Solving large scale optimization models with Julia

Przemysław Szufel

https://szufel.pl/

Preparation of this workshop has been supported by the Polish National Agency for Academic Exchange under the Strategic Partnerships programme, grant number BPI/PST/2021/1/00069/U/00001.

Installing and running Julia

- Download Julia
  - Free and Open Source
  - **https://julialang.org/downloads/**
  - v1.9.2 – the latest stable version
- Programming environment – VS Code
  - https://code.visualstudio.com/download/)
- Jupyter notebook
  - Available via IJulia package

Julia Command Line (REPL)

```
>julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.8.3 (2022-11-14)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

pressing **]** changes REPL to package installation mode `(v1.2) pkg>`

pressing **;** changes REPL to package installation mode `shell>`

pressing **?** changes REPL to help mode `help?>`

to go back to normal mode press **BACKSPACE** `julia>`

Adding Julia packages

- Start Julia REPL

- Press **]** to start the Julia package manager (prompt `(v1.9) pkg>` will be seen)

- Sample package installation command

```
(v1.9) pkg> add PyPlot DataFrames Distributions
```

to go back to normal mode press **BACKSPACE**

Managing packages

(press] for the package management REPL mode)

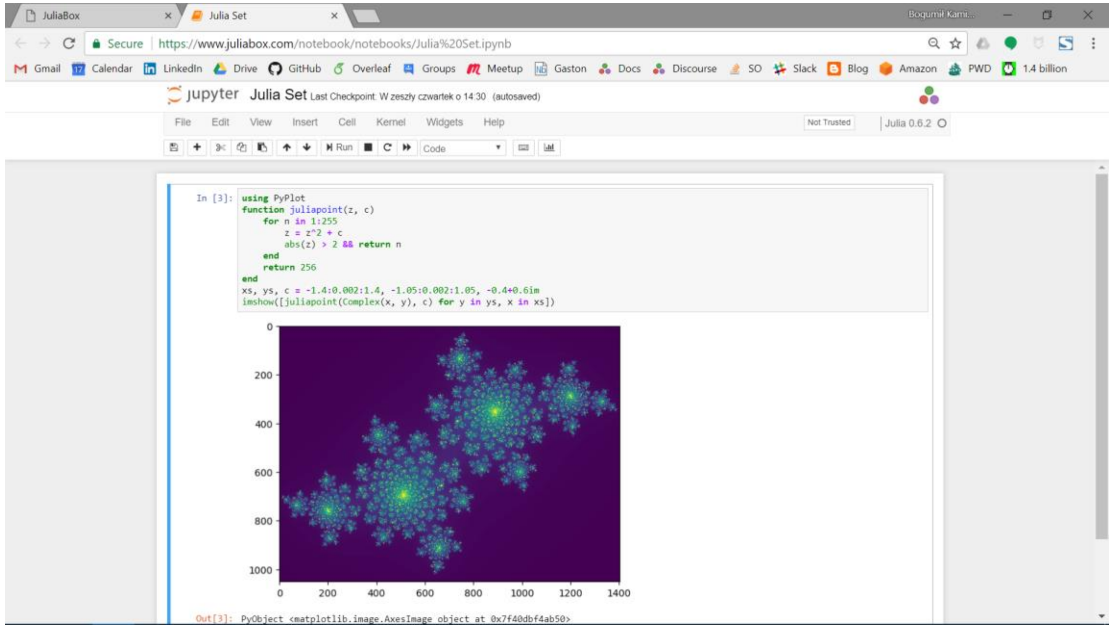
```
(@v1.6) pkg> status
      Status `C:\JuliaPkg\Julia-1.6.3\environments\
  [46ada45e] Agents v4.5.6
  [6e4b80f9] BenchmarkTools v1.2.0
  [336ed68f] CSV v0.8.5
  [34f1f09b] ClusterManagers v0.4.2
  [5ae59095] Colors v0.12.8
  [8f4d0f93] Conda v1.5.2
  [a93c6f00] DataFrames v1.2.2
```

