

InfiniteOpt



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

Dr. Joshua Pulsipher

julia

JUMP

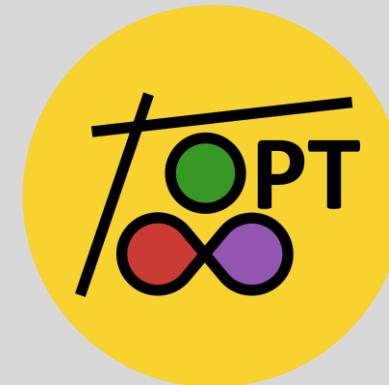
Today's Topics



CODING IN JULIA



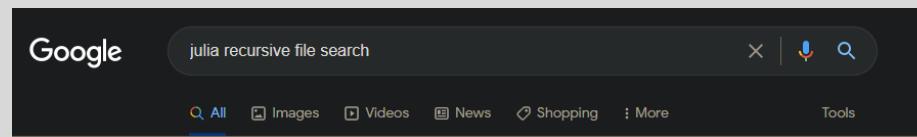
MATHEMATICAL OPTIMIZATION
VIA JUMP.JL



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



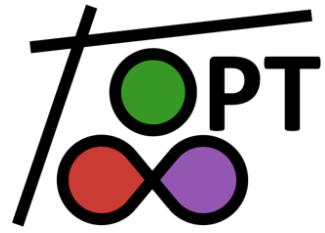
A screenshot of a Stack Overflow post titled "julia recursive file search". The post has 17 answers. The top answer, by user lognlp/patrick, shows a code snippet for using `walkdir`:

```
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 a link to the official documentation: <https://docs.julialang.org/en/v1/base/file/#Base.Filesystem.walkdir>.

Course Schedule

Times	Topic	Duration
8:30 a.m. – 9:45 a.m.	<i>Introduction</i> - The why and what of Julia, JuMP.jl, and InfiniteOpt.jl.	45 min.
9:45 a.m. – 10:15 a.m.	<i>Installation and Setup</i> - Configure software on personal laptop. Online interface provided as an alternative.	30 min.
10:15 a.m. – 10:30 a.m.	Break	15 min.
10:30 a.m. – 11:30 a.m.	<i>Julia: A Practical Introduction</i> – Overview of core types, programmatic syntax, and package management.	60 min.
11:30 a.m. – 11:45 a.m.	Break	15 min.
11:45 a.m. – 12:30 p.m.	<i>JuMP.jl: A Brief Introduction</i> – The basics of modeling and solving mathematical optimization problems in JuMP.jl.	45 min.
12:30 p.m. – 1:30 p.m.	Lunch	60 min.
1:30 p.m. – 2:30 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.
2:30 p.m. – 2:45 p.m.	Break	15 min.
2:45 p.m. – 3:45 p.m.	<i>InfiniteOpt.jl: The Basics</i> – An introduction on how to compactly model and solve complex infinite-dimensional optimization problems.	60 min.
3:45 p.m. – 4:00 p.m.	Break	15 min.
4:00 p.m. – 4:45 p.m.	<i>InfiniteOpt.jl: New Modeling Strategies</i> – A tutorial on how InfiniteOpt.jl enables new formulation/solution approaches.	45 min.
4:45 p.m. – 5:15 p.m.	<i>InfiniteOpt.jl: Deployment Tools</i> – An overview of the API to enable rapid deployment of new modeling/solution techniques.	30 min.
5:15 p.m. – 5:30 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?



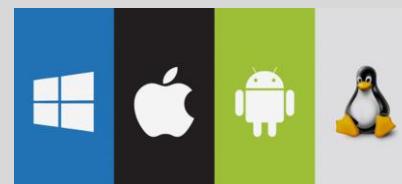
- Speed
 - Up to C speeds



- Parallel Computing
 - Designed for it from the ground up



- Portability
 - Works on any O.S.



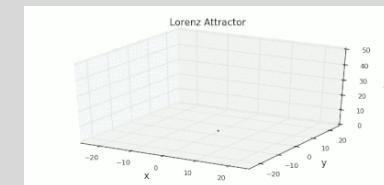
- Generality
 - Can be used for scripts, apps, metaprogramming, etc.



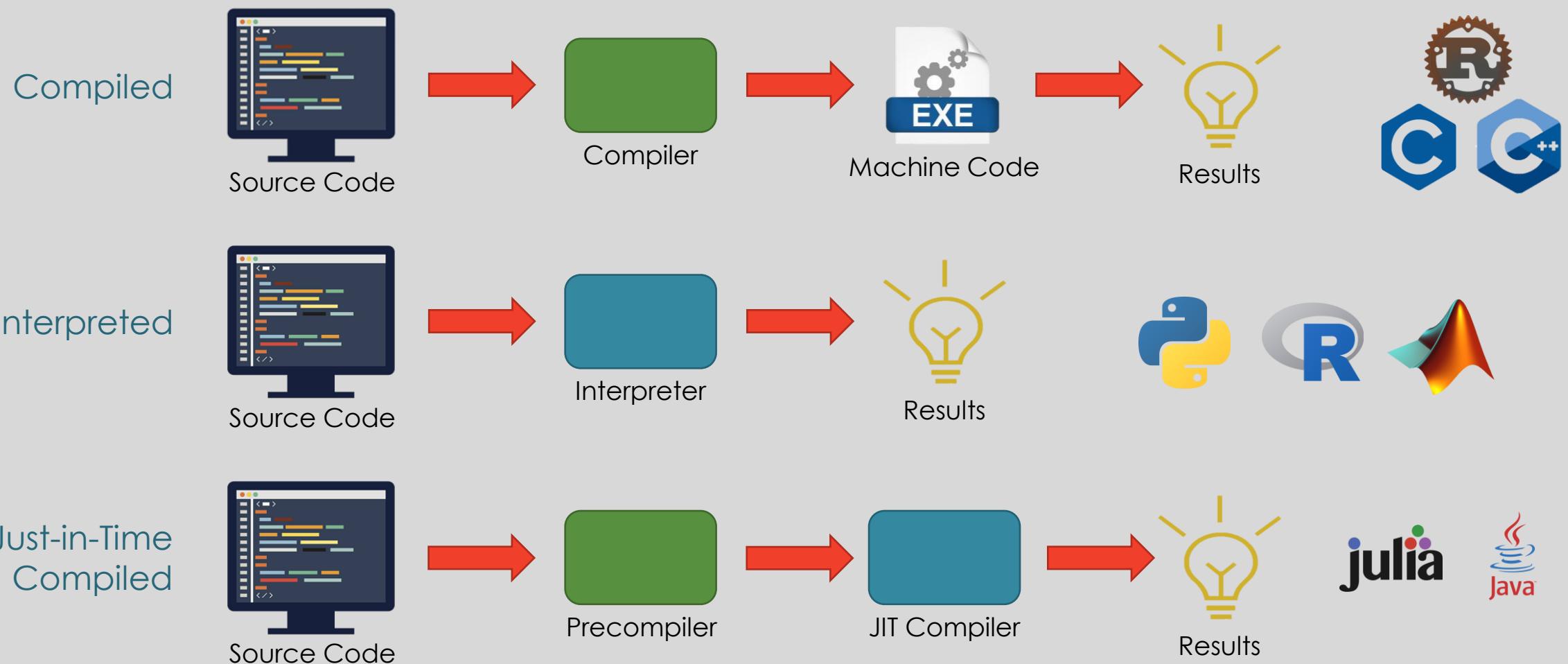
- Open Source
 - It's free!



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



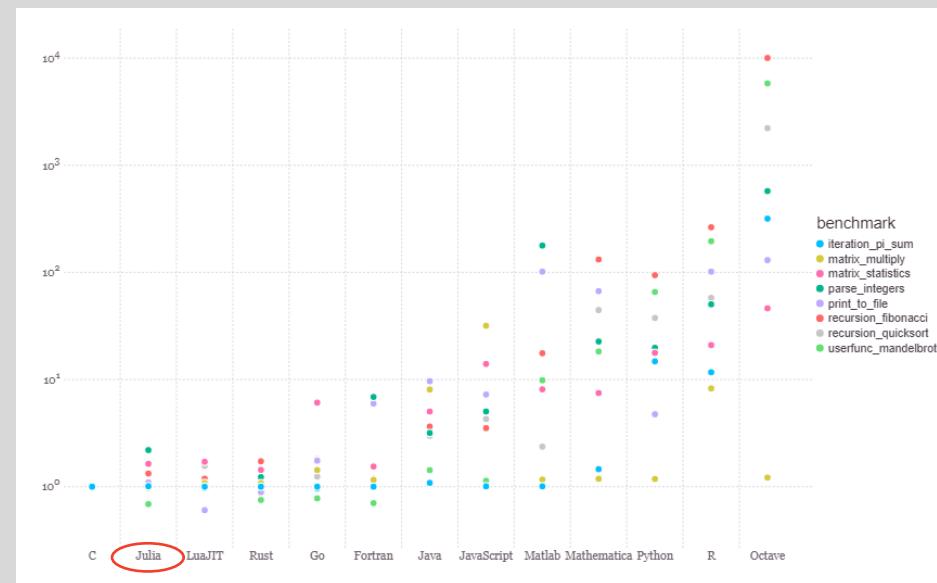
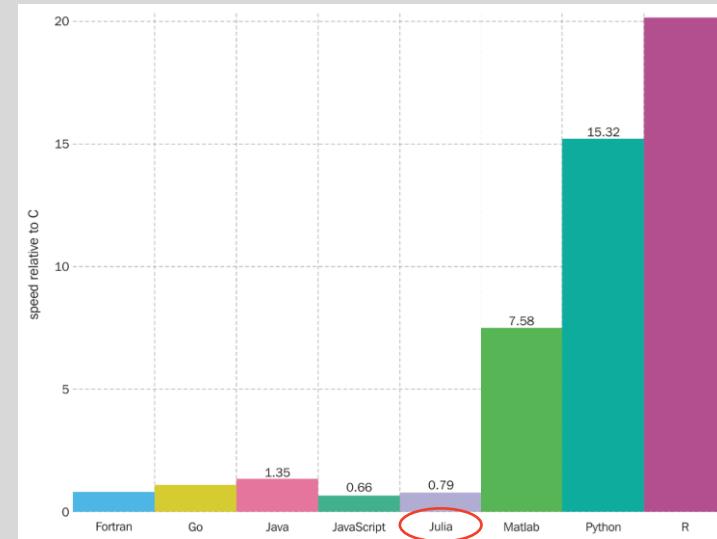
Programming Language Paradigms



*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*

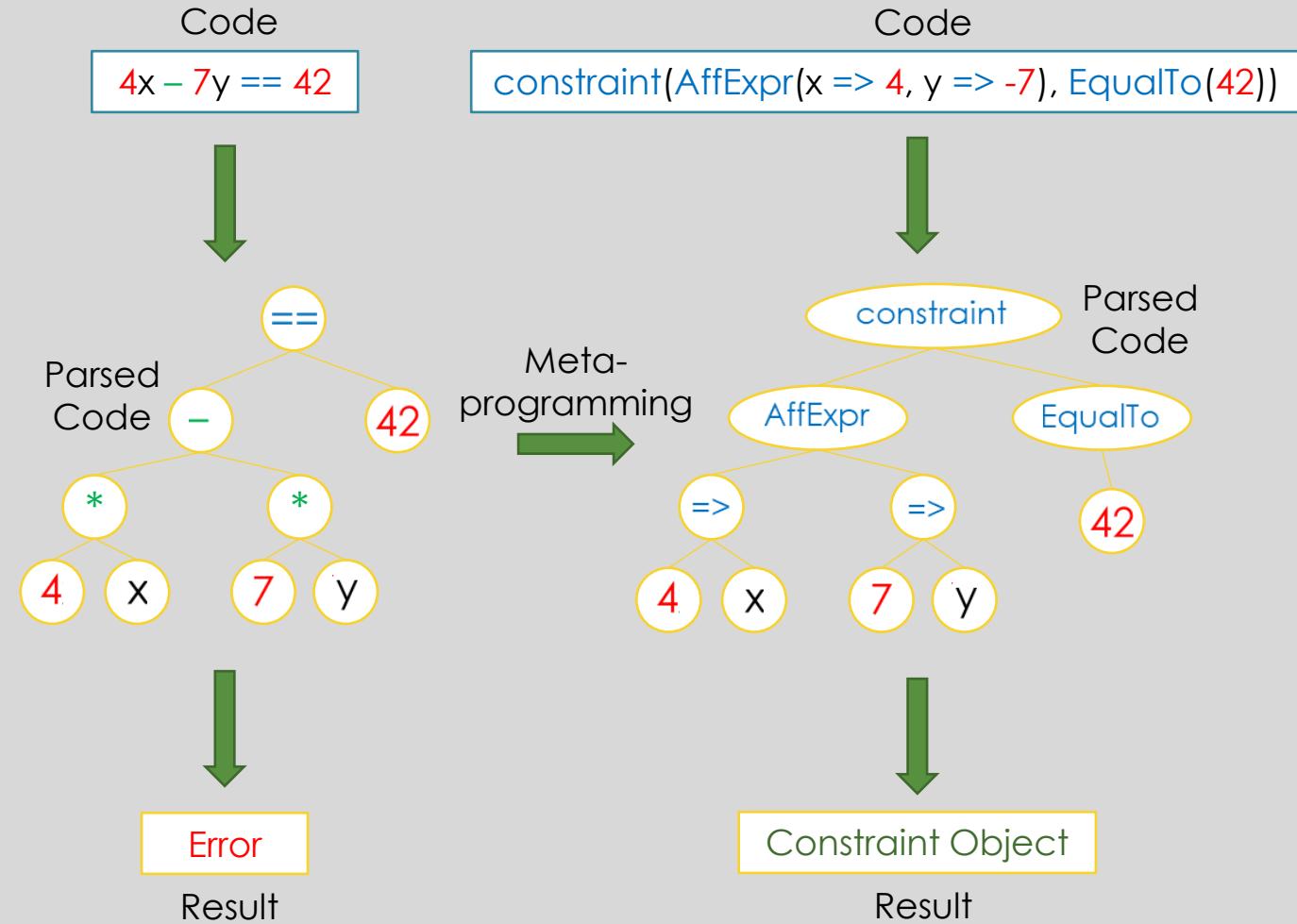


Metaprogramming (Macros)

- 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))
```

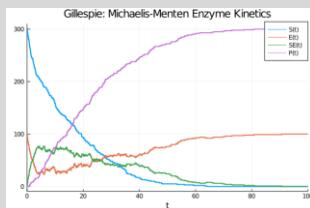


Packages

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



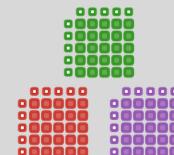
- DifferentialEquations.jl
 - Solve ODEs, PDEs, SDEs, DAEs, more
 - Fastest implementation available



- 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)
```

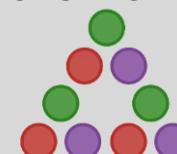
- DataFrames.jl
 - Efficient datasets (like Pandas)



- Flux.jl
 - Efficient/interpretable machine learning



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





Why JuMP.jl?

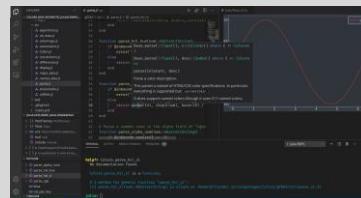
- High-level Syntax
 - Novice friendly

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

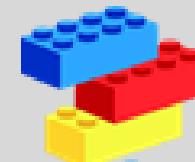
- Solver Library
 - Rapidly try different solvers



- Programmable
 - Easy embed in a data script



- Extensible
 - Can be extended for advanced techniques



- Open Source
 - It's free!

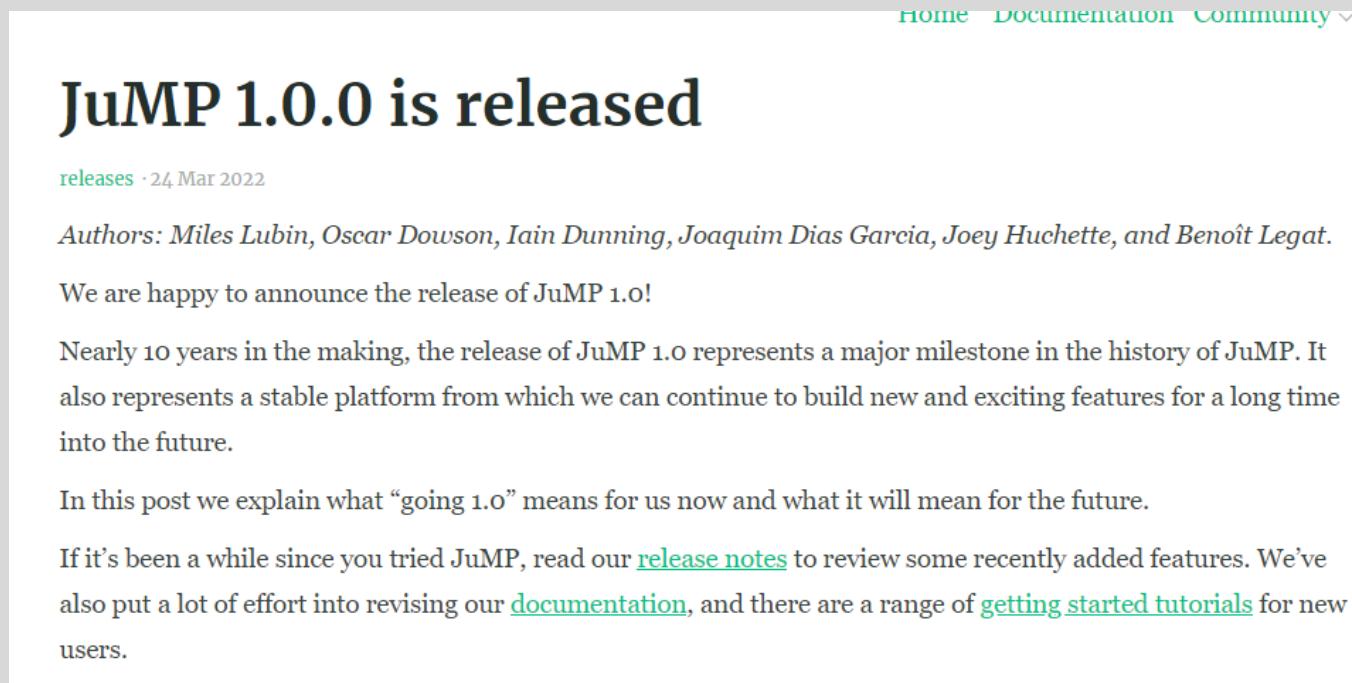


- Performance
 - Highly performant (it's Julia)



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 post titled "JuMP 1.0.0 is released". The post is dated March 24, 2022, and is authored by Miles Lubin, Oscar Dowson, Iain Dunning, Joaquim Dias Garcia, Joey Huchette, and Benoît Legat. The text discusses the major milestone of JuMP reaching version 1.0, its stability, and future plans. It also links to release notes, documentation, and tutorials.

releases · 24 Mar 2022

Authors: Miles Lubin, Oscar Dowson, Iain Dunning, Joaquim Dias Garcia, Joey Huchette, and Benoît Legat.

We are happy to announce the release of JuMP 1.0!

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.

In this post we explain what “going 1.0” means for us now and what it will mean for the future.

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.

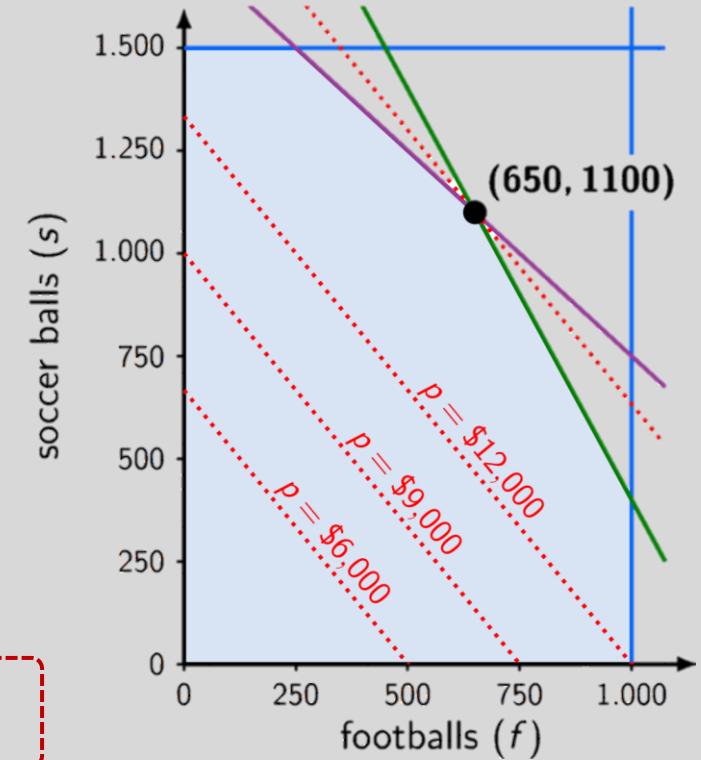
Aspects

- 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**

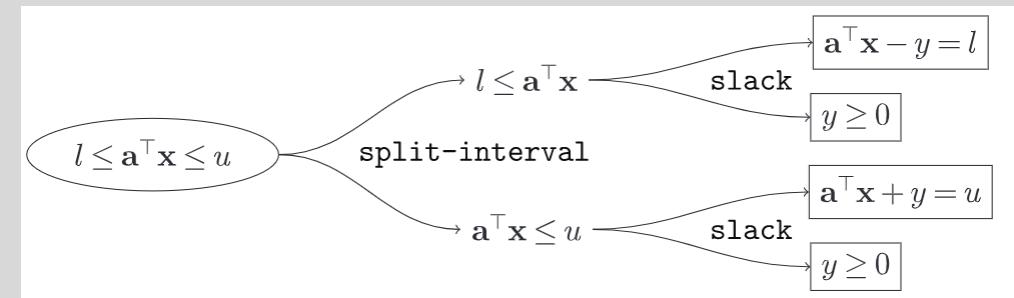
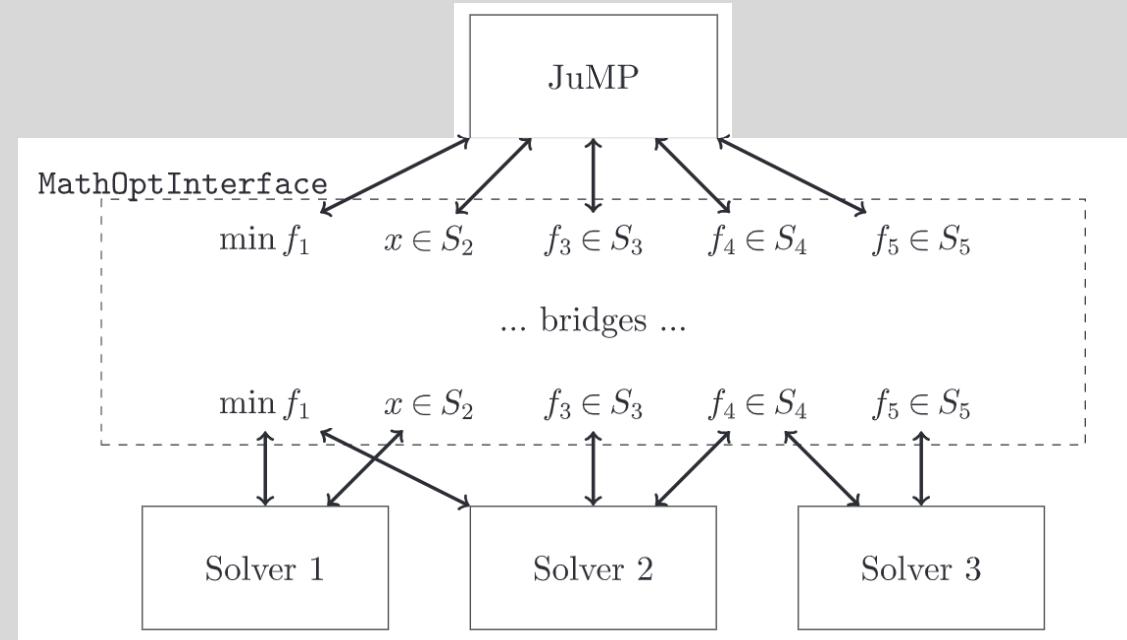
```
6 @constraint(model, 4f + 2s <= 4800)  
7 @constraint(model, f + s <= 1750)  
8 @constraint(model, 0 <= f <= 1000)  
9 @constraint(model, 0 <= s <= 1500)
```

- **Solve**

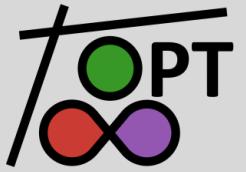
```
10 optimize!(model)  
11 profit = objective_value(model)  
12 f_best = value(f)  
13 s_best = value(s)
```

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

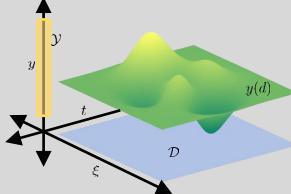


Why InfiniteOpt.jl?



- Unifying Abstraction

- Captures a wide breadth of problem classes



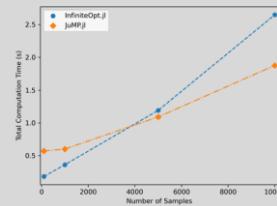
- High-level Syntax

- 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)
```

