

InfiniteOpt



INFINITEOPT.JL: A JULIA
PACKAGE FOR INFINITE-
DIMENSIONAL OPTIMIZATION

Dr. Joshua Pulsipher

julia

JUMP

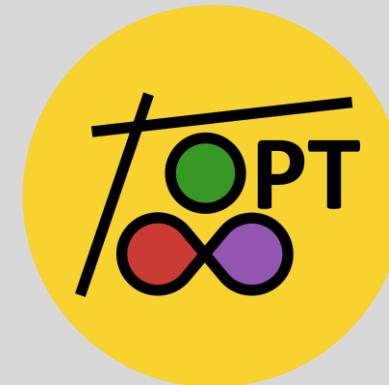
Today's Topics



CODING IN JULIA



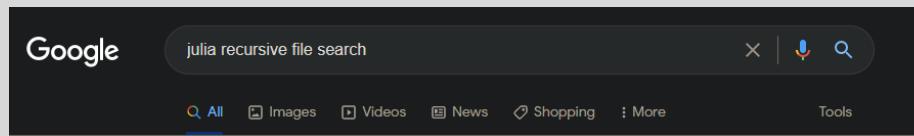
MATHEMATICAL OPTIMIZATION
VIA JUMP.JL



INFINITE-DIMENSIONAL
OPTIMIZATION VIA
INFINITEOPT.JL

Learning Outcomes

- Topic familiarity
 - Can briefly describe what Julia, JuMP.jl, and InfiniteOpt.jl are (and what they might be useful for)
 - Can “Google” the right things
- Programming expertise
 - Can setup basic Julia, JuMP.jl, and InfiniteOpt.jl workflows with appropriate reference materials
- Continued learning
 - Know what/where comes next in developing expertise with these tools
 - Identify resources to facilitate this learning



A screenshot of a Stack Overflow post titled "julia recursive file search". The post has 17 answers. The top answer, by user lognlp/patrick, provides a code snippet using `walkdir`:

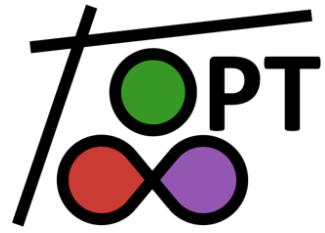
```
for (root, dirs, files) in walkdir("mydir")
    operate_on_files(joinpath.(root, files)) # files is a Vector{String}, can be empty
end
```

The accepted answer, by user Arshul Singh, provides an alternative approach:

```
contents = String[]
for (root, dirs, files) in walkdir("mydir")
    # global contents += if isREPL
    push!(contents, read(joinpath.(root, files), String))
end
```

Course Schedule

Times	Topic	Duration
9:00 a.m. – 9:30 a.m.	<i>Introduction</i> - The why and what of Julia, JuMP.jl, and InfiniteOpt.jl.	30 min.
9:30 a.m. – 10:00 a.m.	<i>Installation and Setup</i> - Configure software on personal laptop. Online interface provided as an alternative.	30 min.
10:00 a.m. – 10:10 a.m.	Break	10 min.
10:10 a.m. – 11:10 a.m.	<i>Julia: A Practical Introduction</i> – Overview of core types, programmatic syntax, and package management.	60 min.
11:10 a.m. – 11:20 a.m.	Break	10 min.
11:20 a.m. – 12:00 p.m.	<i>JuMP.jl: A Brief Introduction</i> – The basics of modeling and solving mathematical optimization problems in JuMP.jl.	40 min.
12:00 p.m. – 1:00 p.m.	Lunch	60 min.
1:00 p.m. – 2:00 p.m.	<i>JuMP.jl: Beyond the Basics</i> – A deeper dive into the core modeling/solution strategies including variables, constraints, containers, and more.	60 min.
2:00 p.m. – 2:10 p.m.	Break	10 min.
2:10 p.m. – 3:20 p.m.	<i>InfiniteOpt.jl: The Basics</i> – An introduction on how to compactly model and solve complex infinite-dimensional optimization problems.	70 min.
3:20 p.m. – 3:30 p.m.	Break	10 min.
3:30 p.m. – 4:15 p.m.	<i>InfiniteOpt.jl: New Modeling Strategies</i> – A tutorial on how InfiniteOpt.jl enables new formulation/solution approaches.	45 min.
4:15 p.m. – 4:45 p.m.	<i>InfiniteOpt.jl: Deployment Tools</i> – An overview of the API to enable rapid deployment of new modeling/solution techniques.	30 min.
4:45 p.m. – 5:00 p.m.	<i>Final Thoughts</i> – Summary of key points and planned future development. Provide resources for further learning.	15 min.



InfiniteOpt

Introduction

Why Julia?



- Speed
 - Up to C speeds



- Parallel Computing
 - Designed for it from the ground up



- Portability
 - Works on any O.S.



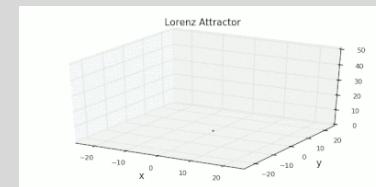
- Generality
 - Can be used for scripts, apps, metaprogramming, etc.



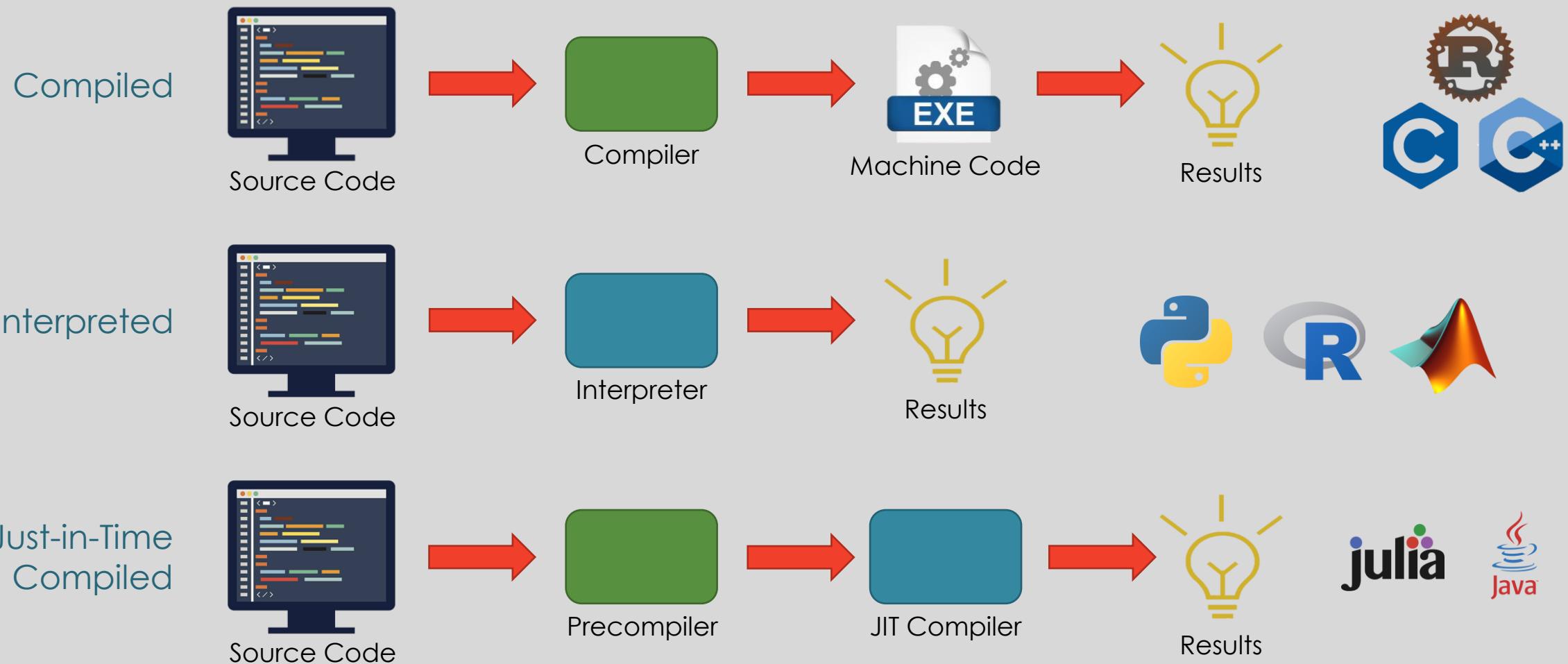
- Open Source
 - It's free!



- Scientific Computing
 - MATLAB-like linear algebra, symbolic math, scripting



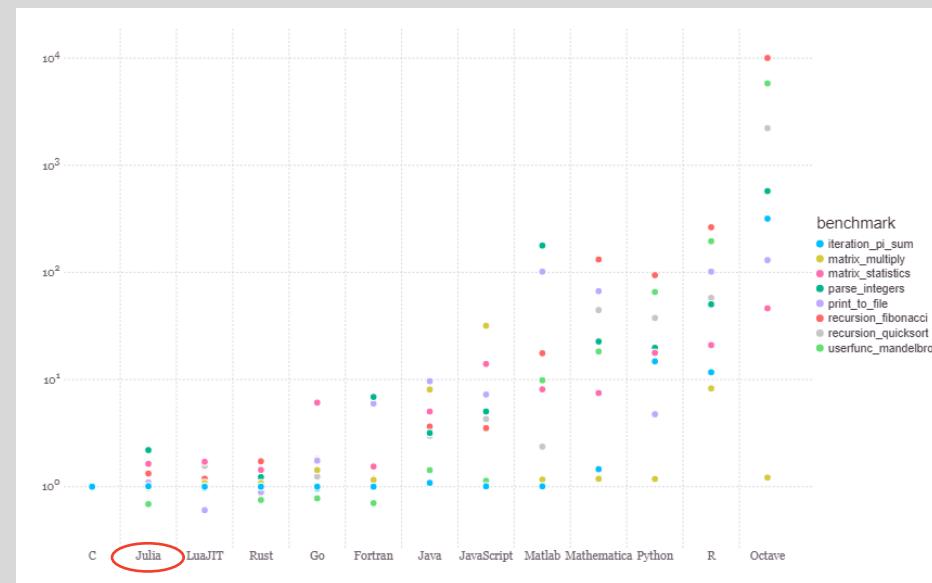
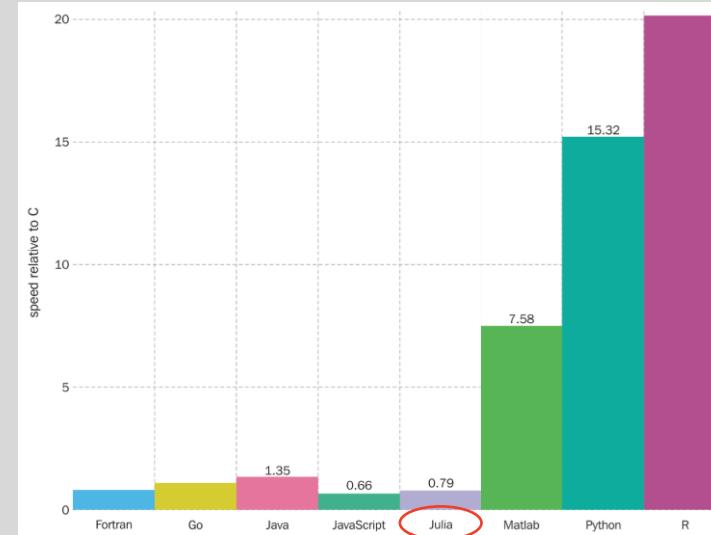
Programming Language Paradigms



*Over simplified

Performance

- Compiled Languages
 - Difficult “low-level” code
 - Cannot be modified while running
 - Not very portable between computers
 - Fast
- Interpreted Languages
 - Easy “high-level” syntax
 - Can be modified while running
 - Highly portable
 - Slow
- JIT Compiled Languages
 - Can be “high-level” (only Julia)
 - Highly portable
 - Can be modified while running
 - Fast*

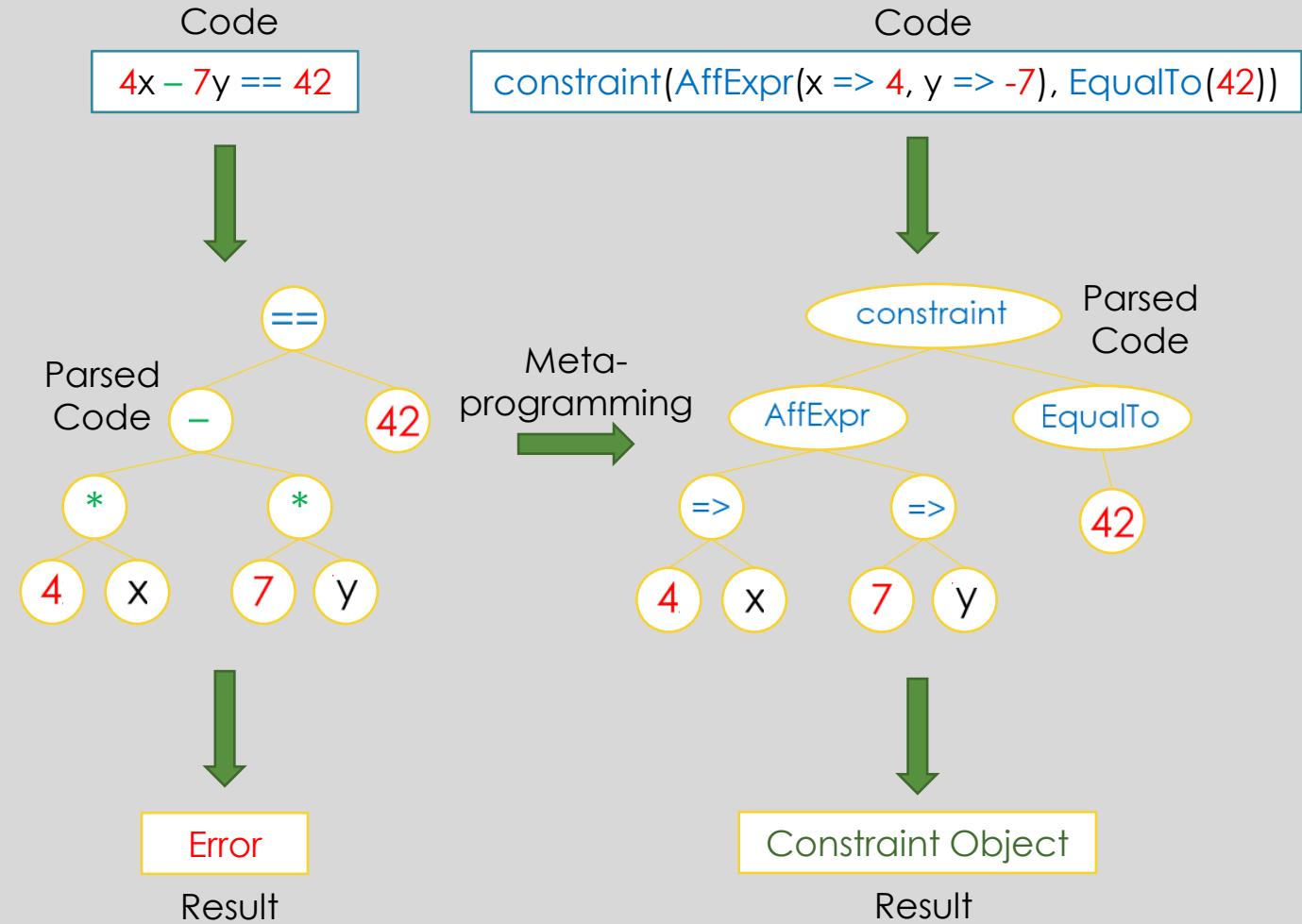


Metaprogramming (Macros)

- Intercept code before it is run
- Write Julia code that writes Julia code
- **Enables easy symbolic coding for users**

```
@constraint(4x - 7y == 42)
```

```
constraint(AffExpr(x => 4, y => -7), EqualTo(42))
```

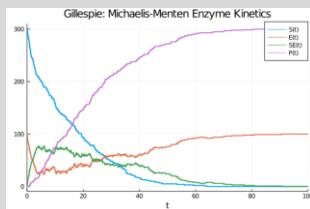


Packages

- ModelingToolkit.jl
 - Symbolic math ~1000 times faster than SymPy



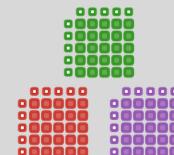
- DifferentialEquations.jl
 - Solve ODEs, PDEs, SDEs, DAEs, more
 - Fastest implementation available



- JuMP.jl
 - Symbolic interface for optimization

```
6  @constraint(model, 4f + 2s <= 4800)
7  @constraint(model, f + s <= 1750)
8  @constraint(model, 0 <= f <= 1000)
9  @constraint(model, 0 <= s <= 1500)
```

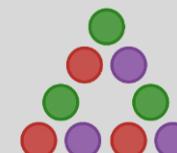
- DataFrames.jl
 - Efficient datasets (like Pandas)



- Flux.jl
 - Efficient/interpretable machine learning



- DistributedArrays.jl (GPU parallelization)
 - Easy parallelization for CPU cores/threads and GPUs





Why JuMP.jl?

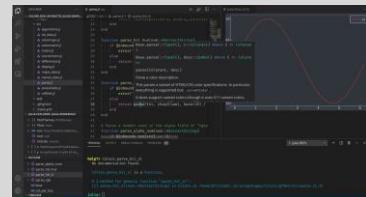
- High-level Syntax
 - Novice friendly

```
6 @constraint(model, 4f + 2s <= 4800)
7 @constraint(model, f + s <= 1750)
8 @constraint(model, 0 <= f <= 1000)
9 @constraint(model, 0 <= s <= 1500)
```

- Solver Library
 - Rapidly try different solvers



- Programmable
 - Easy embed in a data script



- Extensible
 - Can be extended for advanced techniques



- Open Source
 - It's free!

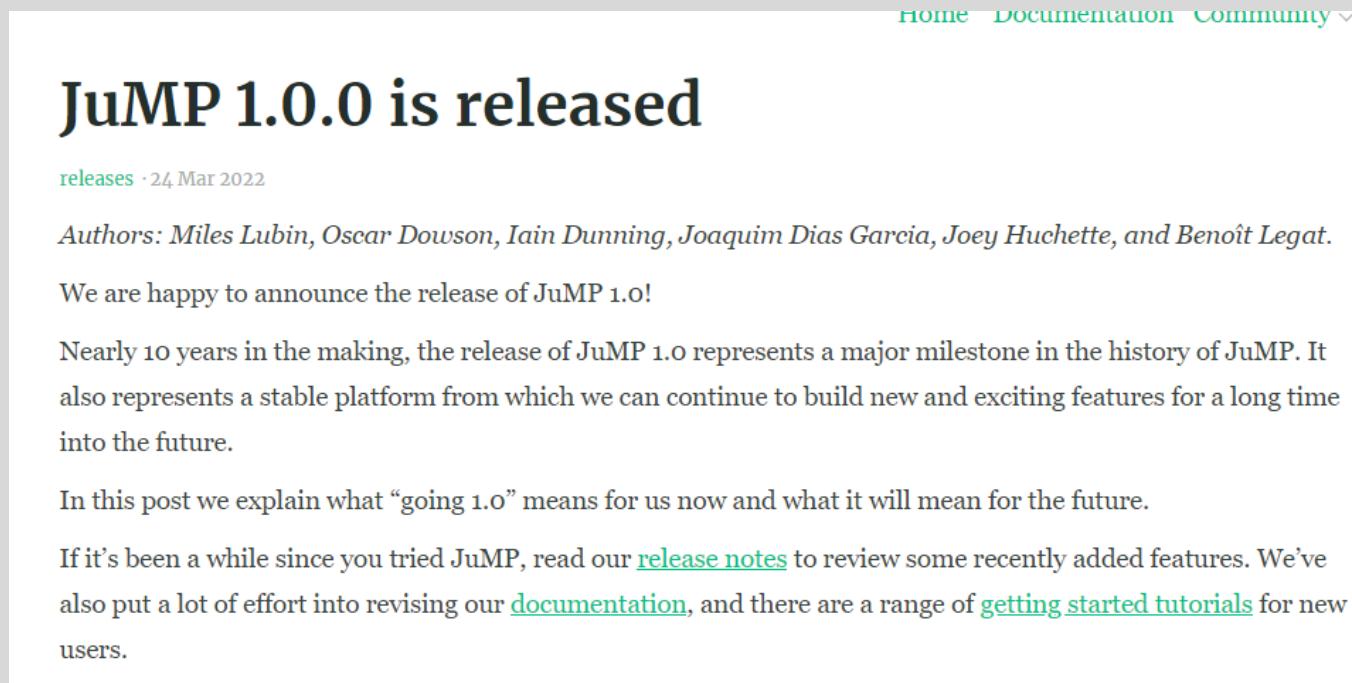


- Performance
 - Highly performant (it's Julia)



JuMP.jl 1.0

- After nearly 10 years, JuMP.jl is now at version 1.0
- “By releasing JuMP 1.0.0, the core contributors are ensuring that all code you write using a 1.x.y release of JuMP will continue to work” until 2.0.0



The screenshot shows a blog-style post on a website. At the top right, there are navigation links: "Home", "Documentation", and "Community". Below the header, the main title "JuMP 1.0.0 is released" is displayed in a large, bold, dark font. Underneath the title, the word "releases" is followed by the date "24 Mar 2022". A short bio follows: "Authors: Miles Lubin, Oscar Dowson, Iain Dunning, Joaquim Dias Garcia, Joey Huchette, and Benoît Legat." The post begins with the text: "We are happy to announce the release of JuMP 1.0!" It continues to explain the significance of the release: "Nearly 10 years in the making, the release of JuMP 1.0 represents a major milestone in the history of JuMP. It also represents a stable platform from which we can continue to build new and exciting features for a long time into the future." The author then discusses what "going 1.0" means: "In this post we explain what ‘going 1.0’ means for us now and what it will mean for the future." Finally, the author encourages users to review recent changes and provides links to documentation and tutorials: "If it’s been a while since you tried JuMP, read our [release notes](#) to review some recently added features. We’ve also put a lot of effort into revising our [documentation](#), and there are a range of [getting started tutorials](#) for new users."

What is Finite Optimization?

Idea: Determine the “best” set of **decisions** for a given problem.