```
(@v1.6) pkg> add RCall
    Updating registry at `C:\JuliaPkg\Julia-1.6.3\registries\General`
    Updating git-repo `https://github.com/JuliaRegistries/General.git`
   Resolving package versions...
   Installed ShiftedArrays ──── v1.0.0
   Installed WinReg ─────────── v0.3.1
   Installed StatsModels ────── v0.6.26
   Installed RCall ──────────── v0.13.12
   Installed CategoricalArrays ─ v0.10.1
    Updating `C:\JuliaPkg\Julia-1.6.3\environments\v1.6\Project.toml`
  [6f49c342] + RCall v0.13.12
    Updating `C:\JuliaPkg\Julia-1.6.3\environments\v1.6\Manifest.toml`
  [324d7699] + CategoricalArrays v0.10.1
  [6f49c342] + RCall v0.13.12
  [1277b4bf] + ShiftedArrays v1.0.0
  [3eaba693] + StatsModels v0.6.26
  [1b915085] + WinReg v0.3.1
    Building RCall → `C:\JuliaPkg\Julia-1.6.3\scratchspaces\44cfe95a-1eb2-52ea-b672-
Precompiling project...
  4 dependencies successfully precompiled in 13 seconds (282 already precompiled)
```

Jupyter notebook

- Jupyter notebook
  - `using Pkg; Pkg.add("IJulia")`
  - `using IJulia`
  - `notebook(dir=".")`

  - Press Ctrl+C to exit



Julia 10,000 feet overview

- Exponential growth, in several areas becomes a standard for scientific and high performance computing
- “walks like Python runs like C”
- Syntax in-between Pyhton/numpy and Matlab
- Compiles to assembly
- Compiles to GPU
