Julia, o cuando un programa libre es mejor que uno de pago.

AUTORÍA: Ana Isabel Muñoz Montalvo

PRESENTA: Carlos Uriarte González

Universidad Rey Juan Carlos | Oficina de Conocimiento y Cultura Libres

III JORNADAS DE CULTURA LIBRE

VICERRECTORADO COMUNIDAD CAMPUS, CULTURA Y DEPORTE

Julia, o cuando un programa libre es mejor que uno de pago.

AUTORÍA: Manuel Arrayás Chazeta

PRESENTA: Carlos Uriarte González

Universidad Rey Juan Carlos | Oficina de Conocimiento y Cultura Libres

III JORNADAS DE CULTURA LIBRE

VICERRECTORADO COMUNIDAD CAMPUS, CULTURA Y DEPORTE

Julia, o cuando un programa libre es mejor que uno de pago.

AUTORÍA: Carlos Uriarte González

PRESENTA: Carlos Uriarte González

Universidad Rey Juan Carlos | Oficina de Conocimiento y Cultura Libres

III JORNADAS DE CULTURA LIBRE
VICERRECTORADO COMUNIDAD CAMPUS, CULTURA Y DEPORTE

Contenidos

1. Julia como lenguaje de Programación
2. Similitudes con otros lenguajes de pago (Matlab)
3. Ventajas
4. Inconvenientes
5. Benchmarking y experiencia propia

Julia como lenguaje de programación

- *Creado en 2009.*
- *Lenguaje multiplataforma y multiparadigma.*
- *Lenguaje compilado de alto nivel.*
- *Tipado dinámico.*

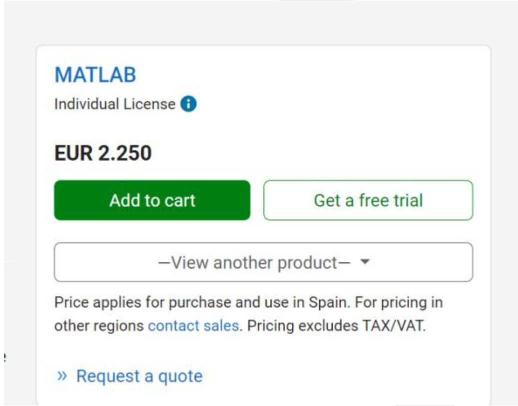
Similitudes (Matlab)

- Facil sintaxis.
- Tipado dinámico
- Extensa librería matemática.

```
julia> c = "Julia"
"Julia"

julia> for i in c
         println(i)
       end
J
u
l
i
a
```

Ventajas

- Al ser compilado, la velocidad de Julia no tiene rival frente a Matlab.
- Además, la velocidad es escalable si agregamos la definición de tipado en las variables.
- Es gratis!!
- Está incluso más preparado para el cálculo matemático.
- Permite el manejo de paquetes de forma sencilla.



Ventajas

Solver
ode45
ode23
ode113
ode78
ode89
ode15s
ode23s
ode23t
ode23tb
ode15i

- `NDBLSRK144` - 14-stage, fourth order low-storage method with optimized stability regions for advection-dominated problems. Fixed timestep only. Like SSPRK methods, `NDBLSRK144` also takes optional arguments `stage_limiter!`, `step_limiter!`.
- `CFRLDDRK64` - 6-stage, fourth order low-storage, low-dissipation, low-dispersion scheme. Fixed timestep only.
- `TSLDDRK74` - 7-stage, fourth order low-storage low-dissipation, low-dispersion scheme with maximal accuracy and stability limit along the imaginary axes. Fixed timestep only.
- `DGLDDRK73_C` - 7-stage, third order low-storage low-dissipation, low-dispersion scheme for discontinuous Galerkin space discretizations applied to wave propagation problems, optimized for PDE discretizations when maximum spatial step is small due to geometric features of computational domain. Fixed timestep only. Like SSPRK methods, `DGLDDRK73_C` also takes optional arguments `stage_limiter!`, `step_limiter!`.
- `DGLDDRK84_C` - 8-stage, fourth order low-storage low-dissipation, low-dispersion scheme for discontinuous Galerkin space discretizations applied to wave propagation problems, optimized for PDE discretizations when maximum spatial step is small due to geometric features of computational domain. Fixed timestep only. Like SSPRK methods, `DGLDDRK84_C` also takes optional arguments `stage_limiter!`, `step_limiter!`.
- `DGLDDRK84_F` - 8-stage, fourth order low-storage low-dissipation, low-dispersion scheme for discontinuous Galerkin space discretizations applied to wave propagation problems, optimized for PDE discretizations when the maximum spatial step size is not constrained. Fixed timestep only. Like SSPRK methods, `DGLDDRK84_F` also takes optional arguments `stage_limiter!`, `step_limiter!`.
- `SHLDDRK64` - 6-stage, fourth order low-stage, low-dissipation, low-dispersion scheme. Fixed timestep only. Like SSPRK methods, SHLDDRK64 also takes optional arguments `stage_limiter!`, `step_limiter!`.
- `RK46NL` - 6-stage, fourth order low-stage, low-dissipation, low-dispersion scheme. Fixed timestep only.
- `ParsaniKetchesonDeconinck3S32` - 3-stage, second order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S82` - 8-stage, second order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S53` - 5-stage, third order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S173` - 17-stage, third order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S94` - 9-stage, fourth order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S184` - 18-stage, fourth order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S105` - 10-stage, fifth order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `ParsaniKetchesonDeconinck3S205` - 20-stage, fifth order (3S) low-storage scheme, optimized for the spectral difference method applied to wave propagation problems.
- `CKLLSRK43_2` - 4-stage, third order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3C` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK95_4S` - 9-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK95_4C` - 9-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK95_4M` - 9-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3C_3R` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3M_3R` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3N_3R` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK85_4C_3R` - 8-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK85_4M_3R` - 8-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK85_4P_3R` - 8-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3N_4R` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK54_3M_4R` - 5-stage, fourth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK65_4M_4R` - 6-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK85_4FM_4R` - 8-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes equations.
- `CKLLSRK75_4M_5R` - 7-stage, fifth order low-storage scheme, optimized for compressible Navier–Stokes