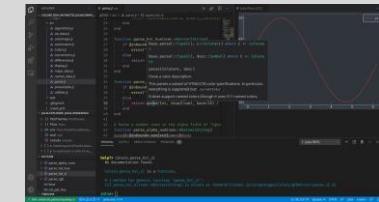
- Performance

- Quickly build complex models at scale



- Built on JuMP.jl

- Draw from an extensive suite of solvers



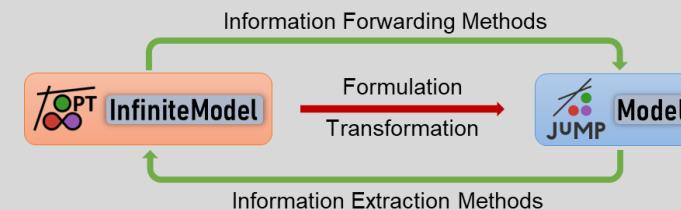
- Open Source

- It's free!



- Enables Accessible Research

- Implement advanced methods for general users



KEY TAKEAWAYS

- InfiniteOpt's intuitive interface makes optimal control problems **easier to model** and enables **new modeling objects**
- InfiniteOpt's transformation API automates and **accelerates advanced solution techniques** on CPUs and GPUs



UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

ACKNOWLEDGEMENTS



Victor Zavala
UW-Madison
Professor



Carl Laird
CMU
Professor



Ignacio Grossmann
CMU
Professor



Hector Perez
RelationalAI
Researcher



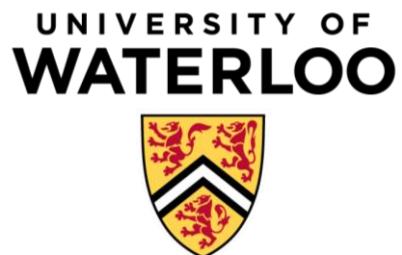
Daniel Ovalle Varela
CMU
PhD Candidate



Stefan Mazzadi
UWaterloo
Incoming MASc



Sungho Shin
MIT
Asst. Professor



Department of
Chemical Engineering



Carnegie Mellon University
Center for Advanced
Process Decision-making



UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

OUTLINE

- Unifying Abstraction
- InfiniteOpt
- New Modeling Objects
- Accelerated Solution Methods

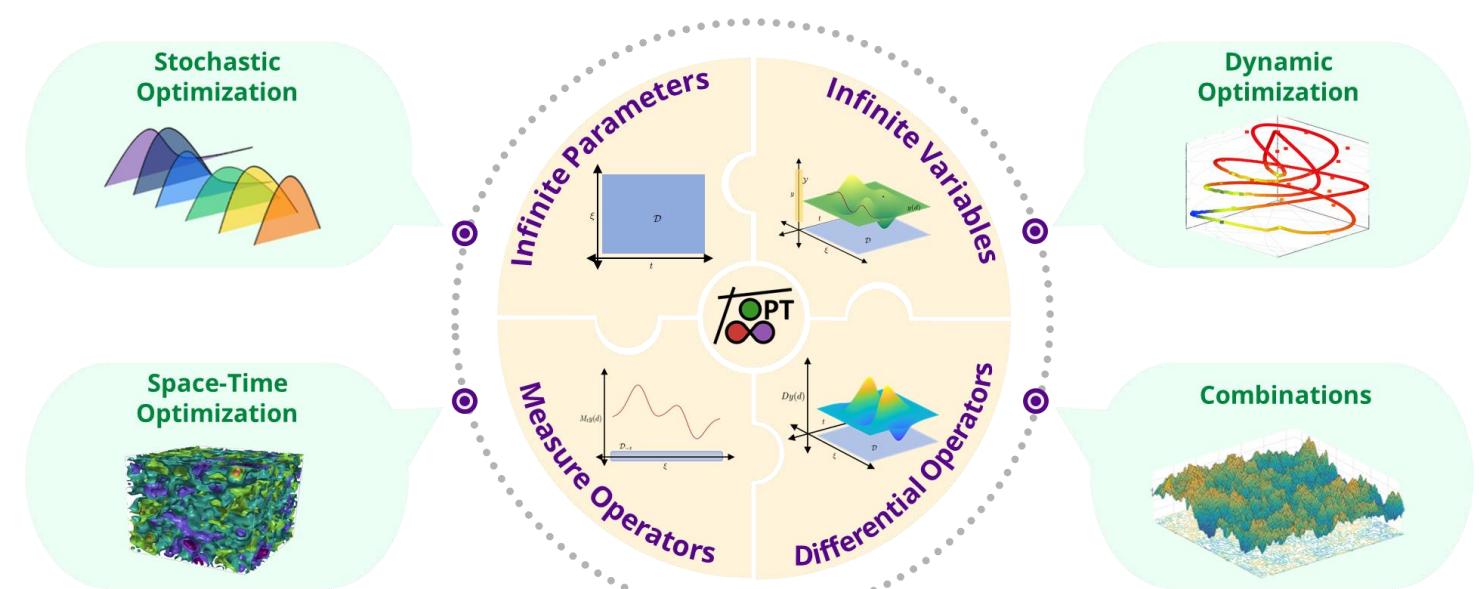


UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

OUTLINE

- Unifying Abstraction
- InfiniteOpt
- New Modeling Objects
- Accelerated Solution Methods



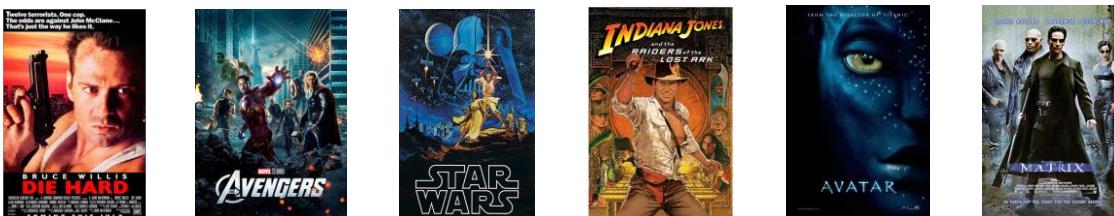
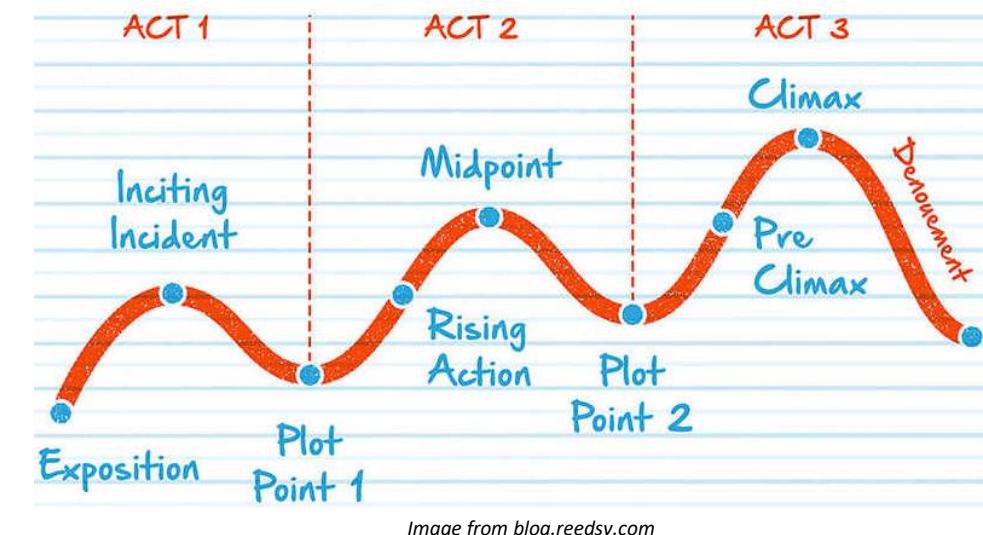
UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

WHAT IS AN ABSTRACTION?

Definition: A general representation of events/objects comprised of essential **attributes**.

Example: 3 Act Storyline



Power of Abstraction

- Better fundamental understanding
- Facilitates **scientific discovery**
- Enable **accessible modeling tools**

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$



MATLAB



UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

DYNAMIC OPTIMIZATION

Dynamic Optimization

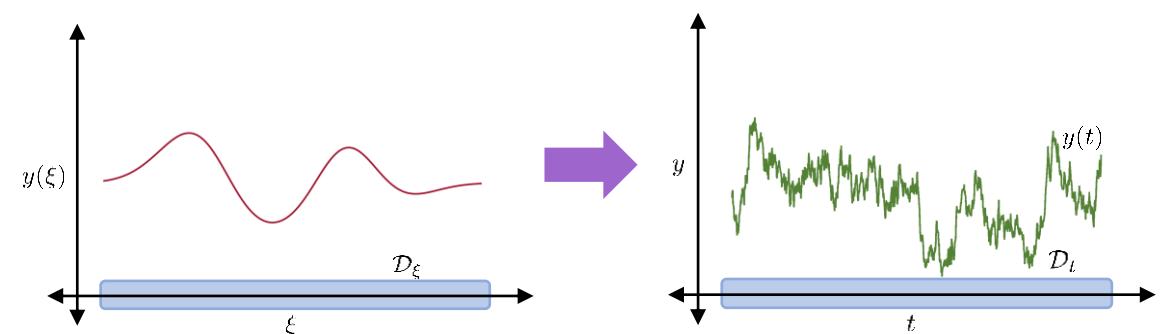
$$\begin{aligned} \min_{\mathbf{y}(\mathbf{t})} \quad & \int_{\mathbf{t} \in \mathcal{D}_t} f(\mathbf{y}(\mathbf{t}), \mathbf{t}) d\mathbf{t} \\ \text{s.t.} \quad & g(\mathbf{y}(\mathbf{t}), \mathbf{t}) \leq 0, \quad \mathbf{t} \in \mathcal{D}_t \end{aligned}$$

Stochastic Optimization

$$\begin{aligned} \min_{\mathbf{y}(\xi)} \quad & \int_{\xi \in \mathcal{D}_\xi} f(\mathbf{y}(\xi), \xi) p(\xi) d\xi \\ \text{s.t.} \quad & g(\mathbf{y}(\xi), \xi) \leq 0, \quad \xi \in \mathcal{D}_\xi \end{aligned}$$

Different problem, similar features

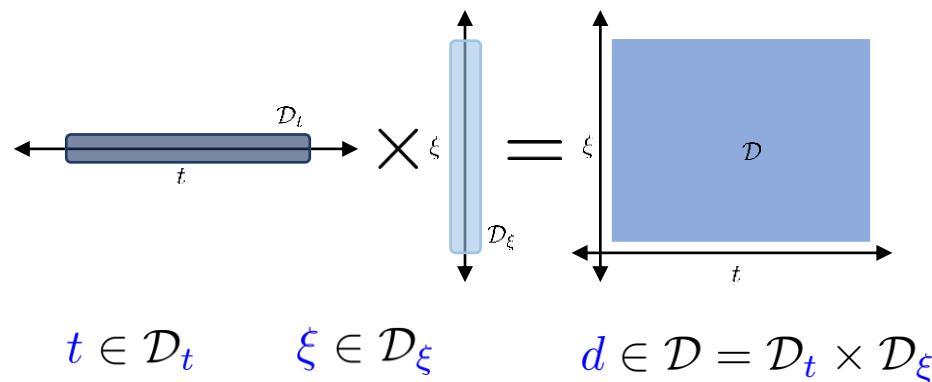
- Replace $\xi \in \mathcal{D}_\xi$ with time $\mathbf{t} \in \mathcal{D}_t$
- Let $p(\xi) = 1$
- Shape $\mathbf{y}(\mathbf{t})$ instead of $\mathbf{y}(\xi)$



INFINITE-DIMENSIONAL OPTIMIZATION

Infinite Parameters

- Index over **continuous domains**



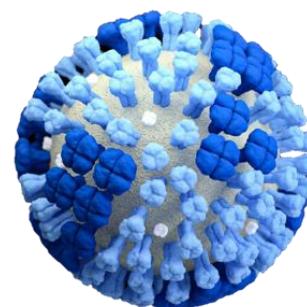
- Example:** Disease Control

- Population dynamics

$$t \in [0, t_f]$$

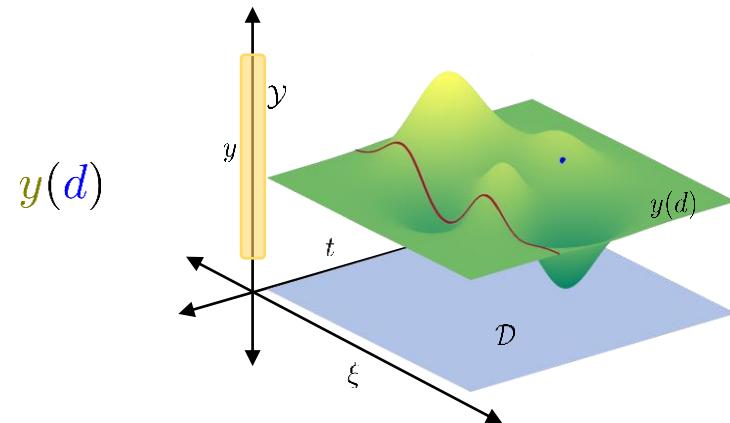
- Uncertain infection rates

$$\xi \in (-\infty, \infty) \sim \mathcal{N}(\mu, \Sigma)$$



Infinite Variables

- Decisions** indexed by infinite parameters



- Example:** Disease Control

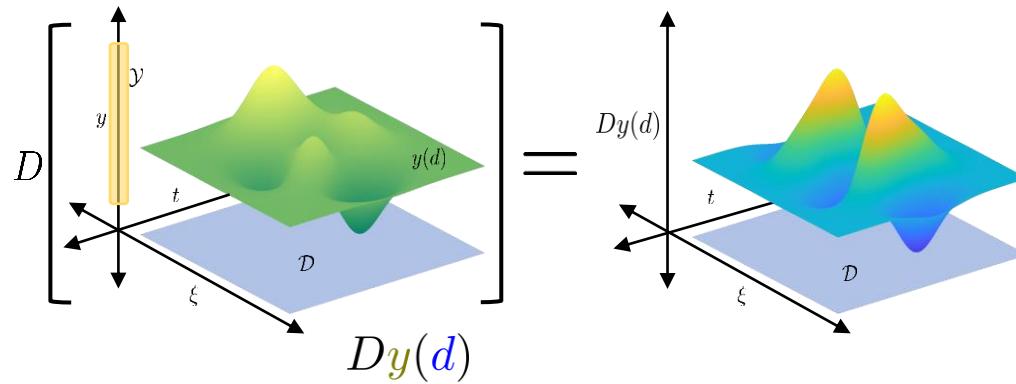
- Population of infected at a particular time and infection rate

$$y_i(t, \xi)$$

INFINITE-DIMENSIONAL OPTIMIZATION

Differential Operators

- Capture of **rate of change** in variables



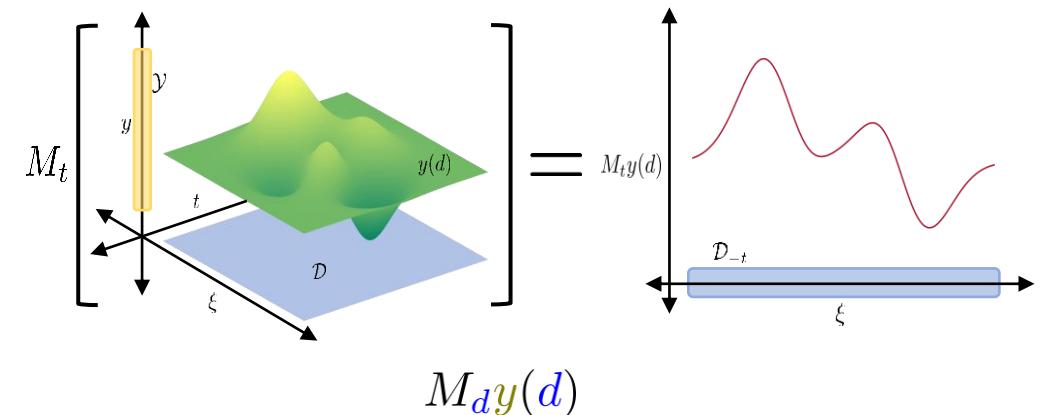
- **Example:** Disease Control

- Time derivative $\frac{\partial y_i(t, \xi)}{\partial t}$

- SEIR model $\frac{\partial y_i(t, \xi)}{\partial t} = \xi y_e(t) - \gamma y_i(t)$

Measure Operators

- **Summarize variables** over continuous domains



- **Example:** Disease Control

- Summarize overall infections

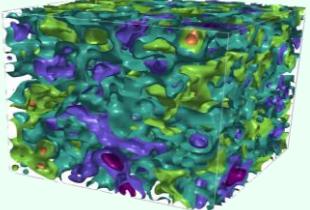
$$\int_{t \in \mathcal{D}_t} \mathbb{E}_{\xi}[y_i(t, \xi)] dt \quad \mathbb{E}_{\xi} \left[\int_{t \in \mathcal{D}_t} y_i(t, \xi) dt \right]$$

UNIFYING ABSTRACTION

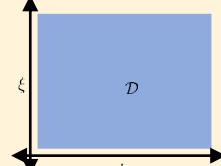
Stochastic
Optimization



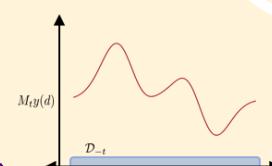
Space-Time
Optimization



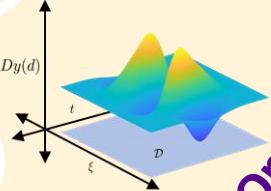
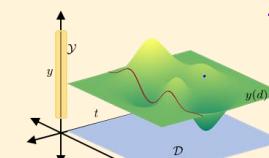
Infinite Parameters