- Distributed computing built into the language (known to scale up to millions of CPU cores)
- Best option for number crunching

Why another language for data science?

Two language problem of data science – programming languages

- are either fast (C++, Fortran)
- or are convenient (Python, R, Matlab)

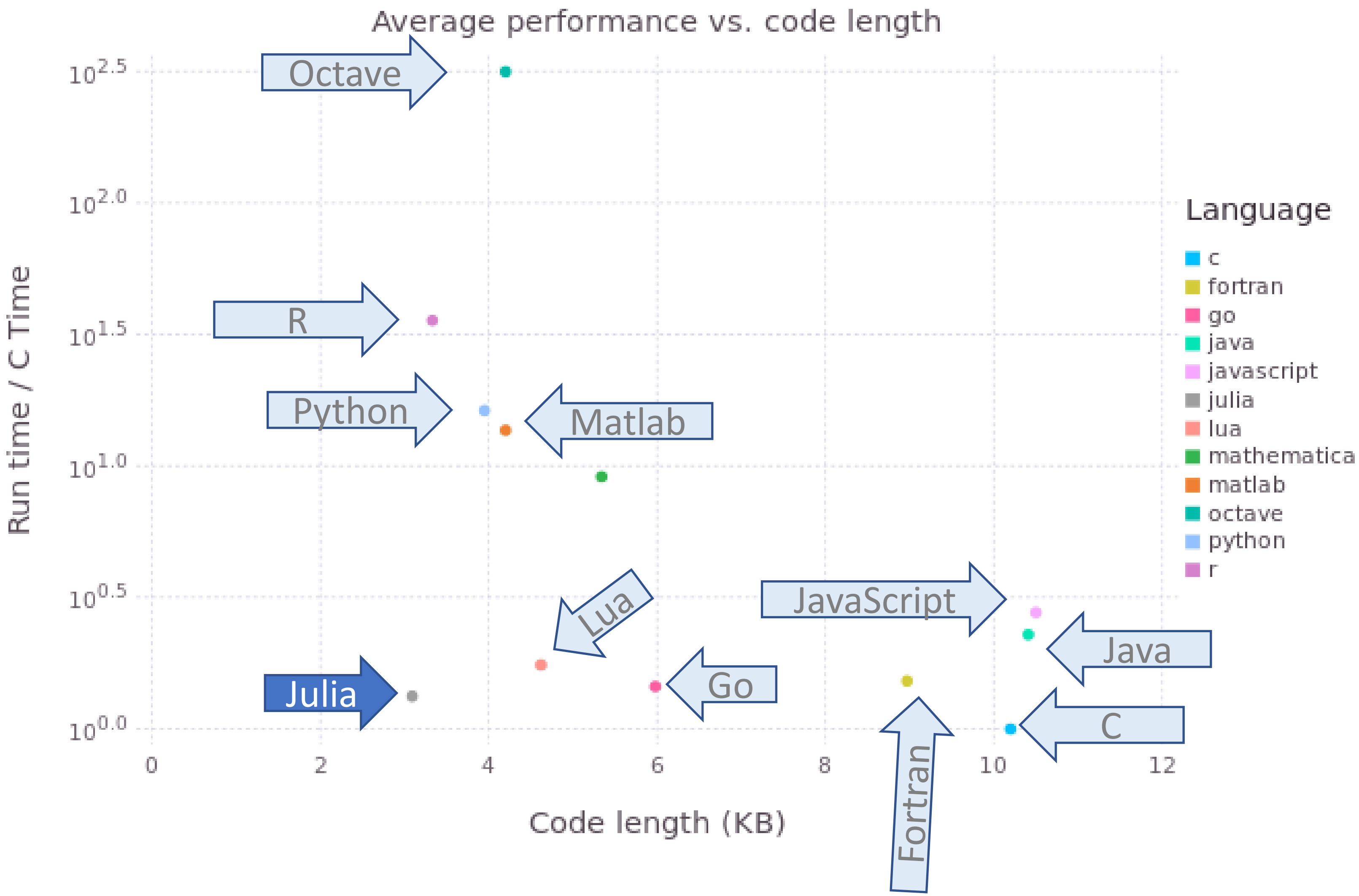
Main features of Julia

1. Efficiency
2. Expressiveness
3. Integrability
4. Metaprogramming – DSLs for various data science subproblems
5. Integration and toolboxes

Methods of achieving high performance in different data science environments

Ecosystem	Glue	Hot code	GPU
R-based	R	RCpp	C
Python-based	Python	Numba/Cython/C	C
Julia-based	Julia	Julia	Julia
Matlab-based	Matlab	C	GPU coder

Language Code Complexity vs Execution Speed

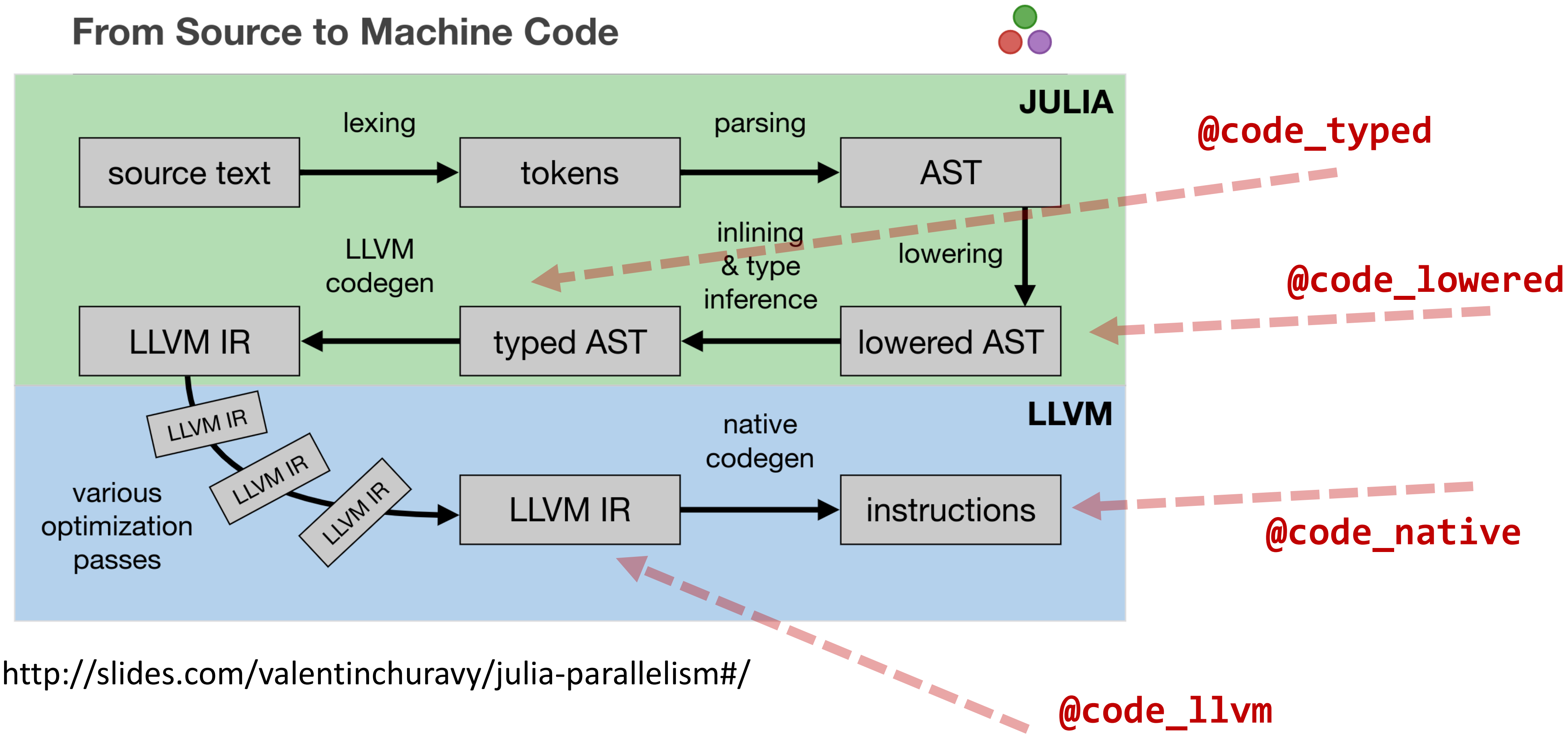


Source: http://www.oceanographerschoice.com/2016/03/the-julia-language-is-the-way-of-the-future/

Key features

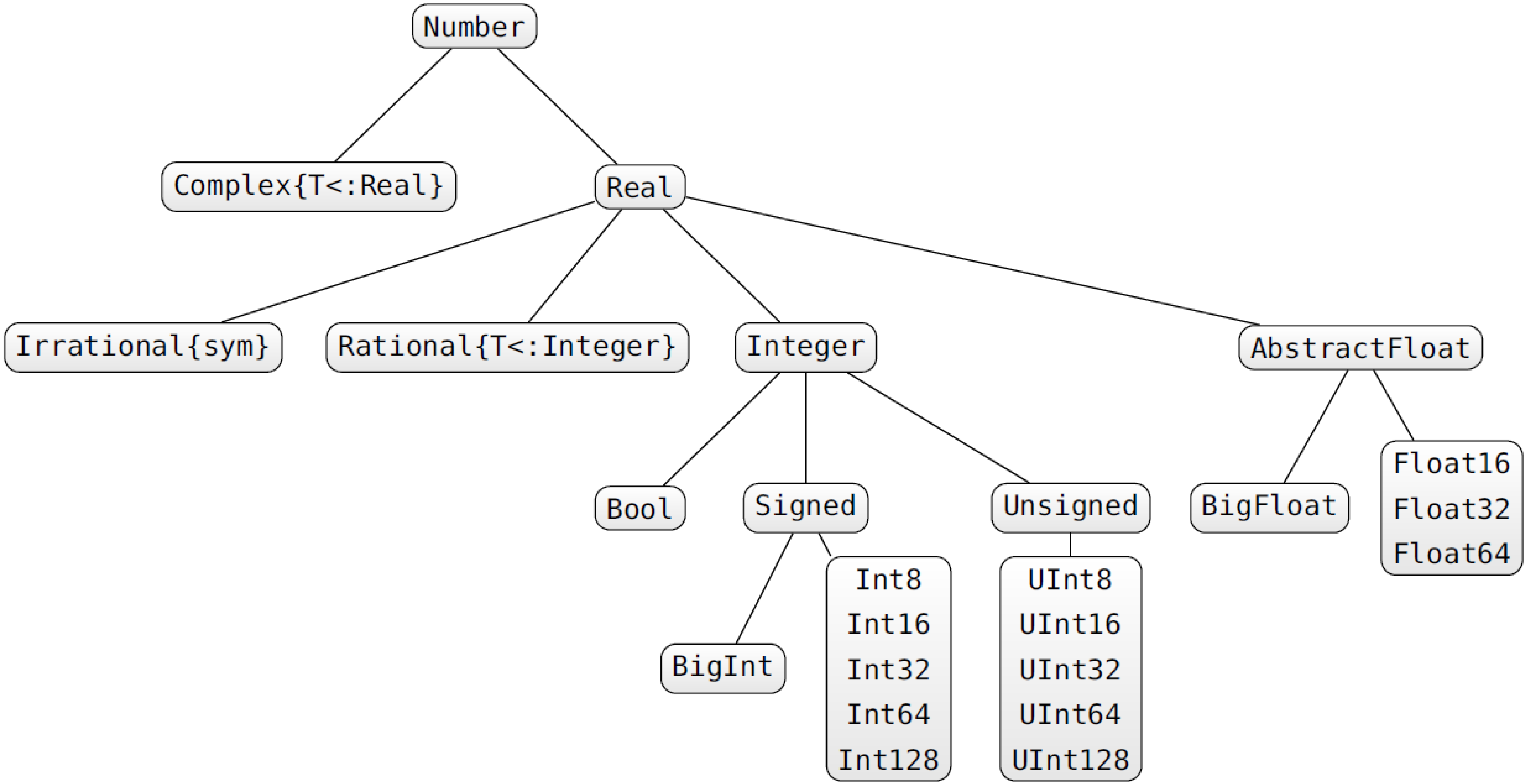
- Performance
  - Dynamically compiled to optimized native machine code
- Scalability
  - SIMD, Threading, Distributed computing
- Modern design of the language
  - multiple dispatch, metaprogramming, type system
- MIT License
  - corporate-use friendly (also package ecosystem)

Julia code compilation process



http://slides.com/valentinchuravy/julia-parallelism#/

