



A unifying modeling abstraction for infinite-dimensional optimization

Joshua L. Pulsipher, Weiqi Zhang, Tyler J. Hongisto, Victor M. Zavala*

Department of Chemical and Biological Engineering, University of Wisconsin-Madison, 1415 Engineering Dr, Madison, WI 53706, USA

ARTICLE INFO

Article history:

Received 21 June 2021

Revised 23 September 2021

Accepted 2 October 2021

Available online 6 October 2021

Keywords:

Infinite-dimensional

Optimization

Measures

Space-time

Random

ABSTRACT

Infinite-dimensional optimization (InfiniteOpt) problems involve modeling components (variables, objectives, and constraints) that are functions defined over infinite-dimensional domains. Examples include continuous-time dynamic optimization (time is an infinite domain and components are functions of time), PDE optimization problems (space and time are infinite domains and components are functions of space-time), as well as stochastic and semi-infinite optimization (random space is an infinite domain and components are a function of such random space). InfiniteOpt problems also arise from combinations of these problem classes (e.g., stochastic PDE optimization). Given the infinite-dimensional nature of objectives and constraints, one often needs to define appropriate quantities (measures) to properly pose the problem. Moreover, InfiniteOpt problems often need to be transformed into a finite dimensional representation so that they can be solved numerically. In this work, we present a unifying abstraction that facilitates the modeling, analysis, and solution of InfiniteOpt problems. The proposed abstraction enables a general treatment of infinite-dimensional domains and provides a measure-centric paradigm to handle associated variables, objectives, and constraints. This abstraction allows us to transfer techniques across disciplines and with this identify new, interesting, and useful modeling paradigms (e.g., event constraints and risk measures defined over time domains). Our abstraction serves as the backbone of an intuitive Julia-based modeling package that we call `InfiniteOpt.jl`. We demonstrate the developments using diverse case studies arising in engineering.

© 2021 Elsevier Ltd. All rights reserved.

1. Introduction

Infinite-dimensional optimization (InfiniteOpt) problems contain parameters that live in infinite-dimensional domains (e.g., time, space, random) (Devolder et al., 2010); the components of these problems (variables, objectives, and constraints) are parameterized over these domains and thus are functions with infinite-dimensional domains (they form manifolds and surfaces). A classical example of an InfiniteOpt problem is continuous-time dynamic optimization (Bertsekas et al., 1995); here, the control trajectory is a function of time and time is a parameter that lives in an infinite-dimensional (continuous) domain. This formulation contrasts with that of a discrete-time dynamic optimization problem, in which the control trajectory is a collection of values defined over a finite set of times (domains are finite). Given the infinite-dimensional nature of variables, objectives, and constraints, one requires specialized techniques to define an InfiniteOpt problem properly. This is done by using *measures*, which are operators that summarize/collapse an infinite-dimensional object into a

scalar quantity. For instance, in dynamic optimization, one often minimizes the integral of the cost over the time domain and, in stochastic optimization, one often minimizes the expected value or variance of the cost. Measures are thus a key modeling element of InfiniteOpt problems that help manipulate the *shape* of infinite-dimensional objects to achieve desired outcomes (e.g., minimize peak/extreme costs or satisfy constraints with high probability). InfiniteOpt problems also often contain differential operators that dictate how components evolve over their corresponding domains; these operators often appear in problems that include differential and algebraic equations (DAEs) and partial differential equations (PDEs).

InfiniteOpt problems encompass a wide range of classical fields such as dynamic optimization (Biegler, 2007), PDE optimization (Hinze et al., 2008), stochastic optimization (Birge and Louveaux, 2011), and semi-infinite optimization (Stein and Still, 2003). One also often encounters InfiniteOpt problems that are obtained by combining infinite-dimensional domains (e.g., space, time, and random domains). This situation arises, for instance, in stochastic dynamic optimization (e.g., stochastic optimal control) problems and in optimization problems with stochastic PDEs. In these problems, one needs to define measures that summarize modeling objects that are defined over the multiple domains (e.g., mini-

* Corresponding author.

E-mail address: victor.zavala@wisc.edu (V.M. Zavala).

minimize the space-time integral of the cost or minimize the expected value of the time integral of the cost). InfiniteOpt problems appear in applications such as continuous-time model predictive control and moving horizon estimation (Rawlings, 2000; Qin and Badgwell, 2003), design under uncertainty (Stankiewicz et al., 2000; Straub and Grossmann, 1993), portfolio planning (Çakmak and Özekici, 2006; Dentcheva and Ruszczyński, 2006), parameter estimation for differential equations (Shin et al., 2019; Biegler et al., 2003), reliability analysis (Pulsipher and Zavala, 2020; Suksuwan and Spence, 2018), optimal power flow (Roald et al., 2015; Lan et al., 2018), and dynamic design of experiments (Georgakis, 2013; Asprey and Macchietto, 2002).

