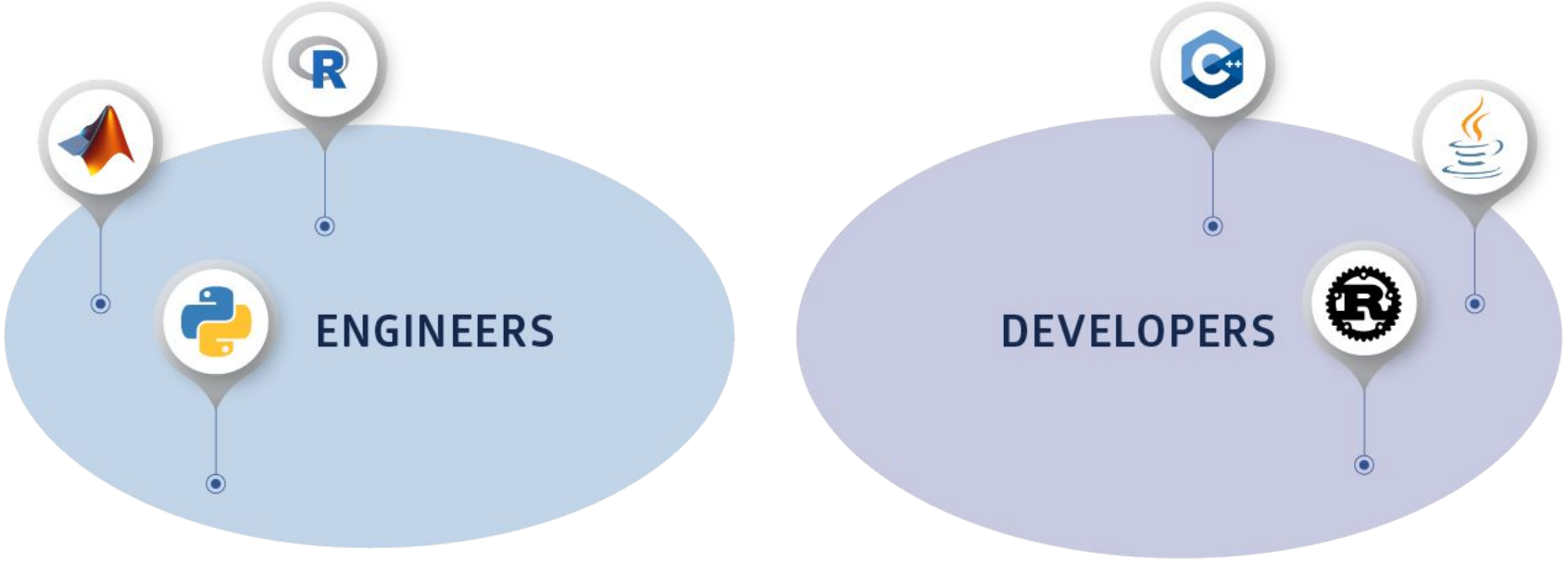
JuliaHub

Satellite Power Analysis

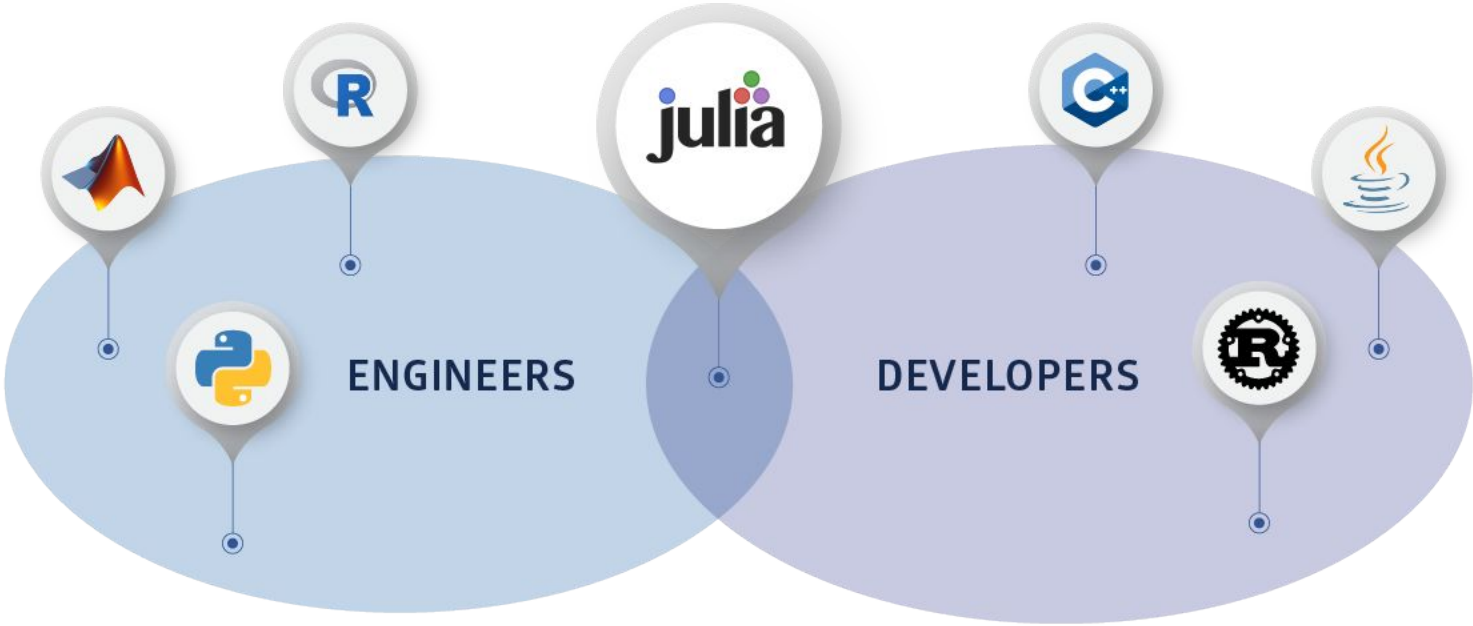
A Fast, Composable, Open-Source Approach