Numeric type hierarchy



Type conversion functions

- `Int64('a')` # *character to integer*
- `Int64(2.0)` # *float to integer*
- `Int64(1.3)` # *inexact error*
- `Int64("a")` # *error no conversion possible*
- `Float64(1)` # *integer to float*
- `Bool(1)` # *converts to boolean true*
- `Bool(0)` # *converts to boolean false*
- `Char(89)` # *integer to char*
- `zero(10.0)` # *zero of type of 10.0*
- `one(Int64)` # *one of type Int64*
- `convert(Int64, 1.0)` # *convert float to integer*
- `parse(Int64, "1")` # *parse "1" string as Int64*

Special types

- `Any` *# all objects are of this type*
- `Union{}` *# subtype of all types, no object can have this type*
- `Nothing` *# type indicating nothing, subtype of Any*
- `nothing` *# only instance of* `Nothing`

Tuples – just like in Python

- `()                  # empty tuple`
- `(1,)                # one element tuple`
- `("a", 1)            # two element tuple`
- `('a', false)::Tuple{Char, Bool} # tuple type assertion`
- `x = (1, 2, 3)`
- `x[1]                # first element`
- `x[1:2]              # (1, 2) (tuple)`
- `x[4]                # bounds error`
- `x[1] = 1            # error - tuple is not mutable`
- `a, b = x            # tuple unpacking a==1, b==2`

Tuples are immutable, and the Julia compiler makes a good use of that!

Arrays

