, Stefan had not met Jeff and Alan for the first year of our collaboration. As I look back at the archives, we had over 10,000 messages from the early days when it was just the three of us programming."

At first, Jeff wrote a simple parser and interpreter and they felt their way around the syntax, ease of use and other issues that needed to be addressed, Shah says. In a few weeks, they had enough of a working language to start writing small programs, libraries and so on. "While Jeff was busy with the language and the runtime, Stefan

and I wrote a standard library as well as the math and scientific libraries," he says.

"Performance was crucial in our minds from day one," Shah continues. "We also knew that we could never write a critical mass of capabilities fast enough if we had to write it all in a low-level language like C. Thus we ate our own dog food and wrote all the libraries in Julia itself, calling well-established C and Fortran libraries in certain cases. At some point, Jeff replaced the interpreter with a compiler that used LLVM to translate Julia programs into efficient executable code. While we always designed Julia to be fast from day one, this is when we started getting a real taste of the speed potential."

On Feb. 14, 2012, Valentine's day, the group published their "**Why we created Julia**" blog post, and the language went viral, says Viral.

"Suddenly the project became a living, breathing open-source project. It took one whole year from there to release version 0.1, on Feb. 14, 2013," he says. "As I look back, over 100 contributors participated in creating the Julia 0.1 release. That number has now grown to over 500 contributors for the latest 0.5 release. The most exciting thing is that this journey made Julia more than just a language; it has become a common platform for computer scientists, engineers, quants, scientists, statisticians, data scientists and other technical folks to come together and collaborate in the form of code."

So why the name, Julia? Is it named after a famous scientist? A wife or ex-girlfriend? Perhaps they're huge Julia Roberts fans? Shah just laughs and says there's no great story there: A friend of Bezanson suggested the name and the group fell in love with it and ran with it.

Perhaps the Ultimate Frisbee angle is the better story to tell, after all.