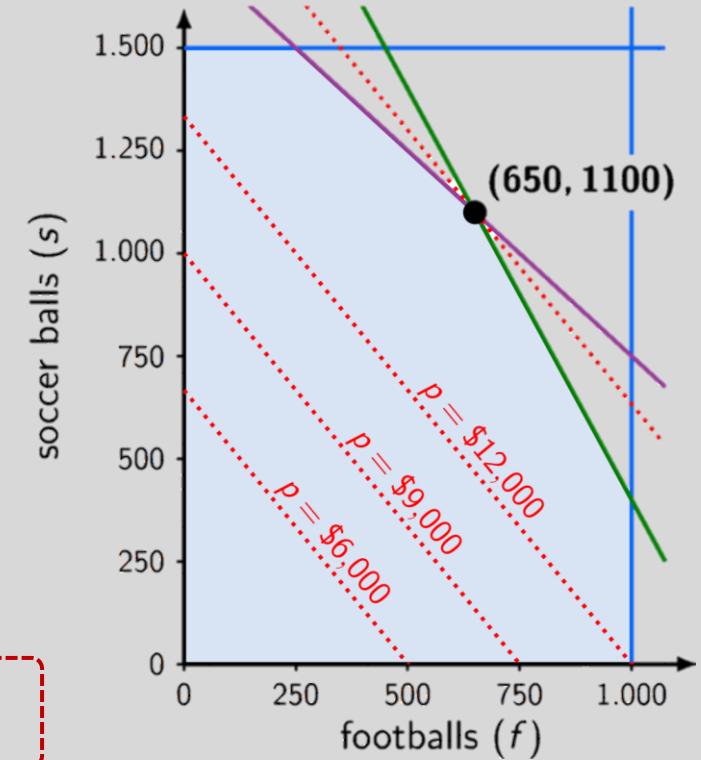
Aspects

- Parameters
 - Fixed quantities/data
- Decision Variables
 - Scalar values to optimize
- Objective
 - Decision criteria
- Constraints
 - Modeling equations
 - Decision requirements

$$\begin{aligned} & \max_{f, s} && 12f + 9s \\ \text{s.t. } & && 4f + 2s \leq 4800 \\ & && f + s \leq 1750 \\ & && 0 \leq f \leq 1000 \\ & && 0 \leq s \leq 1500 \end{aligned}$$

Finite # of decisions

Sports Store Example



Modeling in JuMP.jl

$$\begin{aligned} \max_{f,s} \quad & 12f + 9s \\ \text{s.t.} \quad & 4f + 2s \leq 4800 \\ & f + s \leq 1750 \\ & 0 \leq f \leq 1000 \\ & 0 \leq s \leq 1500 \end{aligned}$$

- Initialize **model**

```
1 using JuMP, Gurobi
2 model = Model(Gurobi.Optimizer)
```

- Define **variables**

```
3 @variable(model, f)
4 @variable(model, s)
```

- Define **objective**

```
5 @objective(model, Max, 12f + 9s)
```

- Define **constraints**

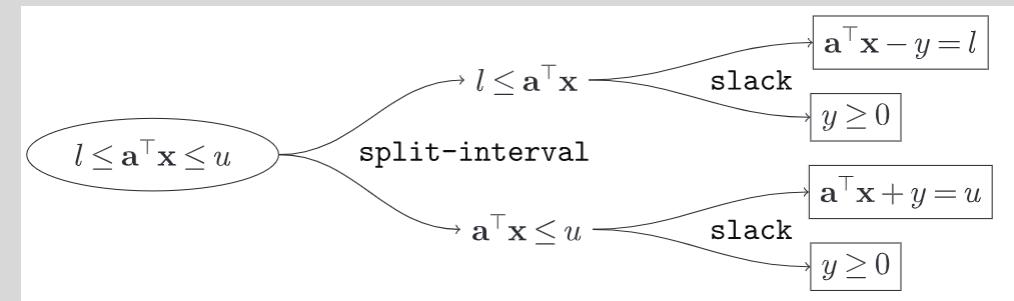
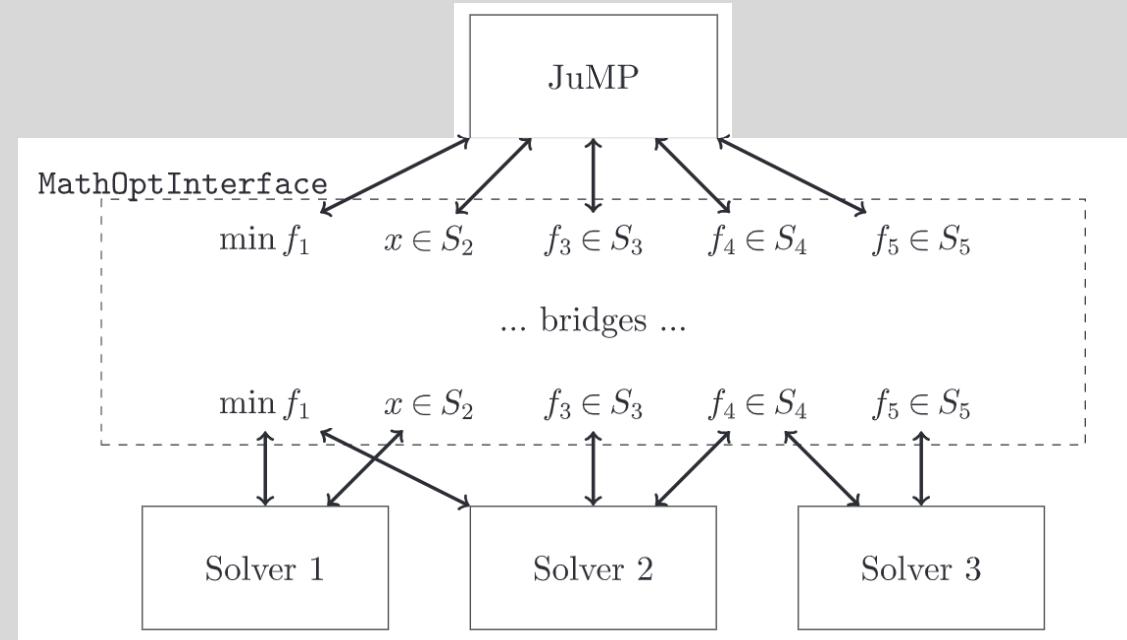
```
6 @constraint(model, 4f + 2s <= 4800)
7 @constraint(model, f + s <= 1750)
8 @constraint(model, 0 <= f <= 1000)
9 @constraint(model, 0 <= s <= 1500)
```

- **Solve**

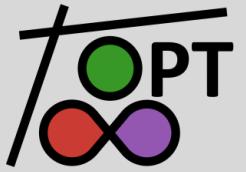
```
10 optimize!(model)
11 profit = objective_value(model)
12 f_best = value(f)
13 s_best = value(s)
```

JuMP.jl Architecture

- JuMP.jl provides the high-level interface (the “macro sugar”)
- Model is stored in MathOptInterface.jl model
 - Constraints are of the form *function* in set
- Bridges convert constraints into form that solvers support
 - Solvers don’t explicitly support every *function-set* combination
- MathOptInterface.jl preserves the constraint mappings
- Solvers interface with MathOptInterface.jl

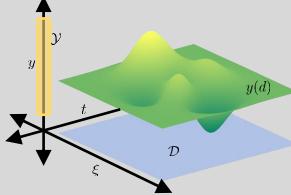


Why InfiniteOpt.jl?



- Unifying Abstraction

- Captures a wide breadth of problem classes



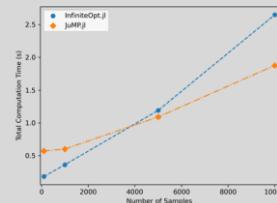
- High-level Syntax

- Compactly express problems

```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, yb ≤ yc * U)
@constraint(m, E(yc, ξ) ≥ α)
@constraint(m, ya(θ) + z[2] == β)
```

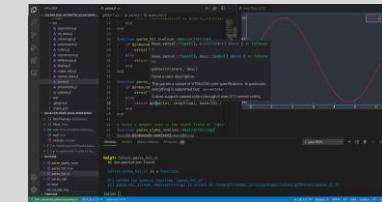
- Performance

- Quickly build complex models at scale



- Built on JuMP.jl

- Draw from an extensive suite of solvers



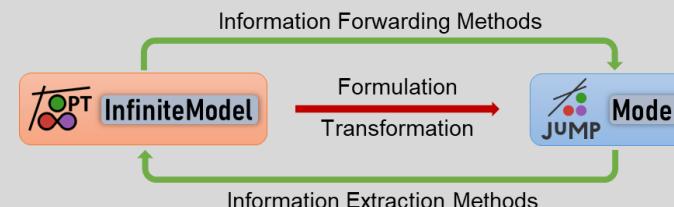
- Open Source

- It's free!



- Enables Accessible Research

- Implement advanced methods for general users



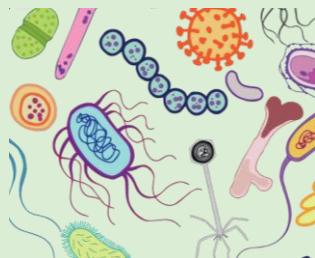
Applications over Infinite Domains

Idea: Many engineering applications are parameterized over **infinite (continuous) domains** (e.g., space & time)

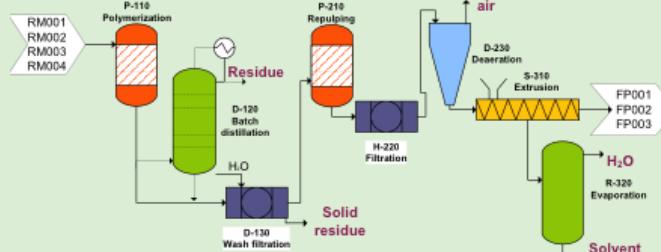
Illustrative Applications



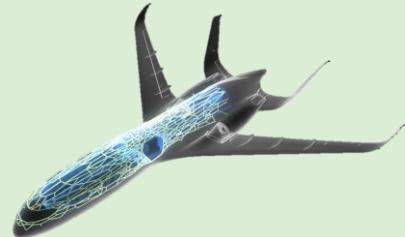
Autonomous Vehicles
(time)



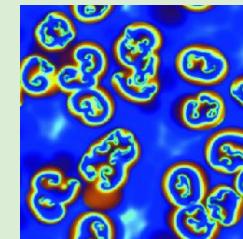
Microbial Communities
(time)



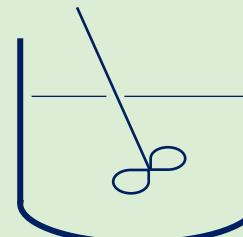
Process Systems
(time)



Structural Design
(space)



Diffusive Processes
(time & space)



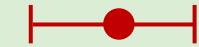
Reactive Systems
(time)

Uncertainty

- Environment



- Fitted parameters



- Forecasting



- Small length scales



What is Infinite Optimization?

Idea: Determine the “best” set of **decision functions** for a given problem.

New Aspects

- Infinite Variables

$$y(t)$$

$$y(t, x)$$

$$y(\xi)$$

- Measures

$$\mathbb{R}[y(\xi)]$$

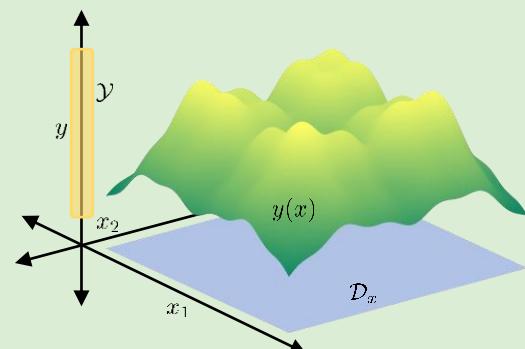
$$\int_{[-1,1]^3} \int_0^T y(t, x) dt dx$$

- Derivatives

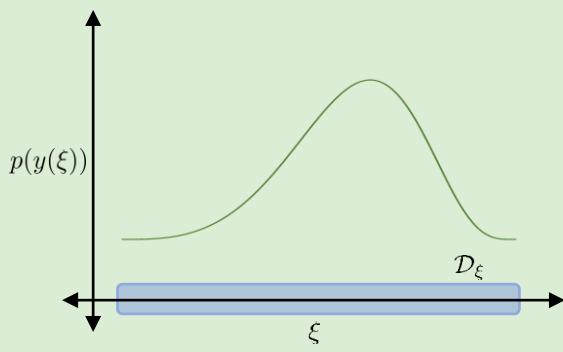
$$\frac{dy(t)}{dt}$$

$$\nabla_x y(t, x)$$

Application Fields



PDE-Constrained



Stochastic



Dynamic

Infinite # of decisions

Motivating Example: Stochastic Optimal Control

Goal: Determine **optimal social distancing policy** to control spread of a contagion and minimize economic impact

Problem Setup

- SEIR disease model

$$\frac{dy_s}{dt} = (y_u - 1)\beta y_s y_i$$

$$\frac{dy_e}{dt} = (1 - y_u)\beta y_s y_i - \xi y_e$$

$$\frac{dy_i}{dt} = \xi y_e - \gamma y_i$$

$$\frac{dy_r}{dt} = \gamma y_i$$

s: susceptible pop.
e: exposed pop.
i: infectious pop.
r: recovered pop.
u: social distancing

- Incubation constant ξ is **uncertain** (statically)

Formulation

$$\begin{aligned} & \min \int_{t \in \mathcal{D}_t} y_u(t) dt \\ \text{s.t. } & \frac{\partial y_s(t, \xi)}{\partial t} = (y_u(t) - 1)\beta y_s(t, \xi) y_i(t, \xi), & \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_e(t, \xi)}{\partial t} = (1 - y_u(t))\beta y_s(t, \xi) y_i(t, \xi) - \xi y_e(t, \xi), & \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_i(t, \xi)}{\partial t} = \xi y_e(t, \xi) - \gamma y_i(t, \xi), & \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_r(t, \xi)}{\partial t} = \gamma y_i(t, \xi), & \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & y_s(0, \xi) = s_0, y_e(0, \xi) = e_0, y_i(0, \xi) = i_0, y_r(0, \xi) = r_0, & \forall \xi \in \mathcal{D}_\xi \\ & y_i(t, \xi) \leq i_{max}, & \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & y_u(t) \in [0, \bar{u}] \\ & \xi \sim \mathcal{U}(\underline{\xi}, \bar{\xi}) \end{aligned}$$

Infinite Domains & Parameters

Infinite Domains

Definition

Domains of **infinite cardinality**
(e.g., continuous spaces)

Notation

$$\mathcal{D}_\ell \subseteq \mathbb{R}^{n_\ell}$$

where $|\mathcal{D}_\ell| = \infty$

Infinite Parameters

Definition

Parameters **living on infinite domains.**

Notation

$$d_\ell \in \mathcal{D}_\ell$$

Problem Domain

Definition

Domain of infinite-dimensional problem.

Notation

$$\mathcal{D} := \prod_{\ell \in \mathcal{L}} \mathcal{D}_\ell$$

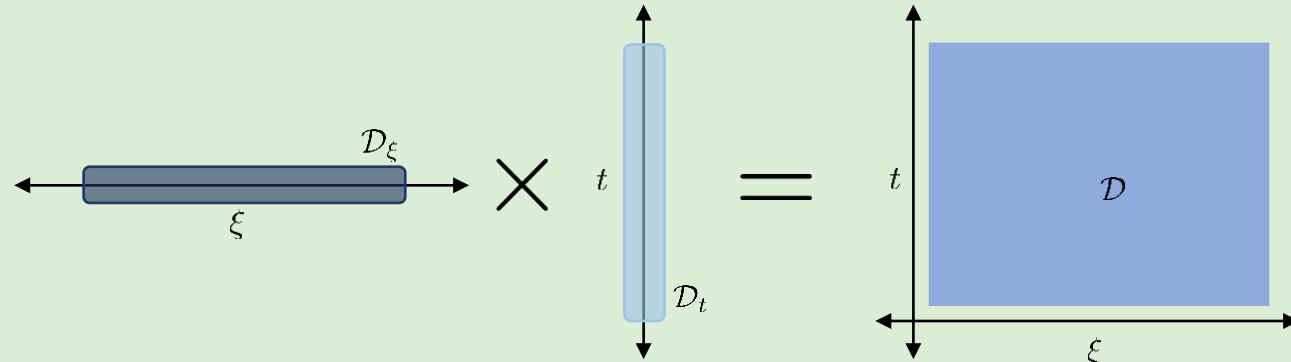
Examples

$$t \in \mathcal{D}_t = [0, 10]$$

$$\xi \in \mathcal{D}_\xi = (-\infty, \infty)^n$$

$$\mathcal{D} = \mathcal{D}_t \times \mathcal{D}_\xi = [0, 10] \times (-\infty, \infty)^n$$

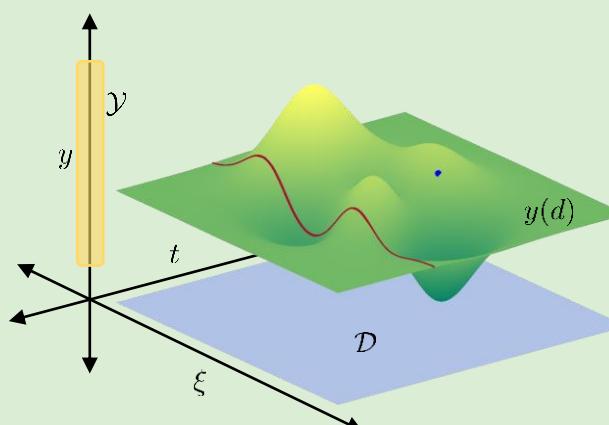
Illustration



Decision Variables

Infinite Variables

Illustration



Definition

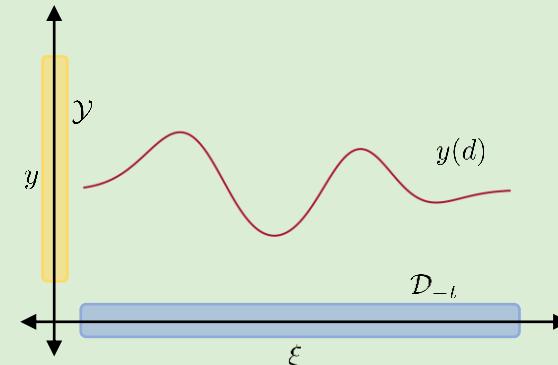
Functions w/ infinite parameter inputs → manifolds to shape

Notation

$$y : \mathcal{D} \mapsto \mathcal{Y} \subseteq \mathbb{R}^{n_y}$$

$$y(t, x, \xi)$$

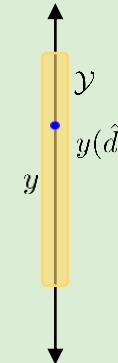
Semi-Infinite Variables



Partially restricted infinite variable

$$y : \mathcal{D}_{-l} \mapsto \mathcal{Y} \subseteq \mathbb{R}^{n_y}$$
$$y(0, x, \xi)$$

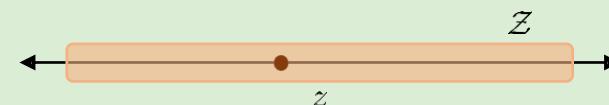
Point Variables



Fully restricted infinite variable

$$y(\hat{d}) \in \mathcal{Y} \subseteq \mathbb{R}^{n_y}$$
$$y(0, 1, 0.43)$$

Finite Variables

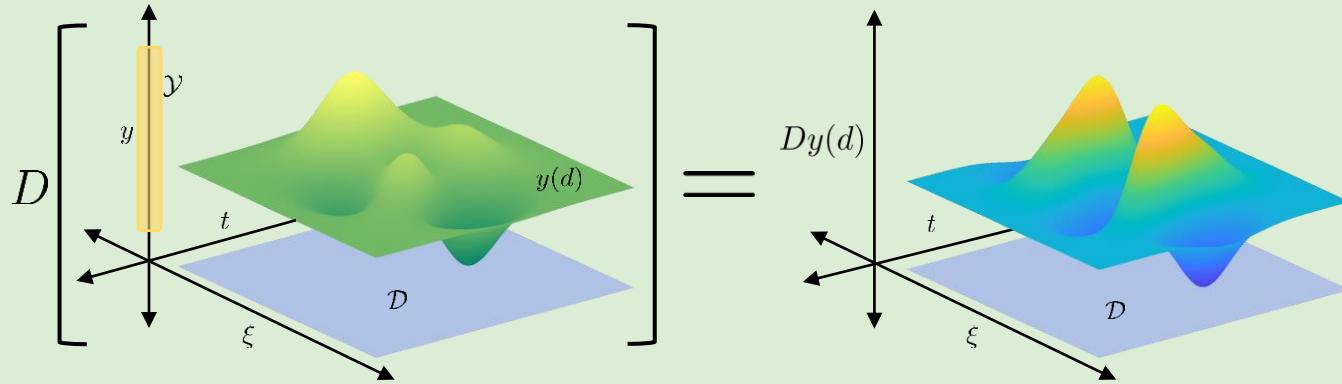


Indexed over finite domains

$$z \in \mathcal{Z} \subseteq \mathbb{R}^{n_z}$$

Operators

Differential Operators



Definition

Capture **how a variable changes** over its infinite domain

Notation

$$Dy : \mathcal{D} \mapsto \mathbb{R}^{n_d}$$
$$\frac{dy(t)}{dt} \quad \nabla_x y(t, x)$$

Definition

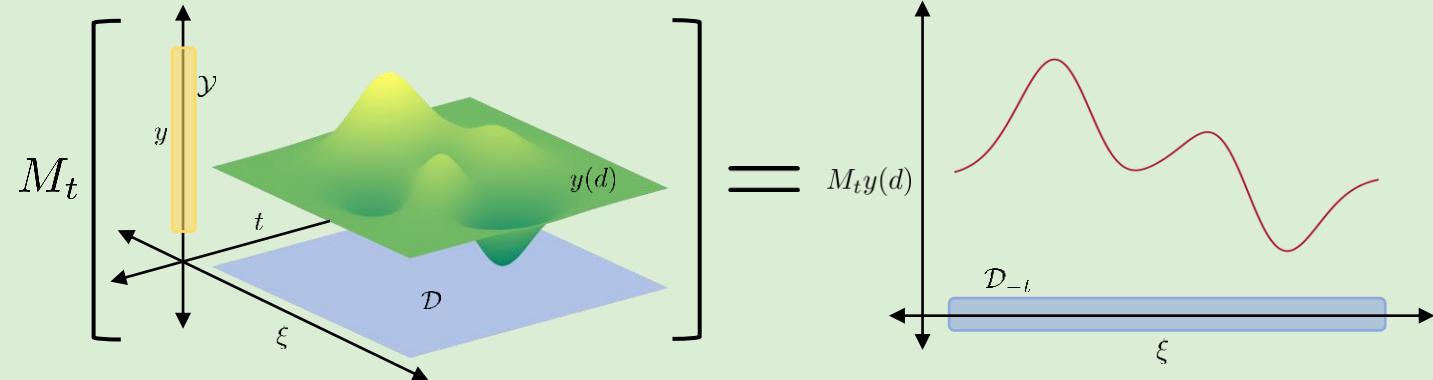
Summarize/collapse **functions** over an infinite domain

Notation

$$M_\ell y : \mathcal{D}_{-\ell} \mapsto \mathbb{R}^{n_m}$$

$$\int_{(t,x) \in \mathcal{D}_{t,x}} y(t, x) dt dx \quad \mathbb{R}[y(\xi)]$$

Measure Operators



Objectives and Constraints

Definition

Scalarize cost function via **measure operators**.