Measure Operators

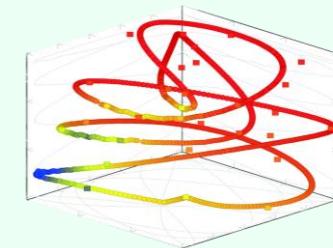


Infinite Variables

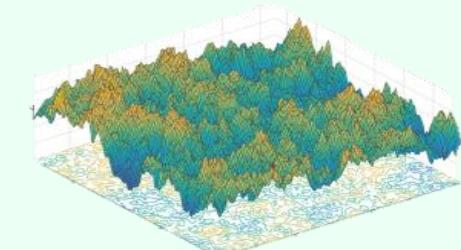


Differential Operators

Dynamic
Optimization



Combinations



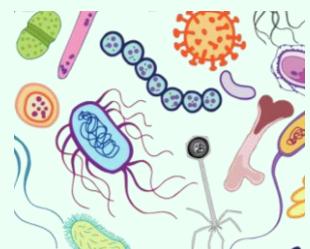
APPLICATIONS

Unifying Abstraction

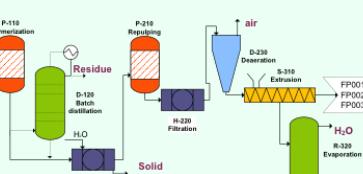
Dynamic Optimization (Time)



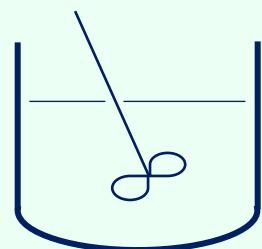
Autonomous Vehicles



Microbial Communities

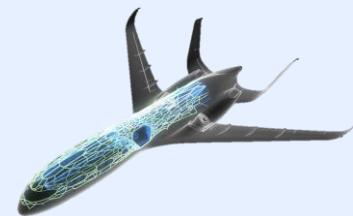


Process Control



Reaction Systems

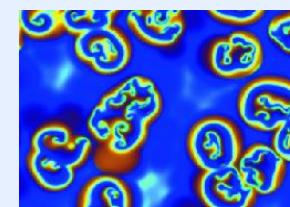
PDE-Constrained Optimization (Space-Time)



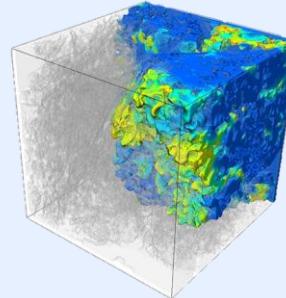
Structural Design



Wildfire Simulation



Diffusive Systems

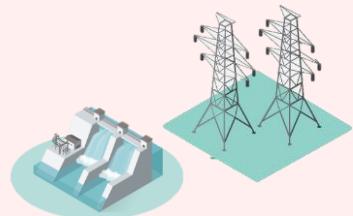


Complex Fluids

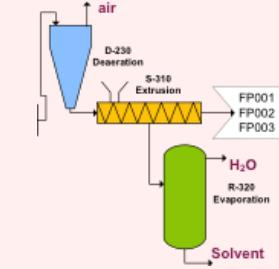
Stochastic Optimization (Uncertainty)



Pharmaceutical Production



Optimal Power Flow



Process Design

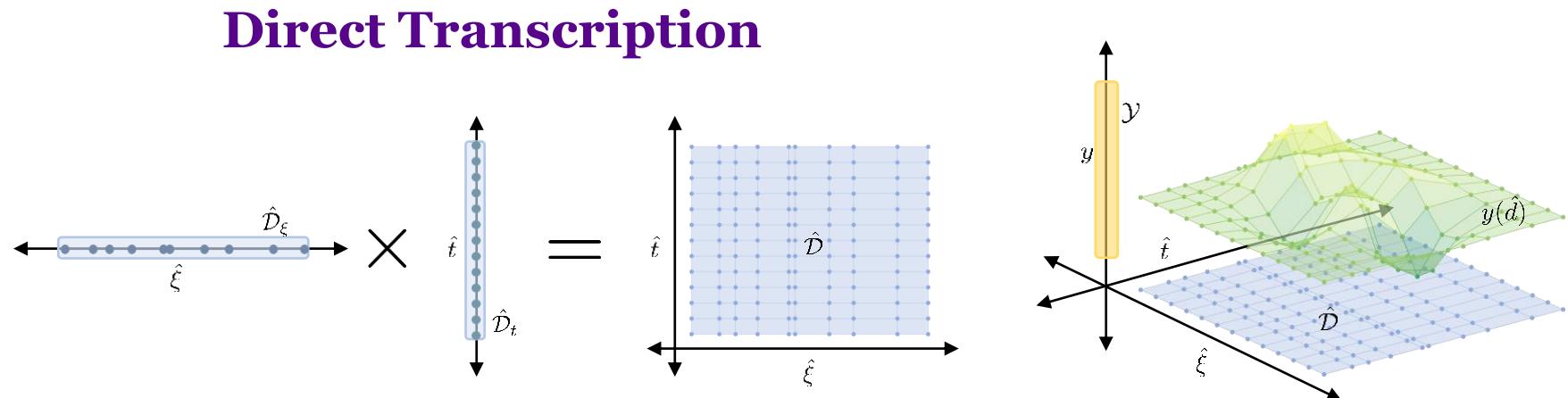


Investment Planning

TRANSFORMING INFINITEOPT PROBLEMS INTO FINITE ONES

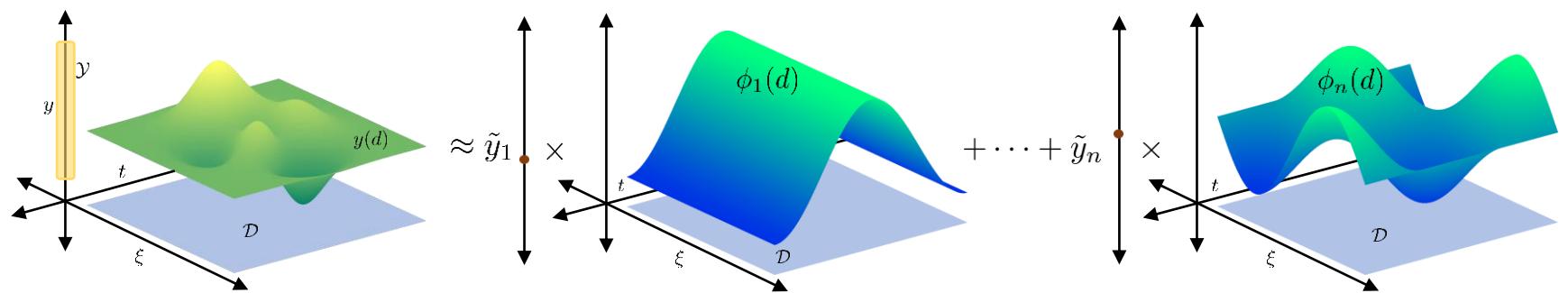
Project onto set of finite points $\hat{\mathcal{D}}$

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



Project onto set of known **basis functions**

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



OUTLINE

- Unifying Abstraction
- **InfiniteOpt**
- New Modeling Objects
- Accelerated Solution Methods



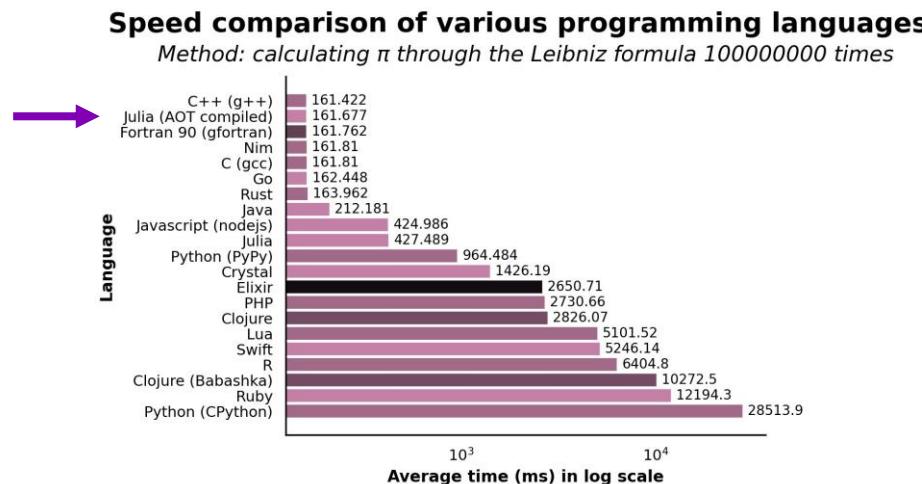
UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

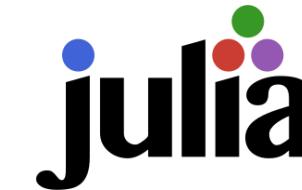
WHY JULIA?



- Fast as C/C++/Fortran, but as **simple** as Python/MATLAB/R



github.com/coin-or/Ipopt.git
220k lines of code



github.com/MadNLP/MadNLP.jl.git
20k lines of code

- JuMP provides intuitive and powerful platform for optimization



- **50+ solver library** + all AMPL and GAMS solvers
- Extensive amount of **constraint types** supported



UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING



Why is it Different?

- Implements **unifying abstraction**
 - Models a wide range of problems
 - Leverages structure to **accelerate solutions**
- Implemented in **julia**
 - Enables **intuitive** symbolic expressions
 - Highly **performant**
- **Extensive resources**
 - Documentation, tutorials, examples, forum, short courses, videos

Basic Usage
Infinite, semi-infinite, point, and finite variables are summarized in the following table:

Name	Variable Type Object	Description	Example
Infinite	Infinite	decision functions	$y(t, x, \xi)$
Semi-Infinite	SemiInfinite	partially evaluated decision functions	$y(t_0, x_0, \xi)$
Point	Point	fully evaluated decision functions	$y(t_0, x_0, \xi_0)$
Finite	NA	classical decision variables	z

Infinite, semi-infinite, point, and finite variables are defined via `@variable` (inherited from JuMP) with their respective variable type object arguments: `Infinite`, `SemiInfinite`, and `Point` (finite variables don't use a variable type object).

Let's first set up a simple space-time model with infinite parameters time `t` and spatial position `x`:

```
Julia> using InfiniteOpt
Julia> model = InfiniteModel();
Julia> @infinite.parameter(model, t in [0, 10])
t
Julia> @infinite.parameter(model, x[1:2] in [-1, 1], independent = true)
2-element Vector{GeneralVariableRef}
x[1]
```



Try it @ <https://github.com/infiniteopt/InfiniteOpt.jl>



Intuitive Modeling API

$$\frac{\partial y_b(t, \xi)}{\partial t} = 2y_b(t, \xi)^2 + y_a(t) - z_1$$

$$\mathbb{E}_\xi [y_c(t, \xi)] \geq \alpha$$

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

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

Impact

- 1000s of downloads
- Use cases in **diverse disciplines**
 - e.g., evolutionary biology, rocketry, economics, autonomous vehicles



UNIVERSITY OF
WATERLOO

FACULTY OF
ENGINEERING

WHAT MODELING OBJECTS DOES IT SUPPORT?

InfiniteOpt Specific

- Infinite parameters
 - Any distribution from Distributions.jl
 - Cartesian domains (e.g., time/space)
- Integrals
- Expectations
- Ordinary differential equations
- Partial differential equations
- Chance constraints

Inherited from JuMP

- Affine/quadratic/nonlinear/mixed-integer models
- Constraint programming
- Vector-valued constraints
- Conic constraints
- Complementarity constraints
- Multi-objectives
- Indicator constraints
- Many many more...

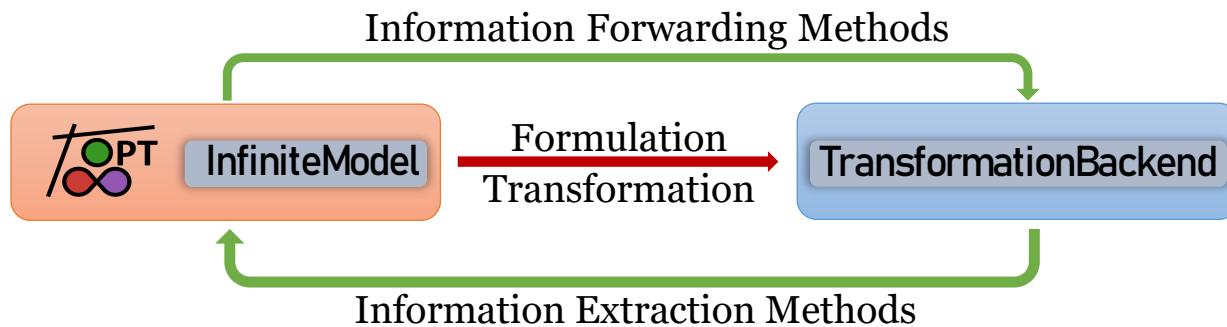


UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

TRANSFORMING INFINITEOPT MODELS

Transformation Paradigm

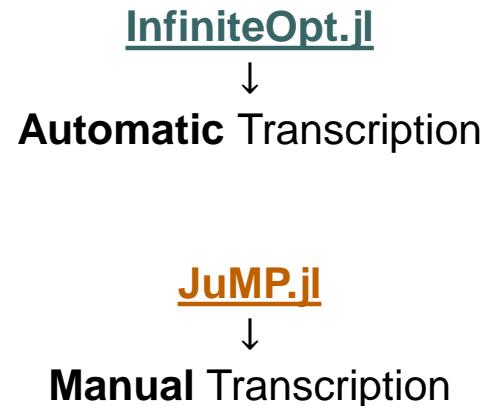
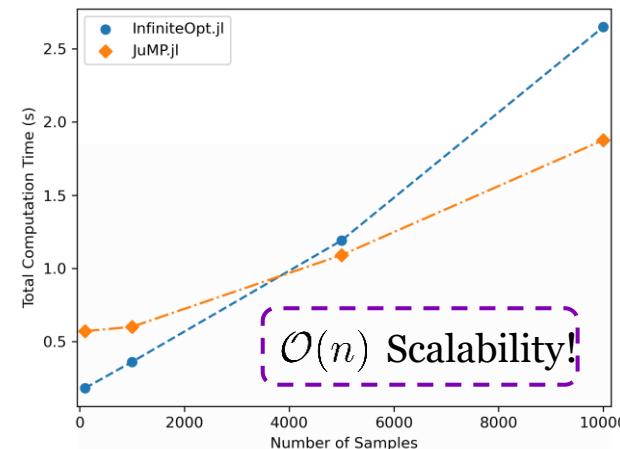


Transformation API

- Highly extensible to **make advanced solution techniques accessible/automated**
- Detailed templates, tutorials, and docs

Automated Transcription (via TranscriptionOpt)

- Many **derivative/measure approximations**
 - Orthogonal collocation, Gauss quadrature, etc.
- **Performant**



MODELING SYNTAX

- 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, tf], 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)
```

- Transform and Solve

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

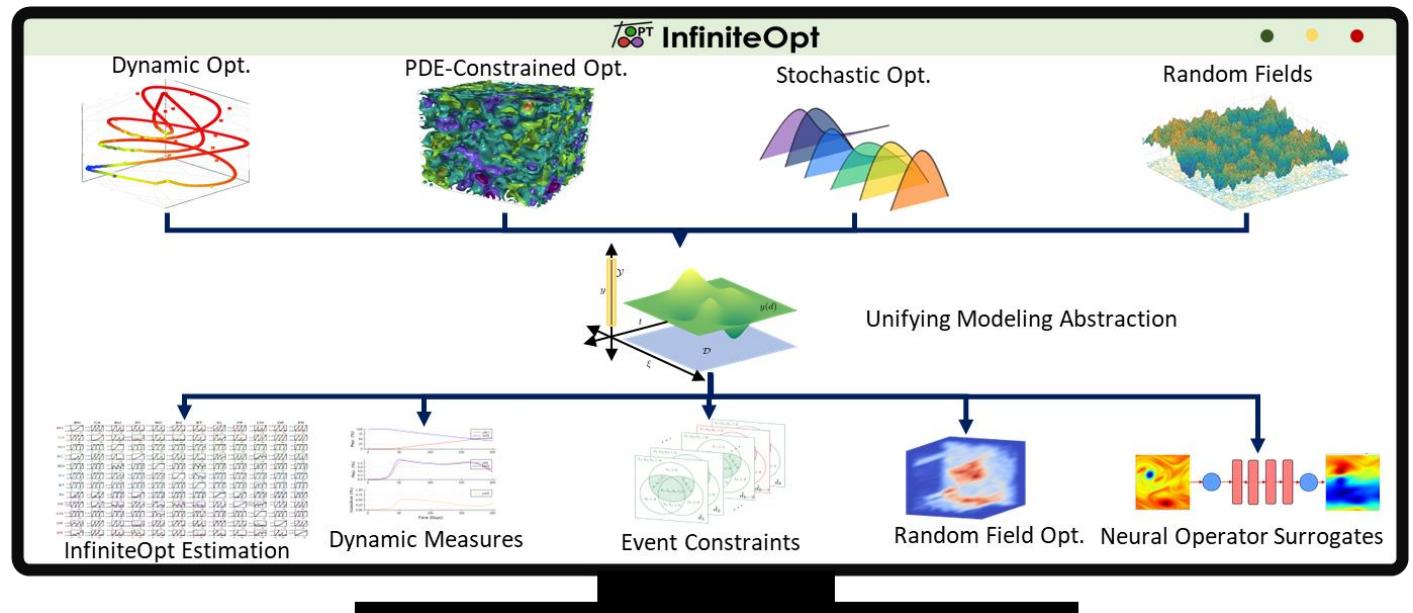


UNIVERSITY OF
WATERLOO

FACULTY OF
ENGINEERING

OUTLINE

- Unifying Abstraction



- New Modeling Objects

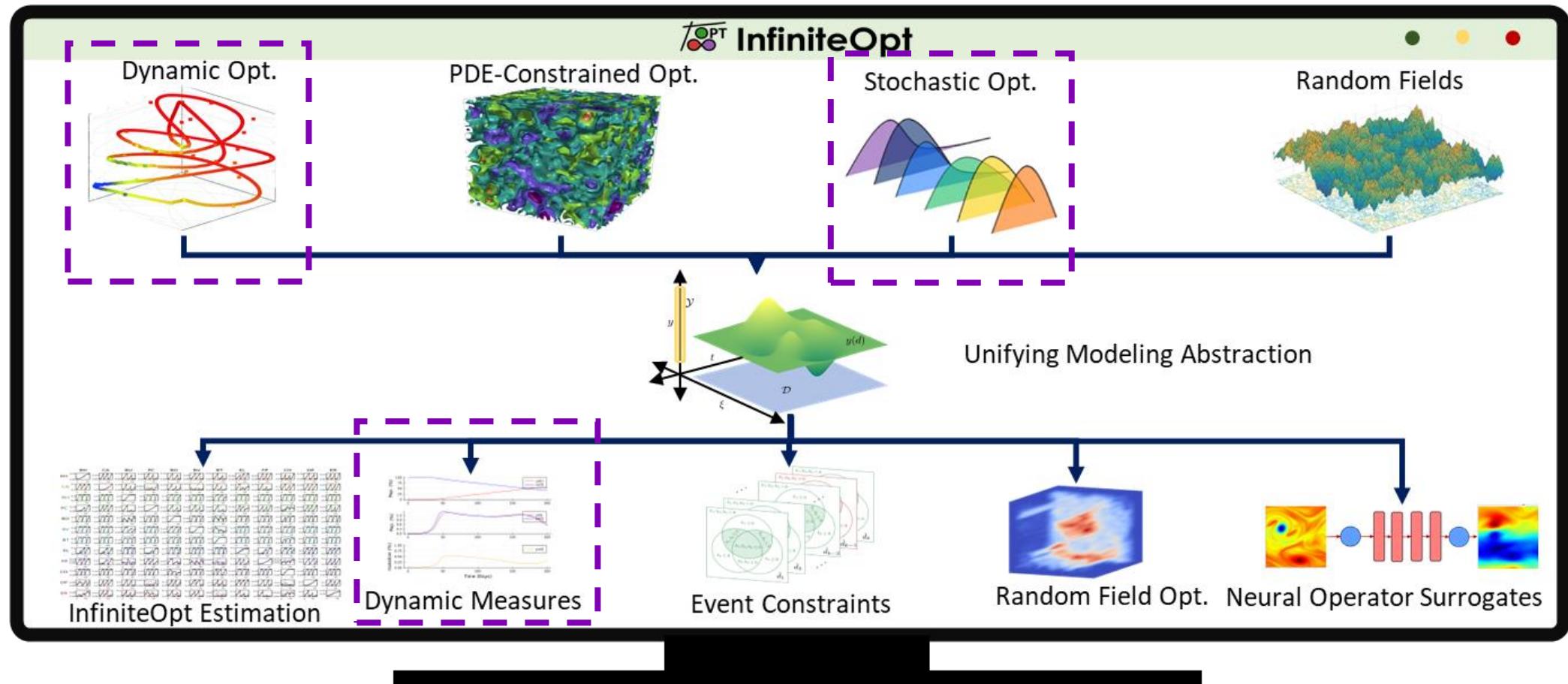
- Accelerated Solution Methods



UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

INNOVATION VIA CROSS POLLINATION



J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A unifying modeling abstraction for infinite-dimensional optimization." 2022

J. L. Pulsipher, B. Davidson, and V. M. Zavala. "New Measures for Shaping Trajectories in Dynamic Optimization." 2022

J. L. Pulsipher, Hongisto, and V. M. Zavala. "Random Field Optimization." 2023

D. Ovalle, H. Perez, I. E. Grossmann, C. D. Laird, **J. L. Pulsipher**. "Event Constrained Programming". In progress

STOCHASTIC OPTIMIZATION

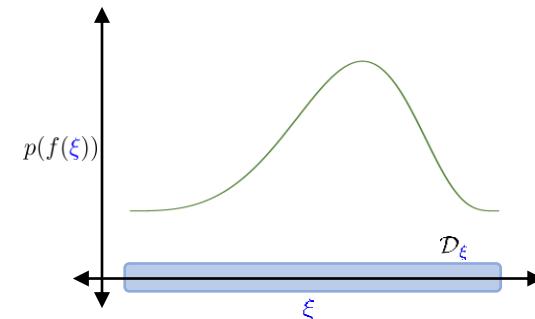
Formulation

$$\begin{aligned} \min_{\textcolor{brown}{z} \in \mathcal{Z}, \textcolor{brown}{y}(\xi) \in \mathcal{Y}} \quad & M_\xi f(\textcolor{brown}{z}, \textcolor{brown}{y}(\xi), \xi) \\ \text{s.t.} \quad & g(\textcolor{brown}{z}, \textcolor{brown}{y}(\xi), \xi) \leq 0, \quad \xi \in \mathcal{D}_\xi \end{aligned}$$

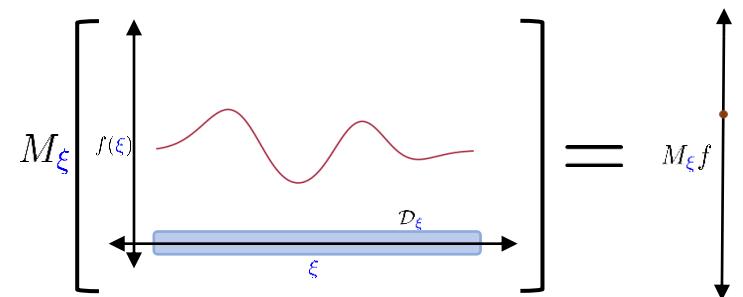
- Random infinite parameter
 $\xi : \Omega \mapsto \mathcal{D}_\xi \quad (\Omega, \mathcal{F}, F)$
- Measured cost function of 1st/2nd stage
- 2nd stage recourse constraints

Risk Measures

- Shape the random **cost density function**



- Directly shape the random **cost function**



STOCHASTIC RISK MEASURES

Expectation

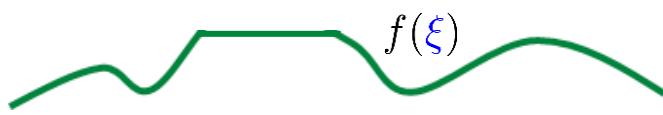
- Integral weighted via **density function**



$$\mathbb{E}_\xi[f(\xi)] := \int_{\xi \in \mathcal{D}_\xi} f(\xi)p(\xi)d\xi$$

CVaR

- Penalize the $1 - \alpha$ **large cost values**



$$\text{CVaR}_\xi(f(\xi); \alpha) := \min_{\hat{f} \in \mathbb{R}} \left\{ \hat{f} + \frac{1}{1-\alpha} \mathbb{E}_\xi[f(\xi) - \hat{f}]_+ \right\}$$

Mean-Variance

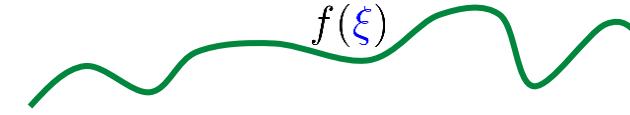
- Penalize the cost spread/fluctuation



$$\mathbb{E}\text{-}\mathbb{V}_\xi[f(\xi)] := \mathbb{E}_\xi[f(\xi)] + \lambda \mathbb{V}_\xi[f(\xi)]$$

Disutility

- Penalize the **unfavorable cost outcomes**



$$\tilde{D}_\xi(f(\xi)) := \inf_{\hat{f} \in \mathbb{R}} \mathbb{E}_\xi[f(\xi) + s(f(\xi) - \hat{f})]$$

TIME-VALUED RISK MEASURES

Expectation

- Integral with **weighting function**

$$\mathbb{E}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t})] := \int_{\textcolor{blue}{t} \in \mathcal{D}_t} f(\textcolor{blue}{t}) p(\textcolor{blue}{t}) dt$$

Mean-Variance

- Penalize the cost **fluctuation**
- Uses weighting function

$$\mathbb{E}\text{-}\mathbb{V}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t})] := \mathbb{E}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t})] + \lambda \mathbb{V}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t})]$$

CVaR

- Penalize the $1 - \alpha$ fraction **largest cost values**
- Uses weighting function

$$\text{CVaR}_{\textcolor{blue}{t}}(f(\textcolor{blue}{t}); \alpha) := \min_{\hat{f} \in \mathbb{R}} \left\{ \hat{f} + \frac{1}{1-\alpha} \mathbb{E}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t}) - \hat{f}]_+ \right\}$$

Disutility

- Penalize the **unfavorable cost outcomes**
- Uses weighting function

$$\tilde{D}_{\textcolor{blue}{t}}(f(\textcolor{blue}{t})) := \inf_{\hat{f} \in \mathbb{R}} \mathbb{E}_{\textcolor{blue}{t}}[f(\textcolor{blue}{t}) + s(f(\textcolor{blue}{t}) - \hat{f})]$$

OPTIMAL DISEASE CONTROL

Formulation

- Limit infected and minimize closures
- Use the SEIR disease model
- Let cost function be control action

$$\min M_t y_u(t)$$

$$\begin{aligned} \text{s.t. } & \dot{y}_s(t) = (y_u(t) - 1) \beta y_s(t) y_i(t), \quad t \in \mathcal{D}_t \\ & \dot{y}_e(t) = (1 - y_u(t)) \beta y_s(t) y_i(t) - \xi y_e(t), \quad t \in \mathcal{D}_t \\ & \dot{y}_i(t) = \xi y_e(t) - \gamma y_i(t), \quad t \in \mathcal{D}_t \\ & \dot{y}_r(t) = \gamma y_i(t), \quad t \in \mathcal{D}_t \\ & y_s(0) = s_0, y_e(0) = e_0, y_i(0) = i_0, y_r(0) = r_0 \\ & y_i(t) \leq \bar{y}_i, \quad t \in \mathcal{D}_t \\ & y_u(t) \in [0, \bar{y}_u], \quad t \in \mathcal{D}_t \end{aligned}$$

Implementation