David Dinh
Ranjan Anantharaman
30 April 2025

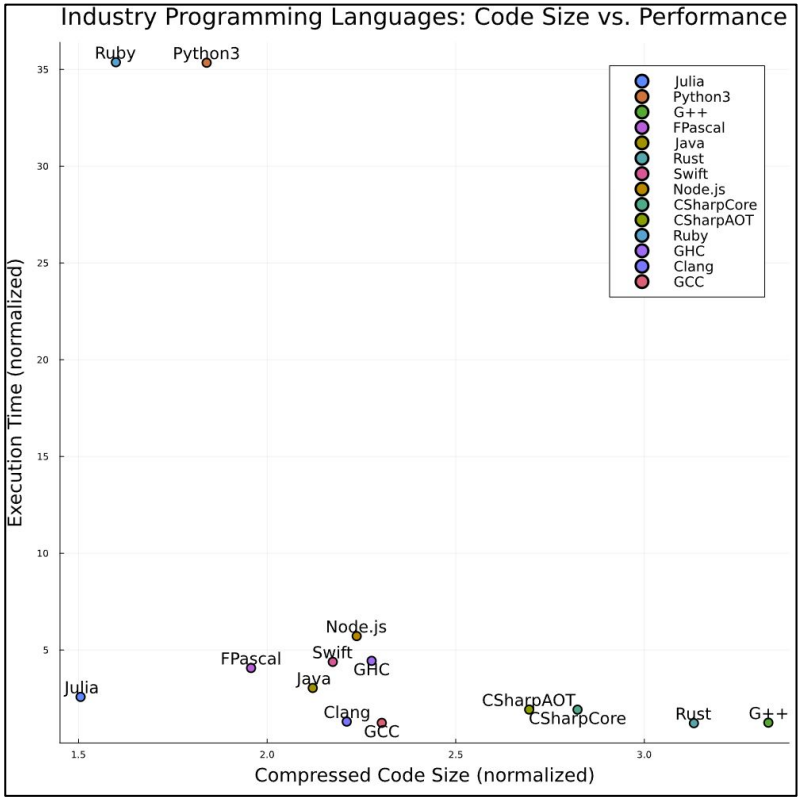
The **two-culture** problem creates divides