The infinite-dimensional nature of modeling objects make InfiniteOpt problems challenging to solve (Nicholson et al., 2018; Vanderbei, 2020; Nocedal and Wright, 2006). Specifically, these problems often need to be *transcribed/transformed* into a finite dimensional representation via discretization. For instance, differential equations and associated domains are often discretized using finite difference and quadrature schemes (Biegler, 2007), while expectation operators and associated random domains are often discretized using Monte Carlo (MC) sampling and quadrature schemes (Chen et al., 2015; Kleywegt et al., 2002). The finite-dimensional representation of the problem can be handled using standard optimization solvers (e.g., Ipopt and Gurobi). Sophisticated transformation techniques are used in different scientific disciplines; for example, projection onto orthogonal basis functions is a technique that is commonly used in PDE and stochastic optimization (Devolder et al., 2010; Koivu and Pennanen, 2010).

Although common mathematical elements of InfiniteOpt problems are found across disciplines, there are limited tools available to model and solve these problems in a unified manner. Powerful, domain-specific software implementations are currently available for tackling dynamic and PDE optimization problems; examples include Gekko, ACADO, and gPROMS (Beal et al., 2018; Houska et al., 2011; Asteasuain et al., 2001). A key limitation of these tools is that the modeling abstraction used is specialized to specific problem classes and are not that easy to extend. On the other hand, there are powerful algebraic modeling languages such as JuMP, CasADi, PETSc, and Pyomo that offer high modeling flexibility to tackle different problem classes; however, these tools require the user to transform InfiniteOpt problems manually (which is tedious and prone to error). These limitations have recently motivated the development of modeling frameworks such as Pyomo.dae (Nicholson et al., 2018); this framework unifies the notion of variables, objectives, and constraints defined over continuous, infinite-dimensional domains (sets). This abstraction facilitates the modeling and transformation (via automatic discretization techniques) of optimization problems with embedded DAEs and PDEs. A limitation of this abstraction is that the notion of continuous sets is limited to space and time and this hinders generalizability (e.g., domains defined by random parameters need to be treated separately). Moreover, this framework provides limited capabilities to define measures (e.g., multi-dimensional integrals and risk functionals).

A unifying abstraction for InfiniteOpt problems can facilitate the development of software tools and the development of new analysis and solution techniques. For instance, it has been recently shown that a graph abstraction unifies a wide range of optimization problems such as discrete-time dynamic optimization (graph is a line), network optimization (graph is the network itself), and multi-stage stochastic optimization (graph is a tree) (Jalving et al., 2019). This unifying abstraction has helped identify new theoretical properties and decomposition algorithms (Jalving et al., 2020; Shin et al., 2021); this has been enabled, in part, via transferring techniques and concepts across disciplines. The limited availability of unifying modeling tools ultimately limits our ability to experiment

with techniques that appear in different disciplines and limits our ability to identify new modeling abstractions to tackle emerging applications.

In this work, we propose a unifying abstraction that facilitates the analysis, modeling, and solution of InfiniteOpt problems (see Fig. 1). Central to our abstraction is the notion of infinite parameters, which are parameters defined over infinite-dimensional domains (e.g., time, space, and random parameters). The proposed abstraction allows us to construct these domains in a systematic manner by using cartesian product operations and to define variables, objectives, and constraints over such domains and subdomains (restricted domains). The ability to handle restricted subdomains allows us to capture infinite-dimensional and standard (finite-dimensional) variables in a unified setting. Another key notion of the proposed abstraction are measure operators; these operators allow us to summarize infinite-dimensional objects over specific domains or subdomains and with this formulate problems with different types of objectives and constraints. The proposed abstraction also incorporates differential operators, which are used to model how variables evolve over their corresponding domains. These modeling elements provide a bridge between different disciplines and enables cross-fertilization. For instance, we show that the proposed abstraction allows us to leverage the use of risk measures (used in stochastic optimization) to shape time-dependent trajectories (arising in dynamic optimization). The proposed abstraction also facilitates the development of new abstractions such as event-constrained optimization problems and optimization problems with space-time random fields. The proposed abstraction forms the basis of a Julia-based modeling package that we call `InfiniteOpt.jl`. In this context, we show that the abstraction facilitates software development and enables a compact and intuitive modeling syntax and the implementation of transformation techniques (e.g., quadrature and sampling).

The paper is structured as follows. Section 2 details the proposed unifying abstraction and highlights its implementation in `InfiniteOpt.jl`. Section 3 reviews problem transformations into finite-dimensional representations through the lens of the abstraction. Section 4 discusses benefits provided by the abstraction. Section 5 presents diverse case studies and Section 6 closes the paper.

2. InfiniteOpt abstraction

In this section, we outline the proposed unifying abstraction for InfiniteOpt problems. Specifically, we discuss the different elements of the abstraction, which include infinite domains and parameters, decision variables, measure operators, and differential operators.

2.1. Infinite domains and parameters

An InfiniteOpt problem includes a collection of infinite domains \mathcal{D}_ℓ with index $\ell \in \mathcal{L}$. An individual infinite domain is defined as $\mathcal{D}_\ell \subseteq \mathbb{R}^{n_\ell}$. The term *infinite* refers to the fact that an infinite domain has infinite cardinality (i.e., $|\mathcal{D}_\ell| = \infty$) and is thus continuous. We also note that each infinite domain \mathcal{D}_ℓ is a subdomain of an n_ℓ -dimensional Euclidean space \mathbb{R}^{n_ℓ} .