Ventajas

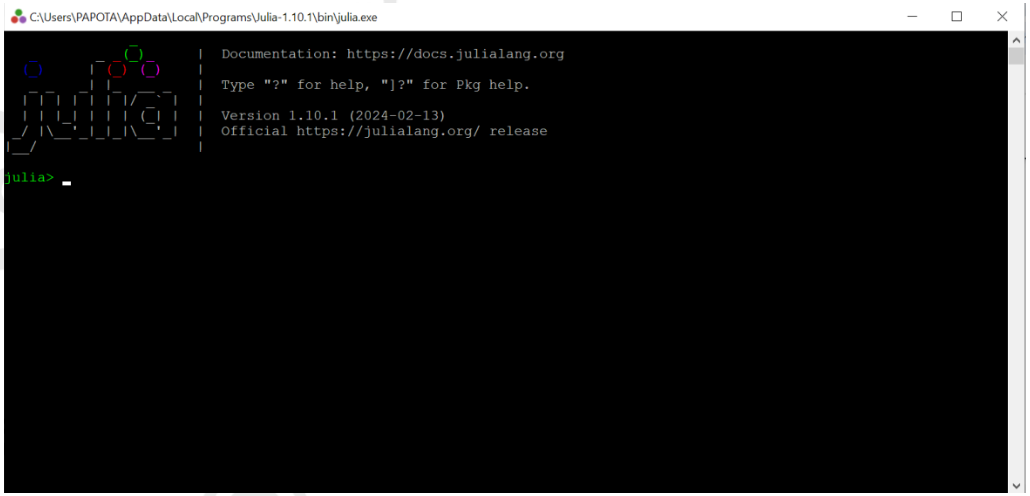
- Es completamente transparente: Sin cajas negras.

>> @edit sin(0.3)