Julia solves the two-culture problem



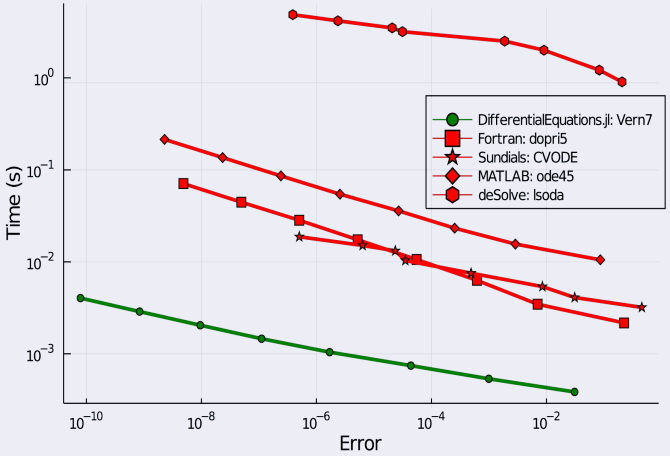
Performance Enables **Higher Fidelity**



Benchmarks

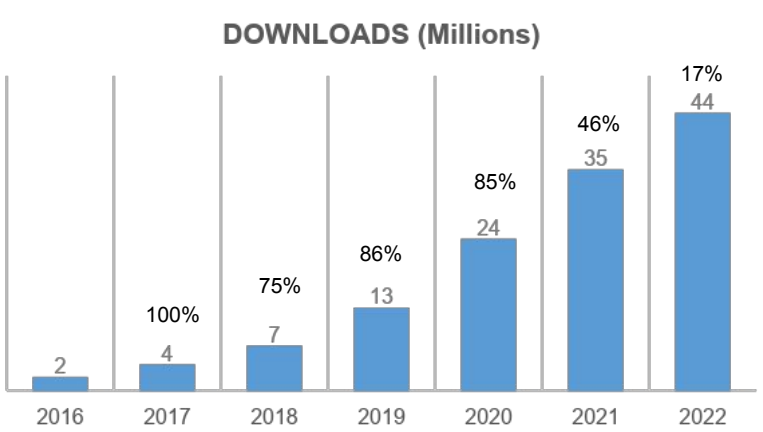
- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than deSolve in R