An infinite domain encompasses different domains encountered in applications; for instance, this can represent a time domain of the form $\mathcal{D}_t = [t_0, t_f] \subset \mathbb{R}$ (with $n_t = 1$), a 3D spatial domain $\mathcal{D}_x = [-1, 1]^3 \subset \mathbb{R}^3$ (with $n_x = 3$), or the co-domain of a multivariate random variable $\mathcal{D}_\xi = (-\infty, \infty)^m \subset \mathbb{R}^m$ (with $n_\xi = m$).

Infinite parameters are parameters that live in the associated infinite domains; specifically, we define the general parameter $d \in \mathcal{D}_\ell$. In our previous examples, the parameters are time $t \in \mathcal{D}_t$, space $x \in \mathcal{D}_x$, and random parameters $\xi \in \mathcal{D}_\xi$.

Traditional Formulations

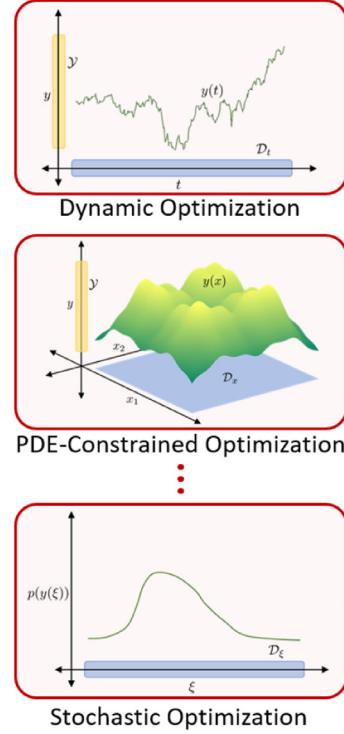


Fig. 1. Summary of the proposed InfiniteOpt abstraction; the abstraction seeks to unify existing problem classes and use this to develop new classes.

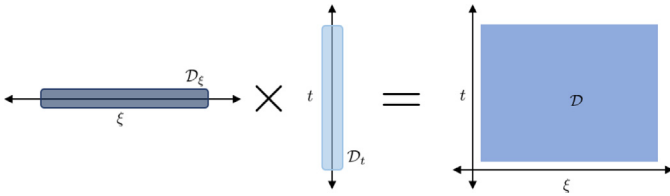


Fig. 2. Cartesian product of random domain \mathcal{D}_ξ and time domain \mathcal{D}_t to produce \mathcal{D} .

The domain of the InfiniteOpt problem is the cartesian product of the infinite domains:

$$\mathcal{D} := \prod_{\ell \in \mathcal{L}} \mathcal{D}_\ell. \quad (1)$$

The construction of the problem domain is exemplified in Fig. 2; here, we see that the domain obtained from the cartesian product of a 1D random domain \mathcal{D}_ξ and a 1D temporal domain \mathcal{D}_t .

2.2. Decision variables

A key feature of an InfiniteOpt problem is that it contains decision variables that are *functions* of infinite-dimensional parameters; as such, decision variables are also infinite-dimensional. In the proposed abstraction, we also consider InfiniteOpt problems that contain finite-dimensional variables (that do not depend on any parameters) and finite-dimensional variables that originate from reductions of infinite variables (e.g., integration over a domain or evaluation of an infinite variable at a point in the domain). To account for these situations, we define different types of variables: infinite, semi-infinite, point, and finite.

Infinite variables $y : \mathcal{D} \mapsto \mathcal{Y} \subseteq \mathbb{R}^{n_y}$ are functions that map an infinite domain \mathcal{D} to the domain \mathcal{Y} . These variables are expressed as:

$$y(d) \in \mathcal{Y}, \quad d \in \mathcal{D}. \quad (2)$$

Fig. 3 shows that infinite variables are functions that can be interpreted as manifolds (also known as surfaces and fields). The goal of the InfiniteOpt problem is to shape these manifolds to achieve predetermined goals (e.g., minimize their mean value or their peaks). Example classes of infinite variables include uncertainty-dependent decision policies arising in stochastic optimization (i.e., recourse variables), time-dependent control/state policies arising in dynamic optimization, or space-time fields arising in PDE optimization. For instance, in a stochastic PDE optimization problem, one might have an infinite variable of the form $y(t, x, \xi)$, which is simultaneously parameterized over time t , space x , and uncertainty ξ . In other words, an infinite variable is equivalent to a collection of finite decision variables indexed over an infinite domain (producing an infinite collection of variables). Fig. 4 illustrates some infinite variables commonly encountered in different disciplines.

Semi-infinite variables $y : \mathcal{D}_{-\ell} \mapsto \mathcal{Y} \subseteq \mathbb{R}^{n_y}$ correspond to infinite variables in which the subdomain \mathcal{D}_ℓ has been restricted/projected to a single point \hat{d}_ℓ ; the restricted domain is denoted as $\mathcal{D}_{-\ell}$. These variables are also functions that map from the infinite domain $\mathcal{D}_{-\ell}$ to the domain \mathcal{Y} :

$$y(d) \in \mathcal{Y}, \quad d \in \mathcal{D}_{-\ell}. \quad (3)$$

We refer to $\hat{d}_\ell \in \mathcal{D}_\ell$ as a support point of the domain. A depiction of how semi-infinite variables are derived from infinite variables via projection is provided in Fig. 5. Example classes mirror those of infinite variables with multiple parameter dependencies; for example, in a stochastic PDE problem, we might want to evaluate the variable $y(t, x, \xi)$ at the support point $t = 0$ (initial time); this gives the semi-infinite variable $y(0, x, \xi)$ and domain $\mathcal{D}_{-t} = \mathcal{D}_x \times \mathcal{D}_\xi$.