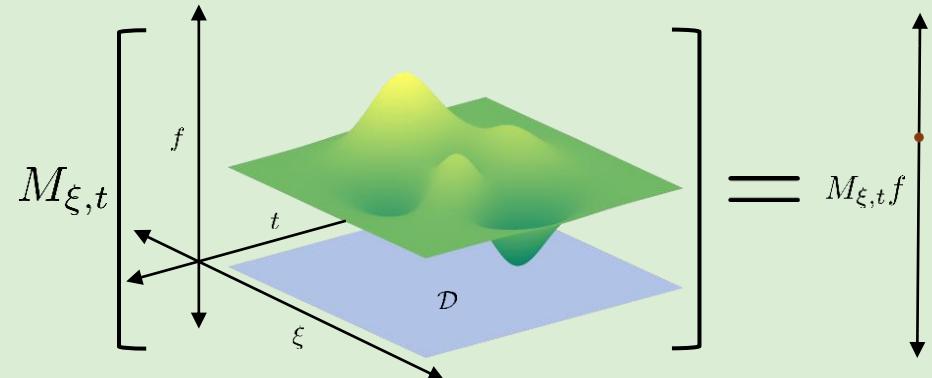
Notation

$$Mf(Dy, y(d), z, d)$$

Objectives

Example

$$\min_{y(t, \xi)} \mathbb{E}_\xi \left[\int_{t \in \mathcal{D}_t} f(y(t, \xi)) dt \right]$$



Algebraic Constraints

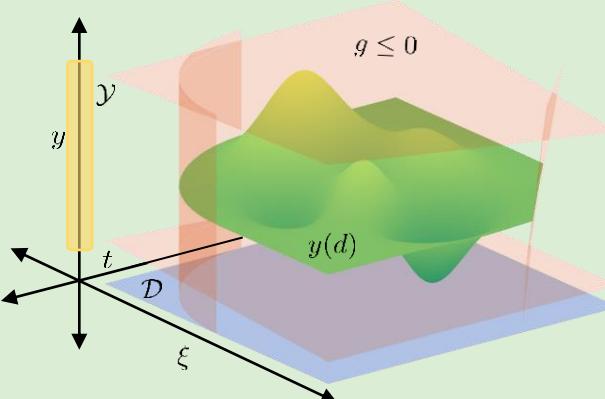
Enforce condition on function g for all values of d .

Notation

$$g(Dy, y(d), z, d) \leq 0, \quad d \in \mathcal{D}$$

$$y^2(t, \xi) + y(t, \xi) \leq 0, \quad t \in \mathcal{D}_t, \quad \xi \in \mathcal{D}_\xi$$

Constraints



Measure Constraints

Enforce constraint on measure M of functions h .

Notation

$$Mh(Dy, y(d), z, d) \geq 0$$

$$\mathbb{P}_\xi(y(t, \xi) \leq 0) \geq \alpha, \quad t \in \mathcal{D}_t$$

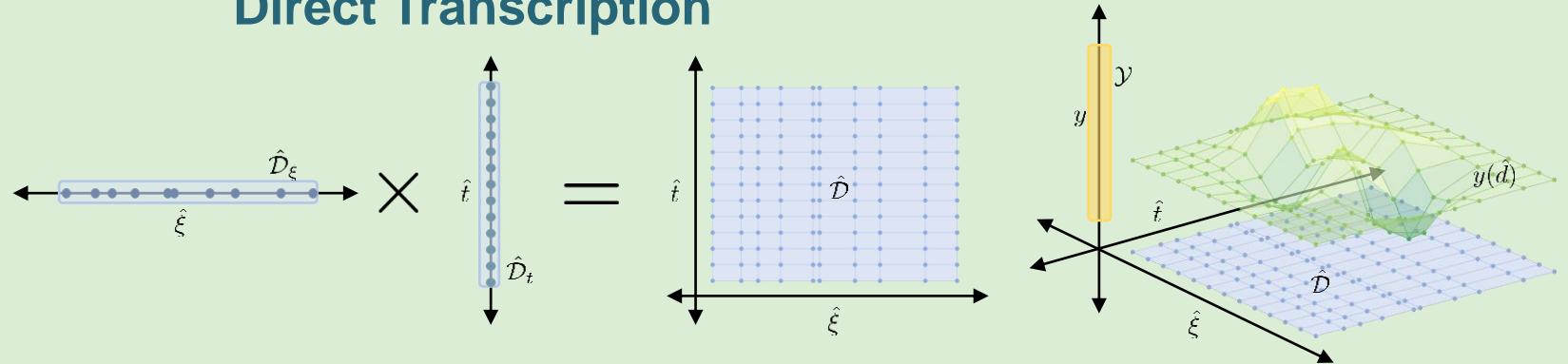
Transformations

Idea: Transform infinite model into a **finite formulation** that is amendable to **conventional optimization solvers**.

Direct Transcription

Project onto set of finite points $\hat{\mathcal{D}}$

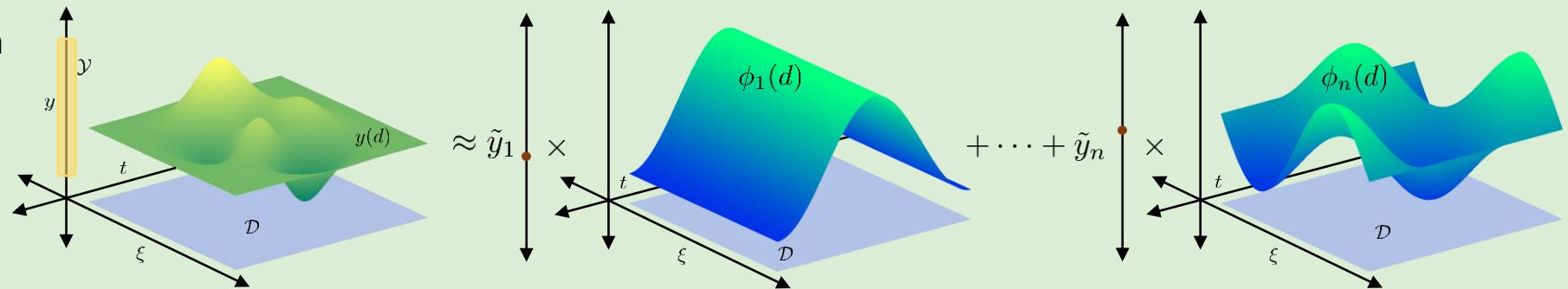
$$\hat{\mathcal{D}} := \prod_{\ell \in \mathcal{L}} \{\hat{d}_{\ell,i} : \hat{d}_{\ell,i} \in \mathcal{D}_\ell, i \in \mathcal{I}_\ell\}$$



Method of Weighted Residuals

Project onto set of known basis functions

$$y(d) \approx \sum_{i \in \mathcal{I}} \tilde{y}_i \phi_i(d)$$



Modeling in InfiniteOpt.jl

- Initialize the **model**

```
using InfiniteOpt, Distributions, Ipopt
m = InfiniteModel(Ipopt.Optimizer)
```

- Add the **infinite parameters**

$$t \in [t_0, t_f] \quad \xi \sim \mathcal{N}(\mu, \Sigma)$$

```
@infinite_parameter(m, t ∈ [t₀, tₙ], num_supports = 10)
@infinite_parameter(m, ξ[1:10] ~ MvNormal(μ, Σ))
```

- Add **variables** and their domain constraints

$$y_a(t) \in \mathbb{R}_+ \quad y_b(t, \xi) \in \mathbb{R}_+ \quad y_c(\xi) \in \{0, 1\} \quad z \in \mathbb{Z}^2$$

```
@variable(m, ya ≥ 0, Infinite(t))
@variable(m, yb ≥ 0, Infinite(t, ξ))
@variable(m, yc, Infinite(ξ), Bin)
@variable(m, z[1:2], Int)
```

- Define the **objective**

$$\min_{y_a(t), y_b(t, \xi), y_c(\xi), z} \int_{t \in \mathcal{D}_t} y_a(t)^2 + 2\mathbb{E}_\xi[y_b(t, \xi)]dt$$

```
@objective(m, Min, ∫(ya^2 + 2 * E(yb, ξ), t))
```

- Add the **constraints**

$$\frac{\partial y_b(t, \xi)}{\partial t} = 2y_b(t, \xi)^2 + y_a(t) - z_1, \quad \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi$$

$$\mathbb{P}(y_b(t, \xi) \leq 0) \geq \alpha, \quad \forall t \in \mathcal{D}_t$$

$$y_a(0) + z_2 = \beta$$

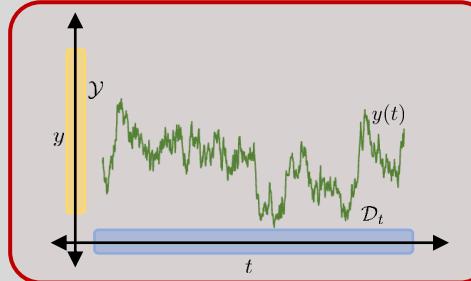
```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, yb ≤ yc * u)
@constraint(m, E(yc, ξ) ≥ α)
@constraint(m, ya(0) + z[2] == β)
```

- **Solve**

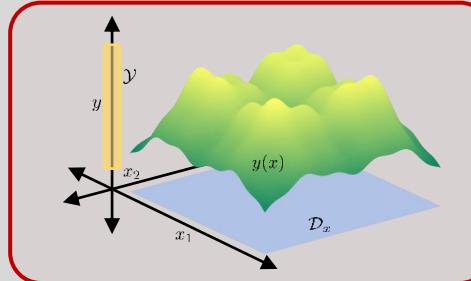
```
optimize!(m)
opt_objective = objective_value(m)
```

Innovating with InfiniteOpt.jl

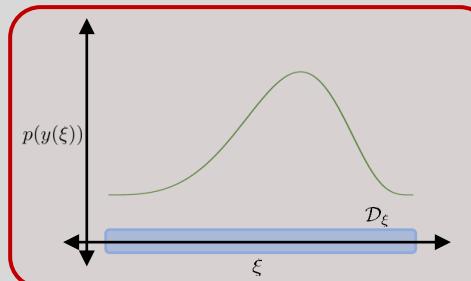
Traditional Formulations



Dynamic Optimization



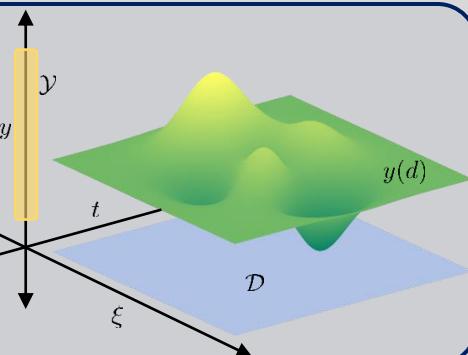
PDE-Constrained Optimization



Stochastic Optimization

InfiniteOpt

Unifying Abstraction

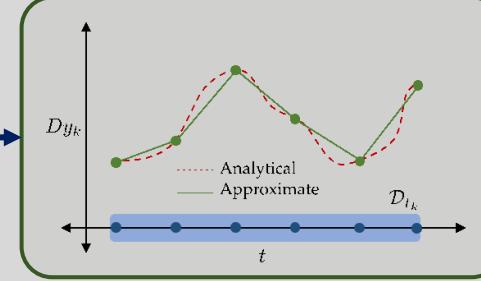


Infinite-Dimensional Optimization

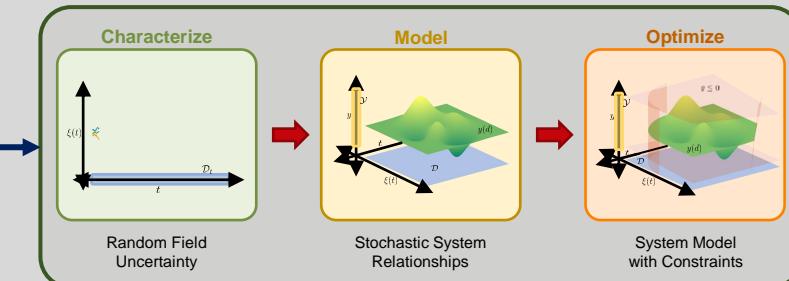
New Formulations



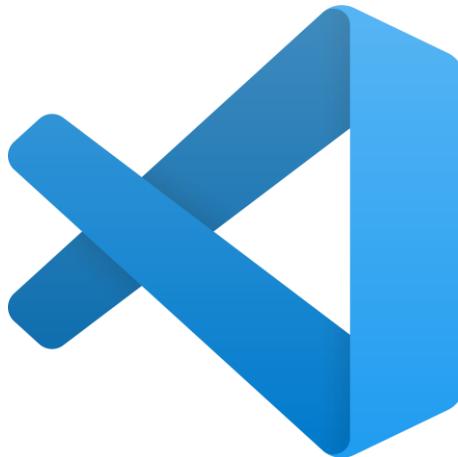
Generalized Shaping Measures



Continuous Time Estimation



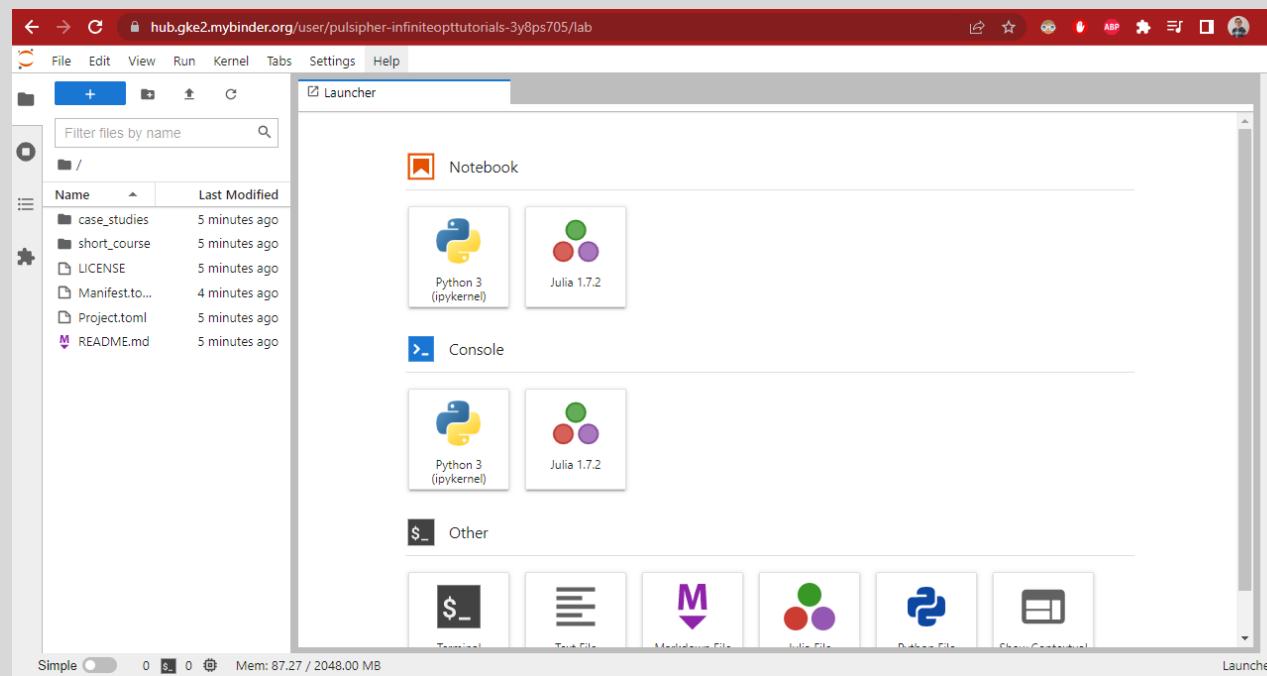
Random Field Optimization



Installation and Setup

Alternative: Binder

- Online interface via Binder: <https://mybinder.org/v2/gh/pulsipher/InfiniteOptTutorials/main>
- This is free, but slow to load
- Limited computing resources for optimizing models
- If you are using ChromeOS or Android then you'll have to use this



Install Julia

- Download latest Julia 1.7 (need at least 1.6) from <https://julialang.org/downloads/>
 - Windows: Choose 64-bit installer
 - macOS: Choose the x86 installer, do not use the ARM build even if you have an M-series processor
 - Linux: Choose the appropriate one for your distribution of Linux
 - Be sure to add it to the PATH (not the default)

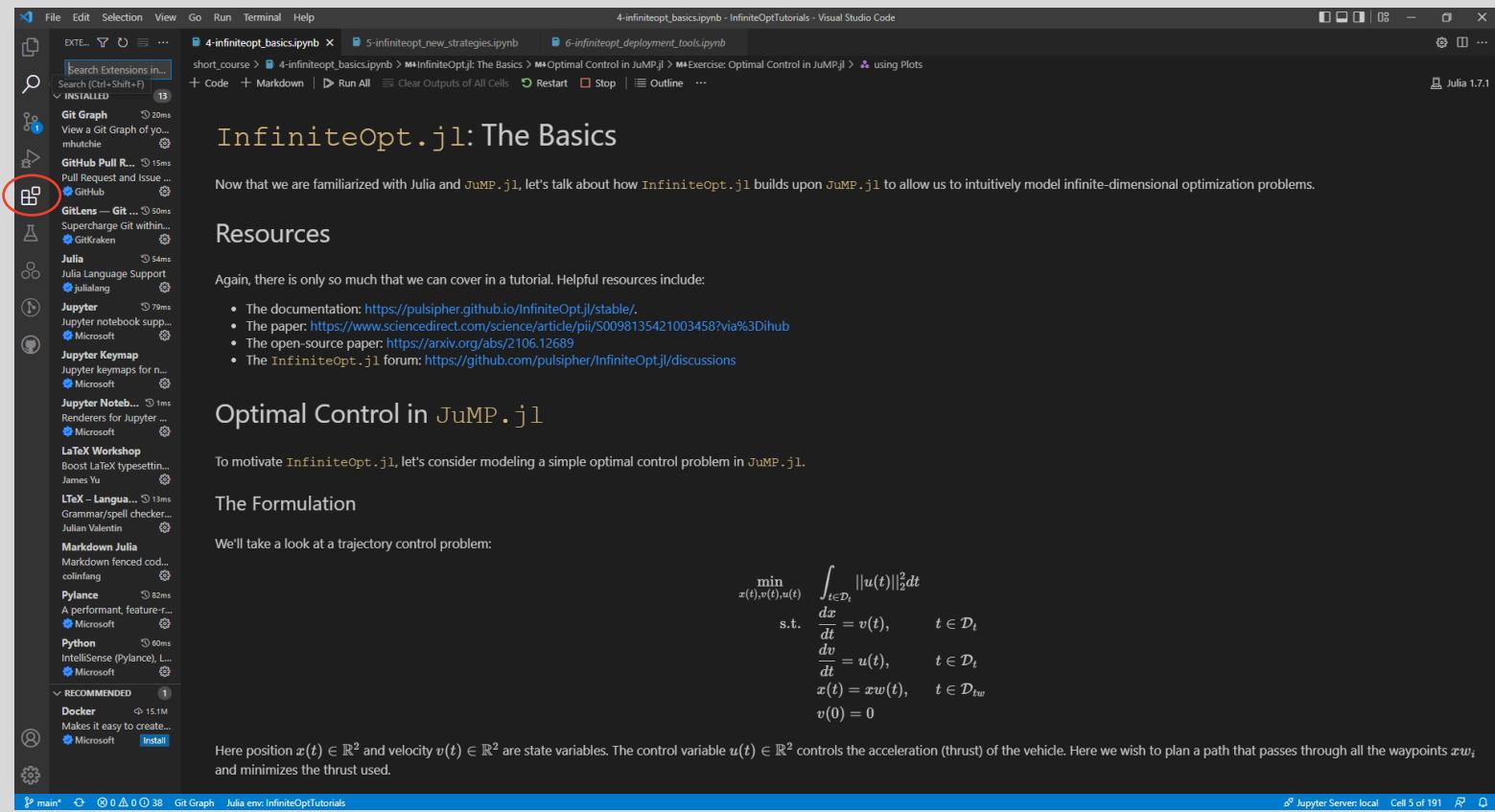
Current stable release: v1.7.2 (Feb 6, 2022)

Checksums for this release are available in both [MD5](#) and [SHA256](#) formats.