```
using InfiniteOpt, Ipopt
# Set the parameters
γ, β, ξ = 0.303, 0.727, 0.3
s0, e0, i0, r0 = 1 - 1e-5, 1e-5, 0, 0

# Define the model
m = InfiniteModel(Ipopt.Optimizer)

# Define the time parameter
@infinite_parameter(m, t ∈ [0, 200], num_supports = 101)

# Add the variables
@variable(m, ys, Infinite(t))
@variable(m, ye, Infinite(t))
@variable(m, yi ≤ 0.02, Infinite(t))
@variable(m, yr, Infinite(t))
@variable(m, 0 ≤ yu ≤ 0.8, Infinite(t))

# Set the time expectation objective
@objective(m, Min, E(yu, t))

# Define the SEIR equations
@constraint(m, ∂(ys, t) == -(1 - yu) * β * ys * yi)
@constraint(m, ∂(ye, t) == (1 - yu) * β * ys * yi - ξ * ye)
@constraint(m, ∂(yi, t) == ξ * ye - γ * yi)
@constraint(m, ∂(yr, t) == γ * yi)
@constraint(m, ys(0) == ys0)
@constraint(m, ye(0) == ye0)
@constraint(m, yi(0) == yi0)
@constraint(m, yr(0) == yr0)

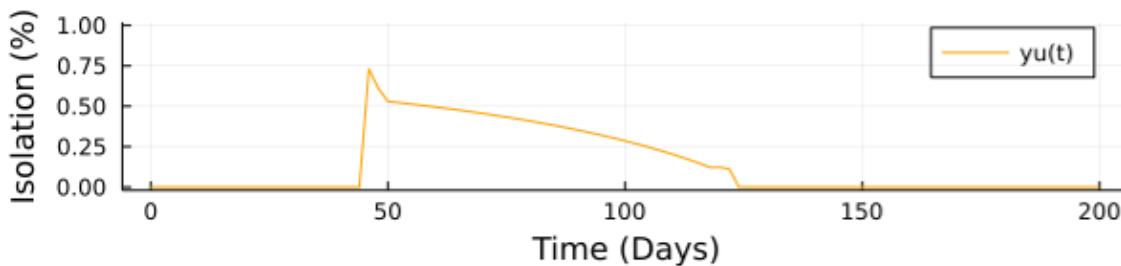
# Solve the model and retrieve results
optimize!(m)
u_opt = value(yu)
ts = value(t)
```



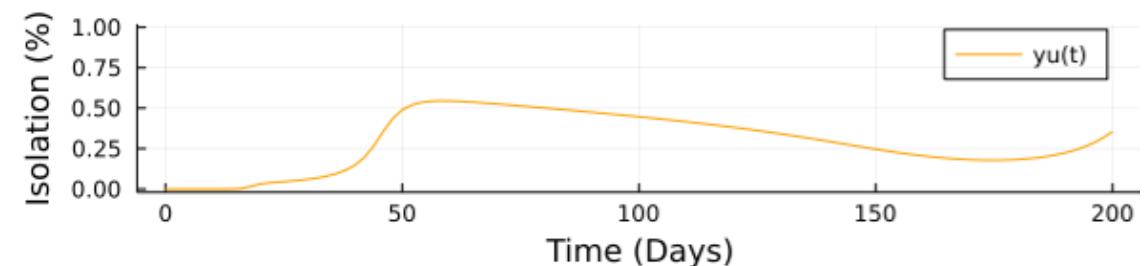
OPTIMAL COST/CONTROL TRAJECTORIES

Expectation (Uniform pdf)

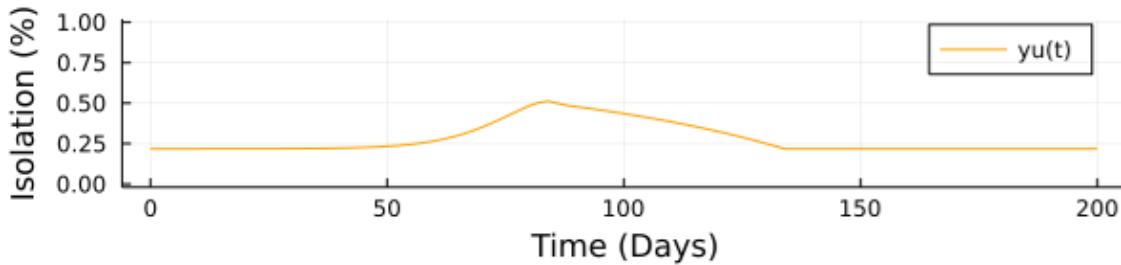
(Same as traditional Bolza objective)



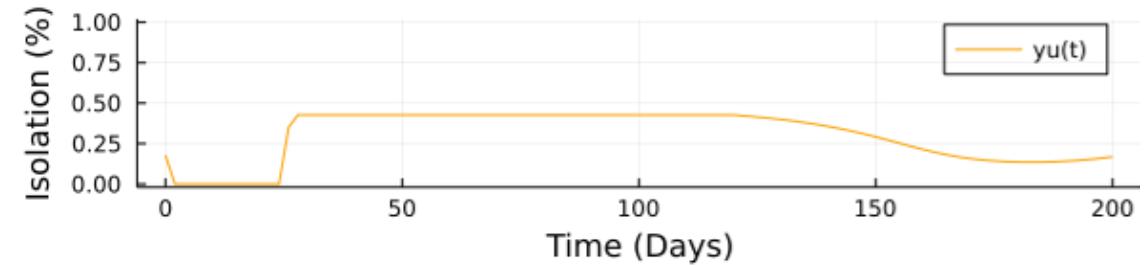
Expectation (Exponential pdf)



Mean-Variance ($\lambda = 8$)

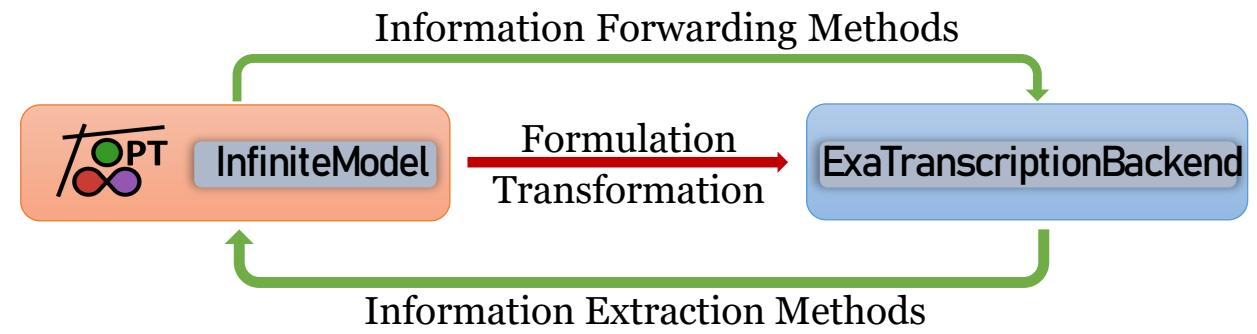


CVaR ($\alpha = 0.9$)



OUTLINE

- Unifying Abstraction
- InfiniteOpt
- New Modeling Objects
- **Accelerated Solution Methods**



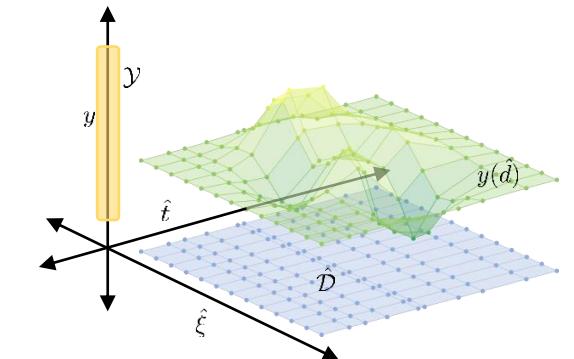
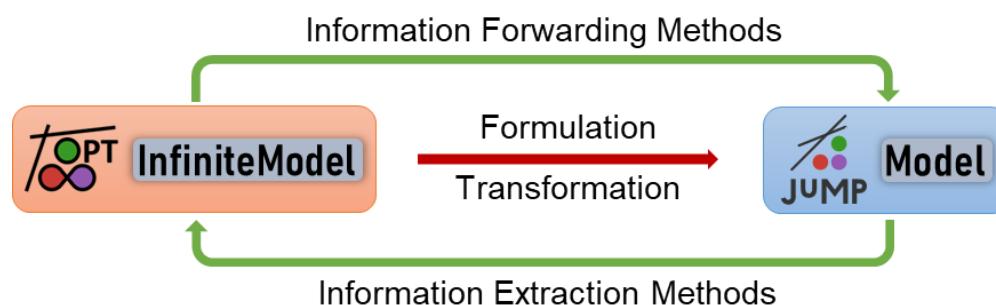
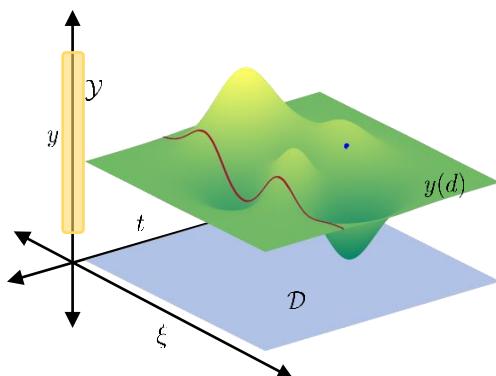
SOLVING INFINITEOPT PROBLEMS VIA TRANSCRIPTIONOPT

- Apply **transformation** to obtain finite JuMP model that can be solved
- InfiniteOpt has a large suite of **discretization** techniques
- Discretized InfiniteOpt problems have **repeated structure**
- Traditional modeling languages like JuMP do not leverage repeated structure

$$\sin^2(y(t)) \leq 42, \quad t \in \mathcal{D}_t$$

↓

$$\sin^2(y_k) \leq 42, \quad k \in \mathcal{K}$$



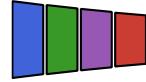
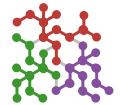
Can we leverage the repeated structure to **accelerate solution performance**?

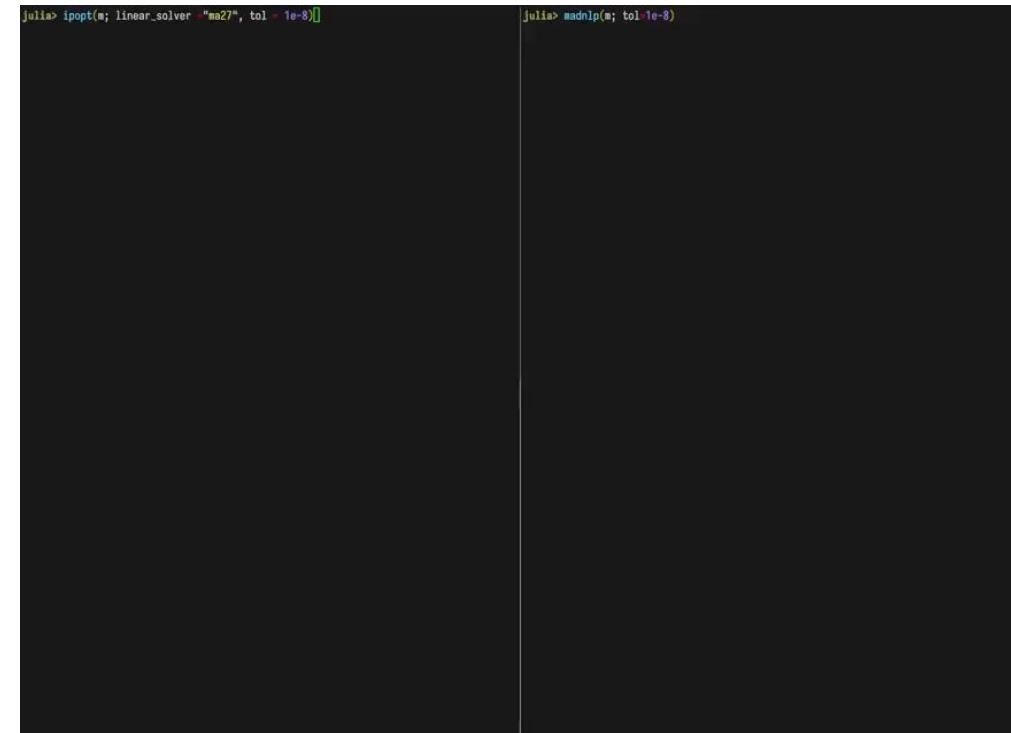


UNIVERSITY OF
WATERLOO

| FACULTY OF
ENGINEERING

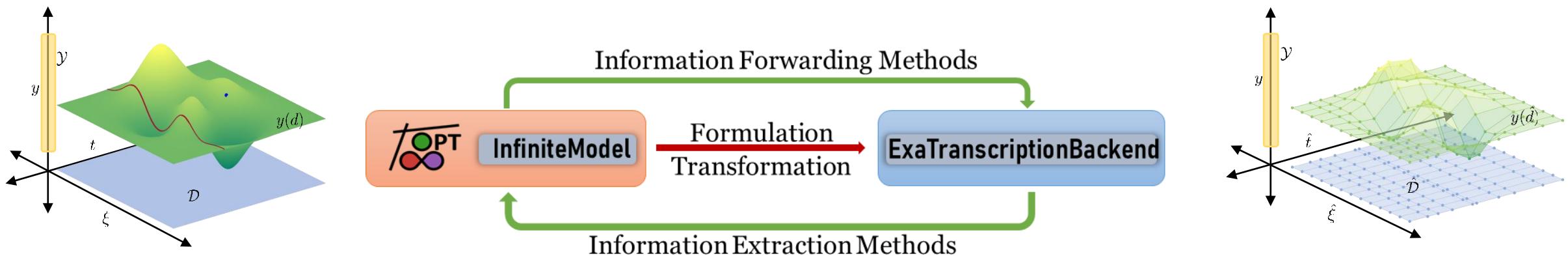
ACCELERATING NLP PERFORMANCE ON CPUS AND GPUS

-  ExaModels uses a single instruction multiple data (SIMD) abstraction
 - Well-suited for optimization problems with **highly repeated structures**
 - Efficiently **compute gradients** for each structure
-  **MadNLP** solves NLPs on GPUs
 - Depends on CUDASS linear solver
 - Obtain solutions **10-20 times faster**
 - Translating InfiniteOpt problems to SIMD is nontrivial



INFINITEEXAMODELS.JL

- Bridges the gap between  InfiniteOpt &  ExaModels
- **Automates transcription** via established transformation interface
- Leverages repeated structure to **drastically reduce model creation time**
 - More efficient than manual transcription directly given to ExaModels



BENCHMARK PROBLEMS

- Compare performance with JuMP, AMPL, ExaModels, and InfiniteExaModels
- Run on CPU with Ipopt and GPU with MadNLP

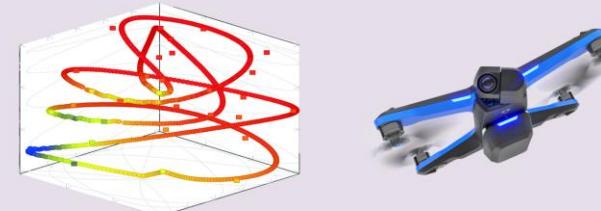
2-Stage Stochastic Program

- Stochastic optimal power flow
- 1,000 to 16,000 random scenarios



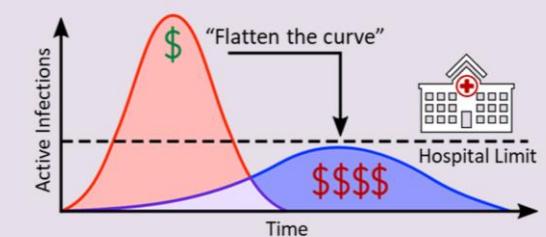
Optimal Control

- Model predictive control of quadcopter
- Track trajectory setpoint and vary grid size



Stochastic Optimal Control

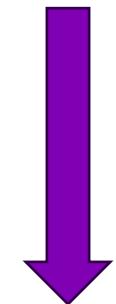
- Control isolation policy to combat disease
- Uncertain transmission rate



NUMERICAL RESULTS (CPU W/ IPOPT)

- AD is **5 – 20 times faster**
- Model build time is **1 – 2 orders-of-magnitude faster**

Approach	Stochastic 2-Stage				Optimal Control				Stochastic Control			
	Build	AD	Solve	Tot.	Build	AD	Solve	Tot.	Build	AD	Solve	Tot.
JuMP.jl	87.1	21.4	63.7	151	143	6.3	28.6	172	10.9	60.7	386	397
AMPL	99.3	11.5	90.3	190	153	5	26.4	179	10.6	23.4	364	375
ExaModelsMOI.jl	86.6	3.05	56.3	143	139	1.1	23.1	162	8.63	3.45	368	376
InfiniteExaModels.jl	1.94	2.03	43	45	9.34	1	22.6	31.9	0.12	3.01	369	369



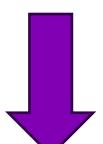
```
1 using InfiniteOpt, InfiniteExaModels, NLPModelsIpopt  
2 model = InfiniteModel(ExaTranscriptionBackend(IpoptSolver))
```

NUMERICAL RESULTS (GPU W/ MADNLP)

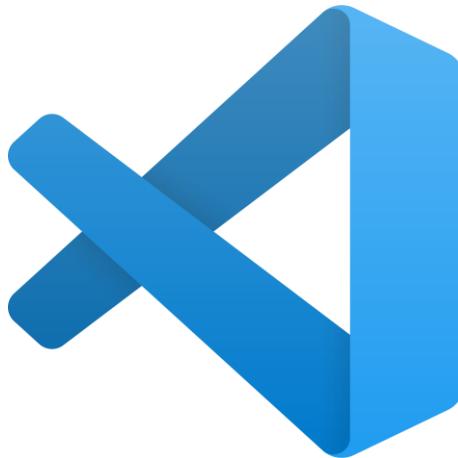
- All AD and solve times are up to **~20 faster on GPU**
- InfiniteExaModels.jl builds models **orders-of-magnitude faster** than ExaModels

Approach	Stochastic 2-Stage				Optimal Control				Stochastic Control			
	Build	AD	Solve	Tot.	Build	AD	Solve	Tot.	Build	AD	Solve	Tot.
ExaModelsMOI.jl	83	0.09	3.15	86.1	133	0.1	6.55	140	8.21	0.51	20.4	28.6
InfiniteExaModels.jl	1.8	0.14	3.03	4.82	8.63	0.1	6.44	15.1	0.06	0.74	21	21

(151 on CPU) **(172 on CPU)** **(397 on CPU)**



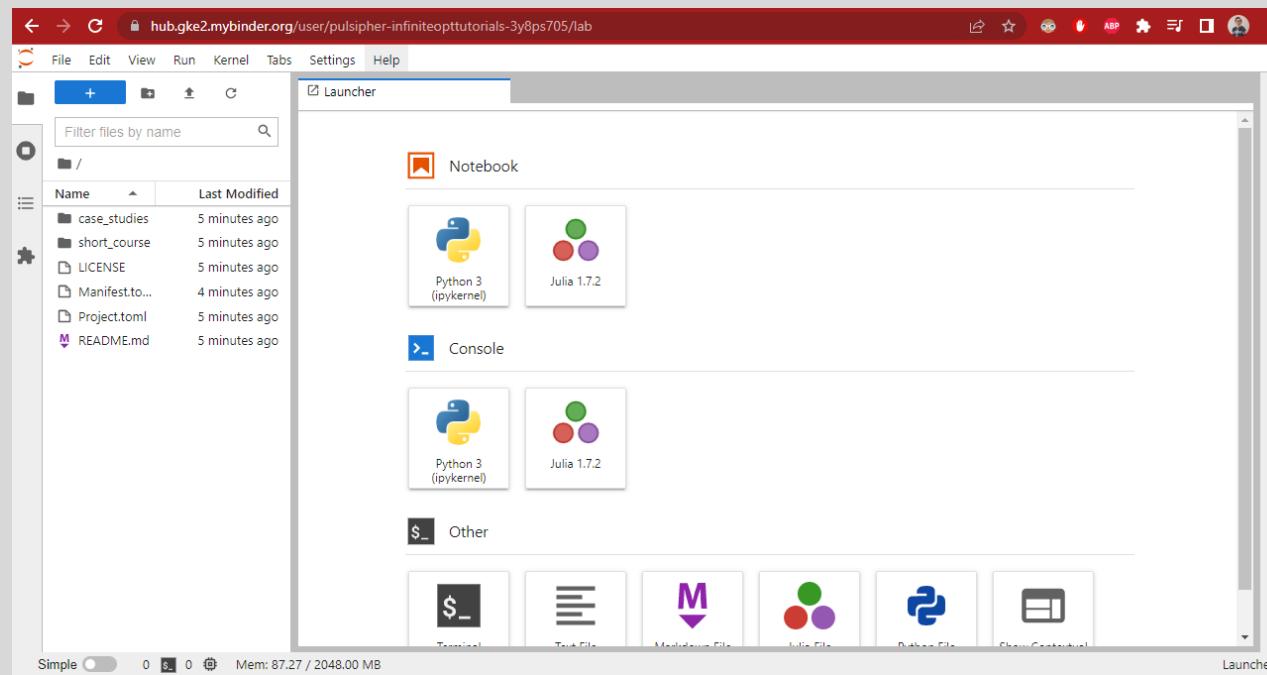
```
1  using InfiniteOpt, InfiniteExaModels, MadNLPGPU, CUDA
2  transform_backend = ExaTranscriptionBackend(MadNLPSolver, backend = CUDABackend())
3  model = InfiniteModel(transform_backend)
```



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 1.10 (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)

Install julia

Install the latest Julia version (v1.10.4 June 4, 2024) from the [Microsoft Store](#) by running this in the command prompt:

```
> winget install julia -s msstore
```

COPY

It looks like you're using Windows. For Linux and MacOS instructions [click here](#)

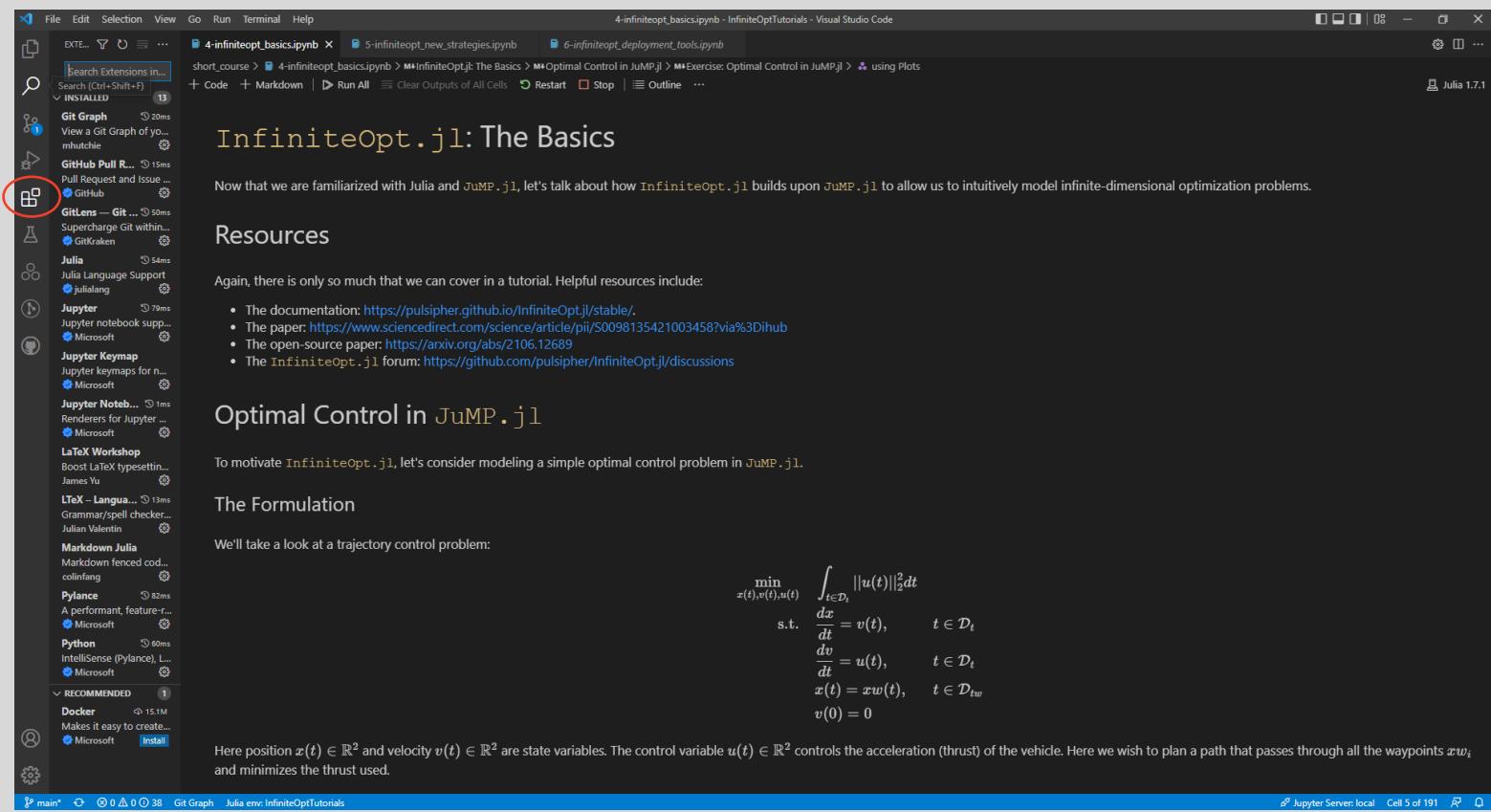
Once installed `julia` will be available via the command line interface.

This will install the `Juliaup` installation manager, which will automatically install julia and help keep it up to date. The command `juliaup` is also installed. To install different julia versions see `juliaup --help`.

Please star us [on GitHub](#). If you use Julia in your research, please [cite us](#). If possible, do consider [sponsoring us](#).

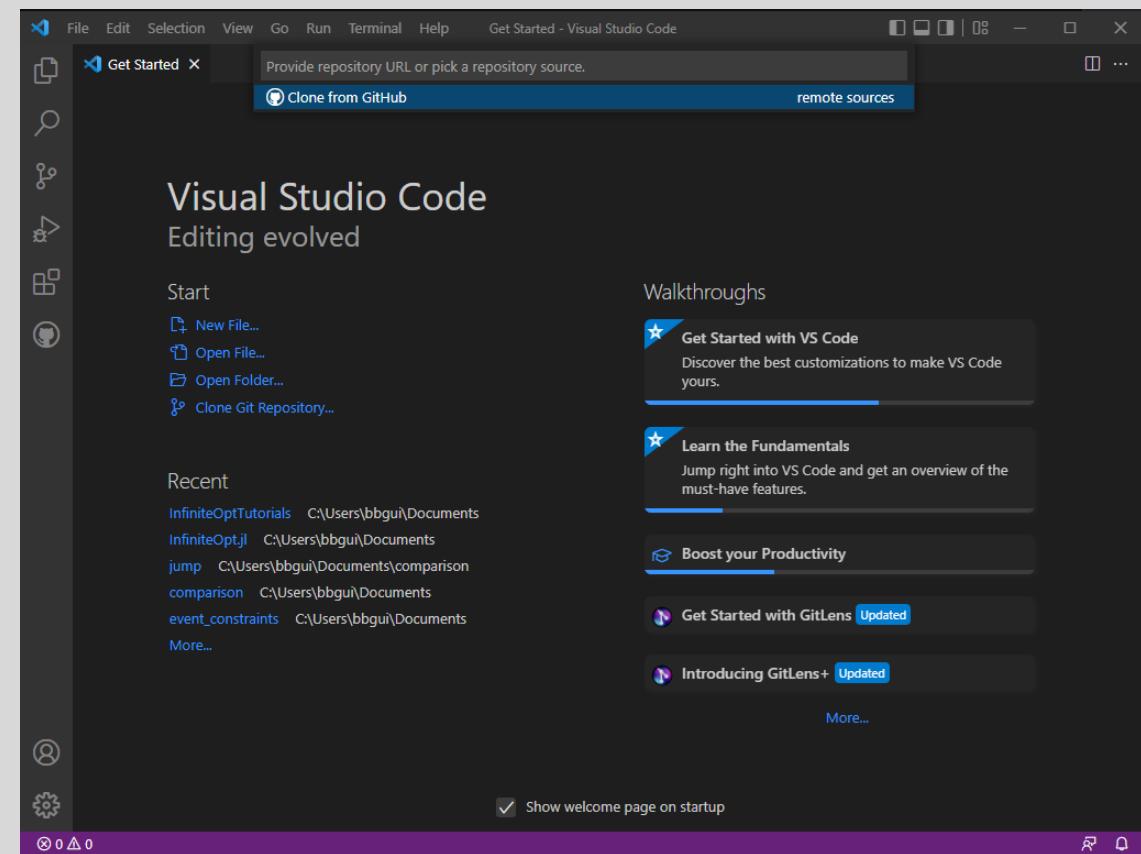
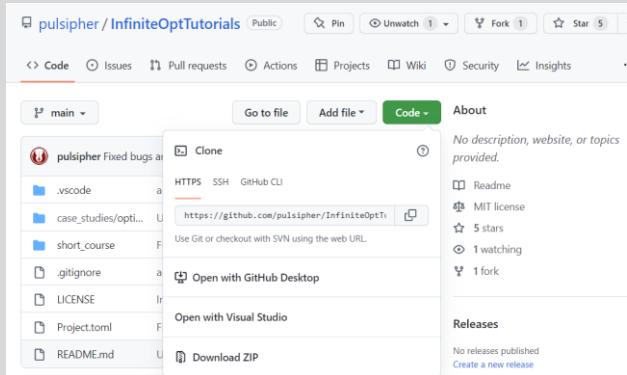
Install Visual Studio Code

- 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
```

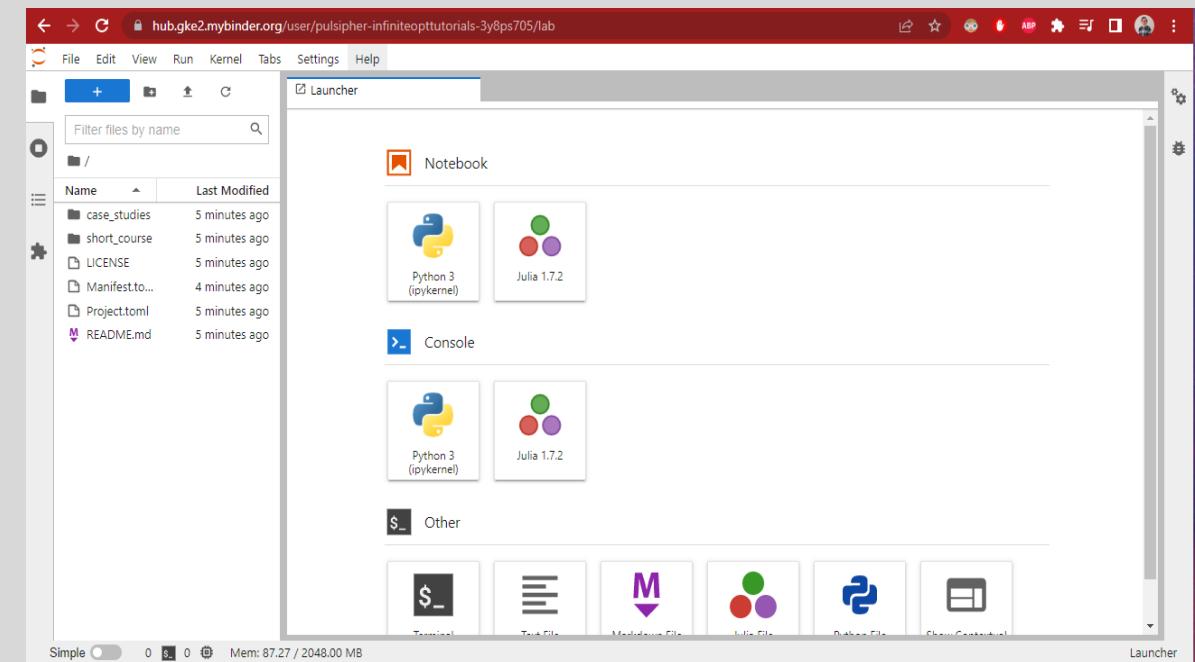
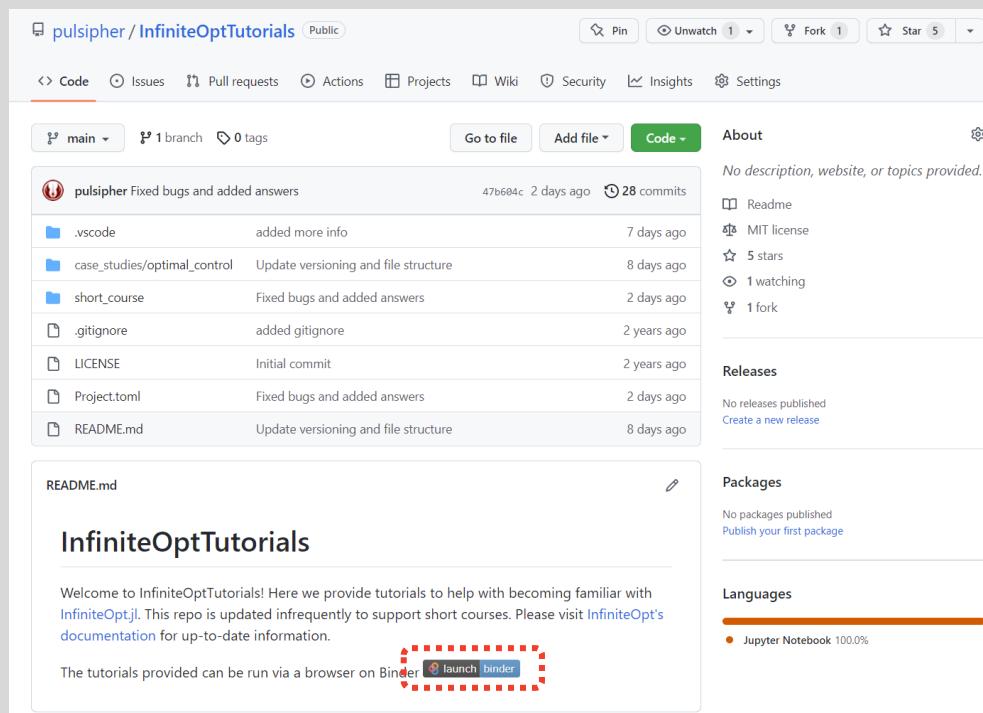
Running Jupyter Notebook

- 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

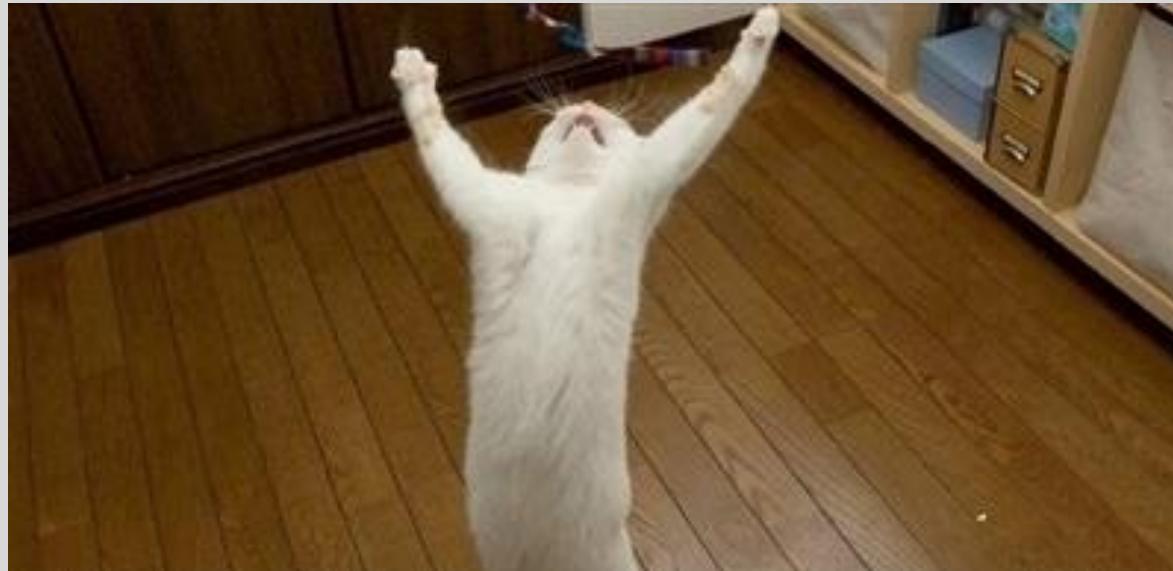
The screenshot shows a Visual Studio Code interface with a dark theme. In the center, a Jupyter Notebook cell displays the title "Julia: A Practical Introduction". Below it, a text block provides a brief tutorial on coding in Julia, mentioning JuMP.jl and InfiniteOpt.jl. At the bottom of the cell, there are two lines of Julia code: `@show 1 + 1` and `@show 1 - 2`. To the left of the notebook, the "EXPLORER" panel is open, showing a file tree for a directory named "INFINITEOPTTUTORIALS". The tree includes files like ".vscode", "case_studies", "short_course", "figures", and several IPython notebook files (e.g., "1-julia_overview.ipynb", "2-jump_overview.ipynb"). Other files visible include "README.md", ".gitignore", "LICENSE", "Manifest.toml", "Project.toml", and "READEME.md". The status bar at the bottom indicates "main*" in the top-left, "Git Graph" in the middle, "Julia env: InfiniteOptTutorials" in the bottom-left, and "Jupyter Server: local Cell 1 of 117" in the bottom-right.