```
Array{Char}(undef, 2, 3, 4)      # 2x3x4 array of Chars
Array{Any}(undef, 2, 3)          # 2x3 array of Any
zeros(5)                    # vector of Float64 zeros
ones(Int64, 2, 1)           # 2x1 array of Int64 ones
trues(3), falses(3)         # tuple of vector of trues and of falses

x = range(1, stop=2, length=5)
  # iterator having 5 equally spaced elements (1.0:0.25:2.0)
collect(x)                  # converts iterator to vector
1:10                        # iterable from 1 to 10
1:2:10                      # iterable from 1 to 9 with 2 skip
reshape(1:12, 3, 4)         # 3x4 array filled with 1:12 values
```

Basics…

Linear optimization

```
using JuMP, HiGHS
m = Model(optimizer_with_attributes(HiGHS.Optimizer))
@variable(m,      x₁ >= 0)
@variable(m,      x₂ >= 0)
@objective(m,     Min, 50x₁ + 70x₂)
@constraint(m,    200x₁ + 2000x₂ >= 9000     )
@constraint(m,    100x₁ +   30x₂ >=  300     )
@constraint(m,      9x₁   +   11x₂ >=   60   )
optimize!(m)
JuMP.value.([x₁,x₂])
```

Note – how to type indexes in Julia

- julia> x
- julia> x_
- julia> x_1
- julia> x_1***<TAB>***
- julia> x_1

… and Integer programming

```
using JuMP, HiGHS
m = Model(optimizer_with_attributes(HiGHS.Optimizer))
@variable(m, x1 >= 0, Int)
@variable(m, x2 >= 0)
@objective(m, Min, 50x1 + 70x2)
@constraint(m, 200x1 + 2000x2 >= 9000)
@constraint(m, 100x1 +   30x2 >=  300)
@constraint(m, 9x1   +   11x2 >=   60)
optimize!(m)
```

How it works - metaprogramming

```
julia> code = Meta.parse("x=5")
:(x = 5)

julia> dump(code)
Expr
  head: Symbol =
  args: Array{Any}((2,))
    1: Symbol x
    2: Int64 5

julia> eval(code)
5

julia> x
5
```

Macros – hello world...

```
macro sayhello(name)
       return :( println("Hello, ", $name) )
end

julia> macroexpand(Main,:(@sayhello("aa")))
:((Main.println)("Hello, ", "aa"))

julia> @sayhello "world!"
Hello, world!
```

Macro @variable

```
julia> @macroexpand @variable(m, x₁ >= 0)
quote
    (JuMP.validmodel)(m, :m)
    begin
        #1###361 = begin
                let
                    #1###361 = (JuMP.constructvariable!)(m, getfield(JuMP, Symbol("#_error#107")){Tuple{Symbol,Expr}}((:m, :(x₁ >= 0))), 0, Inf, :Default, (JuMP.string)(:x₁), NaN)
                    #1###361
                end
            end
        (JuMP.registervar)(m, :x₁, #1###361)
        x₁ = #1###361
    end
end
```

`Calculus.jl` – symbolic differantion at compile time

```
julia> using Calculus

julia> differentiate(:(sin(x)))
:(1 * cos(x))

julia> expr = differentiate(:(sin(x) + x*x+5x))
:(1 * cos(x) + (1x + x * 1) + (0x + 5 * 1))