Windows [help]	64-bit (installer), 64-bit (portable)	32-bit (installer), 32-bit (portable)
macOS x86 (Intel or Rosetta) [help]	64-bit (.dmg), 64-bit (.tar.gz)	
macOS ARM (M-series Processor) [help]	64-bit (experimental)	
Generic Linux on x86 [help]	64-bit (glibc) (GPG), 64-bit (musl) ^[1] (GPG)	32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)	32-bit (ARMv7-a hard float) (GPG)
Generic FreeBSD on x86 [help]	64-bit (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG) GitHub

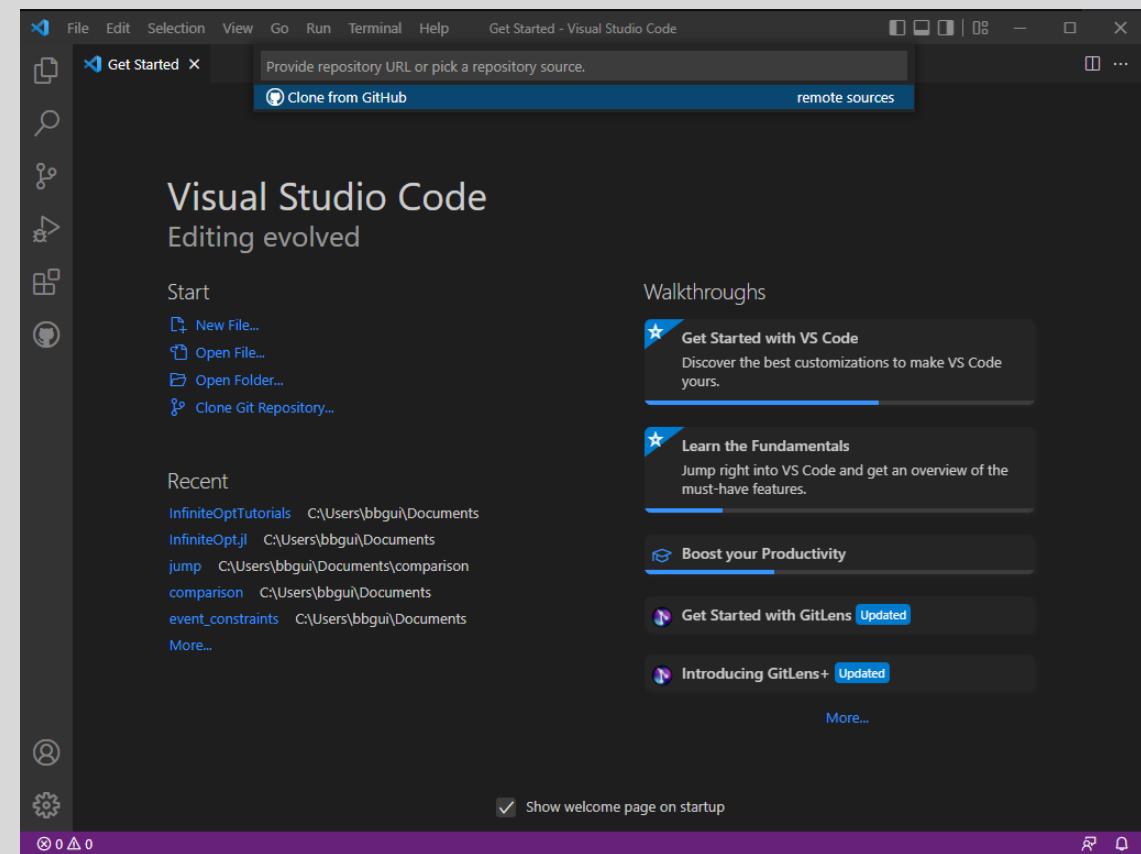
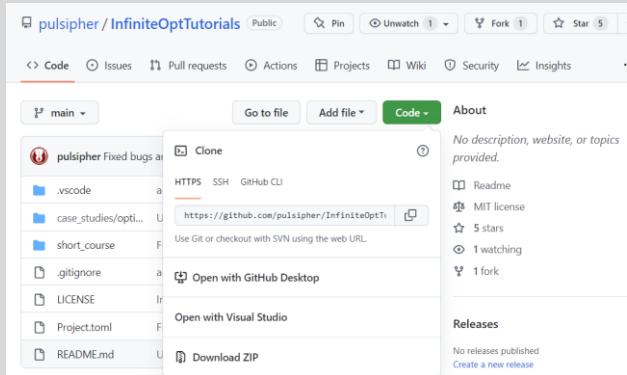
Install Visual Studio Code

- Download from <https://code.visualstudio.com/>
 - The default options are fine
- Install the extensions
 - Julia
 - Jupyter



Download the Course Material

- Located at <https://github.com/pulsipher/InfiniteOptTutorials>
 - Can Google “InfiniteOptTutorials” + “pulsipher”
- Download via VS Code (if git installed)
 - Click “Clone Git Repository”
 - Paste in the link
- Or download it from GitHub
 - Clone it
 - Or download the zip file
 - Or click the “Open with Visual Studio” button
 - Then open the folder in VS Code



Setup the Package Environment

- Now that everything is installed let's setup the Julia environment
- Make sure the open the "InfiniteOptTutorials" folder in VS Code first
- Steps
 - Click on "Julia env: ..." on the button bar
 - Select "InfiniteOptTutorials" from the drop-down bar
 - Now under "View" select "Command Palette"
 - Type "Julia: start repl" and push enter
 - In the Julia REPL type "]" and press enter
 - Now you should see "(InfiniteOptTutorials) pkg>"
 - Type "instantiate" and press enter
- Load the packages
 - Return to `julia>` via backspace
 - Load the packages via `using InfiniteOpt, Plots, HiGHS, Ipopt`

```
(InfiniteOptTutorials) pkg> instantiate
```

```
julia> using InfiniteOpt, Plots, HiGHS, Ipopt
```

Running Jupyter Notebook

- Origins
 - Iterative scripting for Julia, Python, and R
 - That's where the "Ju" comes from
- IJulia
 - Supported via the IJulia package
- VS Code
 - VS code supports Jupyter notebooks directly
 - No setup needed
 - Select the Julia kernel when prompted

The screenshot shows a Visual Studio Code interface with a dark theme. On the left is the Explorer sidebar, which lists a project folder named 'INFINITEOPTTUTORIALS' containing files like '.vscode', 'case_studies', 'short_course', 'figures', and several IPython notebooks (1-julia_overview.ipynb, 2-jump_overview.ipynb, etc.). The main editor area displays a Jupyter notebook titled 'Julia: A Practical Introduction'. The text in the notebook reads:

Here we provide a brief tutorial on coding in Julia, focusing on aspects that are helpful for working with models in `JuMP.jl` and `InfiniteOpt.jl`. As such, this is not intended as a comprehensive guide to coding in Julia. Note that this tutorial draws inspiration from the content provided in https://jump.dev/JuMP.jl/dev/tutorials/getting_started/getting_started_with_julia/#Getting-started-with-Julia.

Resources

- Julia's compendium of learning resources: <https://julialang.org/learning/>
- Julia's documentation: <https://jump.dev/JuMP.jl/stable/>
- Julia community forum: <https://discourse.julialang.org/>
- The help interface in the REPL (accessed via `? followed by the function name of interest`)

Arithmetic

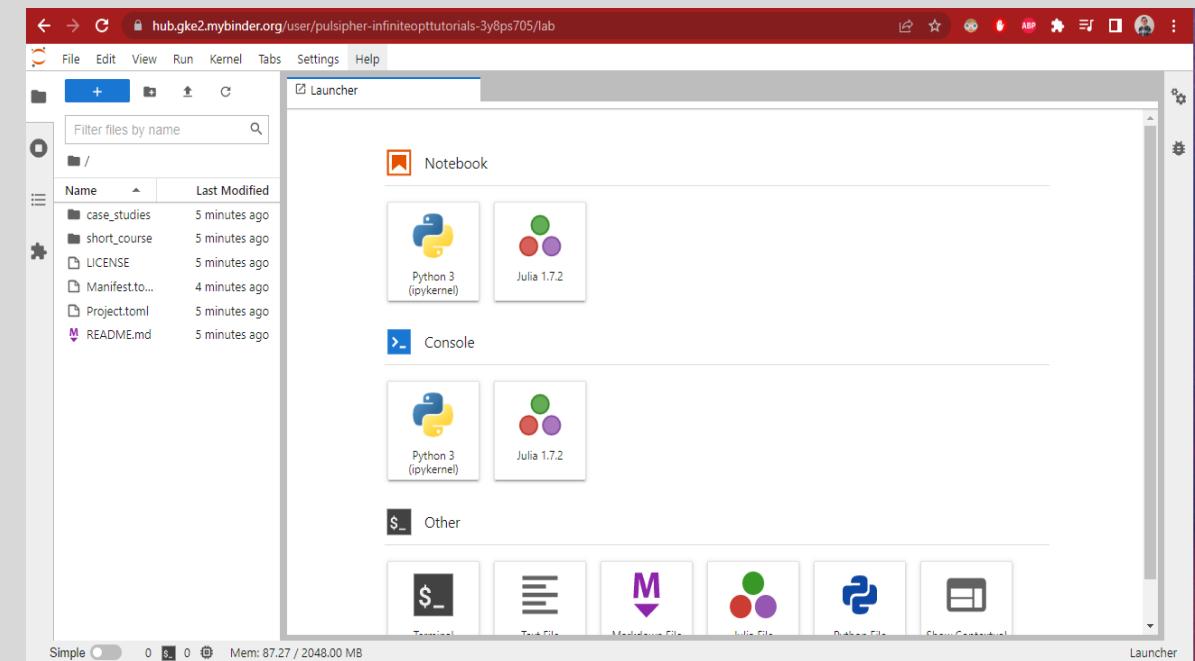
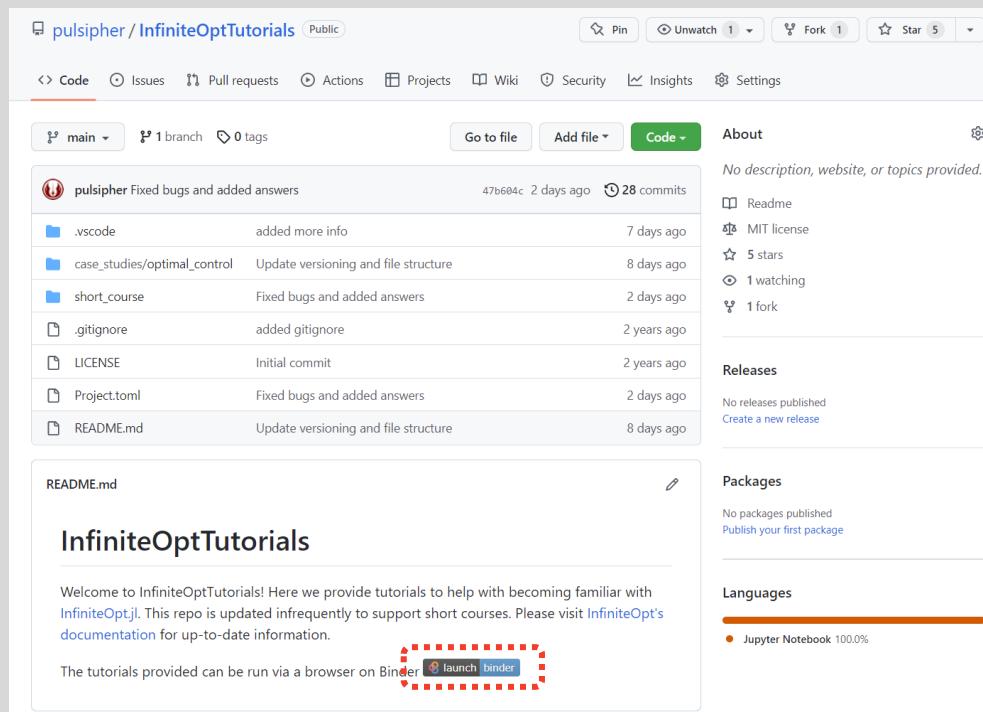
With optimization in mind, we'll be dealing with a lot of mathematical operations (arithmetic). Julia follows a straightforward syntax that is similar to MATLAB.

```
@show 1 + 1  
@show 1 - 2
```

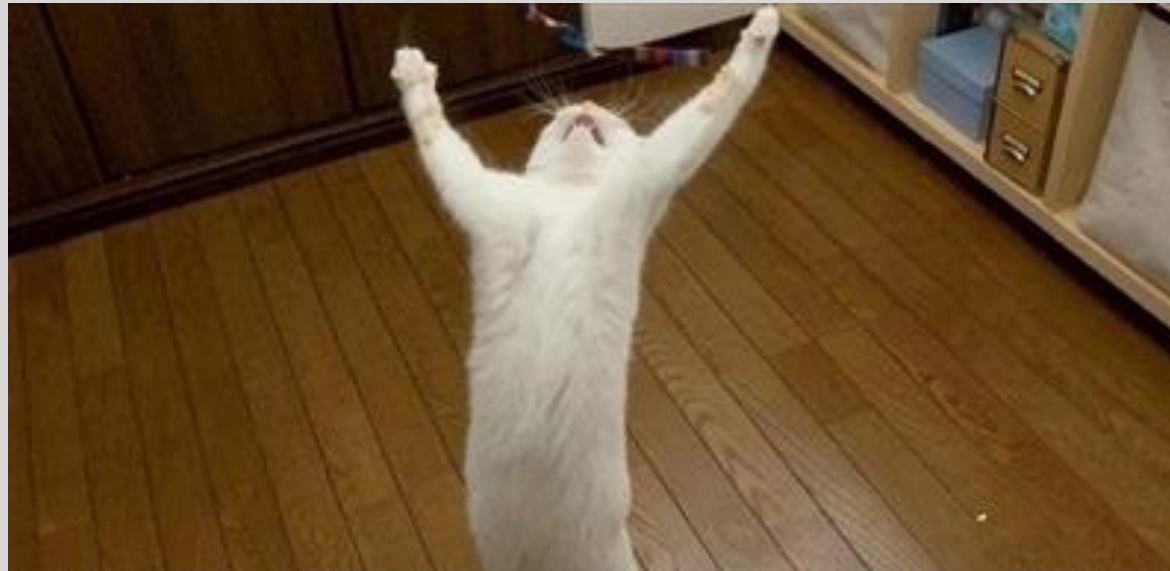
At the bottom, the status bar shows 'main*' in the file tab, and 'Julia env: InfiniteOptTutorials' in the status bar.

Having Trouble? Just use Binder Instead

- Online interface via Binder: <https://mybinder.org/v2/gh/pulsipher/InfiniteOptTutorials/main>
- This is free, but slow to load
- Limited computing resources for optimizing models



10 Minute Break Time

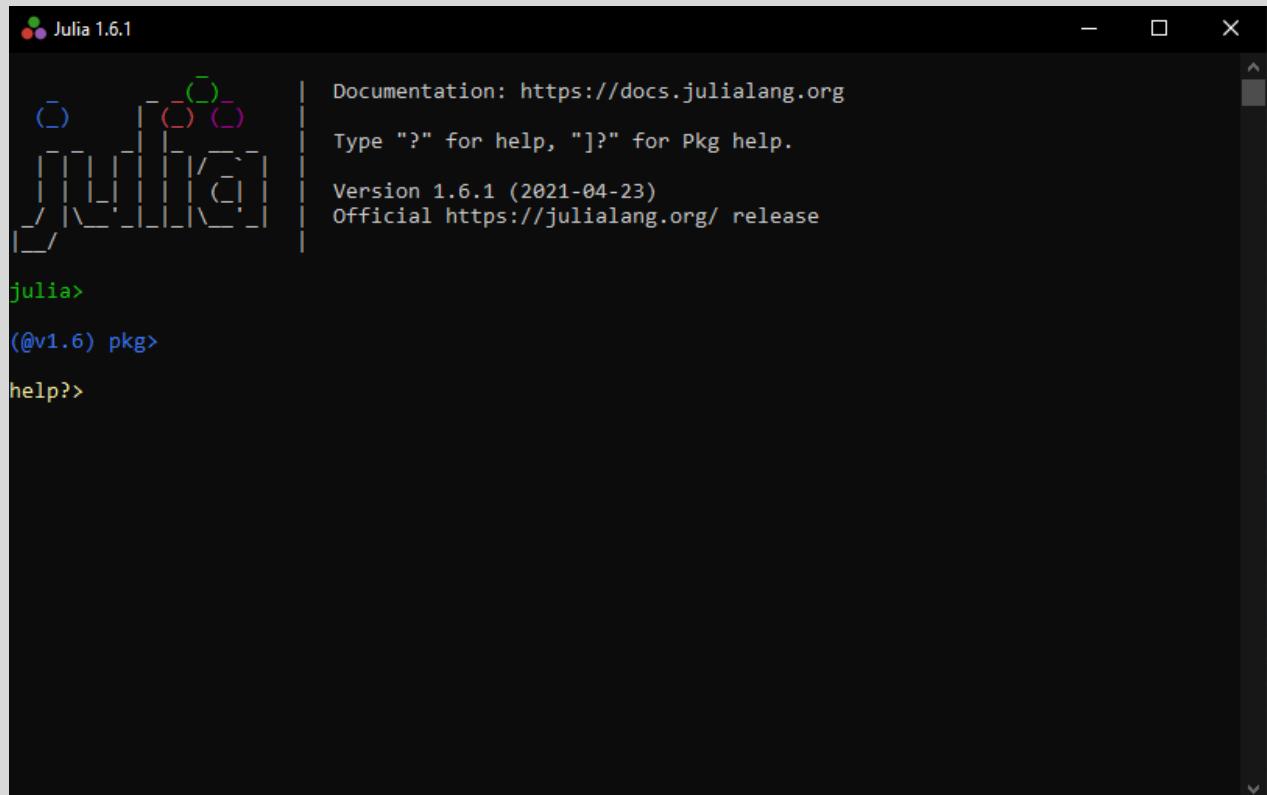




A Practical Introduction

The REPL

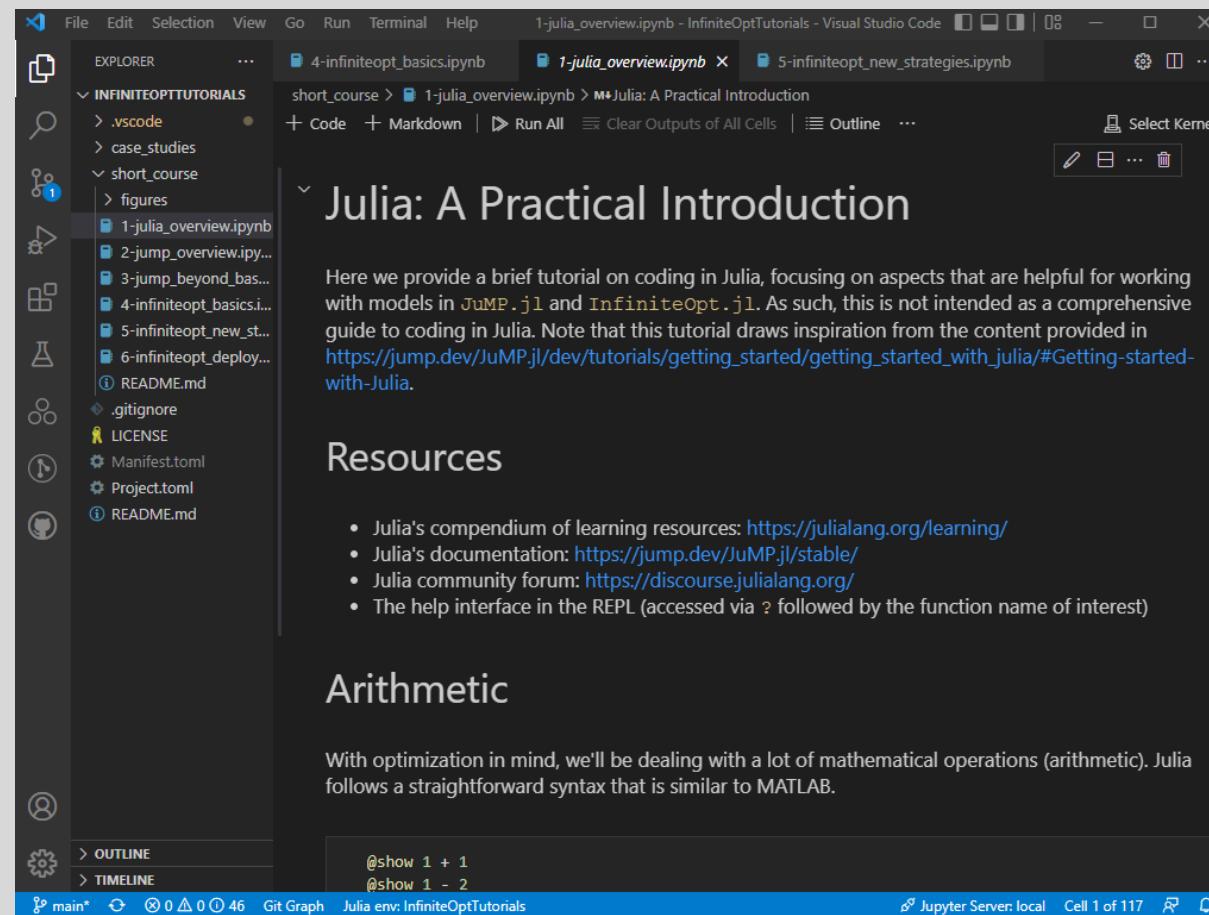
- What is it?
 - Read-Evaluate-Print-Loop
 - Executes commands, code, scripts
- What can I do with
 - Run quick commands
 - Run entire scripts
 - Play around
 - Get help
 - This is analogous to the Python kernel
- Package management
 - Use the package mode
 - More on this later



The screenshot shows the Julia 1.6.1 REPL window. At the top, there's a decorative graphic of colored brackets and parentheses. To the right of the graphic, the text reads: "Documentation: <https://docs.julialang.org>", "Type "?" for help, "]?" for Pkg help.", "Version 1.6.1 (2021-04-23)", and "Official <https://julialang.org/> release". Below this, the Julia prompt "julia>" is followed by "(@v1.6) pkg>". A command "help?>" is typed at the bottom of the window.

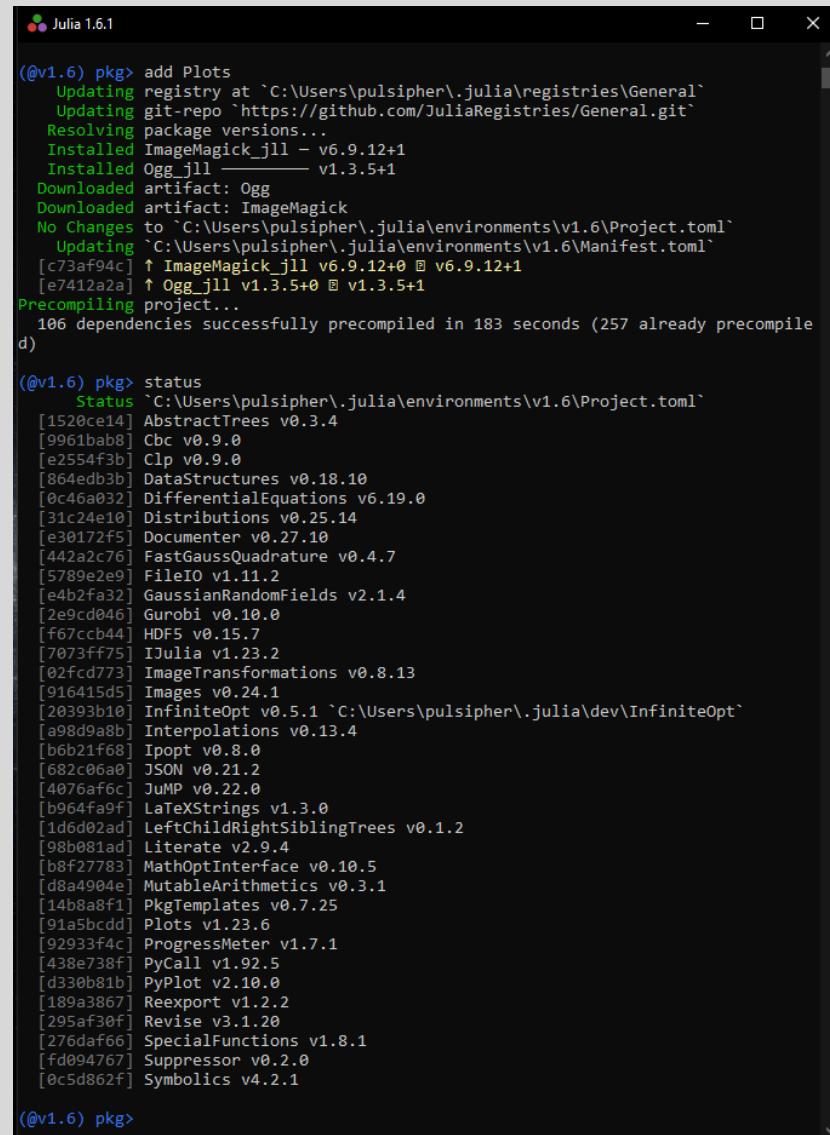
To the Tutorial!