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



15 Minute Break Time

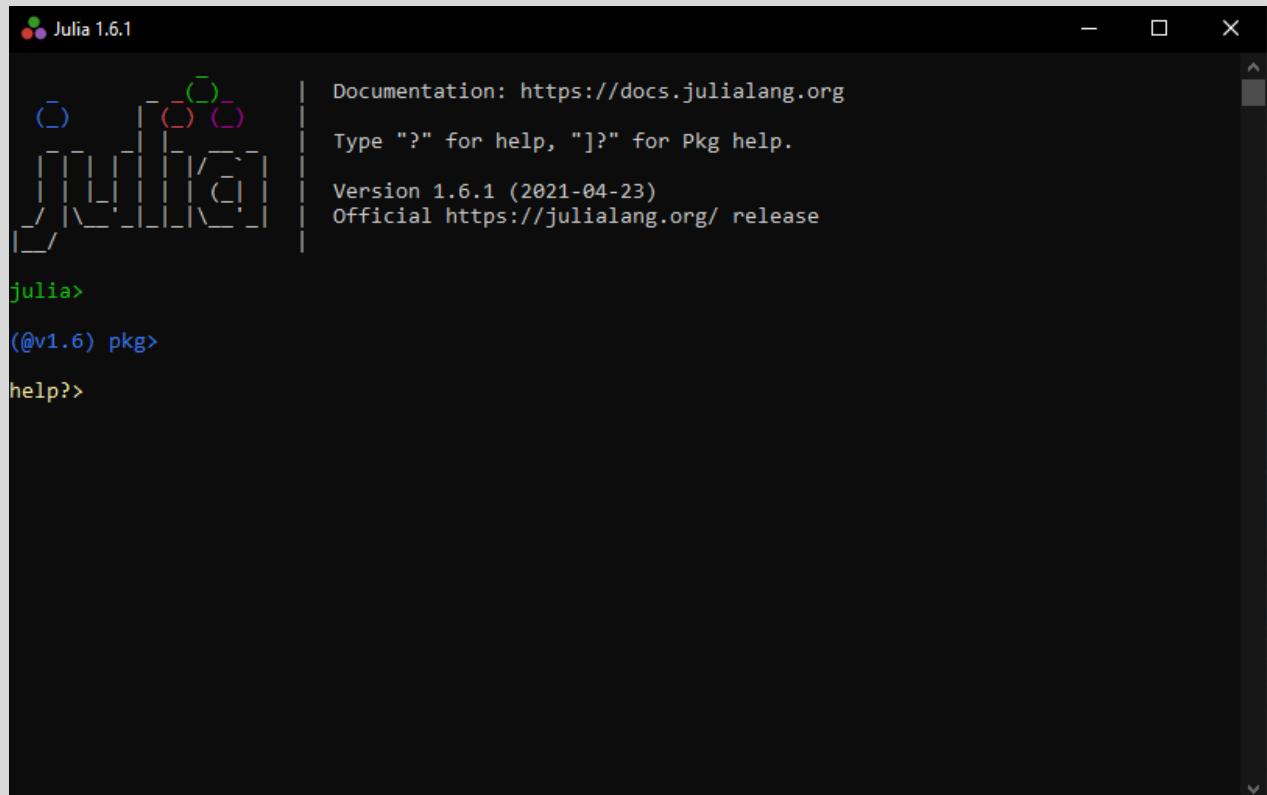




A Practical Introduction

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

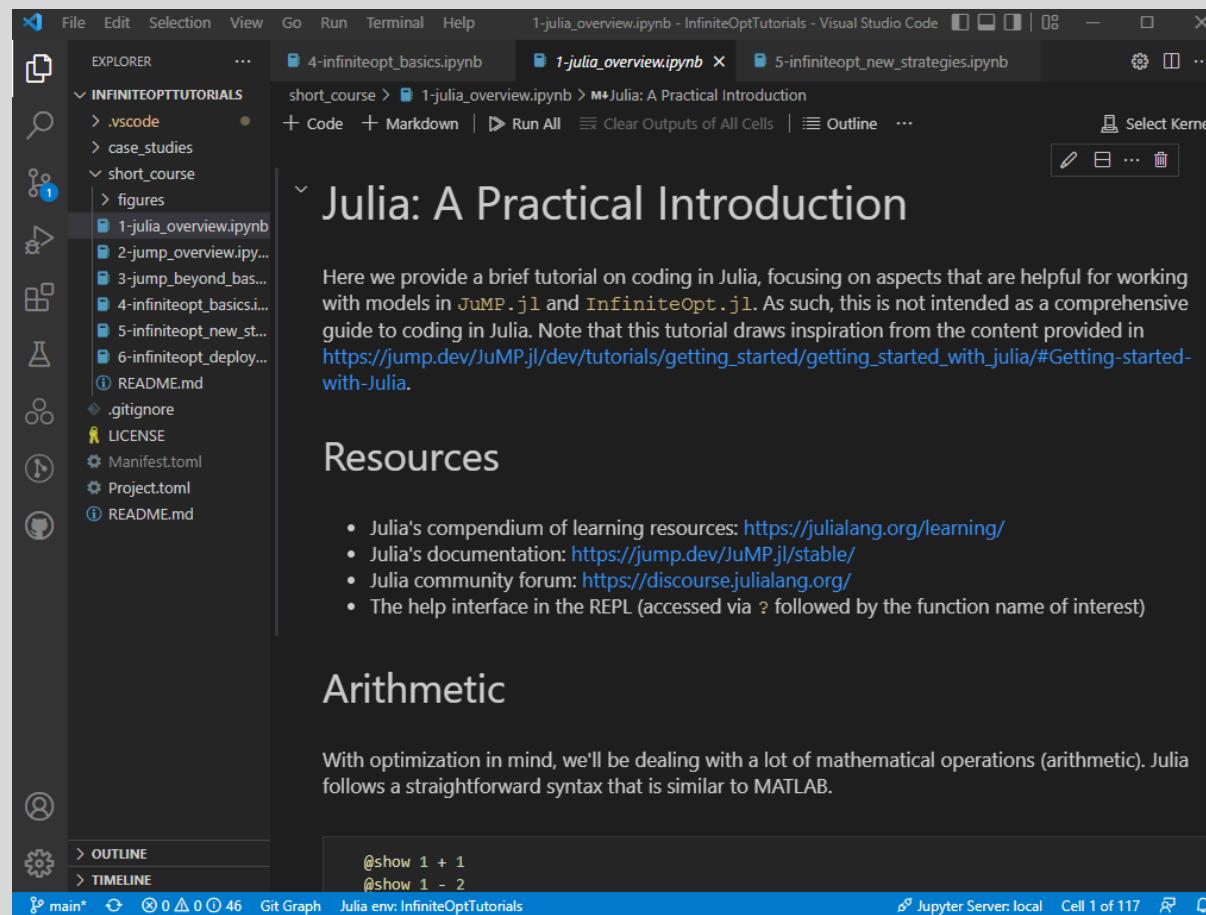


The screenshot shows the Julia 1.6.1 REPL window. The title bar reads "Julia 1.6.1". The window contains the following text:
Documentation: <https://docs.julialang.org>
Type "?" for help, "]?" for Pkg help.
Version 1.6.1 (2021-04-23)
Official <https://julialang.org/> release

julia>
(@v1.6) pkg>
help?>

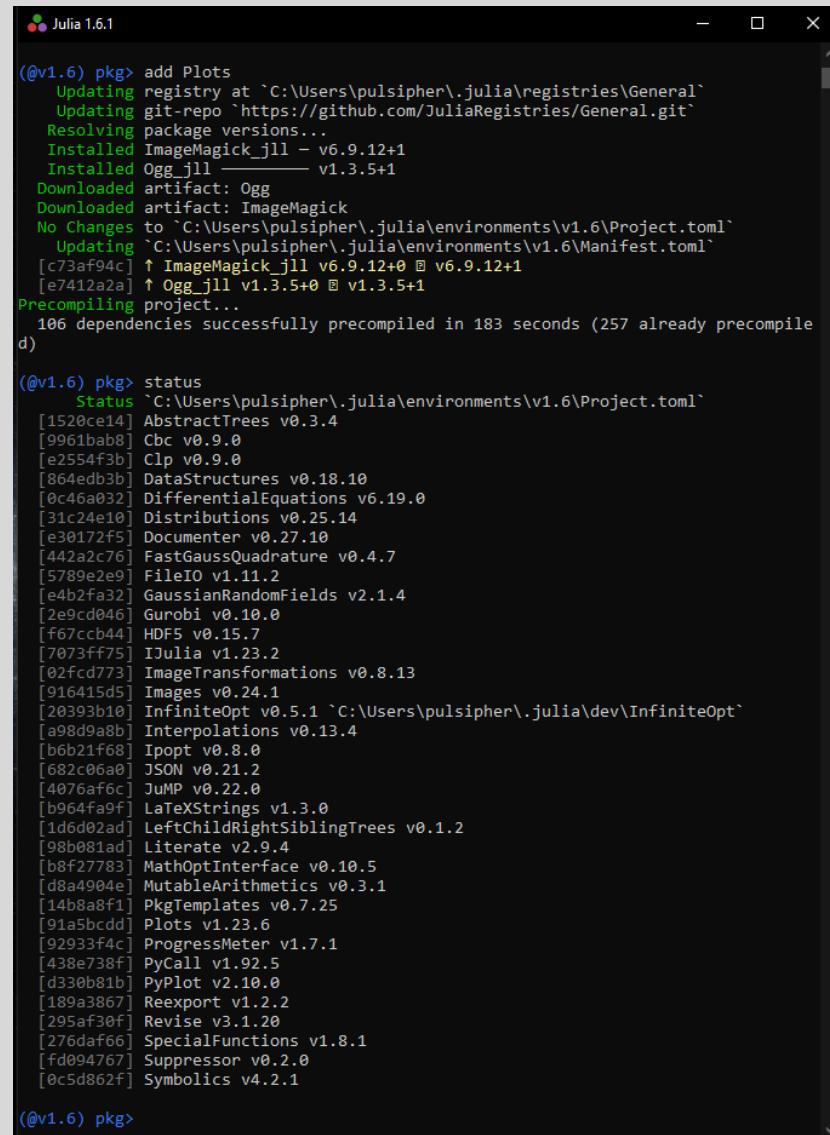
To the Tutorial!

- 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)

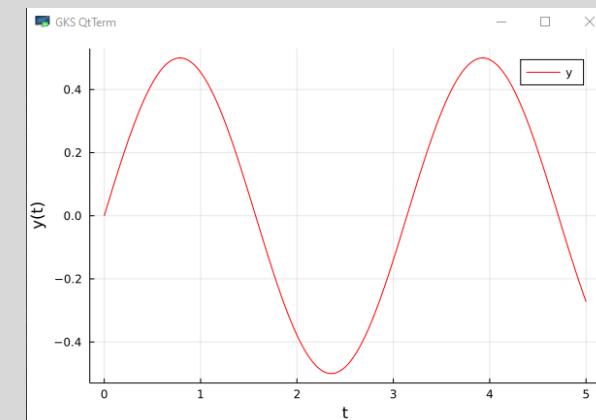
(@v1.6) pkg> status
  Status `C:\Users\pulsipher\.julia\environments\v1.6\Project.toml`...
  [1520ce14] AbstractTrees v0.3.4
  [9961bab8] Cbc v0.9.0
  [e2554f3b] Clp v0.9.0
  [864edb3b] DataStructures v0.18.10
  [0c46a032] DifferentialEquations v6.19.0
  [31c24e10] Distributions v0.25.14
  [e30172f5] 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

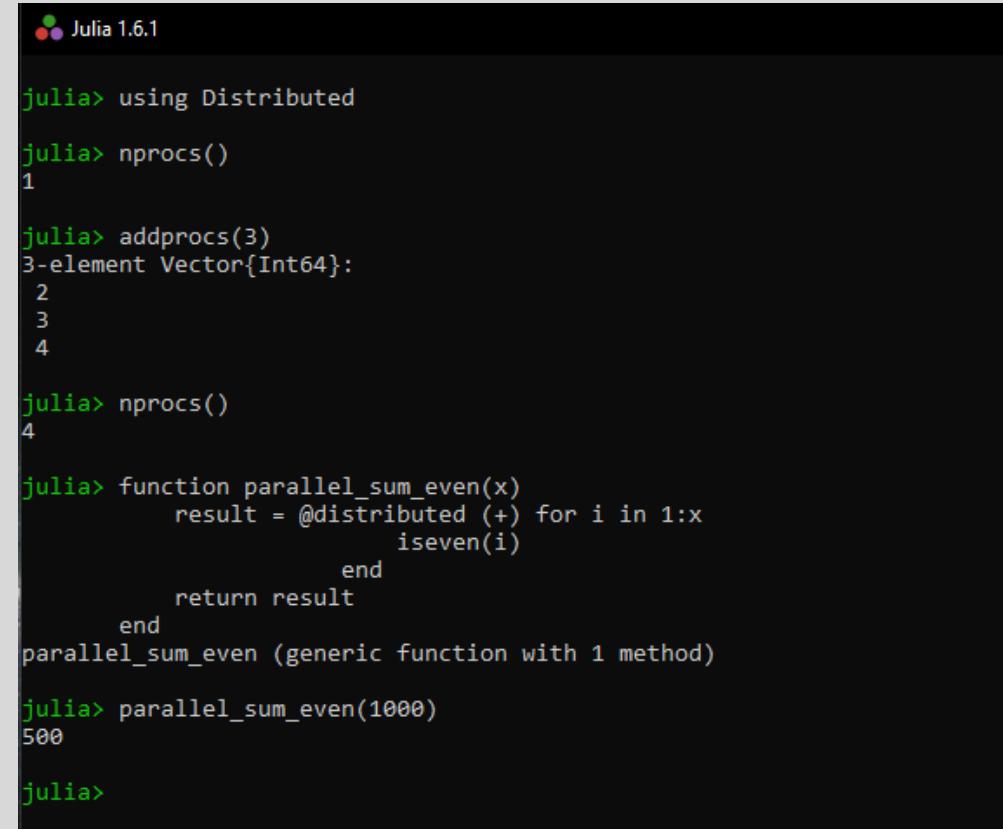
- Plots.jl
 - Efficient go to
 - The documentation is terrible 😞
- PyPlot.jl
 - Simply maps to matplotlib.pyplot
 - Imports all the all the pyplot functions
- Makie.jl
 - Native to Julia
 - Good documentation
 - Can be slow sometimes
- Others
 - GadFly.jl (pretty good 2D library)
 - PGFPlots.jl (plots 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 loaded the `Distributed` package and checked the number of available parallel processes. They then added three new processes using `addprocs(3)`, which returned a vector of process IDs: 2, 3, and 4. After restarting the session with `nprocs()` again, they defined a function `parallel_sum_even(x)` that uses `@distributed` to sum even integers from 1 to x . Finally, they called this function with `parallel_sum_even(1000)` and received the result 500.

```
Julia 1.6.1

julia> using Distributed
julia> nprocs()
1

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

julia> nprocs()
4

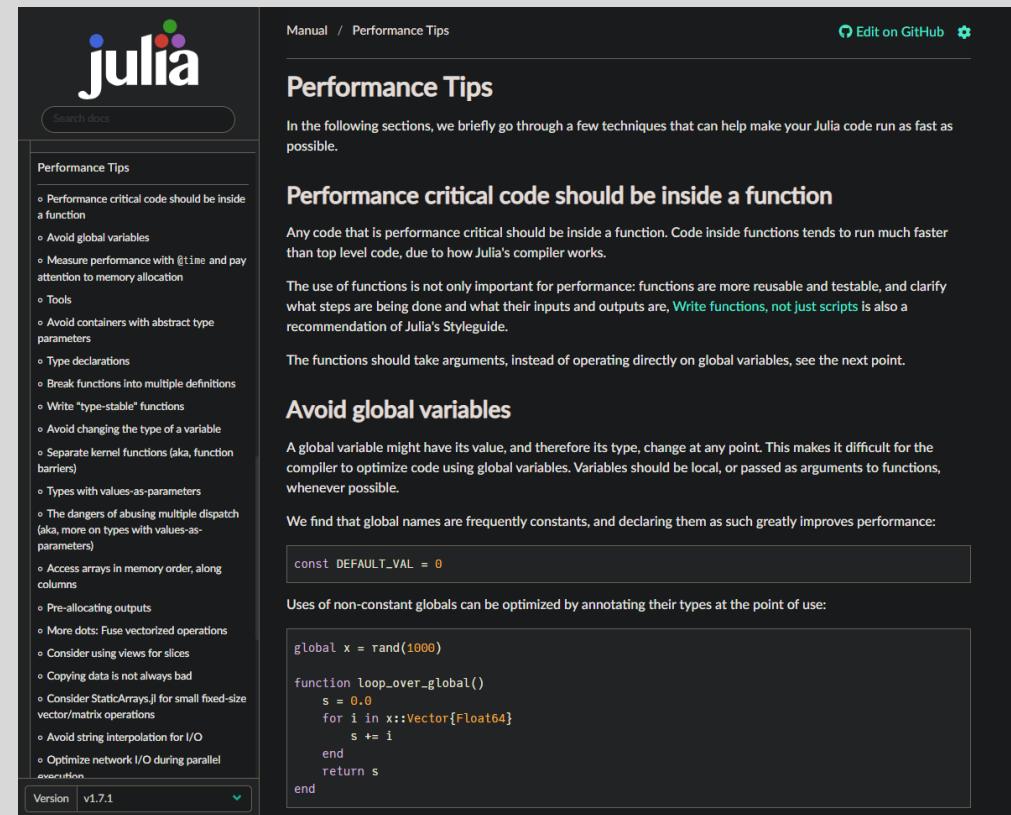
julia> function parallel_sum_even(x)
           result = @distributed (+) for i in 1:x
                     iseven(i)
                 end
           return result
       end
parallel_sum_even (generic function with 1 method)

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 version of the Julia documentation. At the top, there's a navigation bar with 'Manual' and 'Performance Tips'. On the right, there are links to 'Edit on GitHub' and a gear icon. The main content area has a heading 'Performance Tips' with a sub-section 'Performance critical code should be inside a function'. It discusses why functions run faster and provides examples of good and bad practices. Below that is a section 'Avoid global variables' with a note about compiler optimization. A code block shows how to declare a constant. The bottom part of the screenshot shows a code editor with some Julia code, including a function definition and a return statement.

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:

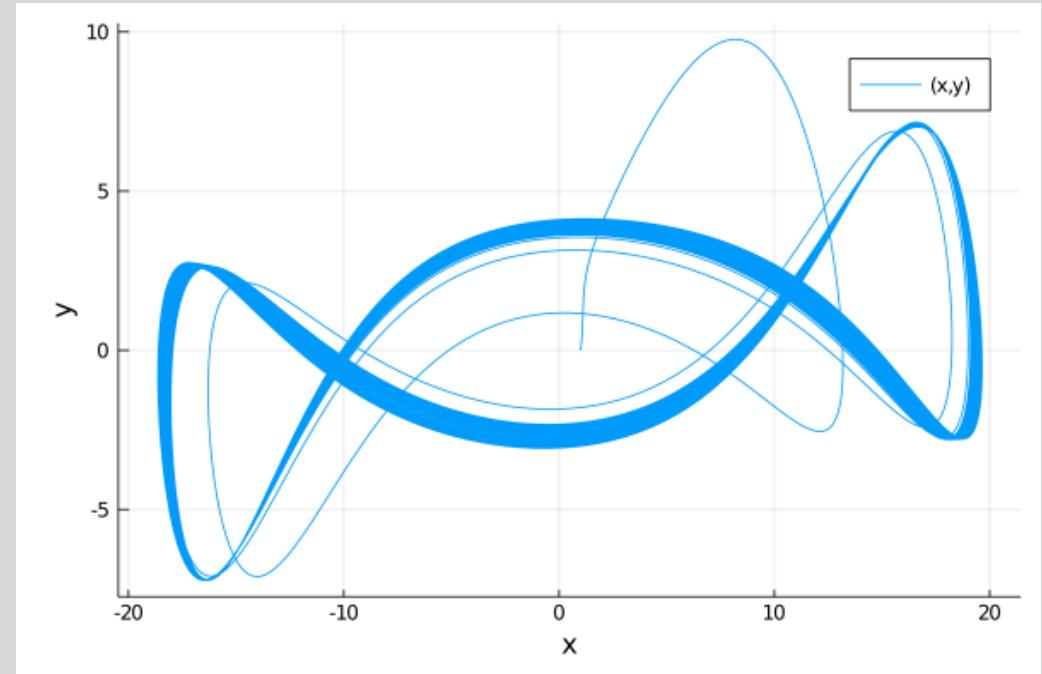
```
global x = rand(1000)

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.