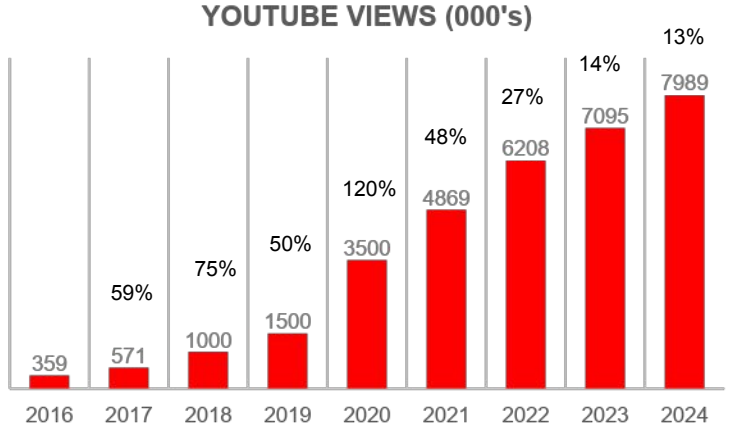
Non-Stiff ODE: Rigid Body System

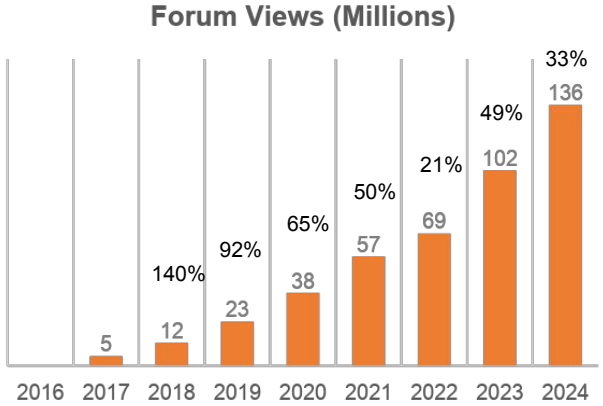


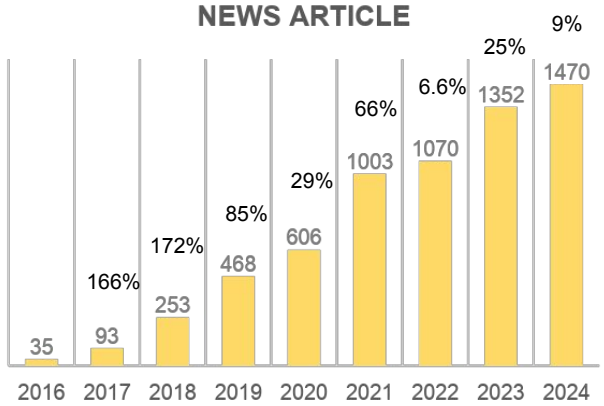
New Graduates Are Entering the Workforce with Julia Proficiency

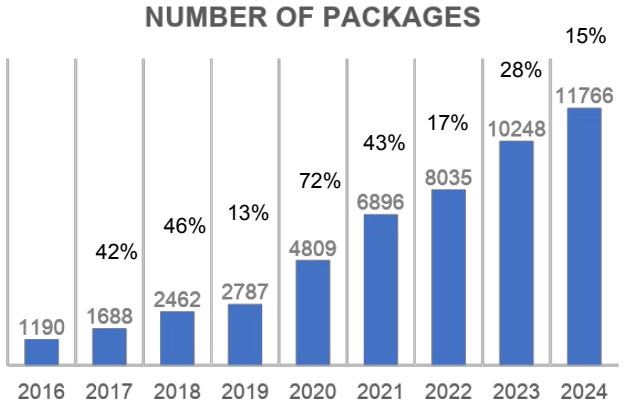
Estimated 1 Million Users of Julia, and growing!