- Open the “1-julia_overview.ipynb” jupyter notebook



Packages

- Libraries of abilities
 - Nearly 7,000 packages are available
 - Access a wealth of functions and capabilities
- Package manager
 - Use the Pkg.jl interface in the REPL
 - Manages dependencies
- Environments
 - An independent collection of packages
 - Stops packages from stepping on each others toes
 - See documentation for details
- Importing
 - Use the `using` or `import` keywords



```
Julia 1.6.1

(@v1.6) pkg> add Plots
  Updating registry at `C:\Users\pulsipher\.julia\registries\General`...
  Updating git-repo `https://github.com/JuliaRegistries/General.git`
  Resolving package versions...
  Installed ImageMagick_jll - v6.9.12+1
  Installed Ogg_jll └── v1.3.5+1
  Downloaded artifact: Ogg
  Downloaded artifact: ImageMagick
  No Changes to `C:\Users\pulsipher\.julia\environments\v1.6\Project.toml`...
  Updating `C:\Users\pulsipher\.julia\environments\v1.6\Manifest.toml`...
  [c73af94c] ↑ ImageMagick_jll v6.9.12+0 → v6.9.12+1
  [e7412a2a] ↑ Ogg_jll v1.3.5+0 → v1.3.5+1
  Precompiling project...
  106 dependencies successfully precompiled in 183 seconds (257 already precompiled)

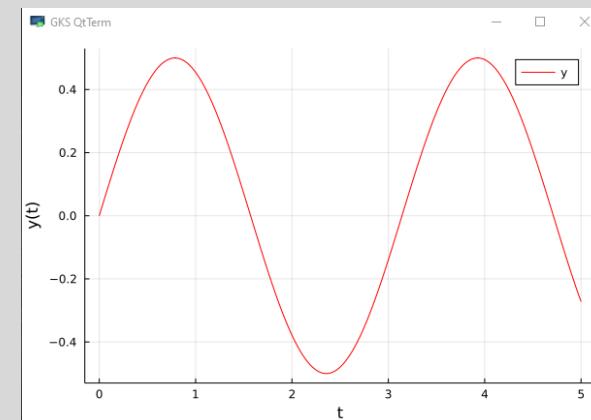
(@v1.6) pkg> status
  Status `C:\Users\pulsipher\.julia\environments\v1.6\Project.toml`...
  [1520ce14] AbstractTrees v0.3.4
  [9961bab8] Cbc v0.9.0
  [e2554f3b] Clp v0.9.0
  [864edb3b] DataStructures v0.18.10
  [0c46a032] DifferentialEquations v6.19.0
  [31c24e10] Distributions v0.25.14
  [e30172f5] Documenter v0.27.10
  [442a2c76] FastGaussQuadrature v0.4.7
  [5789e2e9] FileIO v1.11.2
  [e4b2fa32] GaussianRandomFields v2.1.4
  [2e9cd046] Gurobi v0.10.0
  [f67ccb44] HDF5 v0.15.7
  [7073ff75] IJulia v1.23.2
  [02fcfd73] ImageTransformations v0.8.13
  [916415d5] Images v0.24.1
  [20393b10] InfiniteOpt v0.5.1 `C:\Users\pulsipher\.julia\dev\InfiniteOpt`...
  [a98d9a8b] Interpolations v0.13.4
  [b6b21f68] Ipopt v0.8.0
  [682c06a0] JSON v0.21.2
  [4076af6c] JuMP v0.22.0
  [b964fa9f] LaTeXStrings v1.3.0
  [1dd02ad] LeftChildRightSiblingTrees v0.1.2
  [98b081ad] Literate v2.9.4
  [b8f27783] MathOptInterface v0.10.5
  [d8a4904e] MutableArithmetics v0.3.1
  [14b8a8f1] PkgTemplates v0.7.25
  [91a5bcd] Plots v1.23.6
  [92933f4c] ProgressMeter v1.7.1
  [438e738f] PyCall v1.92.5
  [d330b81b] PyPlot v2.10.0
  [189a3867] Reexport v1.2.2
  [295af30f] Revise v3.1.20
  [276daf66] SpecialFunctions v1.8.1
  [fd094767] Suppressor v0.2.0
  [0c5d862f] Symbolics v4.2.1

(@v1.6) pkg>
```

Plotting

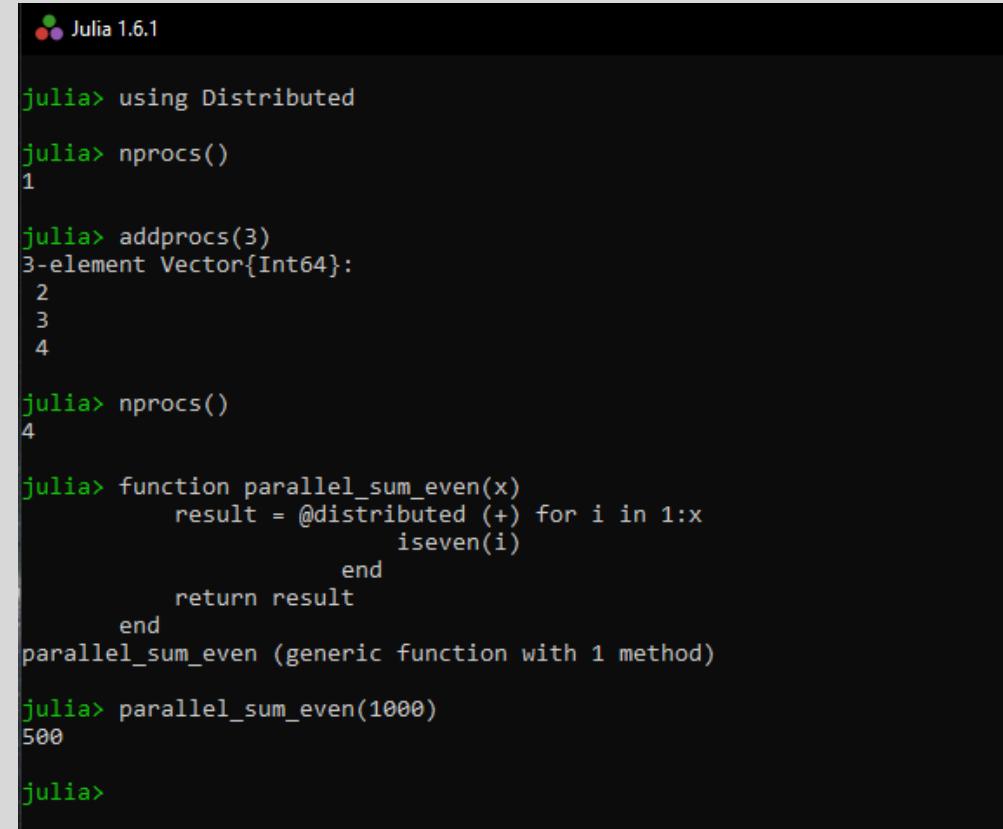
- Plots.jl
 - Efficient go to
 - The documentation is terrible 😞
- PyPlot.jl
 - Simply maps to matplotlib.pyplot
 - Imports all the all the pyplot functions
- Makie.jl
 - Native to Julia
 - Good documentation
 - Can be slow sometimes
- Others
 - GadFly.jl (pretty good 2D library)
 - PGFPlots.jl (plots for LaTeX)
 - UnicodePlots.jl (plot directly in the REPL)

```
Julia 1.6.1
julia> using Plots
julia> y(t) = sin(t) * cos(t)
y (generic function with 1 method)
julia> ts = 0:0.01:5
0.0:0.01:5.0
julia> plot(ts, y.(ts), label = "y", color = "red")
julia> xlabel!("t")
julia> ylabel!("y(t)")
julia>
```



Basic Parallel Computing

- [Distributed.jl](#)
 - Provides the basic CPU parallel computing methods
 - Add parallel resources via `addprocs`
 - Parallel for loops using `@distributed`
 - Other packages like MPI.jl take this further
 - To use arrays look at DistributedArrays.jl
 - Good for “larger jobs” (increased memory)
- [Multi-threading via Threads](#)
 - Operate on for loops with `Threads.@threads`
 - Must start Julia with the # of threads `$ Julia -threads 4`
 - Use cautiously (shared memory)
- [GPU computing](#)
 - Run Julia code on GPUs
 - See [juliagpu.org](#) for info



The screenshot shows a terminal window for Julia 1.6.1. The user has run several commands related to parallel computing:

```
Julia 1.6.1
julia> using Distributed
julia> nprocs()
1

julia> addprocs(3)
3-element Vector{Int64}:
 2
 3
 4

julia> nprocs()
4

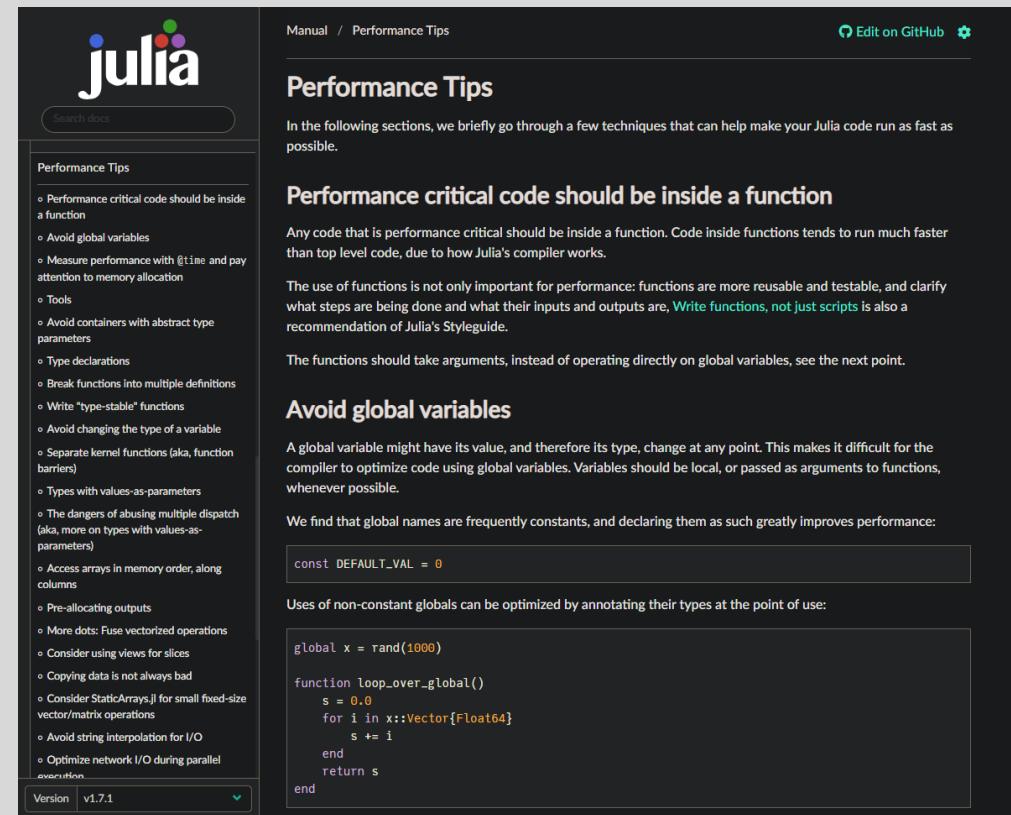
julia> function parallel_sum_even(x)
           result = @distributed (+) for i in 1:x
                   iseven(i)
               end
           return result
       end
parallel_sum_even (generic function with 1 method)

julia> parallel_sum_even(1000)
500

julia>
```

Performance

- The performance tips page
 - <https://docs.julialang.org/en/v1/manual/performance-tips/>
- Julia code can be C-fast
- Can be slow if we have poorly structured code
- Common practices
 - Preallocate instead of appending to arrays
 - Access arrays in memory order
 - Avoid ambiguous types
 - Use function overloading



The screenshot shows a dark-themed web page from the Julia documentation. At the top, there's a navigation bar with links for "Manual" and "Performance Tips". On the right, there are "Edit on GitHub" and settings icons. The main content area has a title "Performance Tips" and a sub-section title "Performance critical code should be inside a function". It discusses why functions run faster than top-level code and provides examples. Below that is a section titled "Avoid global variables" with a note about how global variables can affect optimization. A code block shows a global variable being initialized and used in a loop. The bottom of the page includes a "Version v1.7.1" dropdown.

Manual / Performance Tips Edit on GitHub

Performance Tips

In the following sections, we briefly go through a few techniques that can help make your Julia code run as fast as possible.

Performance critical code should be inside a function

Any code that is performance critical should be inside a function. Code inside functions tends to run much faster than top level code, due to how Julia's compiler works.

The use of functions is not only important for performance: functions are more reusable and testable, and clarify what steps are being done and what their inputs and outputs are. [Write functions, not just scripts](#) is also a recommendation of Julia's Styleguide.

The functions should take arguments, instead of operating directly on global variables, see the next point.

Avoid global variables

A global variable might have its value, and therefore its type, change at any point. This makes it difficult for the compiler to optimize code using global variables. Variables should be local, or passed as arguments to functions, whenever possible.

We find that global names are frequently constants, and declaring them as such greatly improves performance:

```
const DEFAULT_VAL = 0
```

Uses of non-constant globals can be optimized by annotating their types at the point of use:

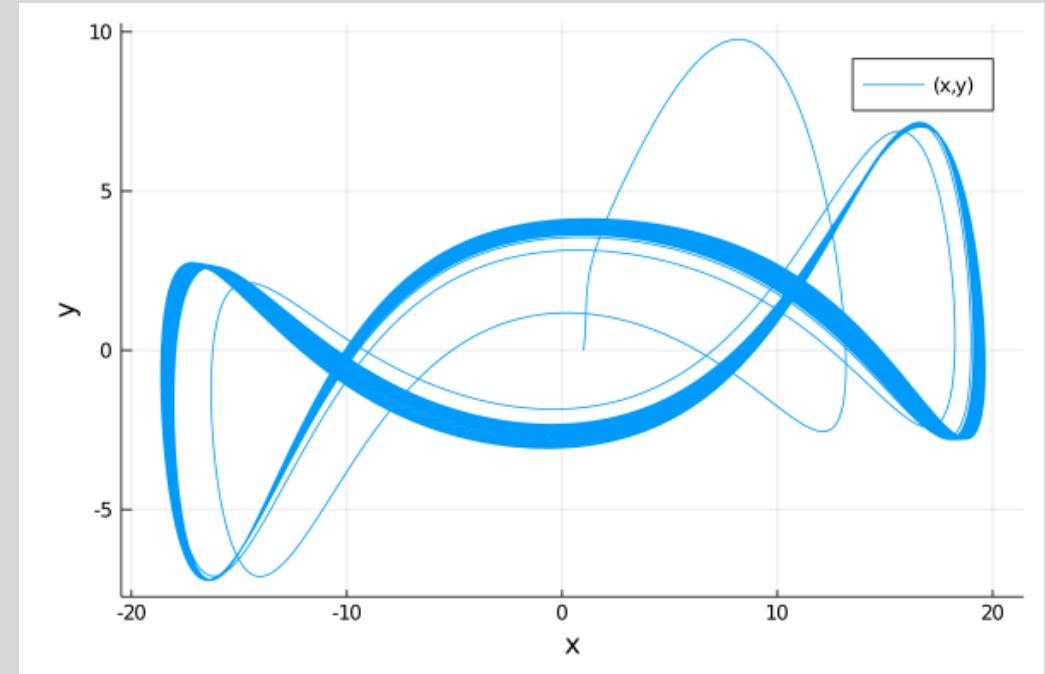
```
global x = rand(1000)

function loop_over_global()
    s = 0.0
    for i in x::Vector{Float64}
        s += i
    end
    return s
end
```

Version v1.7.1

Other Useful Things

- File reading and writing
 - Delimited files via `DelimitedFiles` (built-in)
 - Text files and basic CSVs
 - CSVs via `CSV.jl`
 - Store/read dictionaries via `JSON.jl`
 - Tables via `DataFrames.jl`
- Math stuff
 - Symbolic math with `Symbolics.jl` and `ModelingToolkit.jl`
 - Differential equations with `DifferentialEquations.jl`
 - Statistics with `Statistics.jl` and `Distributions.jl`
- Machine learning
 - `Flux.jl`
 - `Tensorflow.jl` and `ScikitLearn.jl`
 - `Knet.jl`



Resources

- Learn Julia in Y Minutes
 - <https://learnxinyminutes.com/docs/julia/>
- Julia's documentation
 - <https://docs.julialang.org/en/v1/>
- Julia's Discourse forum
 - <https://discourse.julialang.org/>
- Julia's YouTube channel
 - <https://www.youtube.com/user/julialanguage>
 - Check out the tutorials and JuliaCon talks
- Google it!

The screenshot shows the homepage of the Julia Discourse forum. At the top, there is a navigation bar with links for "all categories", "all tags", "Categories" (which is highlighted in orange), "Latest", "New (68)", "Unread (14)", "Top", and a button for "+ New Topic". Below the navigation bar, there is a banner asking "Do you want live notifications when people reply to your posts? Enable Notifications" with a "x" button.

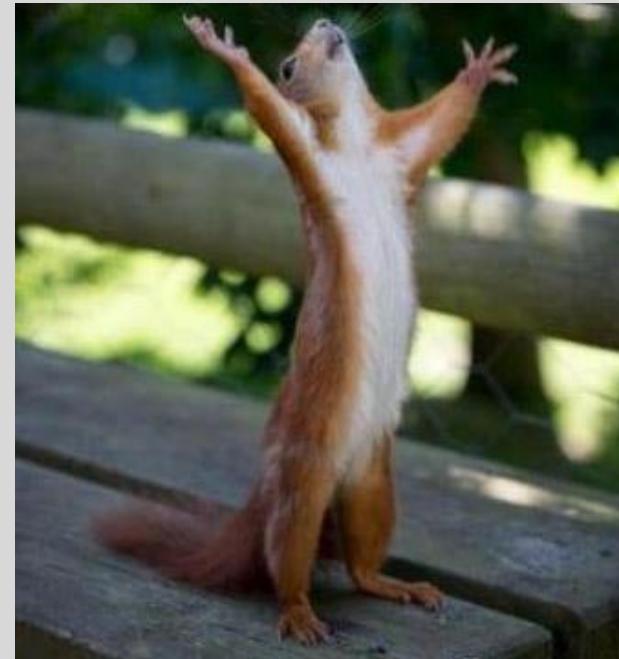
The main content area is divided into several sections:

- Announcements**: 146 topics. Description: Low traffic category for important announcements, mostly Julia releases and JuliaCon.
- New to Julia**: 46 topics. Description: Everybody is new to Julia at some point. Whether you are struggling with the installation or you just can't figure out why you are not seeing the performance everybody else seems to be talking about, this is the place to start!
- General Usage**: 72 topics per week. Description: Questions and discussions about using Julia and its packages. The **Performance** subcategory can be used for dedicated discussions about maximizing speed.
- Specific Domains**: 52 topics per week. Description: Discussion of Julia in various specialized subject domains that have a dedicated community. Don't see your domain? Fear not, post your topic in **general usage**. Subcategories include: Statistics, Numerics, GPU, Biology, Health, and Medicine, Data, Visualization, Optimization (Mathematical), Machine Learning, Modelling & Simulations, Signal Processing, Web Stack, AstroSpace, Finance and Economics, Julia at Scale, Probabilistic programming, Maker, Quantum, Chemistry, and High Energy Physics.
- Tooling**: 15 topics per week. Description: Tools around Julia.