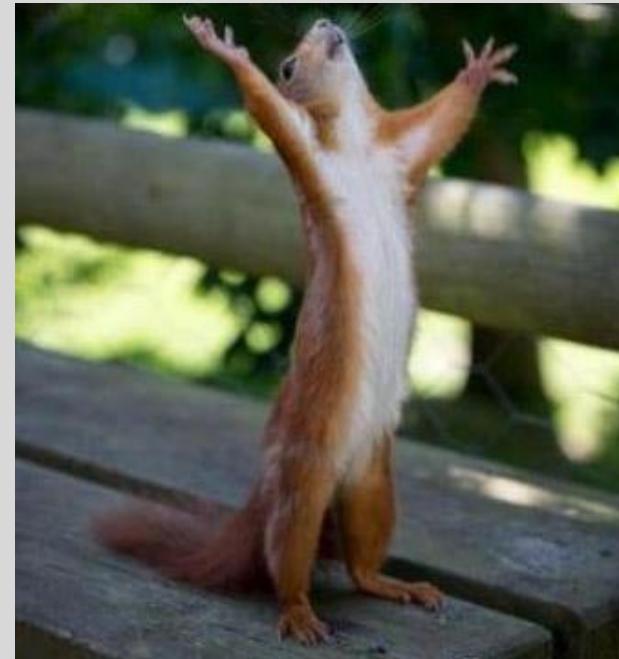
The main content area is divided into several sections:

- 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**. A list of domains follows:
 - Statistics
 - Numerics
 - GPU
 - Biology, Health, and Medicine
 - Data
 - Visualization
 - Optimization (Mathematical)
 - Machine Learning
 - Modelling & Simulation
 - Signal Processing
 - Web Stack
 - AstroSpace
 - Finance and Economics
 - Julia at Scale
 - Probabilistic programming
 - Maker
 - Quantum
 - Chemistry
 - 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)

15 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**

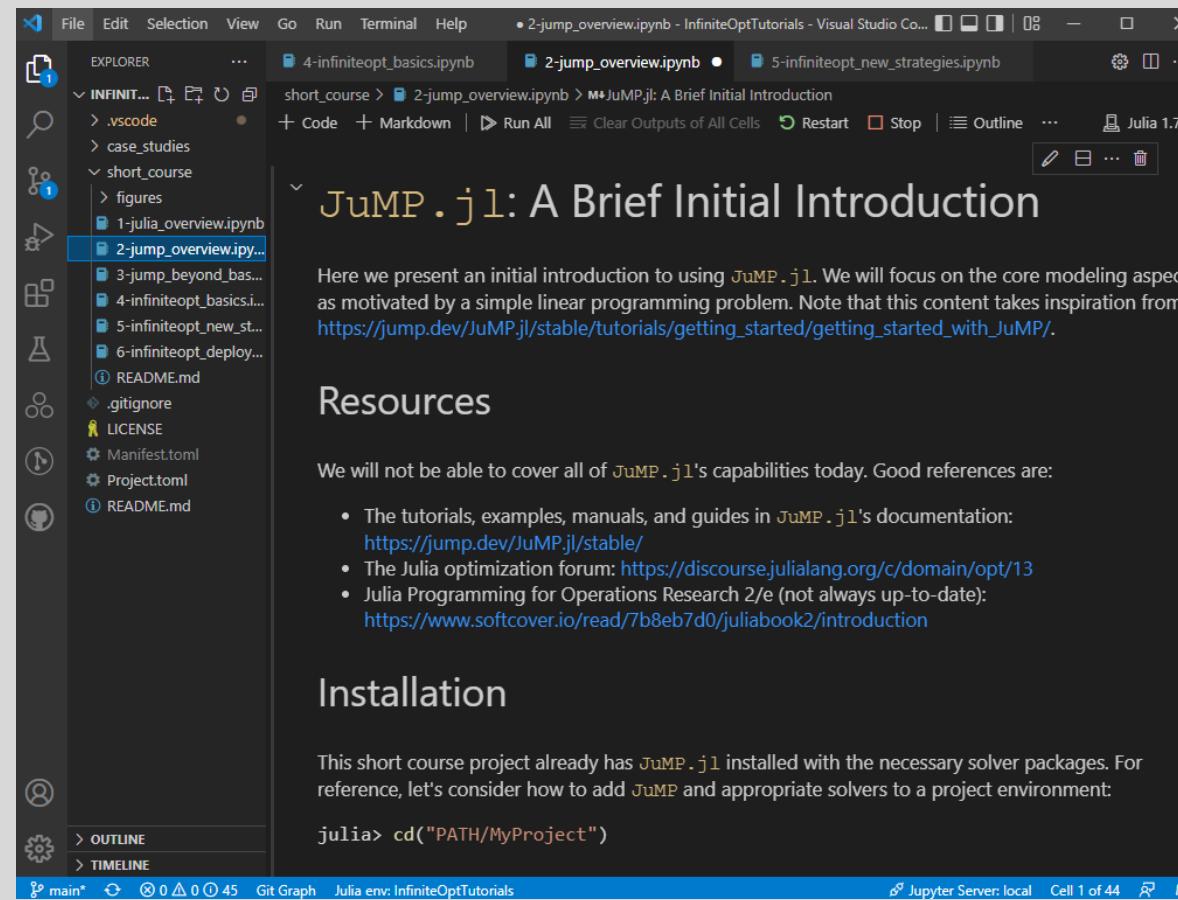
```
6 @constraint(model, 4f + 2s <= 4800)
7 @constraint(model, f + s <= 1750)
8 @constraint(model, 0 <= f <= 1000)
9 @constraint(model, 0 <= s <= 1500)
```

- **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

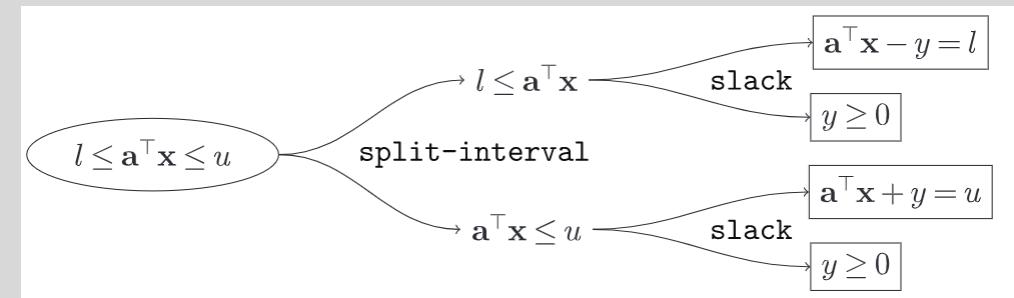
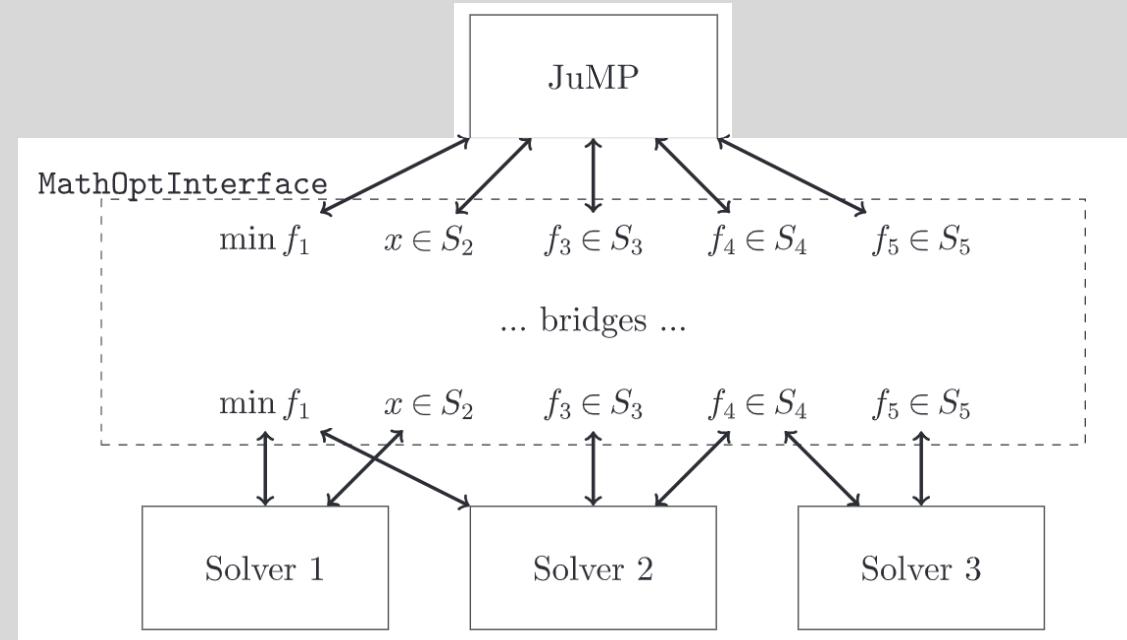




Beyond the basics

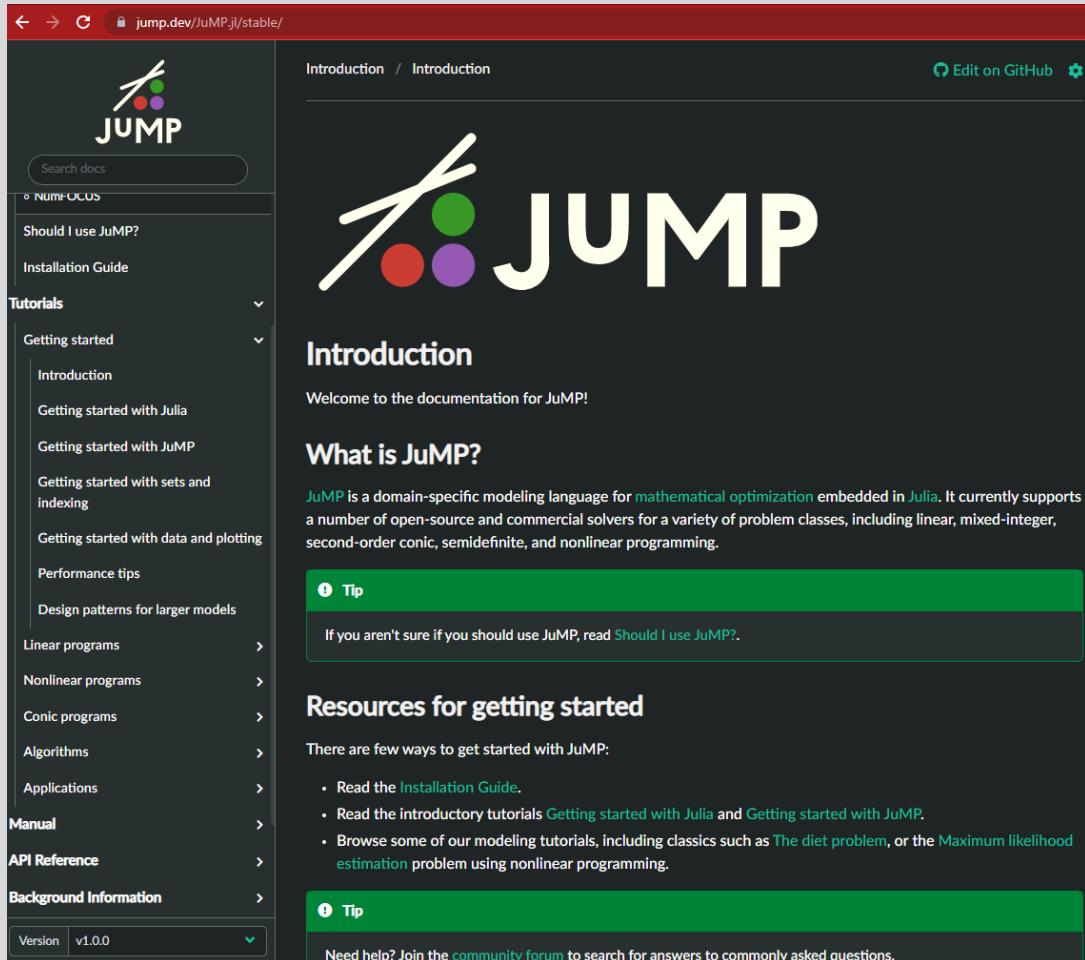
JuMP.jl Architecture

- 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



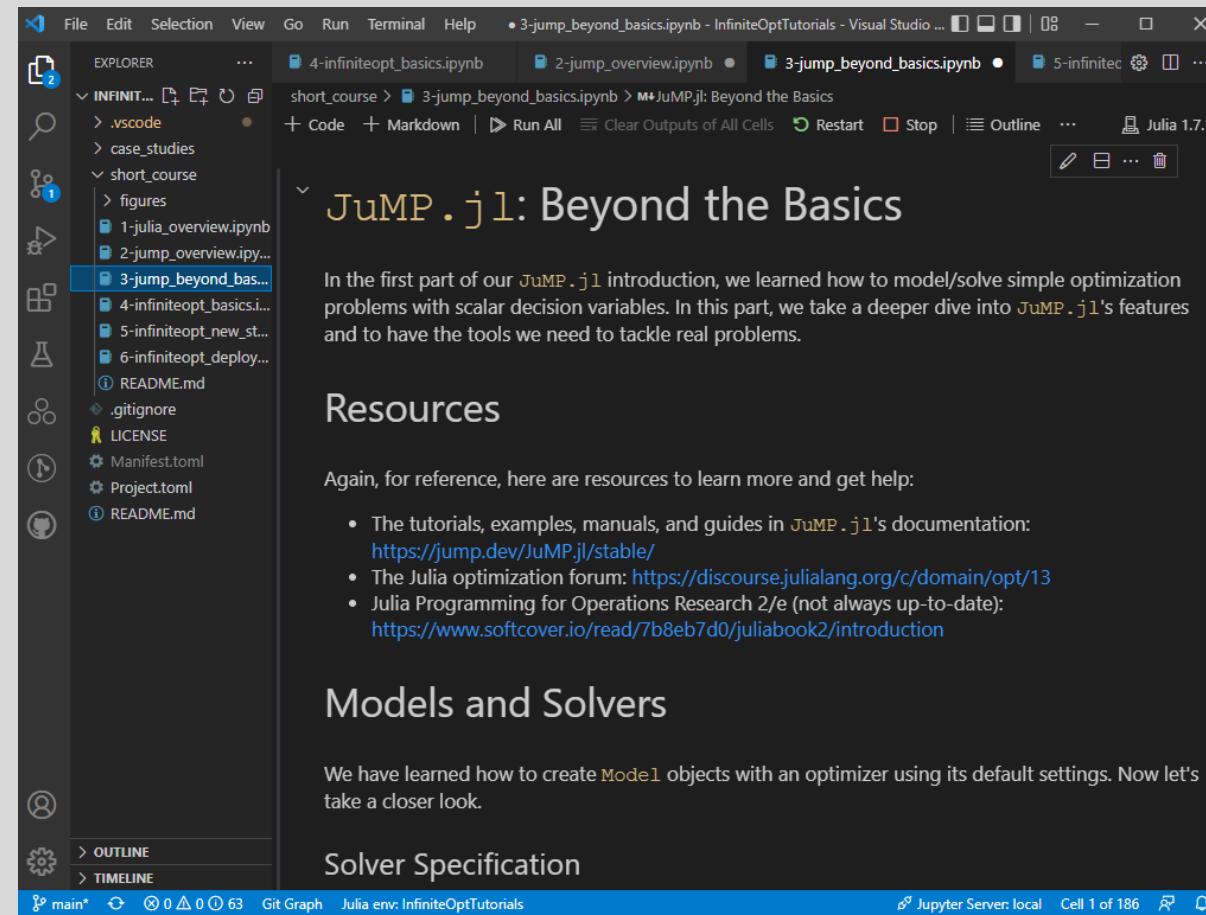
The screenshot shows the JuMP documentation website at jump.dev/JuMP.jl/stable/. The page has a dark theme with a large JuMP logo at the top. The left sidebar contains a navigation menu with sections like NumFOCUS, Should I use JuMP?, Installation Guide, Tutorials (with sub-sections for Getting started, 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 with 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."

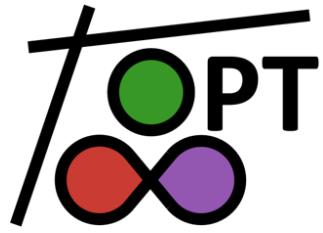
To the Tutorial!

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



15 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**

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

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

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

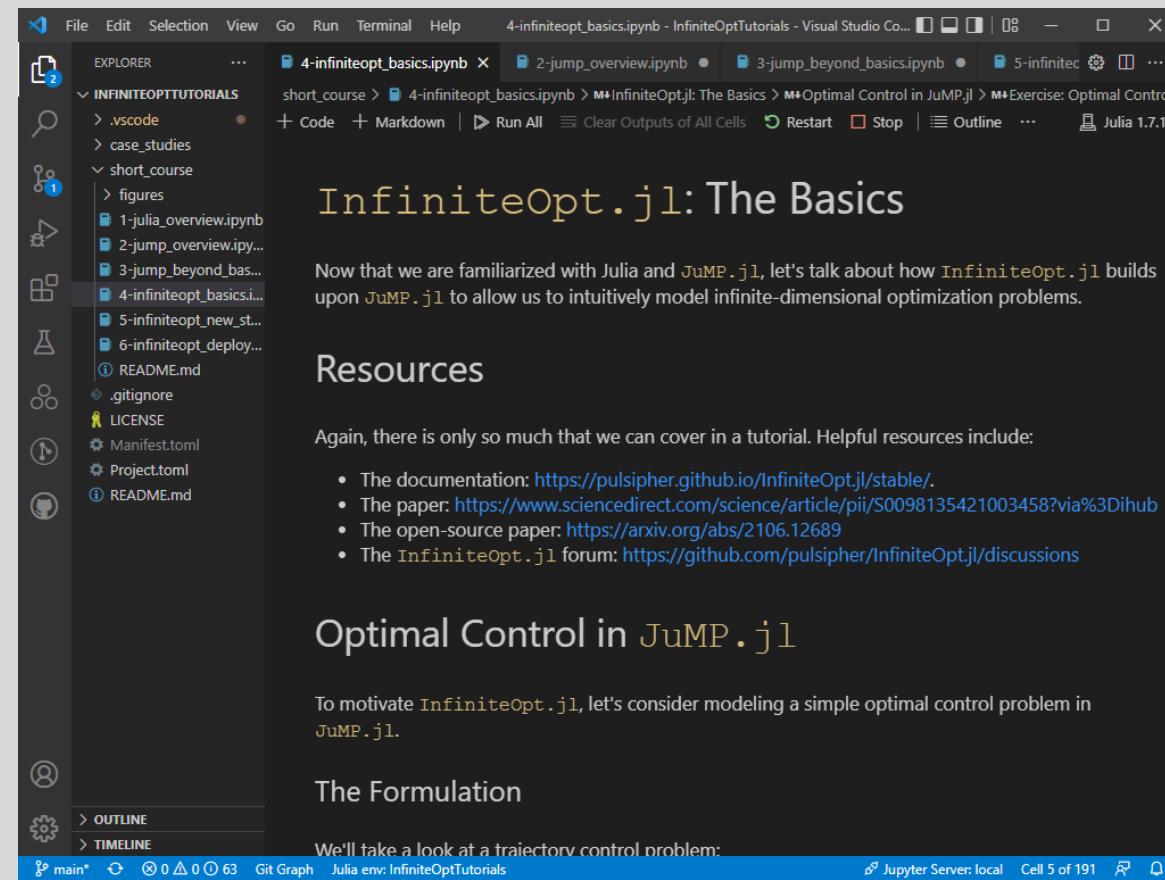
```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, yb ≤ yc * U)
@constraint(m, E(yc, ξ) ≥ α)
@constraint(m, ya(0) + z[2] == β)
```

- **Solve**

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

To the Tutorial!

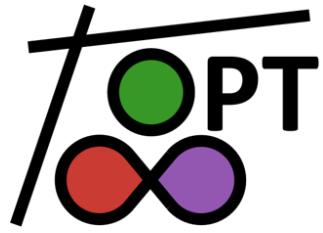
- Open the “4-infiniteopt_basics.ipynb” jupyter notebook



15 Minute Break Time

Why is
InfiniteOpt.jl
so awesome?