Point variables denote infinite variables in which the entire \mathcal{D} is restricted to a single point \hat{d} . These are finite-dimensional variables that can be expressed as:

$$y(\hat{d}) \in \mathcal{Y}. \quad (4)$$

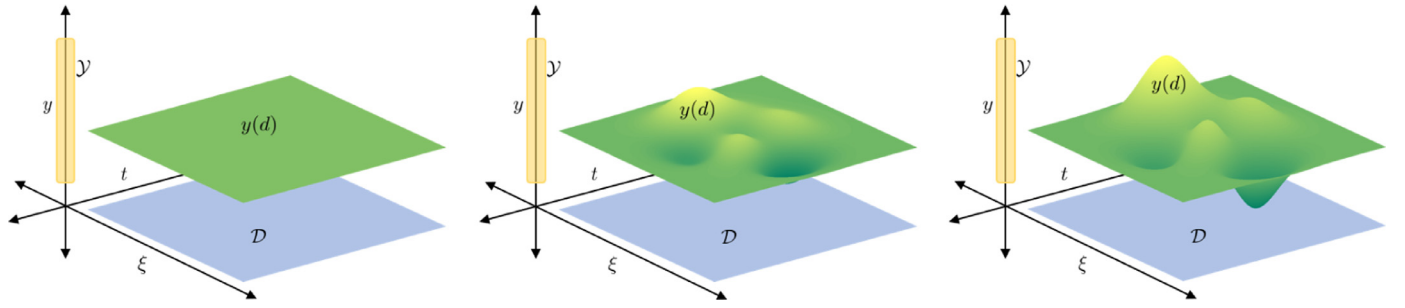


Fig. 3. Depiction of realizations of an infinite variable $y(d)$ with $\mathcal{Y} \subseteq \mathbb{R}$ and $\mathcal{D} \subseteq \mathbb{R}^2$. The horizontal axes define the domain \mathcal{D} and the vertical axis denotes the domain of feasible decisions \mathcal{Y} .

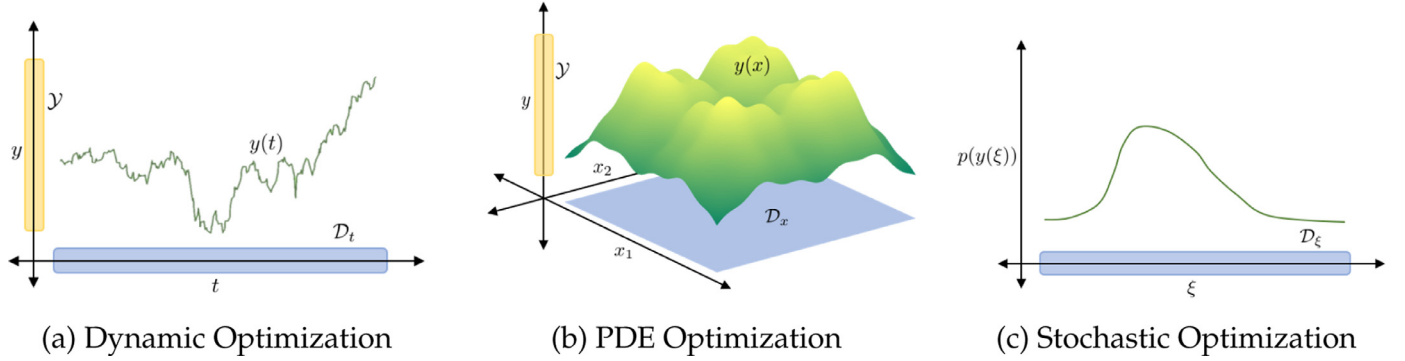


Fig. 4. Example of infinite variables arising in traditional formulations. The recourse variable $y(\xi)$ is visualized in terms of its probability density function (as is customary in stochastic optimization).