On the right side, there is a sidebar titled "Latest" showing recent posts:

- Please read: make it easier to help you (4 replies, Jul 2021)
- Welcome to Discourse (1 reply, Oct 2016)
- Anyone crazy enough to develop a pure Julia GUI toolbox? (119 replies, 30m ago)
- Seeking Julia mentors and projects for GSoc 2022 (1 reply, 31m ago)
- "UndefVarError: V not defined" using simple loop (3 replies, 32m ago)
- Add a prefix to every `println()` output (6 replies, 1h ago)
- Performance issues when working with dict (2 replies, 1h ago)
- Julia vs Zig surprise (9 replies, 1h ago)

10 Minute Break Time





A Brief Introduction

Modeling in JuMP.jl

$$\begin{aligned} \max_{f,s} \quad & 12f + 9s \\ \text{s.t.} \quad & 4f + 2s \leq 4800 \\ & f + s \leq 1750 \\ & 0 \leq f \leq 1000 \\ & 0 \leq s \leq 1500 \end{aligned}$$

- Initialize **model**

```
1 using JuMP, Gurobi
2 model = Model(Gurobi.Optimizer)
```

- Define **variables**

```
3 @variable(model, f)
4 @variable(model, s)
```

- Define **objective**

```
5 @objective(model, Max, 12f + 9s)
```

- Define **constraints**

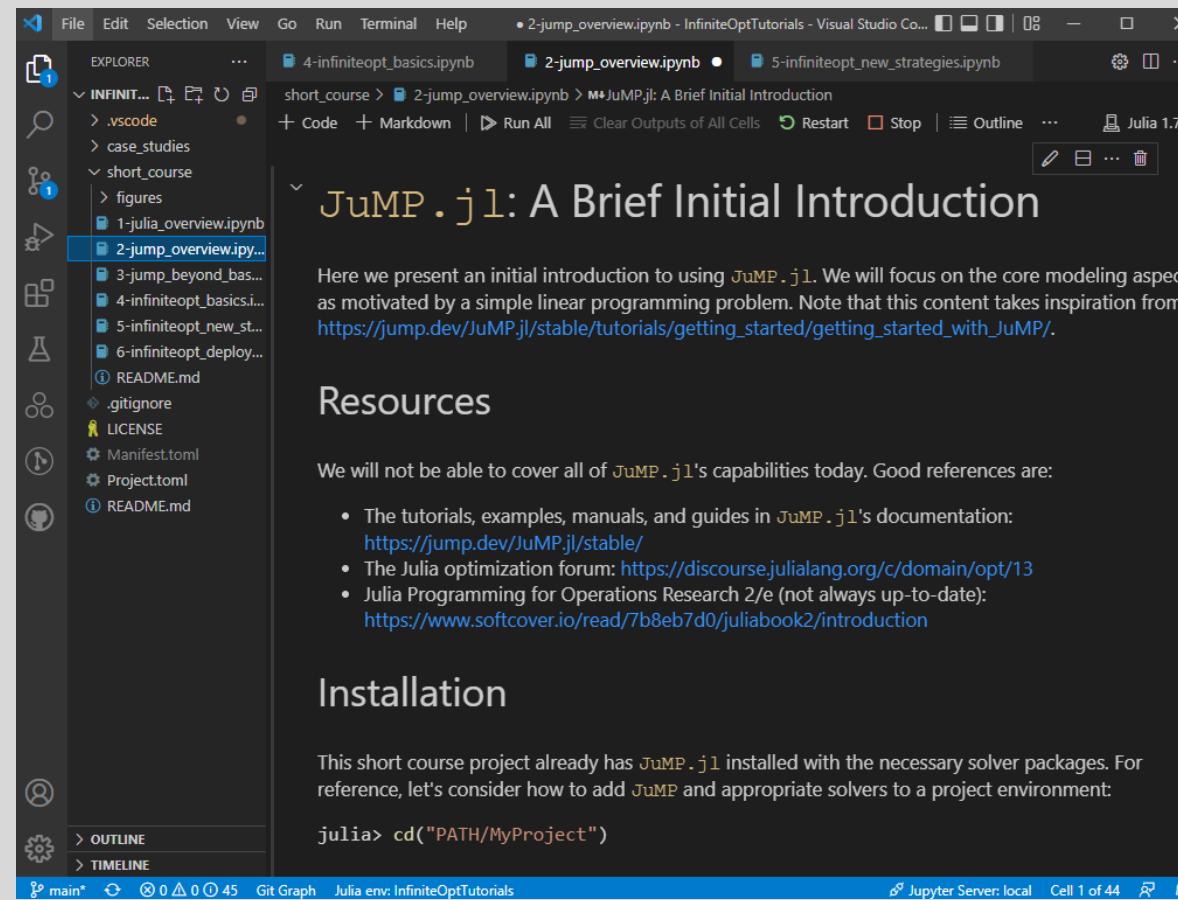
```
6 @constraint(model, 4f + 2s <= 4800)
7 @constraint(model, f + s <= 1750)
8 @constraint(model, 0 <= f <= 1000)
9 @constraint(model, 0 <= s <= 1500)
```

- **Solve**

```
10 optimize!(model)
11 profit = objective_value(model)
12 f_best = value(f)
13 s_best = value(s)
```

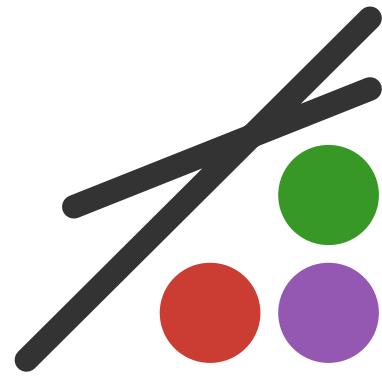
To the Tutorial!

- Open the “2-jump_overview.ipynb” jupyter notebook



1 Hour Lunch Break



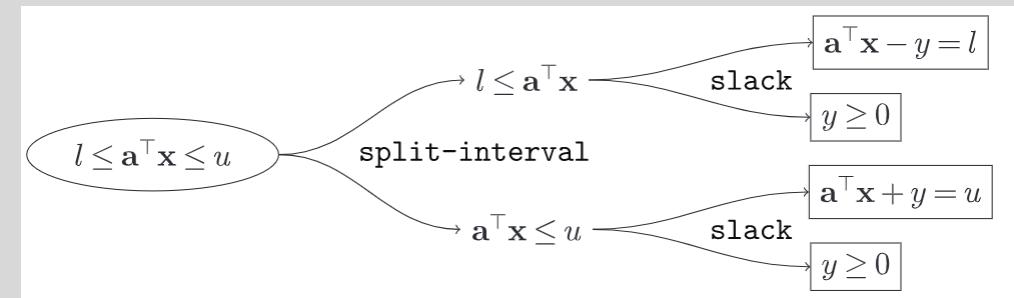
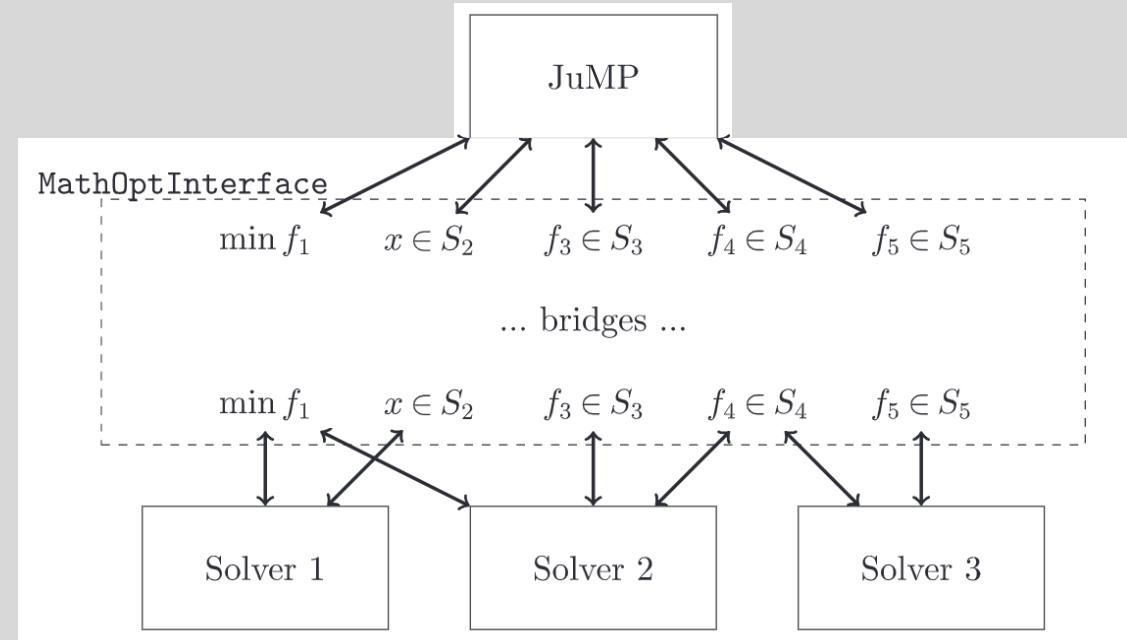


JUMP

Beyond the basics

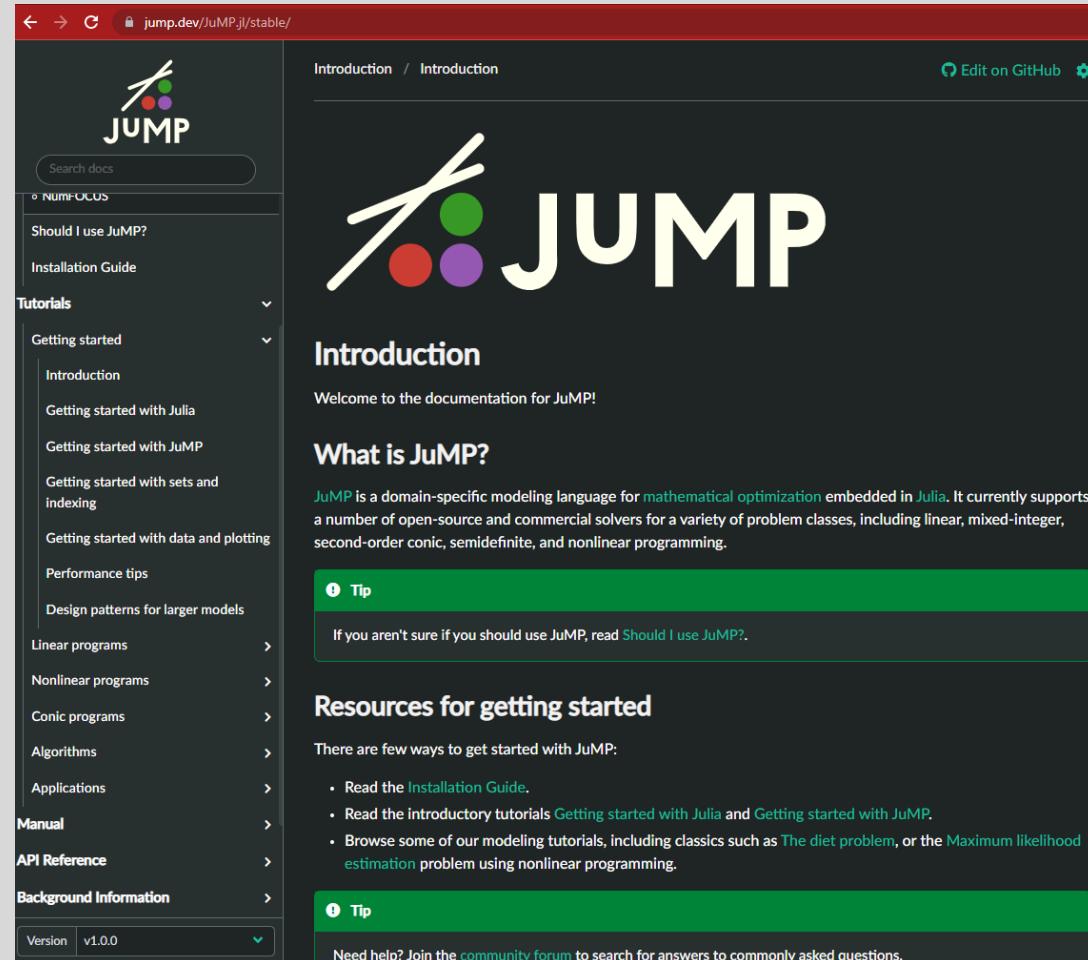
JuMP.jl Architecture

- JuMP.jl provides the high-level interface (the “macro sugar”)
- Model is stored in MathOptInterface.jl model
 - Constraints are of the form *function* in set
- Bridges convert constraints into form that solvers support
 - Solvers don’t explicitly support every *function-set* combination
- MathOptInterface.jl preserves the constraint mappings
- Solvers interface with MathOptInterface.jl



Learning JuMP.jl

- The documentation is full of very useful tutorials



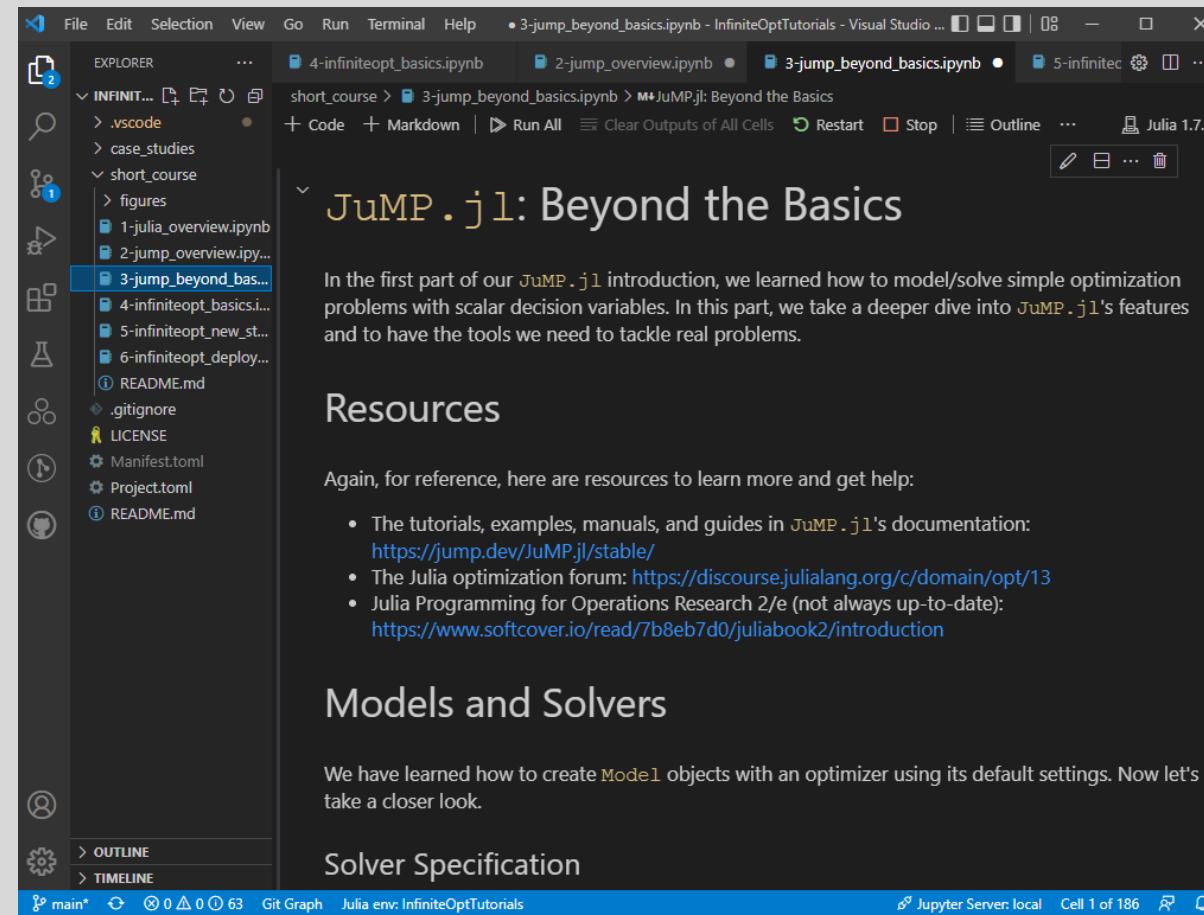
The screenshot shows the JuMP documentation website at jump.dev/JuMP.jl/stable/. The page has a dark theme with a large JuMP logo at the top. The left sidebar contains a navigation menu with sections like NumFOCUS, Should I use JuMP?, Installation Guide, Tutorials (with sub-sections for Getting started, Linear programs, Nonlinear programs, Conic programs, Algorithms, Applications), Manual, API Reference, and Background Information. A "Version v1.0.0" link is at the bottom of the sidebar. The main content area features the JuMP logo again, followed by the title "Introduction". Below it is a "Tip" box containing the text: "If you aren't sure if you should use JuMP, read [Should I use JuMP?](#).". The next section is titled "What is JuMP?" with a brief description: "JuMP is a domain-specific modeling language for [mathematical optimization](#) embedded in [Julia](#). It currently supports a number of open-source and commercial solvers for a variety of problem classes, including linear, mixed-integer, second-order conic, semidefinite, and nonlinear programming." Another "Tip" box follows, stating: "There are few ways to get started with JuMP:

- Read the [Installation Guide](#).
- Read the introductory tutorials [Getting started with Julia](#) and [Getting started with JuMP](#).
- Browse some of our modeling tutorials, including classics such as [The diet problem](#), or the [Maximum likelihood estimation](#) problem using nonlinear programming.

" A final "Tip" box at the bottom encourages users to "Need help? Join the [community forum](#) to search for answers to commonly asked questions."

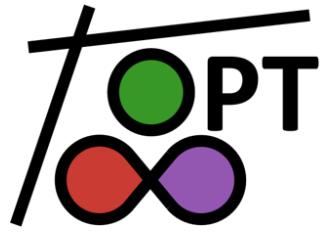
To the Tutorial!

- Open the “3-jump_beyond_basics.ipynb” jupyter notebook



10 Minute Break Time





InfiniteOpt

The Basics

Modeling in InfiniteOpt.jl

- Initialize the **model**

```
using InfiniteOpt, Distributions, Ipopt
m = InfiniteModel(Ipopt.Optimizer)
```

- Add the **infinite parameters**

$$t \in [t_0, t_f] \quad \xi \sim \mathcal{N}(\mu, \Sigma)$$

```
@infinite_parameter(m, t ∈ [t₀, tₙ], num_supports = 10)
@infinite_parameter(m, ξ[1:10] ~ MvNormal(μ, Σ))
```

- Add **variables** and their domain constraints

$$y_a(t) \in \mathbb{R}_+ \quad y_b(t, \xi) \in \mathbb{R}_+ \quad y_c(\xi) \in \{0, 1\} \quad z \in \mathbb{Z}^2$$

```
@variable(m, ya ≥ 0, Infinite(t))
@variable(m, yb ≥ 0, Infinite(t, ξ))
@variable(m, yc, Infinite(ξ), Bin)
@variable(m, z[1:2], Int)
```

- Define the **objective**

$$\min_{y_a(t), y_b(t, \xi), y_c(\xi), z} \int_{t \in \mathcal{D}_t} y_a(t)^2 + 2\mathbb{E}_\xi[y_b(t, \xi)]dt$$

```
@objective(m, Min, ∫(ya^2 + 2 * E(yb, ξ), t))
```

- Add the **constraints**

$$\frac{\partial y_b(t, \xi)}{\partial t} = 2y_b(t, \xi)^2 + y_a(t) - z_1, \quad \forall t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi$$

$$\mathbb{P}(y_b(t, \xi) \leq 0) \geq \alpha, \quad \forall t \in \mathcal{D}_t$$

$$y_a(0) + z_2 = \beta$$

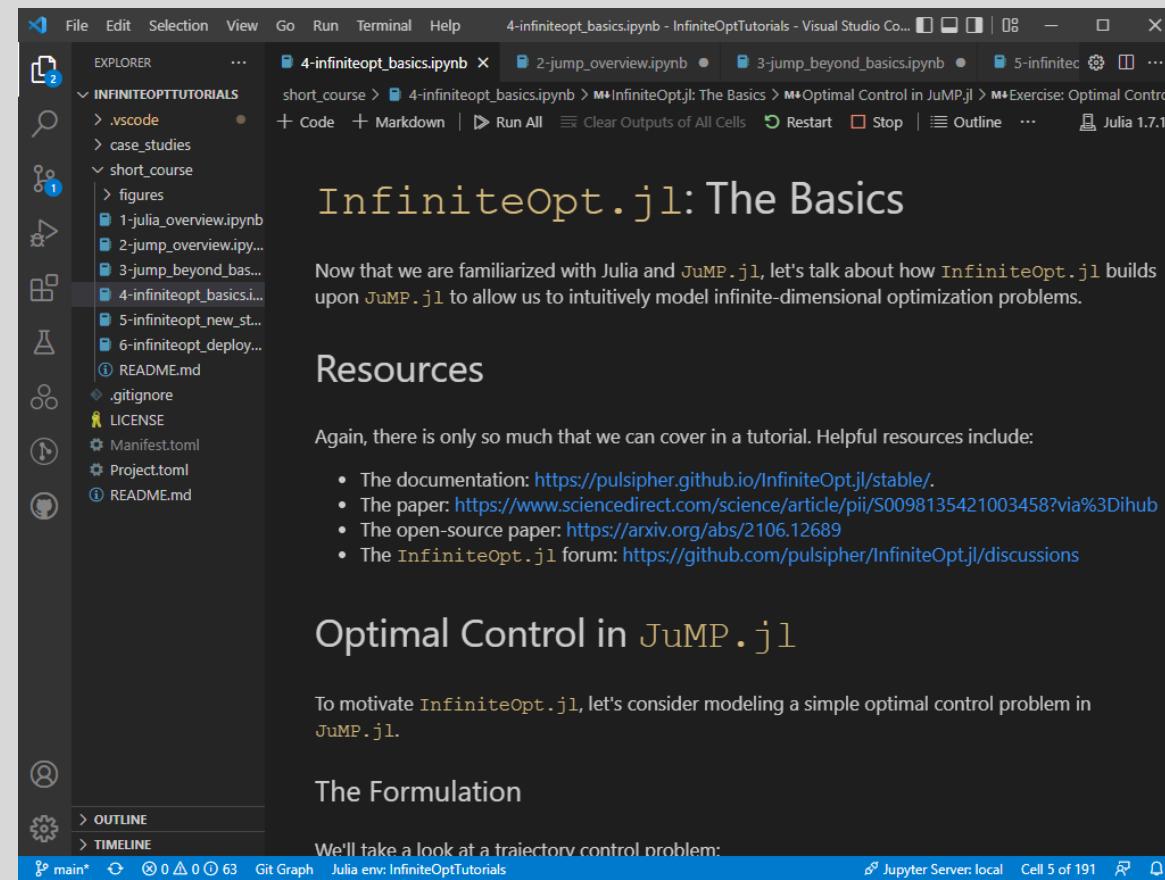
```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, yb ≤ yc * u)
@constraint(m, E(yc, ξ) ≥ α)
@constraint(m, ya(0) + z[2] == β)
```

- **Solve**

```
optimize!(m)
opt_objective = objective_value(m)
```

To the Tutorial!

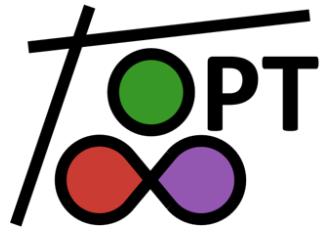
- Open the “4-infiniteopt_basics.ipynb” jupyter notebook



10 Minute Break Time

Why is
InfiniteOpt.jl
so awesome?