julia> x = 0; eval(expr)
```

Some of JuMP Solvers (over 40 as of today)

Solver	Julia Package	License	LP	SOCP	MILP	NLP	MINLP	SDP
Artelys Knitro	KNITRO.jl	Comm.				X	X	
BARON	BARON.jl	Comm.				X	X	
Bonmin	AmplNLWriter.jl CoinOptServices.jl	EPL	X		X	X	X	
Cbc	**Cbc.jl**	**EPL**			**X**			
Clp	Clp.jl	EPL	X					
Couenne	AmplNLWriter.jl CoinOptServices.jl	EPL	X		X	X	X	
CPLEX	**CPLEX.jl**	Comm.	X	X	X			
ECOS	ECOS.jl	GPL	X	X				
FICO Xpress	Xpress.jl	Comm.	X	X	X			
HiGHS	HiGHSMathProgInterface	GPL	X		X			
Gurobi	**Gurobi.jl**	Comm.	X	X	X			
Ipopt	**Ipopt.jl**	**EPL**	X			X		
MOSEK	Mosek.jl	Comm.	X	X	X	X		X
NLopt	NLopt.jl	LGPL				X		
SCS	SCS.jl	MIT	X	X				X

JuMP
Transportation of good among branches

Use case scenario

The Subway restaurant chain in Las Vegas has a total of 118 restaurants in different parts of the city.

18 restaurants have adjacent huge product warehouses that keep ingredients cool and fresh, moreover fresh vegetables are delivered only to those warehouses (rather than to every restaurant) daily at 3am.

Subway has signed a contract with a transportation agency and is billed by the multiple of the weight of transported goods and the distance.

Knowing the amount of available stock at each warehouse and the expected demand at each restaurant (measured in kg), the company needs to decide how the goods should be distributed among warehouses.

Transportation problem statement

- Variables
  - x_{ij} – number of units transported for i-th supplier to j-th requester
  - c_{ij} – unit transportation cost between i-th supplier to j-th requester
- Cost function C

$$C = \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{c}_{ij} \boldsymbol{x}_{ij}$$

- Constraints:

  suppliers have maximum capacity S_i

$$\sum_{j=1}^{n} x_{ij} \leq S_i$$

  demand D_j must be met

$$\sum_{i=1}^{m} x_{ij} \geq D_j$$

Implementation in JuMP

```
m = Model(optimizer_with_attributes(HiGHS.Optimizer));
@variable(m, x[i=1:S, j=1:D])
@objective(m, Min, sum( x[i, j]*distance_mx[i, j] for i=1:S, j=1:D))
@constraint(m, x .>= 0)
for j=1:D
   @constraint(m, sum( x[i, j] for i=1:S) >= demand[j]   )
end
for i=1:S
   @constraint(m, sum( x[i, j] for j=1:D) <= supply[i]   )
end
optimize!(m)
termination_status(m)
```

JuMP
Travelling salesman problem

Use case scenario

The Subway restaurant chain in Las Vegas has a total of 118 restaurants in different parts of the city.

Company’s manager plans to visit all restaurants during a single day.

What is the optimal order that restaurants should be visited?

Traveling salesman problem (TSP)

- Variables:
  - c_{ft} – cost of travel from "f" to "t"
  - x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

$$\text{Min} \sum_{f=1}^{N} \sum_{t=1}^{N} c_{ft} x_{ft}$$

The mathematical problem framing based on: Julia Programming for Operations Research A Primer on Computing, Changhyun Kwon, 2018, see: http://opensourc.es/blog/mip-tsp

TSP

$$\text{Min} \sum_{f=1}^{N}\sum_{t=1}^{N} c_{ft}x_{ft}$$

Variables:

- c_{ft} – cost of travel from "f" to "t"
- x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

Each city visited once

$$\sum_{t=1}^{N} x_{ft} = 1 \quad \forall f \in \{1,\ldots,N\}$$

$$\sum_{f=1}^{N} x_{ft} = 1 \quad \forall t \in \{1,\ldots,N\}$$

City cannot visit itself

$$x_{ff} = 0 \quad \forall f \in \{1,\ldots,N\}$$

Avoid two-city cycles

$$x_{ft} + x_{tf} <= 1 \quad \forall f,t \in \{1,\ldots,N\}$$

Other cycles: /dynamically add a constraint whenever a cycle occurs/

For more details see: http://opensourc.es/blog/mip-tsp

JuMP implementation

```
m = Model(optimizer_with_attributes(HiGHS.Optimizer));
@variable(m, x[f=1:N, t=1:N], Bin)
@objective(m, Min, sum( x[i, j]*distance_mx[i,j] for i=1:N,j=1:N))
@constraint(m, notself[i=1:N], x[i, i] == 0)
@constraint(m, oneout[i=1:N], sum(x[i, 1:N]) == 1)
@constraint(m, onein[j=1:N], sum(x[1:N, j]) == 1)
for f=1:N, t=1:N
    @constraint(m, x[f, t]+x[t, f] <= 1)
end
```

Getting a cycle

```
function getcycle(x_val, N)
    cycle_idx = Vector{Int}()
    push!(cycle_idx, 1)
    while true
        v, idx = findmax(x_val[cycle_idx[end], 1:N])
        if idx == cycle_idx[1]
            break
        else
            push!(cycle_idx, idx)
        end
    end
    cycle_idx
end
```

Adding a constraint…

```
function solved(m, cycle_idx, N)
    println("cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        cc = @constraint(m, sum(x[cycle_idx,cycle_idx])
           <= length(cycle_idx)-1)
        println("added a constraint")
        return false
    end
    return true
end
```

Iterating over the model