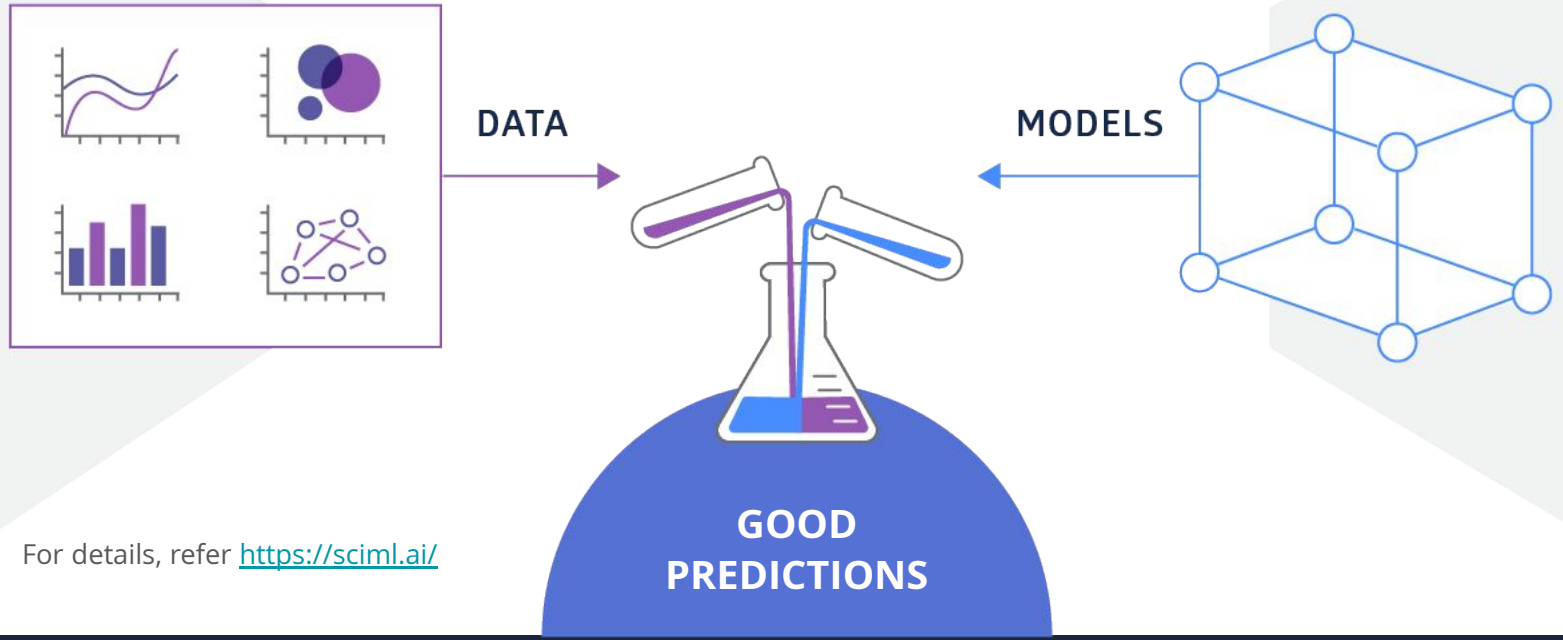




What is SciML?

SciML

Open Source Software for
Scientific Machine Learning

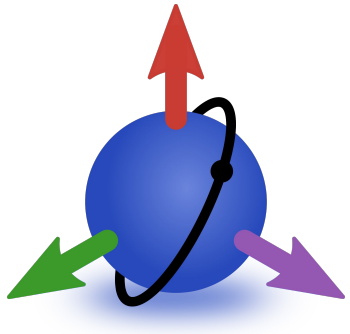


For details, refer https://sciml.ai/

SatelliteToolbox.jl

Open-source Julia package for high-performance space mission analysis

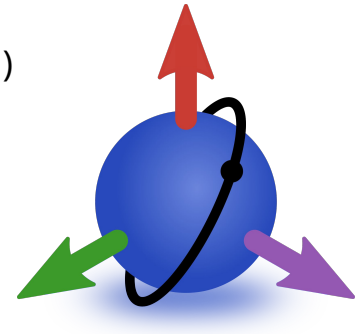
- Development led by Ronan Arraes Jardim Chagas (PhD) at National Institute for Space Research (INPE)
- PhD Research necessitated a simulation of an inertial navigation system consisting of numerous UAVs each with 18 states
- Motivation
  - Underperformance of a well-known software for space mission analysis engineering
    - Would have taken ~6 month to fully simulate his desired configuration
  - Desire to avoid implementation of C features to improve performance
    - High technical burden and also leads to the two-language problem



SatelliteToolbox.jl

Key Features

- Open-source, composable with Julia ecosystem
- Satellite orbit propagation
  - J2, J2 osculating, J4, J4 osculating, SGP4/SDP4, and two-body
- Transformation of reference systems
  - Conventions defined by IAU-76/FK5 and IAU-2006/2010A
  - Geocentric and geodetic coordinates
  - Earth-Centered, Earth-Fixed reference frame to local reference frames
- Computation of atmospheric density
  - Exponential, Jacchia-Roberts 1971, Jacchia-Bowman 2008, and NRLMSISE-00)
- Obtain position vectors of Sun and Moon
- Fetch two-line elements (TLE) of catalogued satellite from online sources
  - Read, parse, and generate TLEs
- Convert time between the required time epochs (UTC, UT1, TT)



JuliaSim

Satellite Power Analysis

Key Insights

Approach

- SatelliteToolbox.jl
  - Orbital mechanics
  - Sun vector
  - Illumination conditions
  - Solar panel orientation
- Julia
  - Solar panel
  - *MPPTracker**
  - *DC-DC Converter**
  - *Battery**
- Simulation

Advantages

- High performance using a single, expressive language
- Enables high-fidelity component-based modeling and analysis in a single framework
- Fully composable with Julia ecosystem
- Open-source