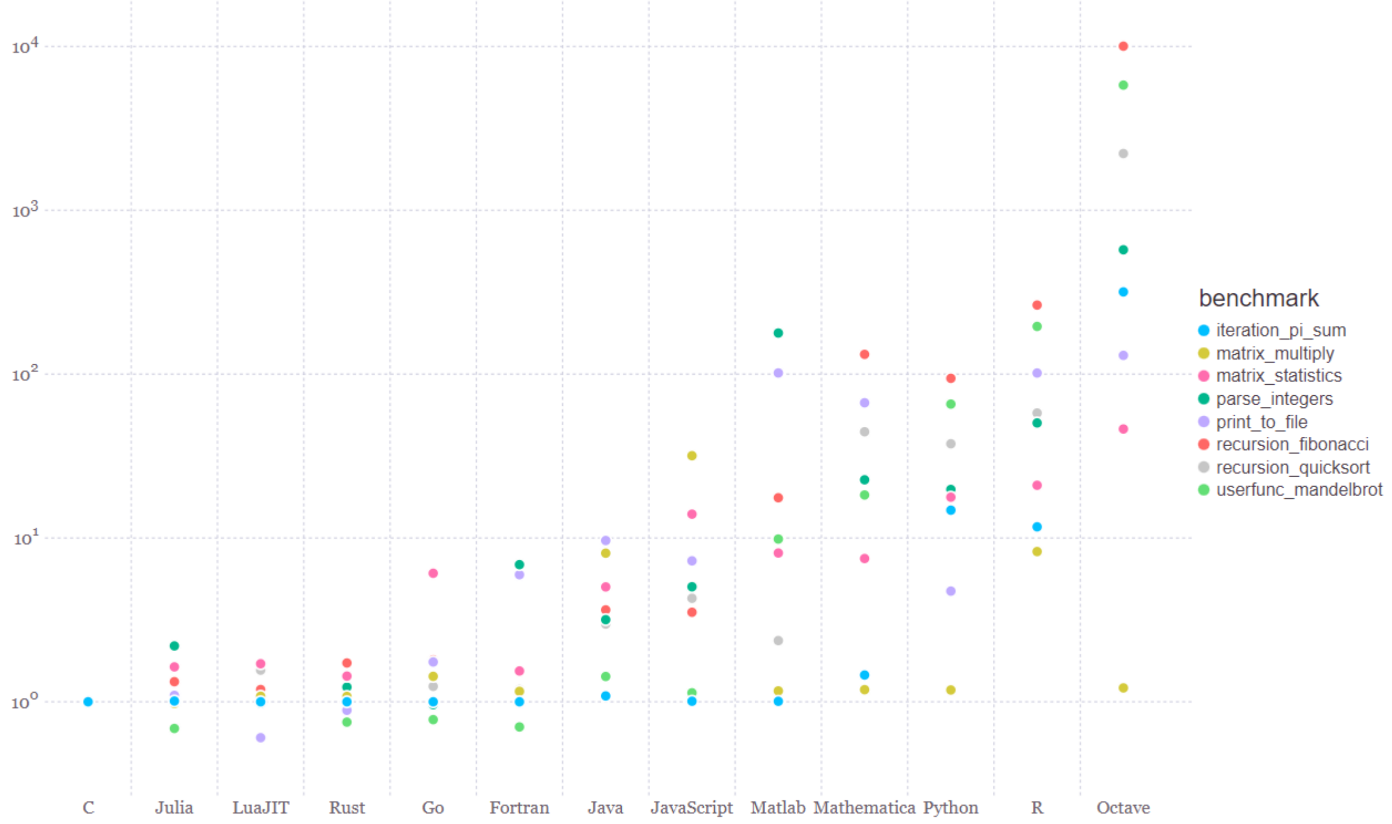

```
# Trigonometric functions
# sin methods
@noinline sin_domain_error(x) = throw(DomainError(x, "sin(x) is only defined for finite x."))
function sin(x::T) where T<:Union{Float32, Float64}
    absx = abs(x)
    if absx < T(pi)/4 #|x| ~<= pi/4, no need for reduction
        if absx < sqrt(eps(T))
            return x
        end
        return sin_kernel(x)
    elseif isnan(x)
        return x
    elseif isinf(x)
        sin_domain_error(x)
    end
    n, y = rem_pio2_kernel(x)
    n = n&3
    if n == 0
        return sin_kernel(y)
    elseif n == 1
        return cos_kernel(y)
    elseif n == 2
        return -sin_kernel(y)
    else
        return -cos_kernel(y)
    end
end

# Coefficients in 13th order polynomial approximation on [0; π/4]
#     sin(x) ≈ x + S1*x³ + S2*x⁵ + S3*x⁷ + S4*x⁹ + S5*x¹¹ + S6*x¹³
# D for double, S for sin, number is the order of x-1
const DS1 = -1.66666666666666324348e-01
const DS2 = 8.33333333332248946124e-03
const DS3 = -1.98412698298579493134e-04
const DS4 = 2.75573137070700676789e-06
const DS5 = -2.50507602534068634195e-08
const DS6 = 1.58969099521155010221e-10

"""
    sin_kernel(yhi, ylo)

Computes the sine on the interval [-π/4; π/4].
"""
@inline function sin_kernel(y::DoubleFloat64)
    y² = y.hi*y.hi
    y⁴ = y²*y²
    r  = @horner(y², DS2, DS3, DS4) + y²*y⁴*@horner(y², DS5, DS6)
    y³ = y²*y.hi
    y.hi-((y²*(0.5*y.lo-y³*r)-y.lo)-y³*DS1)
end
@inline function sin_kernel(y::Float64)
    y² =  y*y
    y⁴ =  y²*y²
    r  =  @horner(y², DS2, DS3, DS4) + y²*y⁴*@horner(y², DS5, DS6)
    y³ =  y²*y
```