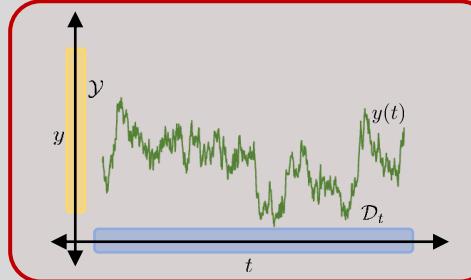


InfiniteOpt

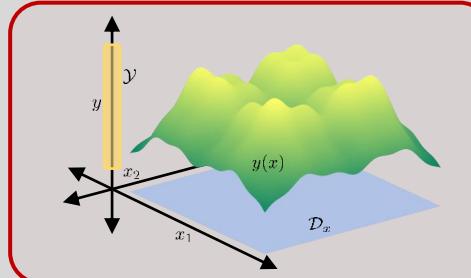
New Modeling Strategies

Innovating with InfiniteOpt.jl

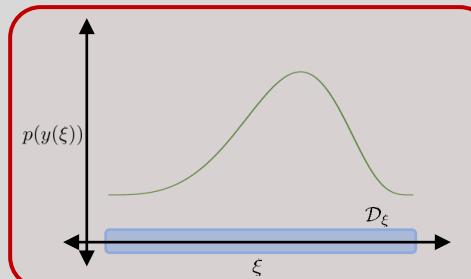
Traditional Formulations



Dynamic Optimization



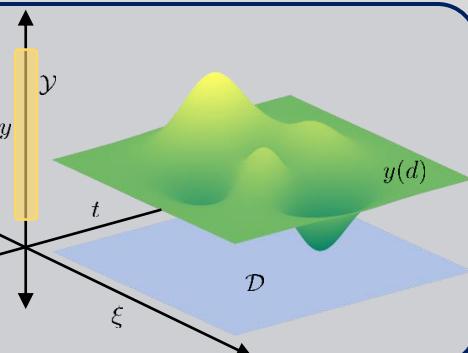
PDE-Constrained Optimization



Stochastic Optimization

InfiniteOpt

Unifying Abstraction

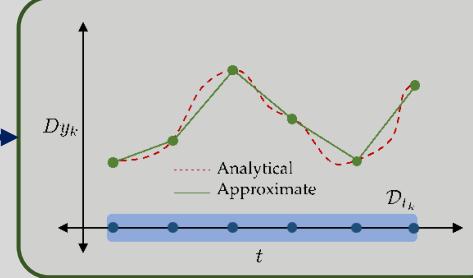


Infinite-Dimensional Optimization

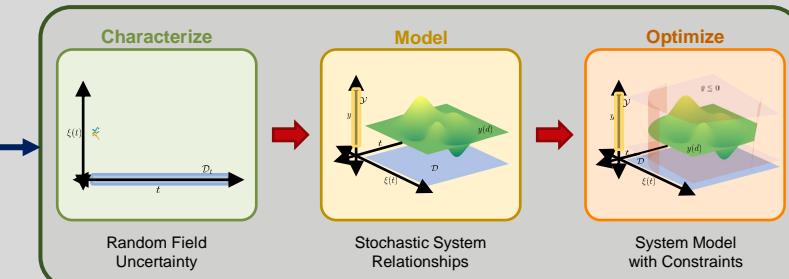
New Formulations



Generalized Shaping Measures



Continuous Time Estimation

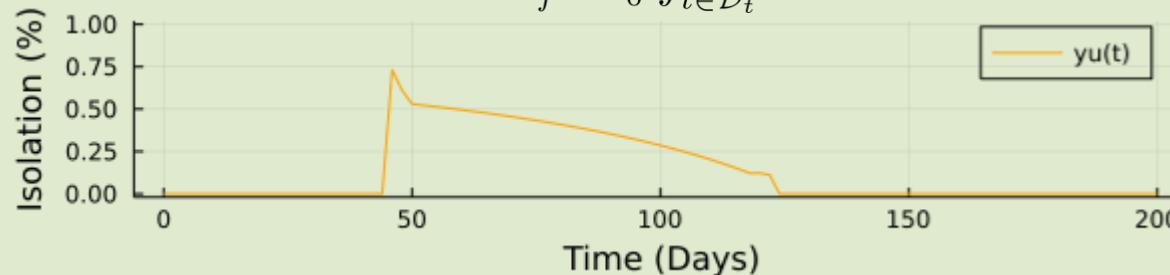


Random Field Optimization

Expectation (Uniform pdf)

- Weight the cost **uniformly**

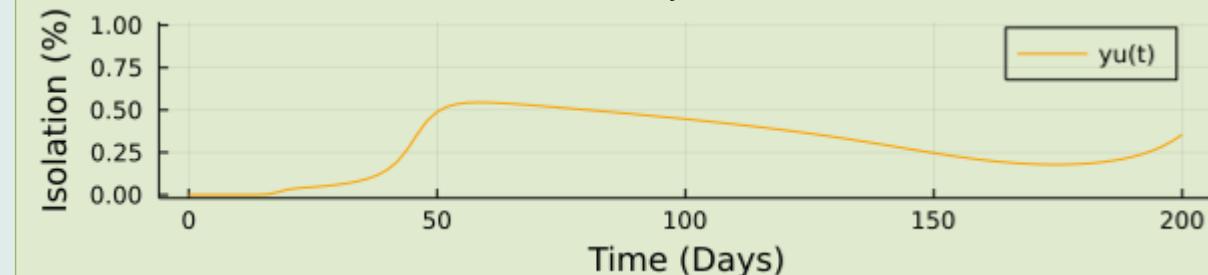
$$\mathbb{E}_t[y_u(t)] = \frac{1}{t_f - t_0} \int_{t \in \mathcal{D}_t} y_u(t) dt$$



Expectation (Exponential pdf)

- Place emphasis on the **initial times**

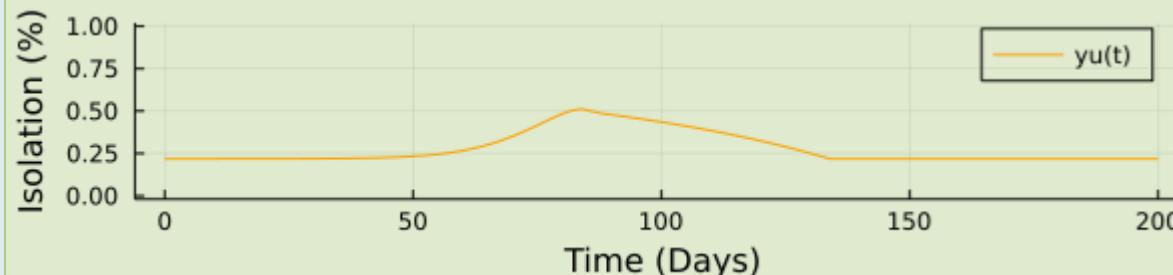
$$\mathbb{E}_t[y_u(t)] = \int_{t \in \mathcal{D}_t} y_u(t) e^{-t} dt$$



Mean-Variance

- Penalize **fluctuations** in the trajectory

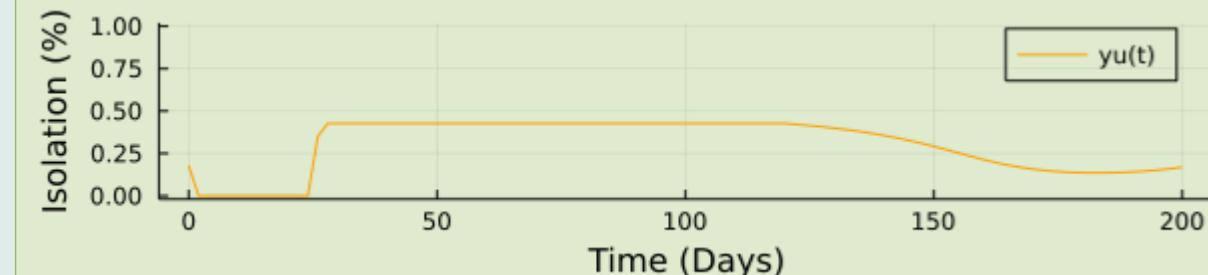
$$\text{EV}_t = \mathbb{E}_t[y_u(t)] + \lambda \mathbb{V}_t[y_u(t)]$$



CVaR

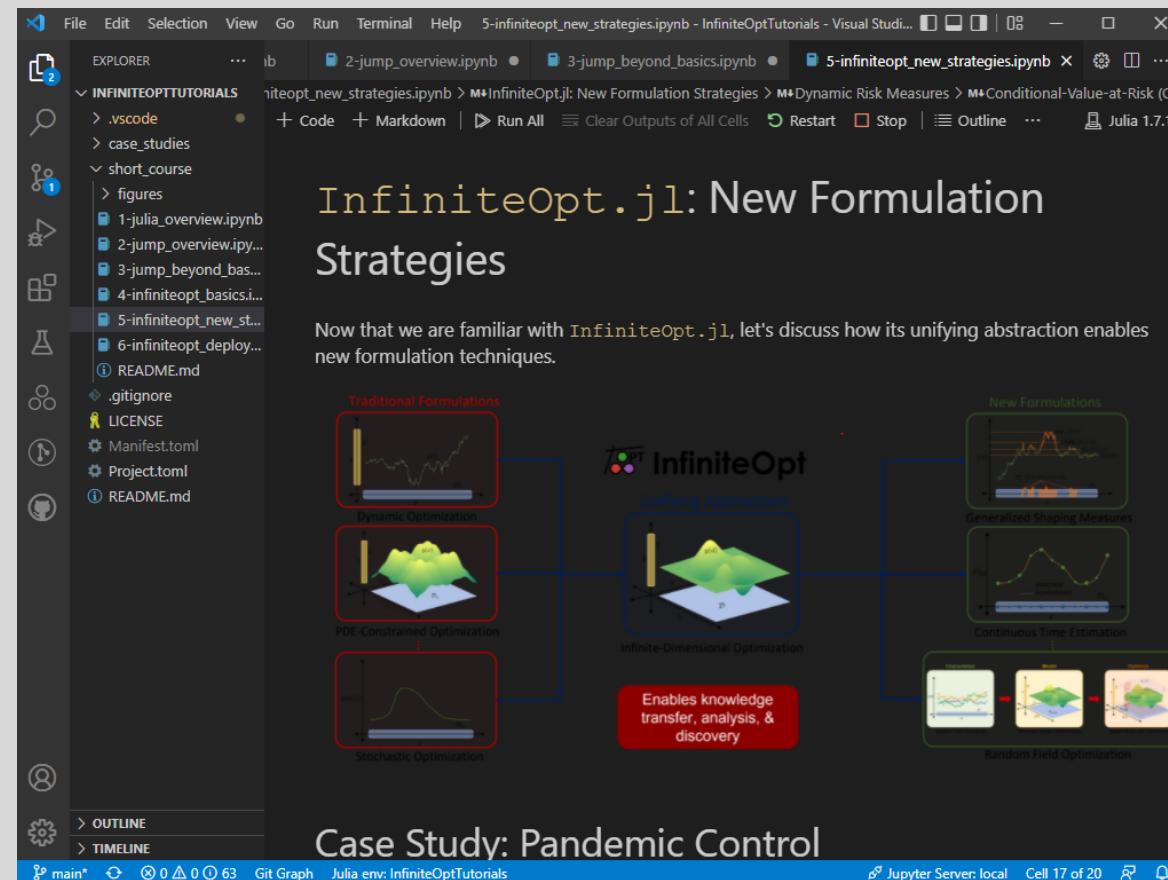
- Only penalize the **high cost peaks**

$$\text{CVaR}_t(y_u(t); \alpha)$$



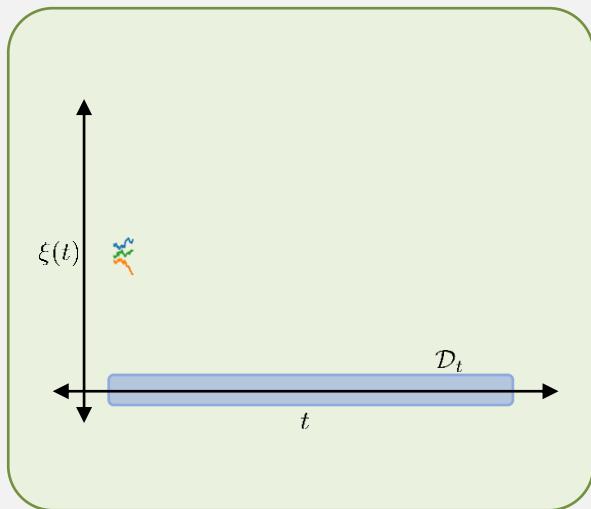
To the Tutorial!

- Open the “5-infiniteopt_new_strategies.ipynb” jupyter notebook



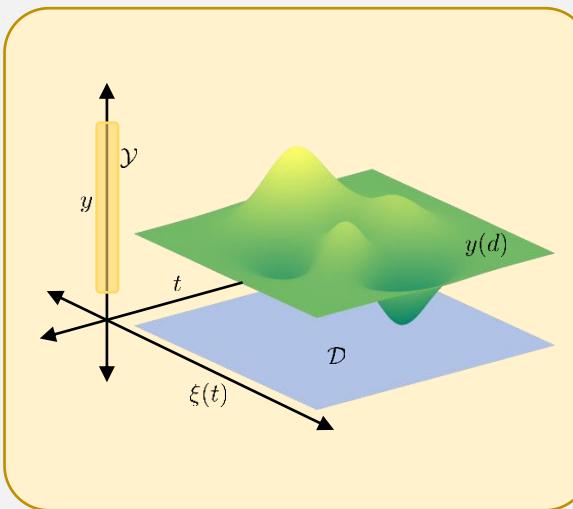
Random Field Optimization

Characterize



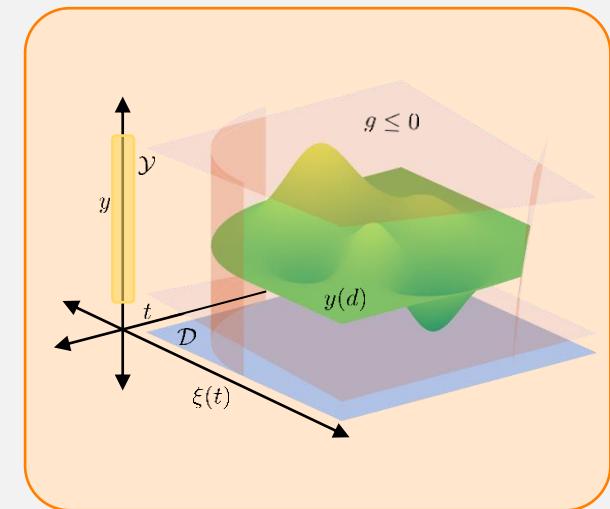
Random Field
Uncertainty

Model



Stochastic System
Relationships

Optimize



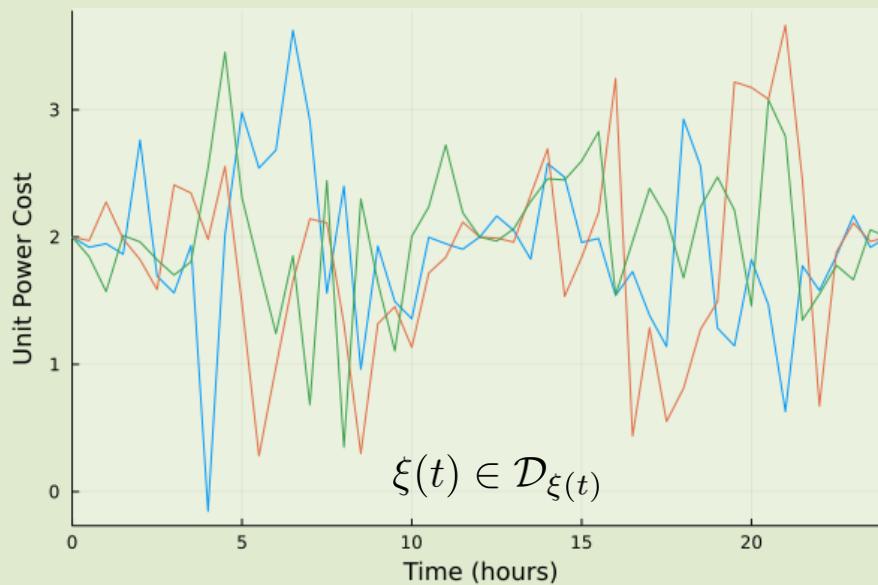
System Model
with Constraints



Idea: Motivate framework with decoupled examples in **time** and **space**

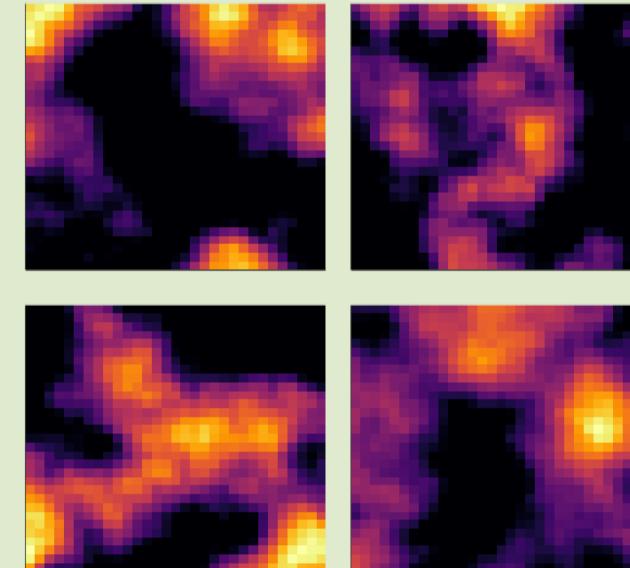
Dynamic Electricity Market

- Random field for **pricing forecast**



Spatial Thermal Diffusivity

- Random field for **uncertain diffusivity**

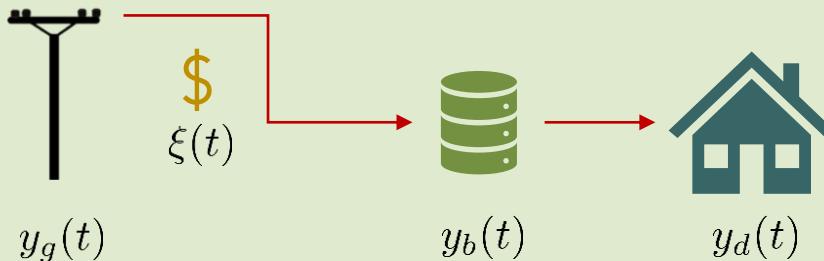


$$\xi(x) \in \mathcal{D}_{\xi(x)}$$

Simplifying Assumption: Fields are continuous and differentiable → avoid stochastic calculus

Battery-Aided Electricity System

- **Store power** via battery to avoid high costs



- **Simulate** with predetermined purchasing

$$\frac{dy_b(t)}{dt} = y_g(t) - y_d(t)$$

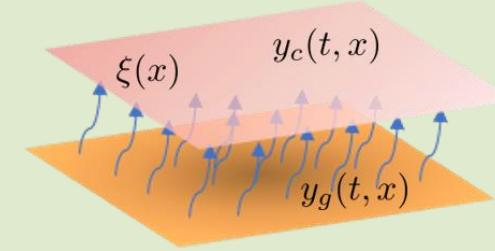
$$f(t) = \xi(t)y_g(t)$$

$$y_b(0) = y_{b0}$$



Transient Thermal Diffusion

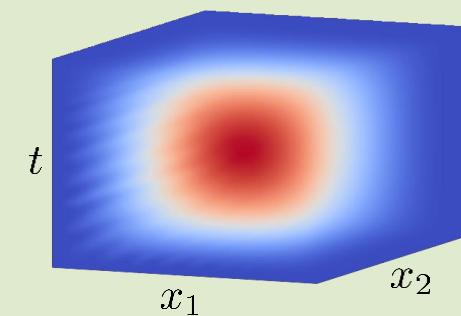
- Control a plate's **temperature distribution**



- **Simulate** with uniform heating

$$\frac{\partial y_c(t, x)}{\partial t} = \xi(x) \left(\frac{\partial^2 y_c(t, x)}{\partial x_1^2} + \frac{\partial^2 y_c(t, x)}{\partial x_2^2} \right) + y_g(t, x)$$

$$y_c(0, x), y_c(t, \text{boundary}) = 0$$





Illustrative Examples

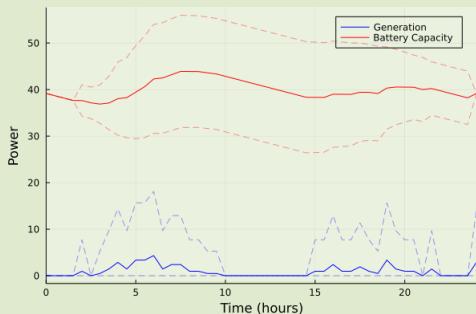
Optimal Battery Design

- Optimally choose the **battery size**

$$\begin{aligned} \min \quad & \mathbb{E}_{\xi(t)} \left[\int_{t \in \mathcal{D}_t} \xi(t) y_g(t, \xi(t)) dt \right] + 0.1z \\ \text{s.t.} \quad & \frac{dy_b(t, \xi(t))}{dt} = y_g(t, \xi(t)) - 1, \quad t \in \mathcal{D}_t, \quad \xi(t) \in \mathcal{D}_{\xi}(t) \\ & 0.2z \leq y_b(t, \xi(t)) \leq z, \quad t \in \mathcal{D}_t, \quad \xi(t) \in \mathcal{D}_{\xi}(t) \\ & y_b(0, \xi(0)) = 0.5z, \quad \xi(0) \in \mathcal{D}_{\xi}(0) \\ & y_b(24, \xi(24)) = 0.5z, \quad \xi(24) \in \mathcal{D}_{\xi}(24) \end{aligned}$$