```
while true
    status = solve(m)
    println(status)
    cycle_idx = getcycle(value.(x), N)
    if solved(m, cycle_idx,N)
        break;
    end
end
```

Gurobi.jl

- Commercial software
- Free for academic use
- Integrates with JuMP via Gurobi.jl

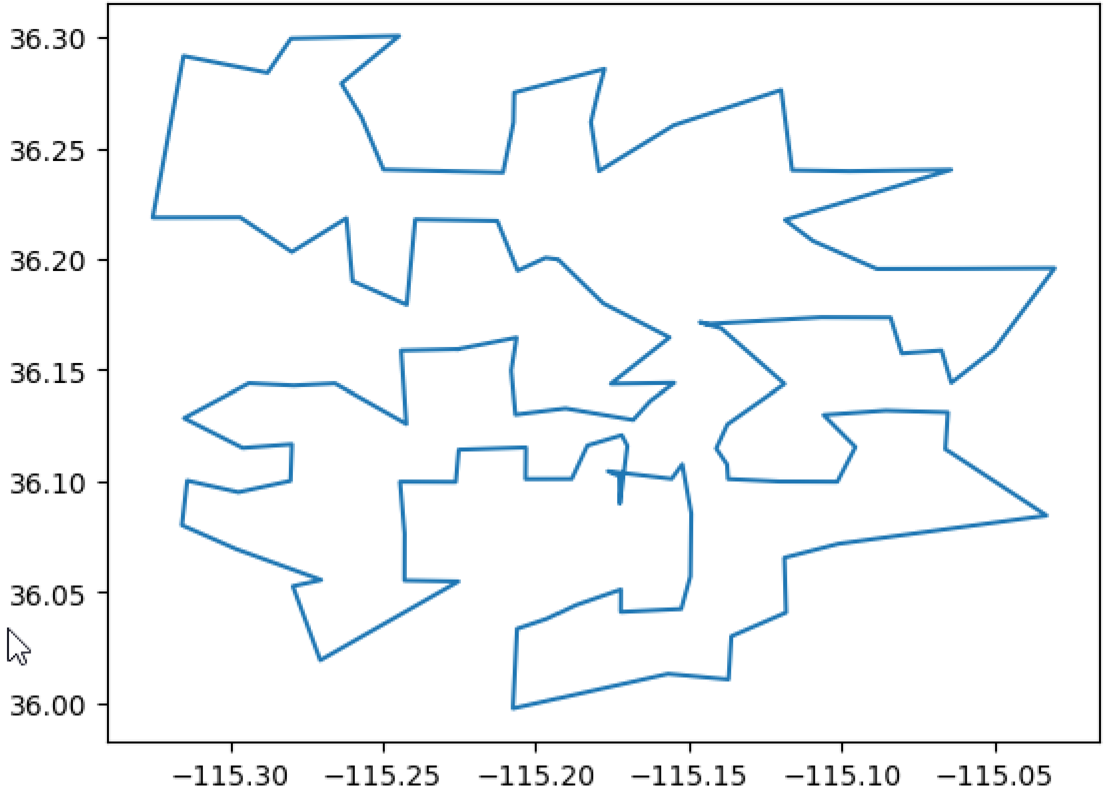
- Supports JuMP Lazy constraints (http://www.juliaopt.org/JuMP.jl/0.18/callbacks.html)

Gurobi callbacks

```
function getcycle(cb, N)
    x_val = callback_value.(Ref(cb), x)
    getcycle(x_val)
end
function callbackhandle(cb)
    cycle_idx = getcycle(cb, N)
    println("Callback! N= $N cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        con = @build_constraint(sum(x[cycle_idx,cycle_idx]) <= length(cycle_idx)-1)
        MOI.submit(m, MOI.LazyConstraint(cb), con)
        println("added a lazy constraint")
    end
end
MOI.set(m, MOI.LazyConstraintCallback(), callbackhandle)
```

TravelingSalesmanHeuristics.jl

```
using TravelingSalesmanHeuristics
sol = TravelingSalesmanHeuristics.solve_tsp(
distance_mx,quality_factor =100)
```



More info:
http://evanfields.github.io/TravelingSalesmanHeuristics.jl/latest/heuristics.html

JuMP
Non-Linear Programming

Simple scenario

Estimate parameters of a quadratic form

$$\boldsymbol{y}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \boldsymbol{x}_i, \text{ where } \boldsymbol{x}_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}$$

for a vector of observed values $\boldsymbol{y}$ to minimize the observed error function

$$\sum_{i=1}^{N} (y(\boldsymbol{x}_i) - y_i)^2$$

Nonlinear optimization Julia

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));

@variable(m, aa[1:2,1:2])

function errs(aa)
   sum((y .- (x * aa ) .* x * [1;1]) .^ 2)
end

@objective(m, Min, errs(aa))

optimize!(m)
```