Desventajas

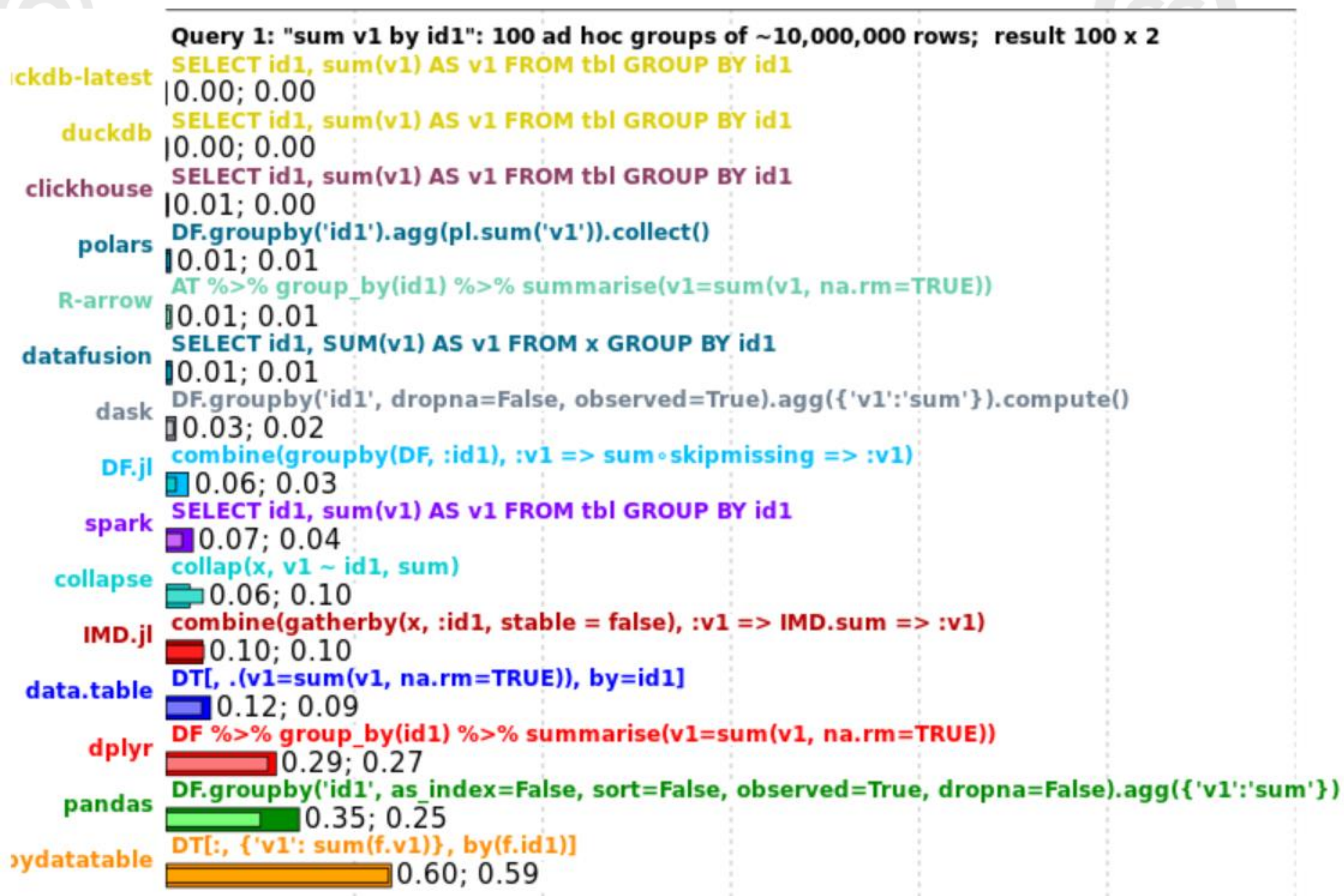
- La principal desventaja radica en su juventud, existe poca información y ejemplos de implementación.
- Carece de un IDE decente.



Benchmark: Algoritmos



Benchmark:Data Frames



LICENCIAS Y CRÉDITOS

Ilustración: "Búho Libre", Sergio Rodríguez Asenjo.
Licencia: Creative Commons Atribución 4.0 Intl.

Estudio benchmark: https://julialang.org/benchmarks/

Licencia: MIT license (https://github.com/JuliaLang/www.julialang.org/blob/main/LICENSE.md)

¡GRACIAS!

MÁS INFORMACIÓN:
HTTPS://OFILIBRE.URJC.ES/

Universidad Rey Juan Carlos | Oficina de Conocimiento y Cultura Libres

VICERRECTORADO COMUNIDAD CAMPUS, CULTURA Y DEPORTE