Orbital Mechanics with SatelliteToolbox.jl

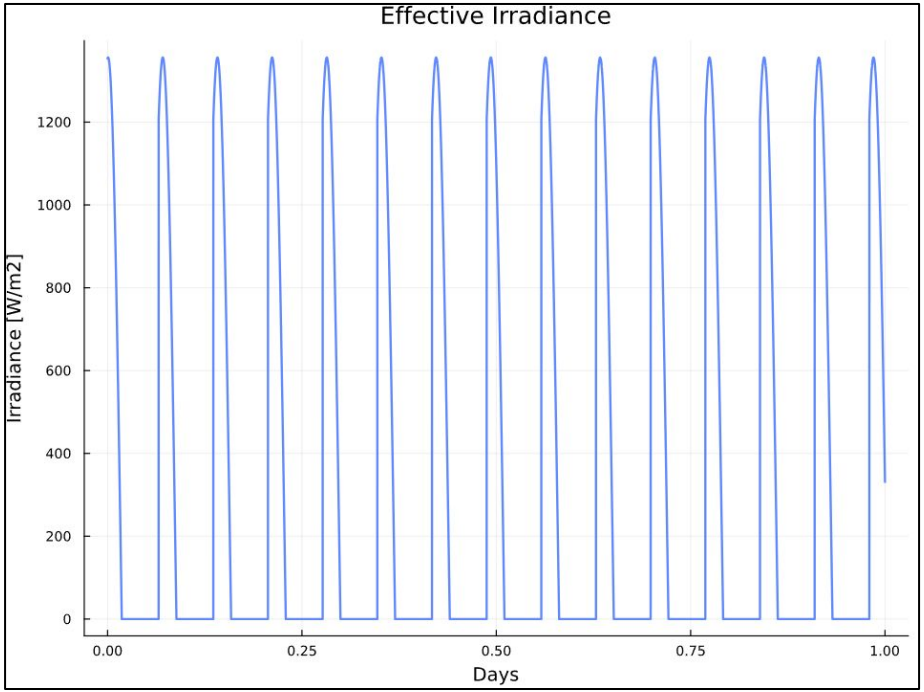
```julia
################ 1. Orbital mechanics ################
# Compute the satellite's position over time using orbital parameters.

jd₀ = date_to_jd(2025, 1, 1, 0, 0, 0)  # start time
days = 30

orb = KeplerianElements(
            jd₀,                # epoch [JD]
            7190.982e3,         # semi-major axis [m]
            0.001111,           # small eccentricity
            98.405 |> deg2rad,  # inclination [rad]
            100    |> deg2rad,  # RAAN [rad]
            90     |> deg2rad,  # argument of perigee [rad]
            19     |> deg2rad   # true anomaly [rad]
        )

orbp = Propagators.init(Val(:J2), orb)

ret = Propagators.propagate!.(orbp, 0:1:(86400*days)) # propagate for 30 days

sat_pos = getindex.(ret,1) # collect the position data
sat_vel = getindex.(ret,2) # collect the velocity data


################ 2. Sun vector ################
# Determine where the sun is in relation to the satellite at any given time.

Δt = 1.0    # 1-minute time step in seconds
times = collect(jd₀:Δt/86400:jd₀ + days)  # 86400 = seconds per day
times_adj = times .- jd₀ # adjusted time array where t=0 corresponds to jd₀
end_time = maximum(times_adj)

sun_pos = [sun_position_mod(jd) for jd in times] # vector from Earth center to Sun
sun_vec = sun_pos .- sat_pos # vector from Satellite to Sun

```

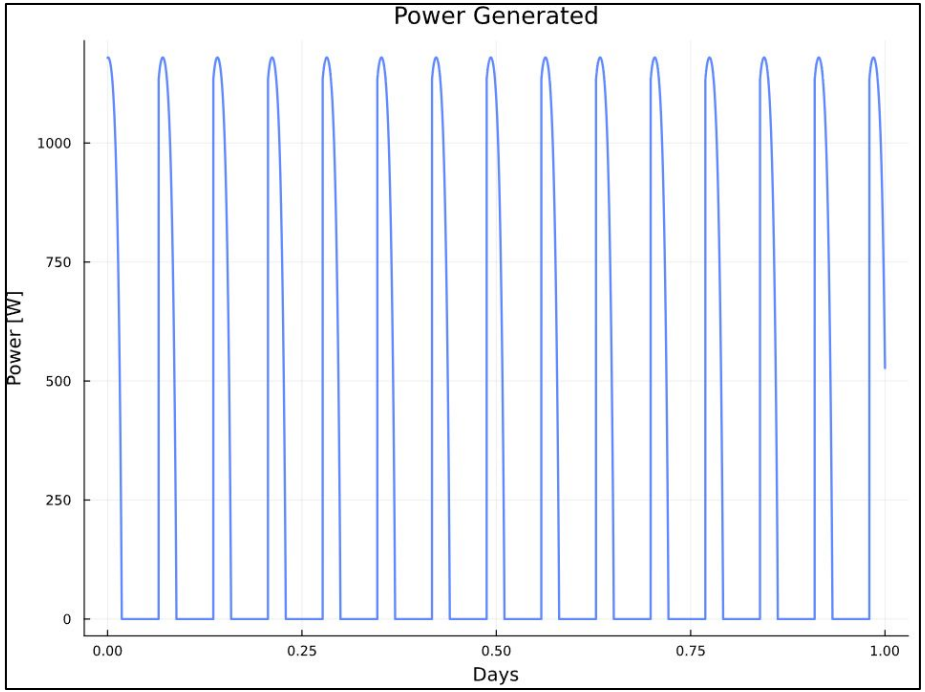
```julia
################ 3. Illumination condition ################
# Figure out whether the satellite is in sunlight or eclipse.

sat_condition = lighting_condition.(sat_pos, sun_pos)
sunlight = Int.(sat_condition .== :sunlight)   # 1 if satellite is in sunlight, 0 otherwise
sunlight_interp =  QuadraticInterpolation(sunlight, times_adj)

################ 4. Solar panel orientation ################
# Compute the angle between the panel normal and the sun vector.

sun_unit = sun_vec./norm.(sun_vec) # unit vector from Satellite to Sun
panel_norm = sat_vel./norm.(sat_vel) # unit norm for solar panels; assume panels face direction of velocity

theta = acos.(dot.(panel_norm, sun_unit)) # angle between solar panel normal and Sun direction [rad]
theta_interp =  QuadraticInterpolation(theta, times_adj)
```