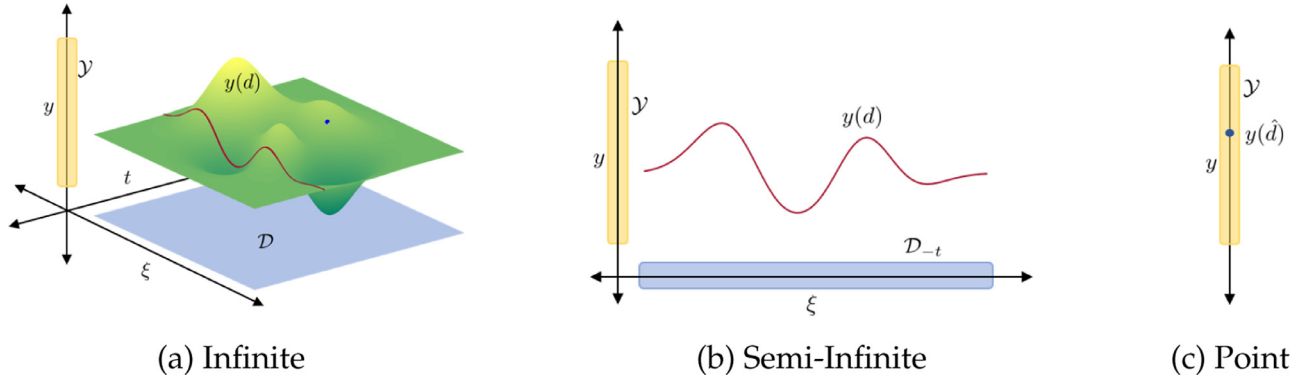


Fig. 5. Illustration of how a semi-infinite variable $y(d)$, $d \in \mathcal{D}_{-t}$ and a point variable $y(\hat{d})$ are obtained from an infinite variable $y(d)$, $d \in \mathcal{D}$ via restriction/projection. Semi-infinite and point variables are realizations of an infinite variable and live in the domain \mathcal{Y} .

Fig. 5 illustrates how point variables relate to other variable classes. Examples of point variables include random variables evaluated at a specific scenario/sample/realization of the uncertain parameter or a space-time variable evaluated at a specific point in the domain (e.g., boundary conditions). Point variables are also used in dynamic optimization to specify so-called point constraints (these ensure that a time trajectory satisfies a set of constraints at specific time points). Point variables can be thought of as variables that are held constant in an infinite domain.

The proposed abstraction also considers finite variables:

$$z \in \mathcal{Z} \subseteq \mathbb{R}^{n_z} \quad (5)$$

where \mathcal{Z} denotes the feasible set. These are variables that are not parameterized over an infinite domain. Examples of finite variables arising in applications are first-stage decision variables and design variables arising in stochastic optimization or model parameters estimated in a dynamic optimization problem. Fig. 6 shows that this variable is analogous to the point variable depicted in Fig. 5c.

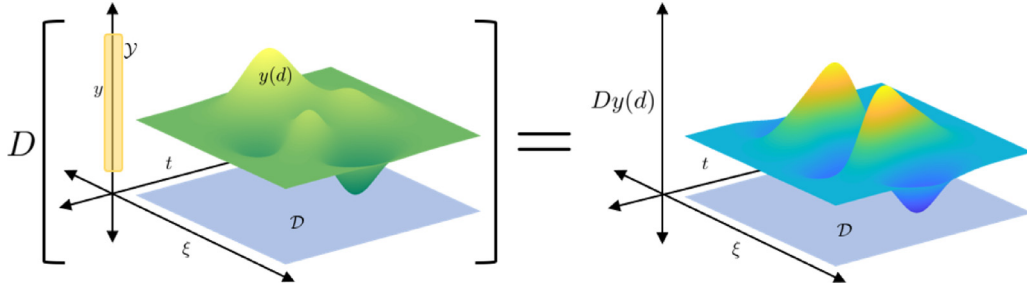
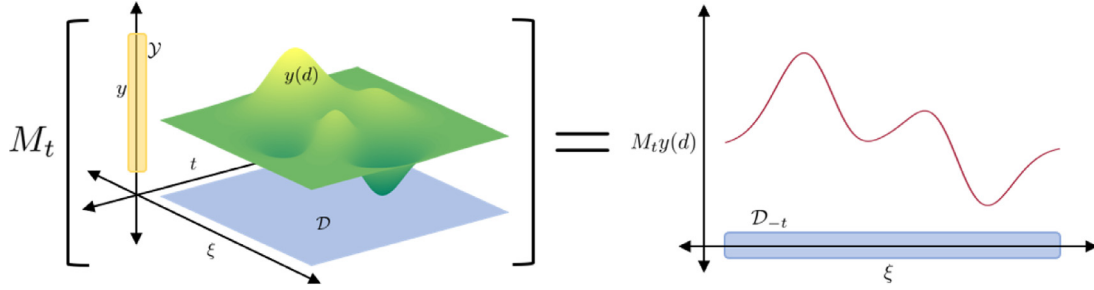


Fig. 6. A depiction of a finite variable $z \in \mathcal{Z} \subseteq \mathbb{R}$.

In fact, this highlights that a point variable is a finite variable; the difference is that a point variable is derived from a restriction of an infinite variable. However, we note that, technically speaking, a finite variable can be seen as a special case of an infinite variable in which the domain is a point. As such, infinite variables provide a unifying framework to capture different types of variables.

2.3. Differential operators

Differential operators are a typical modeling element of InfiOpt problems. These operators capture how a given decision variable (an infinite-dimensional function) changes over its associated

Fig. 7. Depiction of a differential operator D acting on the infinite variable $y(d)$.Fig. 8. Measure operator M_t that acts on domain \mathcal{D}_t of the infinite variable $y(d)$, $d \in \mathcal{D}$. This operation returns the semi-infinite variable $m(\xi) := M_t y$, $\xi \in \mathcal{D}_\xi$.

domain; for example, in a dynamic optimization problem, we want to know how quickly a state/control variable changes in time.

The proposed abstraction defines a differential operator of the general form:

$$D : \mathcal{D} \mapsto \mathbb{R}. \quad (6)$$

Differential operators are applied to infinite variables $y(d)$, $d \in \mathcal{D}$. These operators capture partial derivatives over individual parameters and more sophisticated operators that simultaneously operate on multiple parameters such as the Laplacian operator (typically encountered in PDE optimization). Fig. 7 illustrates a differential operator acting on an infinite variable.