Use case scenario

(source: Hart et al, Pyomo-optimization modeling in python, 2017)

Simulate dynamics of disease outbreak in a small community of 300 individuals (e.g. children at school)

Three possible states of a patient:

- susceptible (S)
- infected (I)
- recovered (R)

<u>Infection spread model :</u>

- N – population size
- α, β – model parameters

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N}$$

$$S_i = S_{i-1} - I_i$$

Optimization problem for finding parameters α and β

S - susceptible

I - infected

N – population size

α, β – model parameters

SI - time indices {1,2,3,...}

C_i - known input (the actual number of infected patients)

$$\min \sum_{i \in SI} \left(\varepsilon_i^I\right)^2$$

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N} \;\; \forall \; i \in SI \setminus \{1\}$$

$$S_i = S_{i-1} - I_i \;\forall\; i \in SI \setminus \{1\}$$

$$C_i = I_i + \varepsilon_i^I$$

$$0 \leq I_i,\; Si \leq N$$

$$0.5 \leq \beta \leq 70$$

$$0.5 \leq \alpha \leq 1.5$$

Model implementation in JuMP

- Input data (disease dynamics)

```
obs_cases = vcat(1,2,4,8,15,27,44,58,55,32,12,3,1,zeros(13))
```

Full model specification in JuMP

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));
@variable(m, 0.5 <= α <= 1.5)
@variable(m, 0.05 <= β <= 70)
@variable(m, 0 <= I_[1:SI_max] <= N)
@variable(m, 0 <= S[1:SI_max]  <= N)
@variable(m, ε[1:SI_max])
@constraint(m, ε .== I_ .- obs_cases  )
@constraint(m, I_[1] == 1)
for i=2:SI_max
   @NLconstraint(m, I_[i] == β*(I_[i-1]^α)*S[i-1]/N)
end
@constraint(m, S[1] == N)
for i=2:SI_max
   @constraint(m, S[i] == S[i-1]-I_[i])
end
@NLobjective(m, Min, sum(ε[i]^2 for i in 1:SI_max))
```

JuMP
Non-Linear Programming
for estimation of model parameters

Simple scenario

Estimate parameters of a quadratic form

$$\boldsymbol{y}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \boldsymbol{x}_i, \text{ where } \boldsymbol{x}_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}$$

for a vector of observed values ***y*** to minimize the observed error function

$$\sum_{i=1}^{N} (y(\boldsymbol{x}_i) - y_i)^2$$

Nonlinear optimization Julia

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));

@variable(m, aa[1:2,1:2])

function errs(aa)
   sum((y .- (x * aa ) .* x * [1;1]) .^ 2)
end

@objective(m, Min, errs(aa))

optimize!(m)
```

Use case scenario
(source: Hart et al, Pyomo-optimization modeling in python, 2017)

Simulate dynamics of disease outbreak in a small community of 300 individuals (e.g. children at school)

Three possible states of a patient:

- susceptible (S)
- infected (I)
- recovered (R)

<u>Infection spread model :</u>

- N – population size
- α, β – model parameters

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N}$$

$$S_i = S_{i-1} - I_i$$

Optimization problem for finding parameters α and β

S - susceptible

I - infected

N – population size

α, β – model parameters

SI - time indices {1,2,3,...}

C_i - known input (the actual number of infected patients)

$$\min \sum_{i \in SI} \left(\varepsilon_i^I\right)^2$$

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N} \quad \forall\ i \in SI \setminus \{1\}$$

$$S_i = S_{i-1} - I_i \ \forall\, i\ \in SI \setminus \{1\}$$

$$C_i = I_i + \varepsilon_i^I$$

$$0 \leq I_i,\ Si \leq N$$

$$0.5 \leq \beta \leq 70$$

$$0.5 \leq \alpha \leq 1.5$$

Model implementation in JuMP

- Input data (disease dynamics)

```
obs_cases = vcat(1,2,4,8,15,27,44,58,55,32,12,3,1,zeros(13))
```

Full model specification in JuMP

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));
@variable(m, 0.5 <= α <= 1.5)
@variable(m, 0.05 <= β <= 70)
@variable(m, 0 <= I_[1:SI_max] <= N)
@variable(m, 0 <= S[1:SI_max]  <= N)
@variable(m, ε[1:SI_max])
@constraint(m, ε .== I_ .- obs_cases  )
@constraint(m, I_[1] == 1)
for i=2:SI_max
   @NLconstraint(m, I_[i] == β*(I_[i-1]^α)*S[i-1]/N)
end
@constraint(m, S[1] == N)
for i=2:SI_max
   @constraint(m, S[i] == S[i-1]-I_[i])
end
@NLobjective(m, Min, sum(ε[i]^2 for i in 1:SI_max))
```