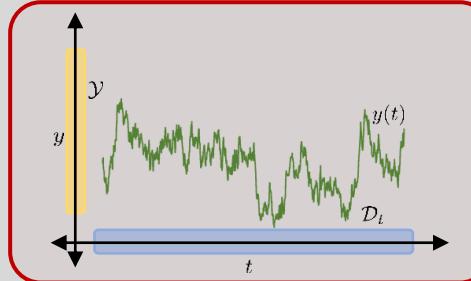


InfiniteOpt

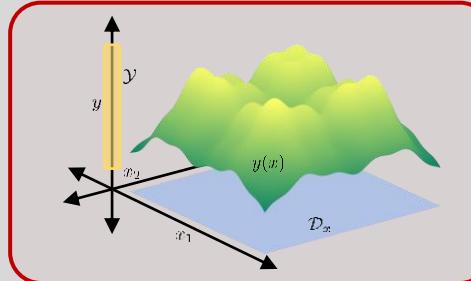
New Modeling Strategies

Innovating with InfiniteOpt.jl

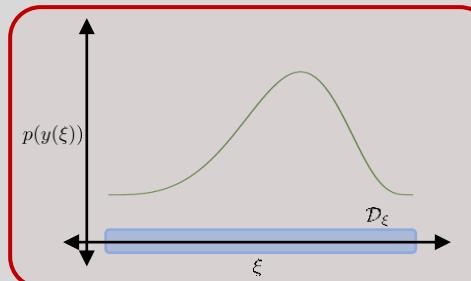
Traditional Formulations



Dynamic Optimization



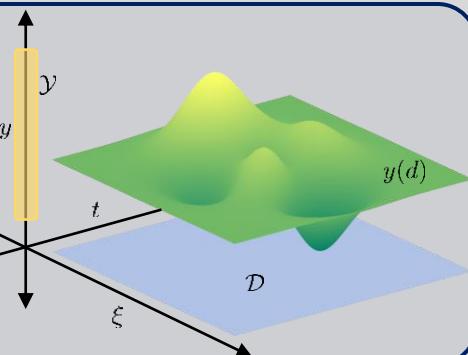
PDE-Constrained Optimization



Stochastic Optimization

InfiniteOpt

Unifying Abstraction

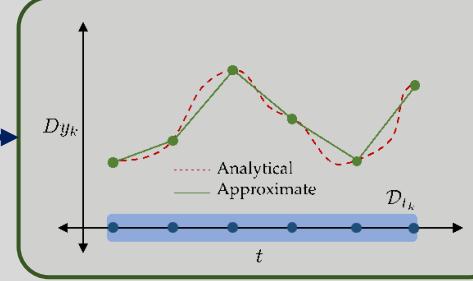


Infinite-Dimensional Optimization

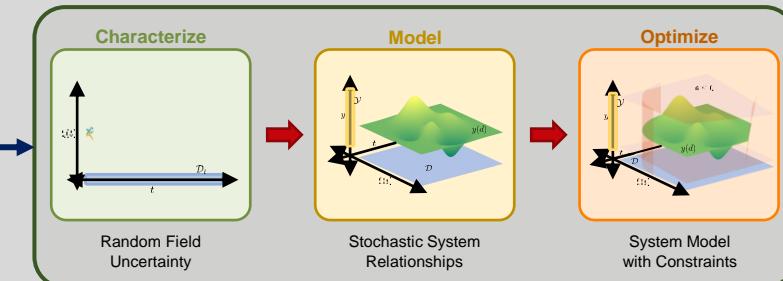
New Formulations



Generalized Shaping Measures



Continuous Time Estimation

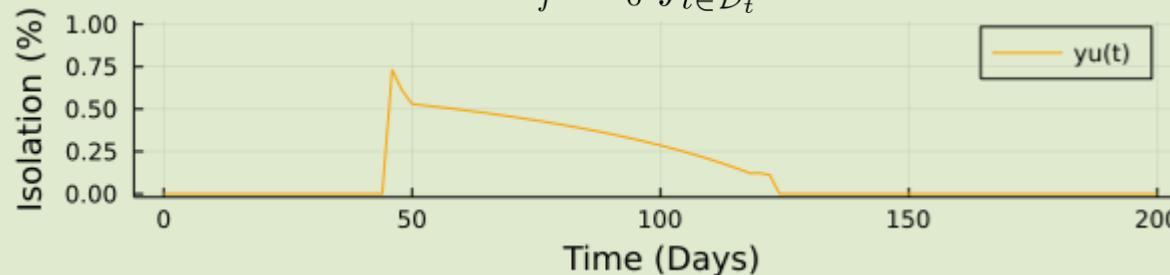


Random Field Optimization

Expectation (Uniform pdf)

- Weight the cost **uniformly**

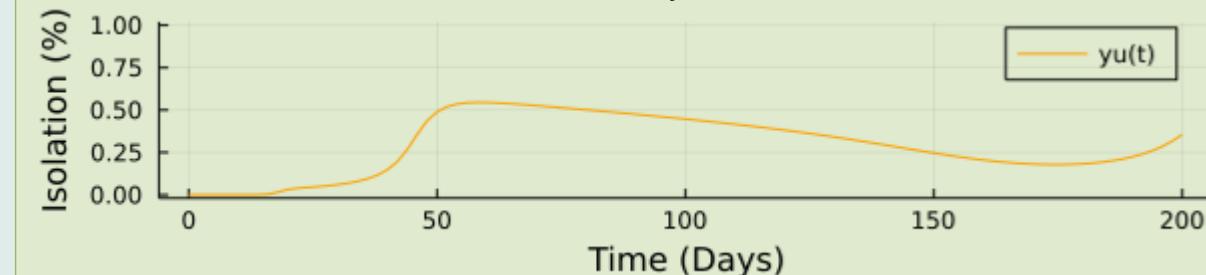
$$\mathbb{E}_t[y_u(t)] = \frac{1}{t_f - t_0} \int_{t \in \mathcal{D}_t} y_u(t) dt$$



Expectation (Exponential pdf)

- Place emphasis on the **initial times**

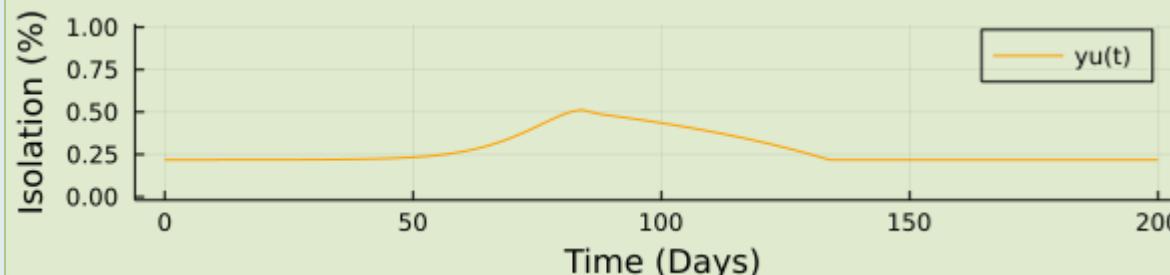
$$\mathbb{E}_t[y_u(t)] = \int_{t \in \mathcal{D}_t} y_u(t) e^{-t} dt$$



Mean-Variance

- Penalize **fluctuations** in the trajectory

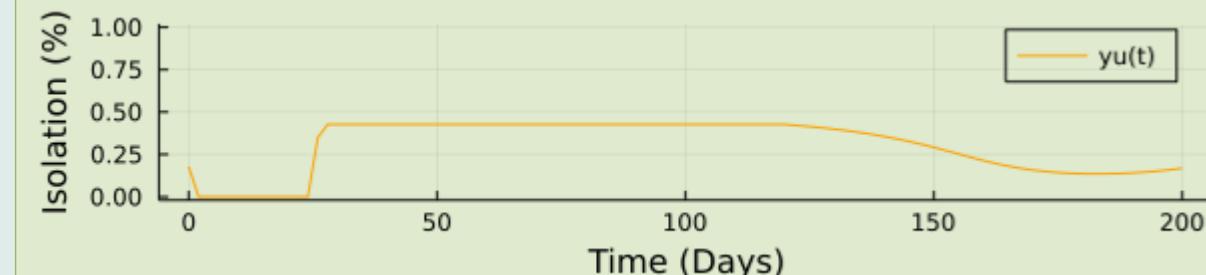
$$\text{EV}_t = \mathbb{E}_t[y_u(t)] + \lambda \mathbb{V}_t[y_u(t)]$$



CVaR

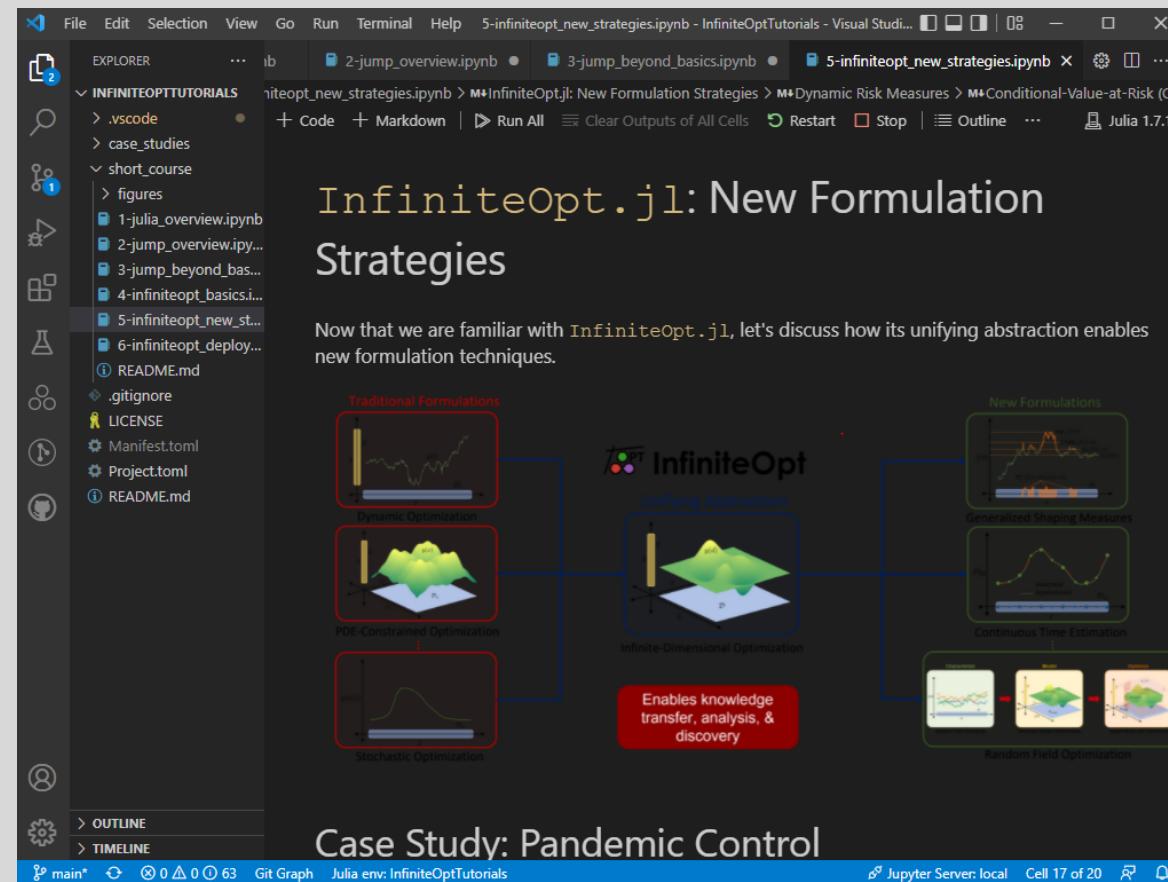
- Only penalize the **high cost peaks**

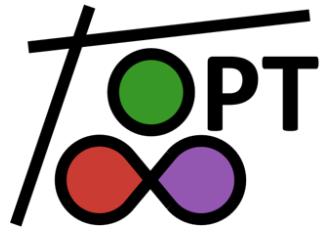
$$\text{CVaR}_t(y_u(t); \alpha)$$



To the Tutorial!

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



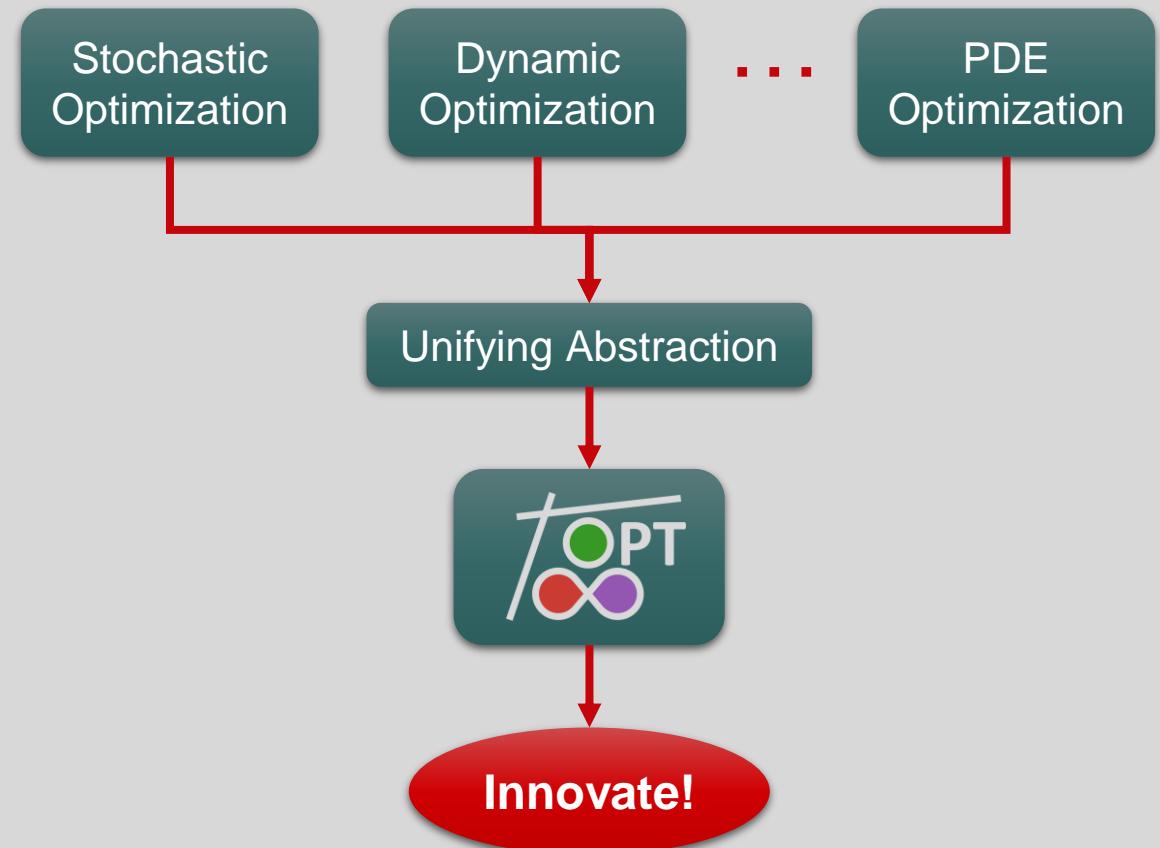


InfiniteOpt

Deployment Tools

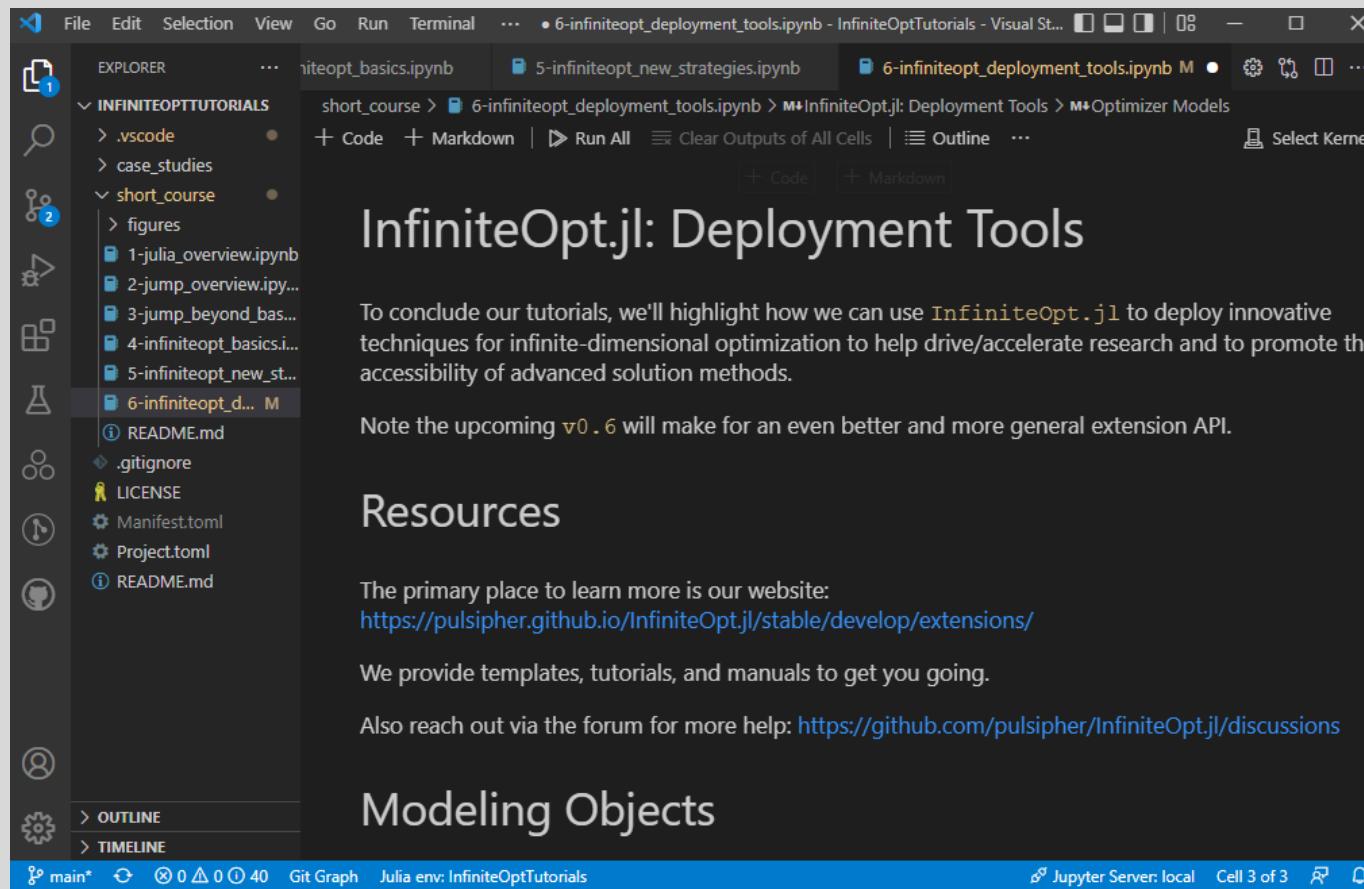
Want to deploy your cool techniques?

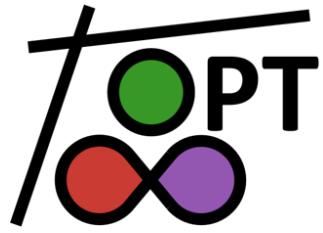
- Almost all modeling aspects can be extended
 - **Infinite domain**
 - **Support generation**
 - **Measure** representation/evaluation
 - **Derivative** evaluation methods
 - Model **transformation**
 - More!
- Extensive **documentation, tutorials, & templates**



To the Tutorial!

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



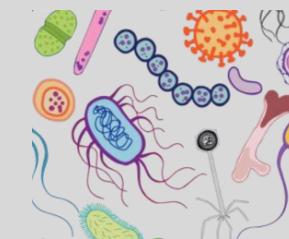
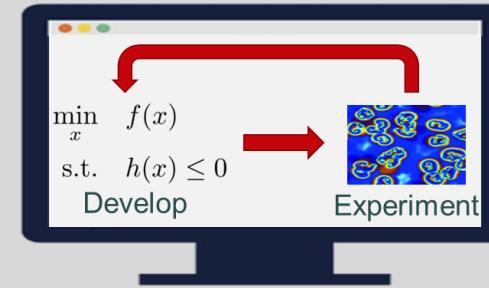


InfiniteOpt

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
- Acceleration
 - Solve problems faster on CPU/GPU via InfiniteExaModels.jl
- Application
 - Offline/online decision making (e.g., optimal control)
 - Any InfiniteOpt problem class

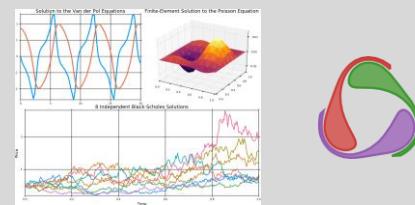


Future Development: Modeling Objects

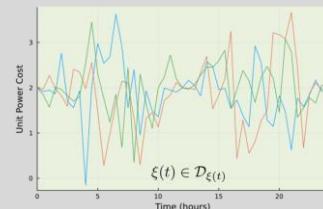
- Non-rectangular domains
 - Leverage packages like Gridap.jl



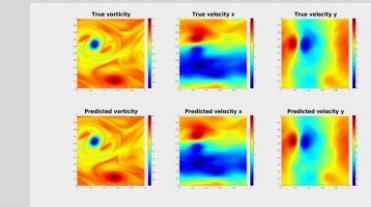
- Optimal Control Toolbox
 - Automatic initialization via DifferentialEquations.jl
 - Simulation



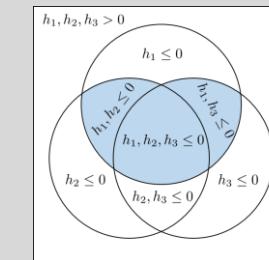
- Infinite Parameter Functions
 - Embed random fields uncertainty



- Infinite-Dimensional Surrogates
 - Embed neural operator models

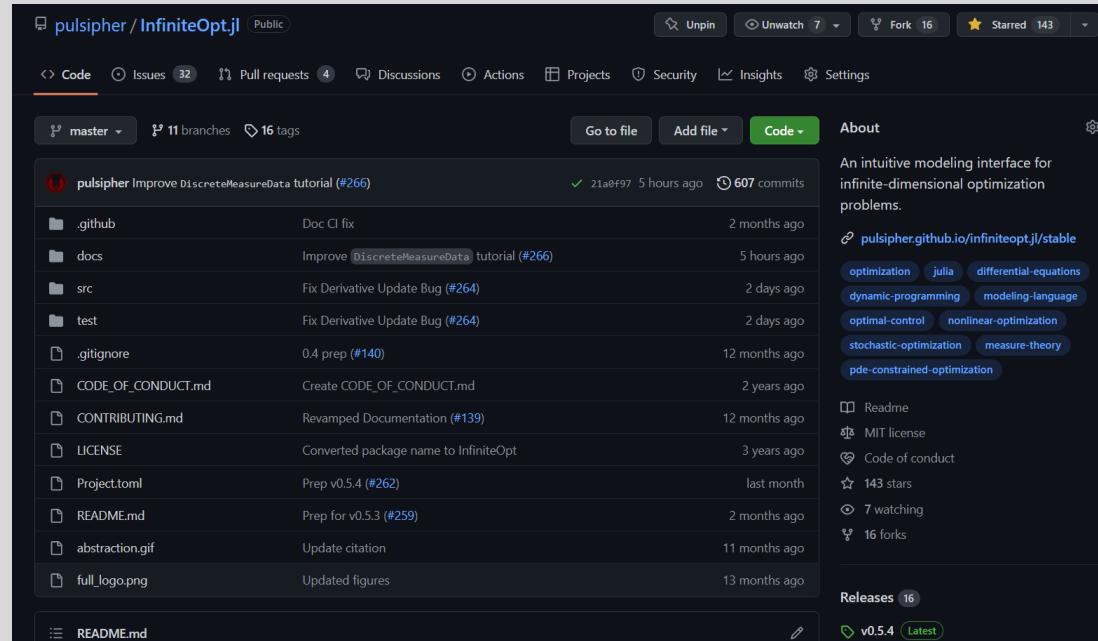


- Event Constraints
 - Generalization of chance constraints
 - Connection to generalized disjunctive programming



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!