Solar Panel Model with Julia

```julia
@component function SolarPanelSimple(; name, G=1361, A=5, η_ref=0.3, T_ref=300, β=0.004, α=0.9, ϵ=0.8, σ=5.67e-8)
  params = @parameters begin
    (G::Float64 = G)
    (A::Float64 = A)
    (η_ref::Float64 = η_ref)
    (T_ref::Float64 = T_ref)
    (β::Float64 = β)
    (α::Float64 = α)
    (ϵ::Float64 = ϵ)
    (σ::Float64 = σ)
  end
  vars = @variables begin
    θ(t), [input = true]
    in_sunlight(t), [input = true]
    G_eff(t)
    T(t)
    η(t)
    P(t)
  end
  defaults = Dict([
  ])
  eqs = Equation[
    T ~ ((α * G_eff) / (ϵ * σ)) ^ (1 / 4)
    G_eff ~ max(G * cos(θ) * in_sunlight, 0)
    η ~ η_ref * (1 - β * (T - T_ref))
    P ~ G_eff * A * η
  ]
  return ODESystem(eqs, t, vars, params; systems = [], defaults, name)
end
```

Simulation Results

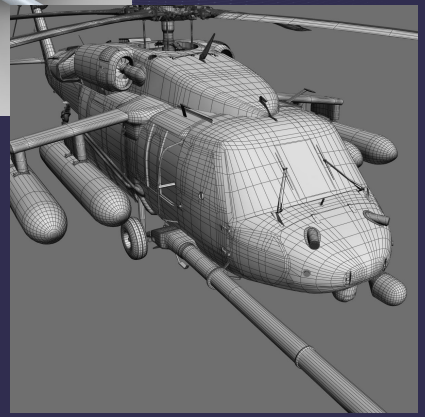




JuliaHub

Julia is the open-source programming language designed for scientific & technical computing

- Julia is widely adopted in the scientific and defense communities for its speed, scalability, and ease of use.
- Strong academic pedigree (MIT) and board (GE and Boeing)