Differential operators map the infinite domain \mathcal{D} to the scalar domain \mathbb{R} . The output of a differential operator is an infinite variable $Dy(d)$, $d \in \mathcal{D}$ that inherits the domain of the argument $y(d)$. For example, consider the infinite variable $y(t)$, $t \in \mathcal{D}_t$; the partial derivative operator is an infinite variable $y'(t) := \partial y(t)/\partial t$, $t \in \mathcal{D}_t$. Some other specific examples include the partial derivative $\frac{\partial y(t, x, \xi)}{\partial t}$, $(t, x, \xi) \in \mathcal{D}$ and the Laplacian $\Delta y(x)$, $x \in \mathcal{D}_x$. Note that derivatives of the form $\frac{\partial}{\partial \xi}$ are not typically used in stochastic optimization problems; however, the proposed abstraction allows for this operator to be defined. This modeling feature can be used, for instance, to control how a random variable changes in the uncertainty space (this can be used to manipulate the shape of its probability density function).

2.4. Measure operators

Measure operators are key modeling constructs that are used to summarize functions by collapsing them to a single quantity. For example, in a dynamic optimization problem, one typically minimizes the time-integral of the cost (a scalar quantity). The proposed abstraction defines a measure operator of the general form:

$$M_\ell : \mathcal{D} \mapsto \mathbb{R} \quad (7)$$

Here, the index ℓ indicates that the operator is applied on the subdomain \mathcal{D}_ℓ and thus has the effect of restricting the domain. As such, the output of a measure operator is a semi-infinite variable

that lives in the restricted domain $\mathcal{D}_{-\ell}$. Fig. 8 illustrates such a measure operator.

Measure operators are a key feature of InfiniteOpt problems; specifically, objective functions and constraints are often expressed in terms of measure operators. For instance, consider a field $y(t, x, \xi)$ arising in a stochastic PDE problem; one can define measure operator that computes the time-integral $m(x, \xi) := \int_{t \in \mathcal{D}_t} y(t, x, \xi) dt$, $(x, \xi) \in \mathcal{D}_x \times \mathcal{D}_\xi$ and note that the output of this operation is a semi-infinite variable $m(x, \xi)$ that lives in $\mathcal{D}_{-t} = \mathcal{D}_x \times \mathcal{D}_\xi$. One can also define an operator that computes the expectation $m(t, x) := \int_{\xi \in \mathcal{D}_\xi} y(t, x, \xi) p(\xi) d\xi$, $(t, x) \in \mathcal{D}_t \times \mathcal{D}_x$ (where $p(\cdot)$ is the probability density function of ξ); this operation gives a semi-infinite variable $m(t, x)$ that lives in $\mathcal{D}_{-\xi} = \mathcal{D}_t \times \mathcal{D}_x$.

The expectation is a measure operator that is of particular interest in stochastic optimization because this can be used to compute different types of risk measures and probabilities; for instance, in the previous example, one might want to compute a probability of the form:

$$\mathbb{P}_\xi(y(t, x, \xi) \in \mathcal{Y}), \quad (t, x) \in \mathcal{D}_t \times \mathcal{D}_x. \quad (8)$$

This is the probability that $y(t, x, \xi)$ is in the domain \mathcal{Y} and can be computed by using an expectation operator:

$$\begin{aligned} \mathbb{P}_\xi(y(t, x, \xi) \in \mathcal{Y}) &= \mathbb{E}_\xi[\mathbb{1}[y(t, x, \xi) \in \mathcal{Y}]] \\ &= \int_{\xi \in \mathcal{D}_\xi} \mathbb{1}[y(t, x, \xi) \in \mathcal{Y}] p(\xi) d\xi. \end{aligned} \quad (9)$$

where $\mathbb{1}[\cdot]$ is the indicator function and the argument of this function is the event of interest. We recall that the indicator function returns a value of 0 if the event is not satisfied or a value of 1 if the event is satisfied. An important observation is that the indicator function can be used to define a wide range of measures and over different types of domains; for instance, the measure $\int_{t \in \mathcal{D}_t} \mathbb{1}[y(t) > \bar{y}] dt$ denotes the amount of time that the function $y(t)$, $t \in \mathcal{D}_t$ crosses the threshold \bar{y} .

Measure operators can also be used to summarize infinite variables over multiple subdomains; for example, one can consider the following measures:

$$M_{t,x} y = \int_{t \in \mathcal{D}_t} \int_{x \in \mathcal{D}_x} y(t, x, \xi) dx dt, \quad \xi \in \mathcal{D}_\xi \quad (10a)$$

$$M_{t,x,\xi} y = \mathbb{E}_{\xi} \left[\int_{t \in \mathcal{D}_t} \int_{x \in \mathcal{D}_x} y(t, x, \xi) dx dt \right] \quad (10b)$$

$$M_{t,x,\xi} y = \mathbb{E}_{\xi} \left[\int_{t \in \mathcal{D}_t} \int_{x \in \mathcal{D}_x} \mathbb{1}[y(t, x, \xi) \in \mathcal{Y}] dx dt \right]. \quad (10c)$$

One can thus see that a wide range of measures can be envisioned.

