Introduction to the Julia language

Outline

- motivations
- julia as a numerical language
- types and methods
- about performance

Why yet another MATLAB like language ?

- *scripting languages* such as MATLAB, Scipy, Octave etc... are efficient to prototype algorithms (more than C++/Fortran)
- MATLAB is not free neither open source. It still remains the *lingua franca* in numerical algorithms.
- Octave is very slow (but highly compatible with MATLAB)
- Python along Numpy and Scipy is a beautiful language, but a little bit slow and do not support any parallelism as a built-in feature.
- accelerators such as Cython, Numba, Codon have pros/cons (what would be a long term choice ? do I have to learn a new syntax for different applications ?, ...)
- R is well suited for statistics, but suffer from legacy syntax

Why do not try a new language for numerical computation ?

A language for numerical computation

- Julia's syntax is very similar to langages as MATLAB, Python or Scilab: moderate learning curve
- does not require *vectorized* code to run fast(JIT compiler)
- uses references (also for function arguments)
- indices start at `begin` (or `1`) and finish at `end` (akin to Fortran)
- use brackets `[i,j,...]` for indexing
- supports broadcasting (`A .= B .+ C .* D` or `@. A = B + C * D`)
- native nD arrays

Functional aspects

- anonymous functions : `(x -> x^2)(2)`
- closures : `f(x) = x + y`
- `map`, `reduce`, `filter`, …
- variadic arguments, keyword argumets : `f(args...; kwargs...)`
- comprehension lists, generators
- convention used: in the `Base` module, function modifying their input arguments have an additional trailing `!`, e.g. inplace sort: `sort!`

Parallelism

- Julia has a built-in support for distributed (`Distributed`) and shared memory (`Threads`) parallelism
- `MPI.jl` for which wraps `MPI` libraries for differents `ABI`s for classical two-sided communication
- `CUDA.jl` for GPU computations

External libraries calls

- sometimes you need to call a C/Fortran code
- "no boilerplate" philosophy : Julia does not require Mexfiles, Swig or other wrapping system
- code must be in a shared library
- various syntax:

```
ccall((:function, "lib"), return_type, (type_1,...,type_n), (arg_1, ..., arg_n))
@ccall "lib".function(arg_1::type_1, ..., arg_n::type_n)::return_type
```

- automatic generation of wrapper for C libraries (`Clang.jl`), lack of automatic C++ generators

Types

- there is a graph type in Julia reflecting the hierarchy of types: `Int64 <: Signed <: Integer <: Real <: Number <: Any`
- both abstract and concrete types are supported
- user can annotate the code with operator \texttt{::} "is an instance of"
- Julia supports
  - primitive types: `Int64, Float32, Bool, Char, ...`
  - composite types

```
struct Foo
  bar
  baz::Int32
  qux::Float64
end
```

- union types: `const OptionalInt = Union{Nothing,Int64}`
- parametric composite types (concrete or abstract):

```
struct Point{T}
  x::T
  y::T
end
```

- `Tuple`, `NamedTuple`, `VarArg` types
- type aliases
- function types

Multiple dispatch

- main idea : define piecewisely methods or functions depending on their arguments types
- let us define `f`

```
f(x::Float64, y::Float64) = 2x + y
f(x::Int, y::Int) = 2x + y
f(2., 3.) # returns 7.0
f(2, 3) # returns 7.0
f(2, 3.) # throws an ERROR: MethodError: no method matching f(::Int64, ::Float64)
```

- but if we define `g`

```
g(x::Number, y::Number) = 2x + y
g(2.0, 3) # now returns 7.0
```

- no automatic or magic conversions : for operators, arguments are promoted to a (user-definable) common type and use the specific implementation

- parametric methods

```
myappend(v::AbstractVector{T}, x::T) where {T <: Real} = [v..., x]
```

Performance (TODO)

- to prove that Julia is fast language, we did some tests
- benchmarks sources taken from the Julia site and modified

Conclusion

pros

- a true language, with lots of powerful features
- Julia is fast!

cons

- small community (package development)
- ...

not adressed

- meta-programming aspects : macros
- reflection

more information

- official website
- official forum
- documentation
- FAQ