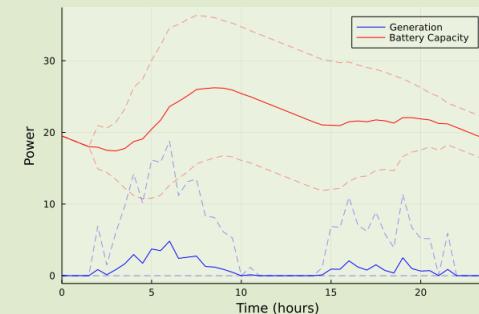
- **Results**

Deterministic



$z = 78.3$, cost = 14.7

Stochastic



$z = 39$, cost = 12

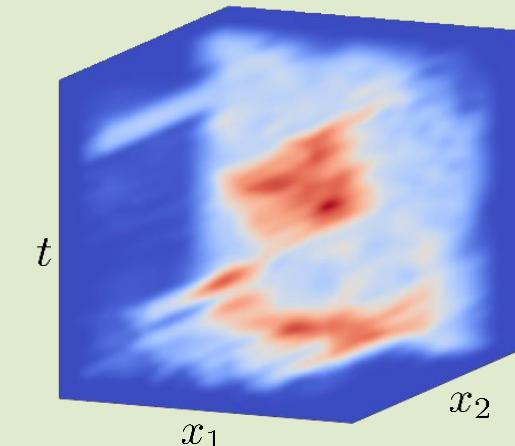
Optimal Temperature Distribution

- Optimally **control heating** to reach setpoint

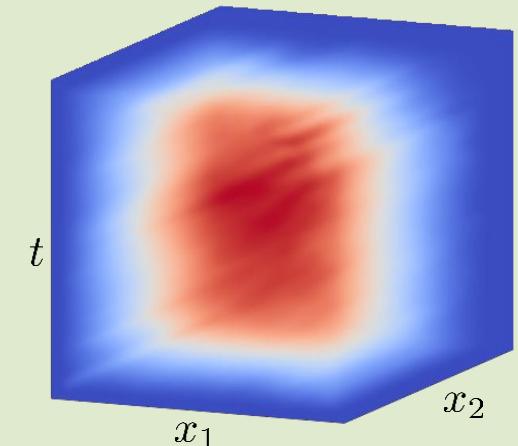
$$\begin{aligned} \min \quad & \mathbb{E}_{\xi(x)} \left[\int_{(t,x) \in \mathcal{D}_{t,x}} (y_c(t, x, \xi(x)) - \bar{y}_c(x))^2 dt dx \right] \\ \text{s.t.} \quad & \frac{\partial y_c(t, x, \xi(x))}{\partial t} = \xi(x) \nabla_x^2 y_c(t, x, \xi(x)) + y_g(t, x), \quad (t, x) \in \mathcal{D}_{t,x}, \quad \xi(x) \in \mathcal{D}_{\xi(x)} \\ & y_c(0, x, \xi(x)), y_c(t, \text{boundary}, \xi(\text{boundary})) = 0, \quad (t, x) \in \mathcal{D}_{t,x}, \quad \xi(x) \in \mathcal{D}_{\xi(x)} \end{aligned}$$

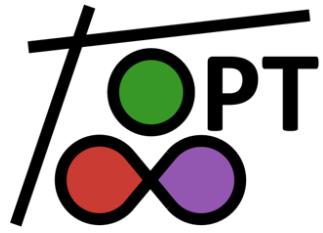
- **Results**

Heating



Average Temperature



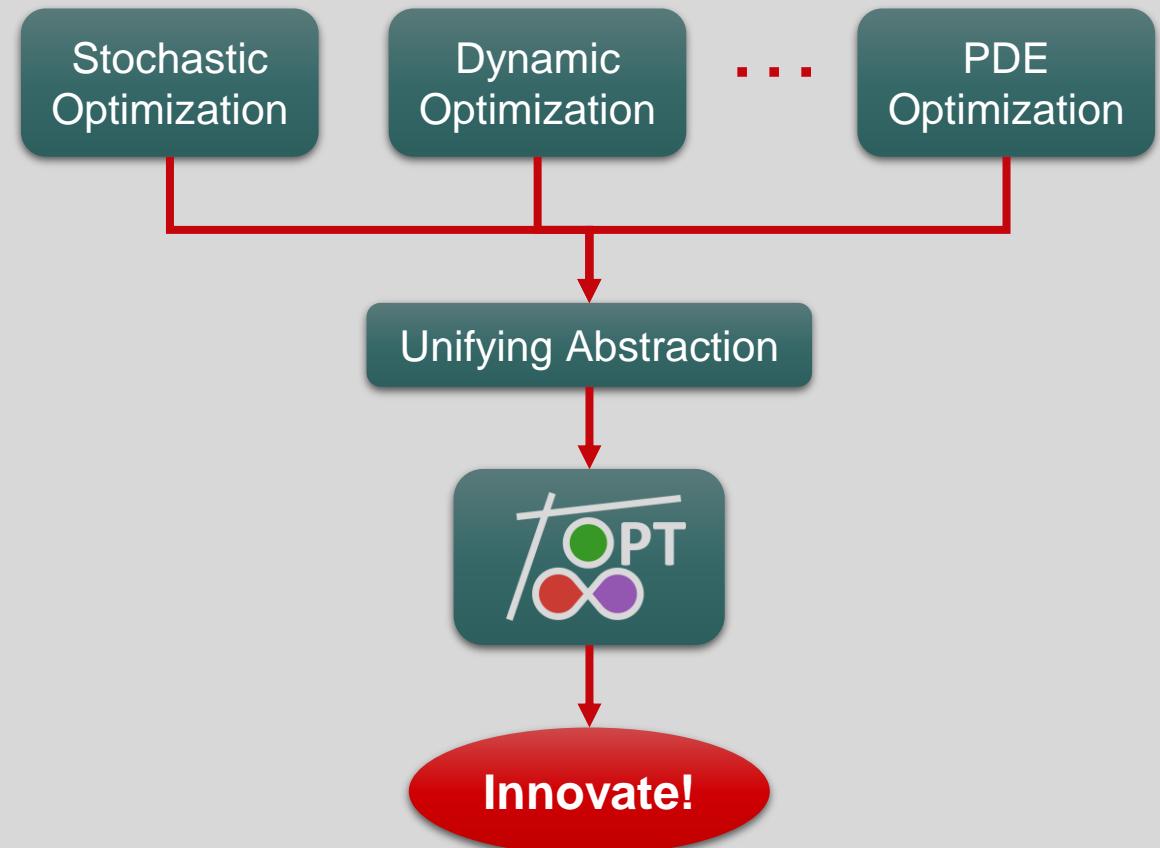


InfiniteOpt

Deployment Tools

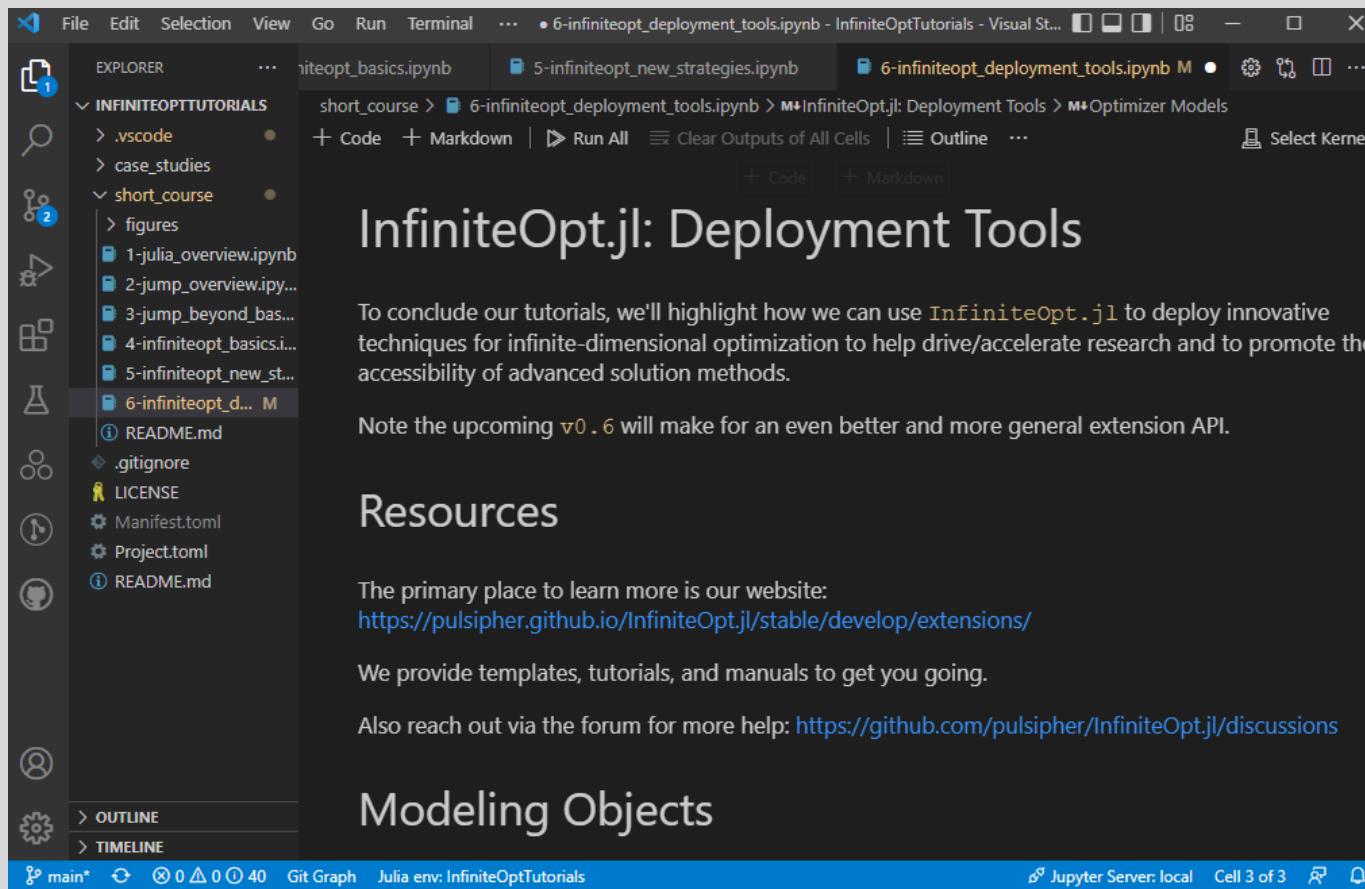
Want to deploy your cool techniques?

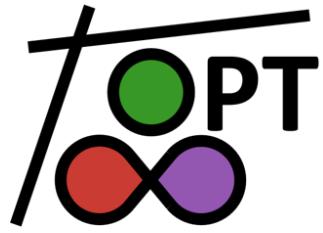
- Almost all modeling aspects can be extended
 - **Infinite domain**
 - **Support generation**
 - **Measure** representation/evaluation
 - **Derivative** evaluation methods
 - Model **transformation**
 - More!
- Extensive **documentation, tutorials, & templates**



To the Tutorial!

- Open the “6-infiniteopt_deployment_tools.ipynb” jupyter notebook



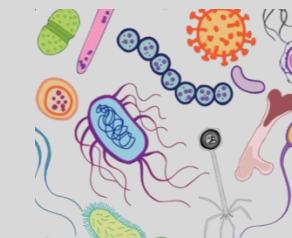
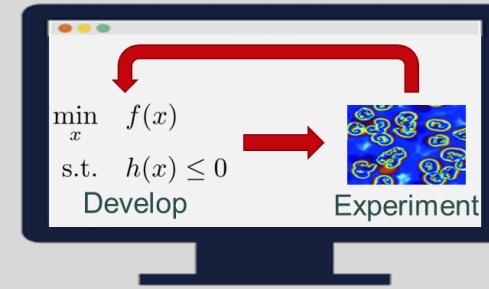


InfiniteOpt

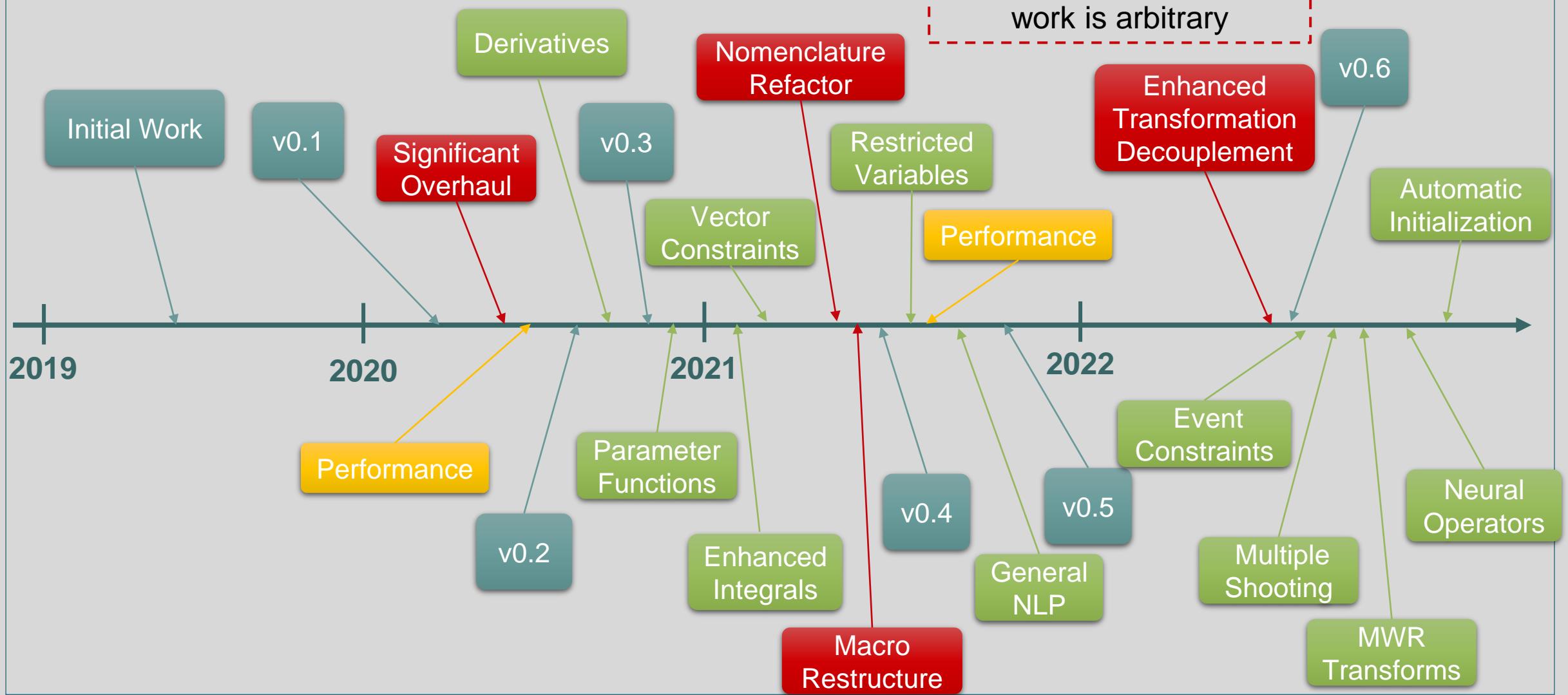
Final Thoughts

When should I use InfiniteOpt.jl?

- Research
 - Sandbox InfiniteOpt problem approaches
 - Deploy your innovations → accessibility
 - Rapid comparisons
- Learning
 - High-level interface for novices
 - Features to interrogate and learn
- Application
 - Offline decision making (e.g., optimal control)
 - Any InfiniteOpt problem class



Development Timeline

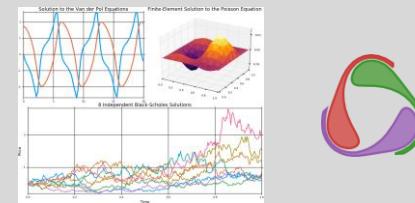


Future Development: Modeling Objects

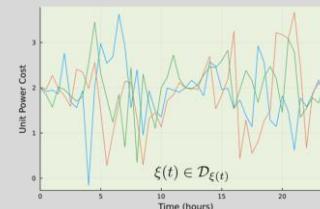
- Non-rectangular domains
 - Leverage packages like Gridap.jl



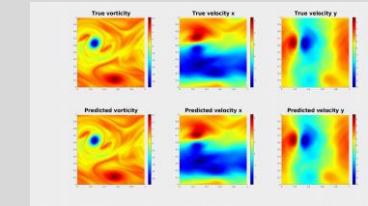
- Optimal Control Toolbox
 - Automatic initialization via DifferentialEquations.jl
 - Simulation



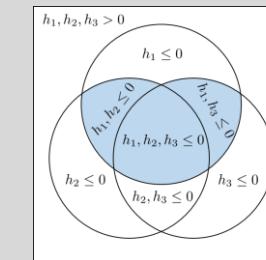
- Infinite Parameter Functions
 - Embed random fields uncertainty



- Infinite-Dimensional Surrogates
 - Embed neural operator models

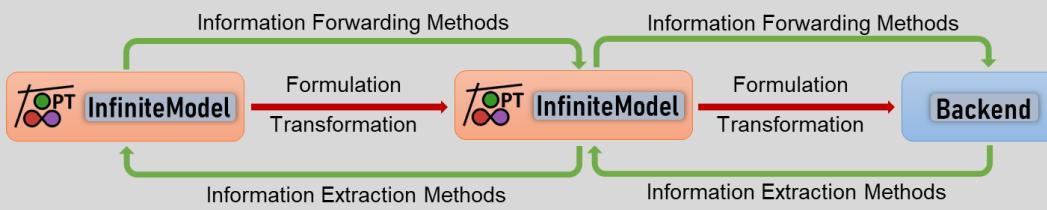


- Event Constraints
 - Generalization of chance constraints
 - Connection to generalized disjunctive programming

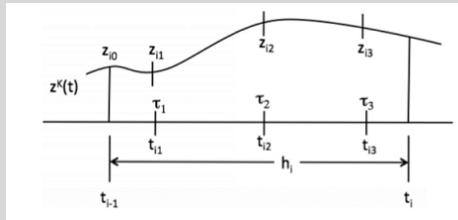


Future Development: Transformations

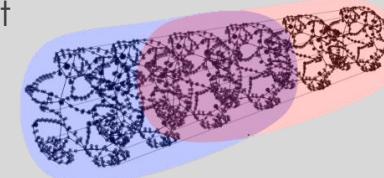
- Transformation Decoupling Rewrite
 - Fully decouple TranscriptionOpt
 - Allow arbitrary (non-JuMP) models



- Enhanced Transcription
 - Comprehensive collocation support
 - Efficient multivariate quadrature
 - Adaptive finite element intervals



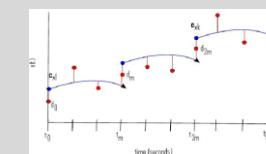
- Automatic Decomposition
 - Connection to Plasmo.jl/MadNLP.jl
 - Connection to Parapint



- Symbolic Simplification via ModelingToolkit.jl
 - Automatic order reduction



- DAE Solver Transformations
 - Multiple shooting
 - Adjoint methods via DifferentialEquations.jl



Future Development: Enhanced Nonlinear

- Leverage JuMP.jl's NLP Overhaul
 - Vector-valued NLP
 - Modular AD backends
 - Replace our `NLPExpr`s with JuMP.jl objects
 - Support black-box functions

NumFOCUS signs agreement with LANL to improve nonlinear support in JuMP

announcements · 21 Feb 2022

The [JuMP Steering Committee](#) is pleased to announce that we, through [NumFOCUS](#), have signed an agreement with [Los Alamos National Laboratory \(LANL\)](#) to improve nonlinear support in JuMP.

The agreement runs until September 2023.

At a high-level, the agreement seeks to improve nonlinear support in JuMP on two key fronts:

- improving the automatic differentiation performance in JuMP by adding the ability to use different automatic differentiation backends
- improving the modeling ergonomics of JuMP by adding the ability to create nonlinear expressions using the `@constraint` and `@objective` macros, and identifying ways to add vector-valued nonlinear functions.

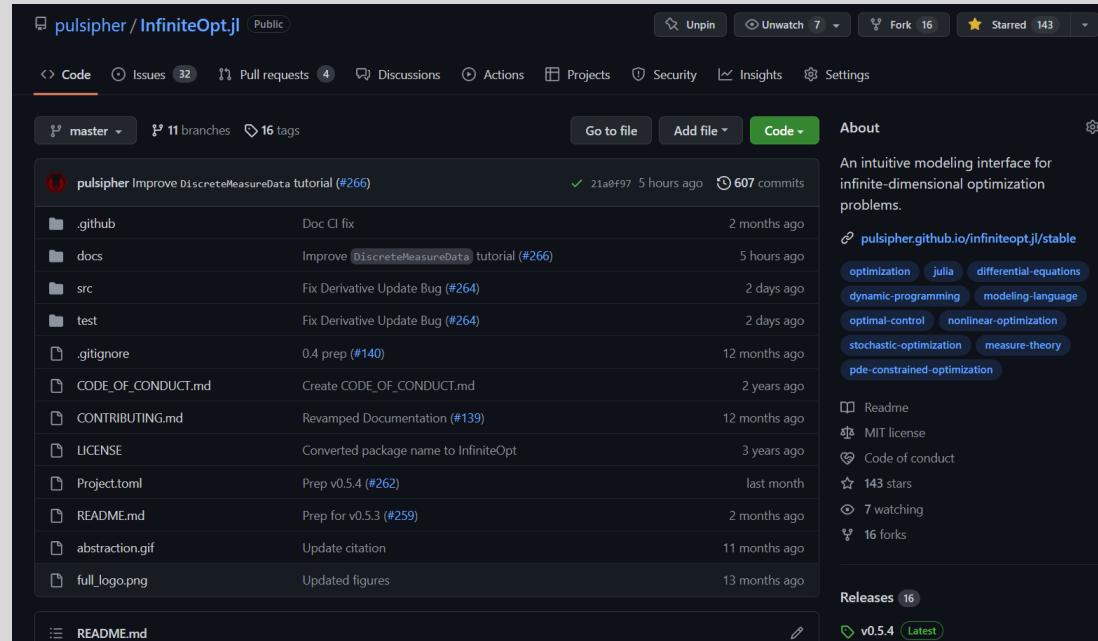
In addition, there is scope to work on and improve other aspects of nonlinear programming in JuMP.

To discuss our plans or to get involved by suggesting improvements:

- Read and comment on our work-in-progress Google doc "[Developing state-of-the-art nonlinear support in JuMP](#)"
- Join the [monthly developer calls](#)
- Join the [developer chatroom](#) for updates and an invitation to semi-regular nonlinear developer calls.

How can I help?

- Get started with our tutorials: <https://pulsipher.github.io/InfiniteOpt.jl/stable/>
- Check out the contribution guide: https://pulsipher.github.io/InfiniteOpt.jl/stable/develop/start_guide/
- Reach out on the forum: <https://github.com/pulsipher/InfiniteOpt.jl/discussions>
- Give InfiniteOpt.jl a star!



What were those resources, again?

- Julia
 - Julia learning: <https://julialang.org/learning/>
 - Documentation: <https://jump.dev/JuMP.jl/stable/>
 - Forum: <https://discourse.julialang.org/>
 - YouTube: <https://www.youtube.com/c/TheJuliaLanguage>
- JuMP.jl
 - Documentation: <https://jump.dev/JuMP.jl/stable/>
 - Forum: <https://discourse.julialang.org/c/domain/opt/13>
 - Free Textbook: <https://www.softcover.io/read/7b8eb7d0/juliabook2/introduction>
- InfiniteOpt.jl
 - Documentation: <https://pulsipher.github.io/InfiniteOpt.jl/stable/>
 - Forum: <https://github.com/pulsipher/InfiniteOpt.jl/discussions>

That's all!