2.5. Objectives

In InfiniteOpt problems, objective functions are functions of infinite variables; as such, objectives are infinite variables. Minimizing or maximizing an infinite-dimensional function does not yield a well-posed optimization problem. This situation is similar in spirit to that appearing in multi-objective optimization problem, in which we seek to simultaneously minimize/maximize a finite collection of objectives (in an InfiniteOpt problem, the collection is infinite).

To deal with ill-posedness, one often resorts to scalarization techniques; the idea is to reduce/summarize the infinite-dimensional function into a single scalar quantity. The goal of this scalarization procedure is to manipulate the shape of the infinite-dimensional objective (e.g., minimize its mean value or its extreme value). Scalarization is performed by using measure operators; for instance, in the context of multi-objective optimization, one scalarizes the objectives by computing a weighted summation of the objectives. In an InfiniteOpt setting, this weighting is done by computing a weighted integral of the objective. For instance, in dynamic optimization, we often have a time-dependent objective function $f(t) := f(y(t), t)$, $t \in \mathcal{D}_t$; here, we can notice that the objective depends on an infinite variable and is thus also an infinite variable. We can scalarize this variable by using the measure $M_t f := \int_{t \in \mathcal{D}_t} f(t) w(t) dt$ with a weighting function satisfying $w: \mathcal{D}_t \rightarrow [0, 1]$ and $\int_{t \in \mathcal{D}_t} w(t) dt = 1$ (note that this measure is a time-average of the objective trajectory).

In space-time PDE optimization, the objective is defined over an infinite domain $\mathcal{D} = \mathcal{D}_t \times \mathcal{D}_x$ that depends on decision variables $y(t, x) \in \mathcal{Y}$; as such, the objective is given by the field $f(t, x) := f(y(t, x), t, x)$, $t \in \mathcal{D}_t, x \in \mathcal{D}_x$. One can scalarize this field by using a measure:

$$M_{t,x} f = \int_{(t,x) \in \mathcal{D}_{t,x}} f(t, x) w(t, x) dt dx, \quad (11)$$

with weighting function satisfying $w: \mathcal{D}_{t,x} \rightarrow [0, 1]$ and $\int_{(t,x) \in \mathcal{D}_{t,x}} w(t, x) dt dx = 1$. One can think of this measure as a space-time average of the objective.

In stochastic optimization, we have infinite-dimensional objectives of the form $f(\xi) := f(z, y(\xi))$, $\xi \in \mathcal{D}_{\xi}$, where $y(\xi)$ is a recourse variable (an infinite variable). Scalarization can be achieved by using the expectation operator:

$$M_{\xi} f = \mathbb{E}_{\xi} [f(\xi)] = \int_{\xi \in \mathcal{D}_{\xi}} f(\xi) p(\xi) d\xi \quad (12)$$

where $p(\xi)$ is the probability density function satisfying $p(\xi) \geq 0$ and $\int_{\xi \in \mathcal{D}_{\xi}} p(\xi) = 1$. This measure is illustrated in Fig. 9.

Average measures as those described previously are intuitive and widely used in practice; however, in Section 4.1 we will see that one can envision using a huge number of measures to perform scalarization. The huge number of choices arises from the fact that one can manipulate the shape of an infinite-dimensional function in many different ways (by focusing on different features of the function); for instance, one can minimize the peak of the function or minimize its variability. In the field of stochastic optimization, for instance, one aims to manipulate the shape of the

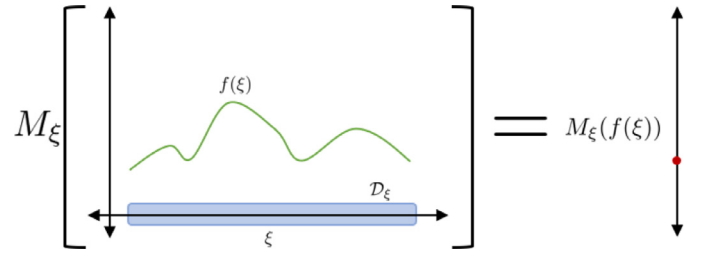


Fig. 9. Depiction of measure operator M_{ξ} acting on infinite variable $f(\xi) = f(z, y(\xi))$.

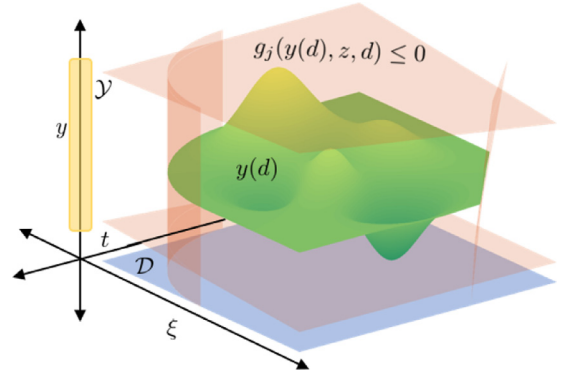


Fig. 10. Depiction of infinite-dimensional constraints $g_j(y(d), z, d) \leq 0$ defined over an infinite domain \mathcal{D} .

infinite-dimensional objective by selecting different risk measures (summarizing statistics) such as the variance, median, quantile, worst/best -case value, or probabilities. We will see that one can borrow risk measures used in stochastic optimization to summarize infinite variables in other domains (e.g., space-time); this leads to interesting approaches to shape complex manifolds/fields arising in complex InfiniteOpt problems.