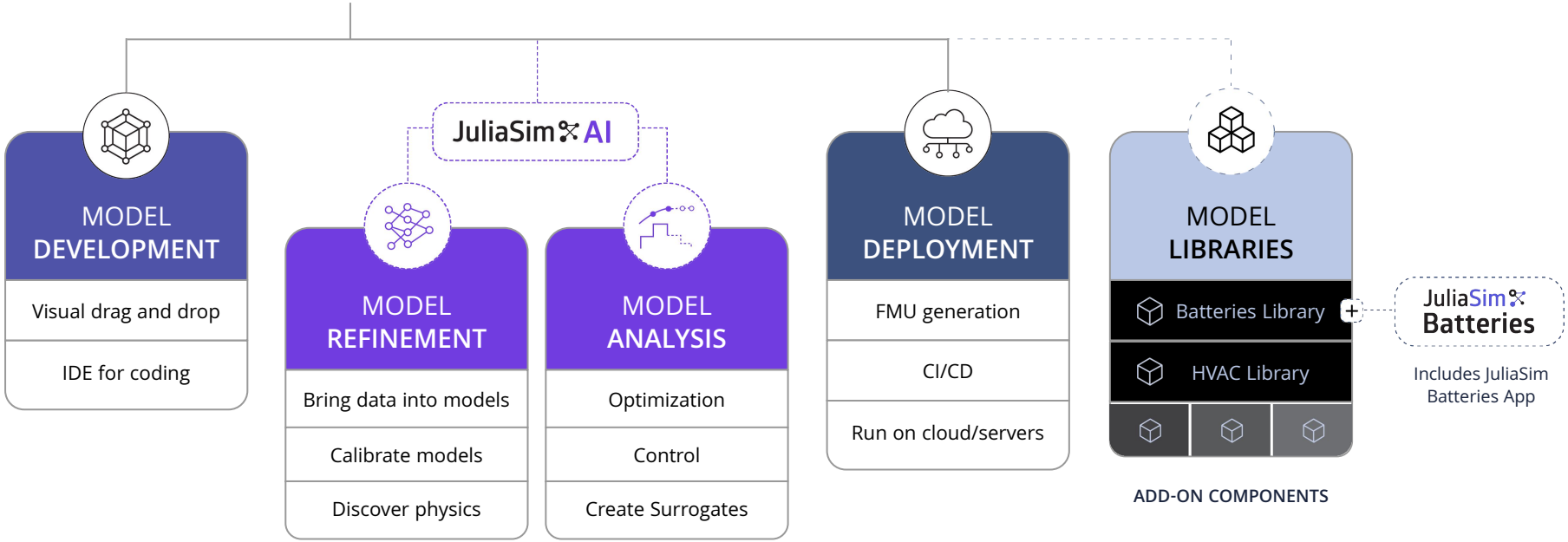


JuliaHub provides value through

- Software development of Julia based applications
- Enterprise based modeling and simulation ecosystem (JuliaSim)
- Consulting, deployment and mentoring around the Julia language

JuliaSim

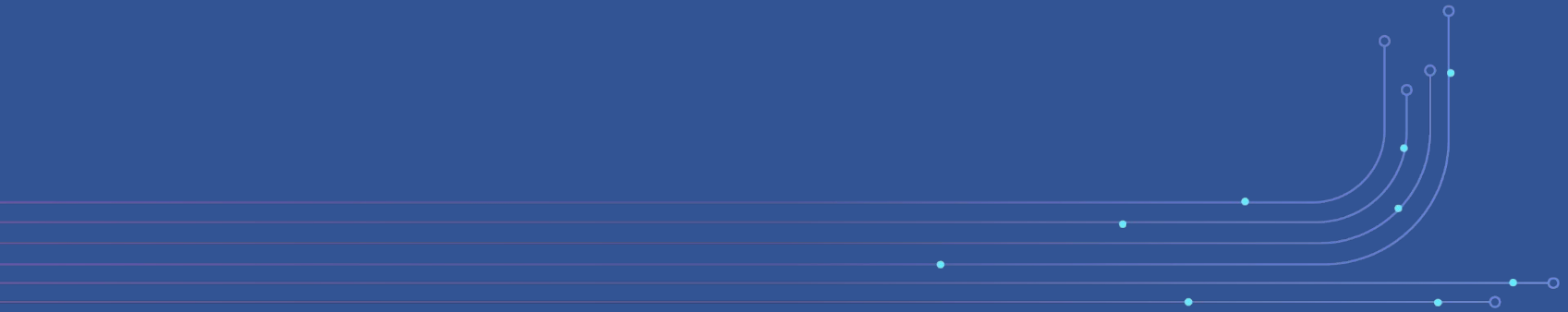
Modern Modeling and Simulation Powered by Machine Learning



BUILT ON OPEN SOURCE COMPONENTS SciML julia

Summary

- Julia
- Satellite Power Analysis
- JuliaHub + JuliaSim



David Dinh
david.dinh@juliahub.com

Ranjan Anantharaman
ranjan.anantharaman@juliahub.com