2.6. Constraints

As in the case of objectives, constraints in InfiniteOpt problems depend on infinite variables and are thus infinite variables themselves. One thus need to use specialized techniques to handle constraints and with this ensure that the problem is well-posed. A key observation that arises in this context is that constraints are treated differently than objectives; specifically, one typically seeks to impose bounds on constraint values and one can handle collections of constraints simultaneously. For instance, in semi-infinite optimization problems, one enforces constraints of the form:

$$g_j(y(d), d) \leq 0, \quad j \in \mathcal{J}, d \in \mathcal{D}. \quad (13)$$

In vector form, this collection of constraints can be expressed as:

$$g(y(d), d) \leq 0, \quad d \in \mathcal{D}. \quad (14)$$

where $g(\cdot)$ is a vector function that contains the constraint collection $g_j(\cdot)$ $j \in \mathcal{J}$. We can see that the constraint functions $g_j(\cdot)$ are required to take value below zero for all values of the parameter $d \in \mathcal{D}$. Moreover, we can see that the constraints $j \in \mathcal{J}$ are all enforced at once. Fig. 10 illustrates this constraint system. This particular approach to enforcing constraints is also widely used in dynamic optimization and stochastic optimization. For instance, in the context of dynamic optimization, one may seek to keep time trajectories for controls/states below a certain threshold value for all times in a time horizon. In the context of stochastic optimization, one may seek to satisfy the demand of a product for all pos-

sible realizations of uncertainty (in this context the constraints are said to be enforced *almost surely* or with probability of one).

These types of constraints are defined in our abstraction using the general form:

$$g(Dy, y(d), z, d) \leq 0, \quad d \in \mathcal{D}. \quad (15)$$

These encapsulate the above use cases and are exemplified by the following PDE optimization constraints that include differential operators, path constraints, and point constraints (e.g., boundary conditions):

$$\begin{aligned} g(Dy(t, x), y(t, x), t, x) &= 0, \quad (t, x) \in \mathcal{D}_t \times \mathcal{D}_x \\ g(y(t, x), t, x) &\leq 0, \quad (t, x) \in \mathcal{D}_t \times \mathcal{D}_x \\ g(y(\hat{t}, \hat{x}), \hat{t}, \hat{x}) &\leq 0. \end{aligned} \quad (16)$$

Constraints that follow the form of (15) can be quite restrictive for certain applications, since they need to hold for every value parameter $d \in \mathcal{D}$. One can relax this requirement by instead enforcing the constraint on a selected set of points in the domain \mathcal{D} or by enforcing constraints on a measure of the constraint functions. For instance, consider the set of constraint functions $h_k(Dy(d), y(d), z, d)$, $k \in \mathcal{K}$; we can aim to enforce constraints on expected values of such functions as:

$$\int_{\xi \in \mathcal{D}_\xi} h_k(y(\xi), \xi) p(\xi) d\xi \geq 0, \quad k \in \mathcal{K}. \quad (17)$$

Given that there are a wide range of measures that can help shape functions over infinite-dimensional domains, one can also envision different approaches to enforce constraints. For instance, in stochastic optimization, one typically uses scalar chance (probabilistic) constraints of the form:

$$\mathbb{P}_\xi(h_k(y(\xi), \xi) \leq 0) \geq \alpha, \quad k \in \mathcal{K}. \quad (18)$$

This set of constraints require that each constraint function $h_k(\cdot)$ is kept below zero to a certain probability level α . In stochastic optimization, one also often enforces joint chance constraints:

$$\mathbb{P}_\xi(h_k(y(\xi), \xi) \leq 0, \quad k \in \mathcal{K}) \geq \alpha. \quad (19)$$

The joint chance constraint can also be expressed in vector form as:

$$\mathbb{P}_\xi(h(y(\xi), \xi) \leq 0) \geq \alpha. \quad (20)$$

Joint chance constraints require that the constraint functions $h(\cdot)$ are kept (jointly) below zero with a certain probability level α . We will see that joint chance constraints allow us to enforce constraints on probability of *events* and we will see that this provides a flexible modeling construct to capture complex decision-making logic. For instance, we might want to ensure that the temperature of a system is higher than a certain value *and* that the concentration of the system is lower than a certain value with a given probability. Joint chance constraints can also be interpreted as a generalization of other constraint types; for instance, if we set $\alpha = 1$, the constraint (19) is equivalent to (13).

The above measure constraints can be expressed in the following general form:

$$Mh(Dy, y(d), z, d) \geq 0. \quad (21)$$

For instance, the chance constraint (20) can be expressed as:

$$M_\xi h \geq 0 \quad (22)$$

with

$$M_\xi h = \mathbb{E}_\xi[\mathbb{1}[h(y(\xi), \xi) \leq 0]] - \alpha. \quad (23)$$

2.7. InfiniteOpt formulation

We summarize the previous elements to express the InfiniteOpt problem in the following abstract form:

$$\begin{aligned} \min_{y(\cdot) \in \mathcal{Y}, z \in \mathcal{Z}} \quad & Mf(Dy, y(d), z, d) \\ \text{s.t.} \quad & g(Dy, y(d), z, d) \leq 0, \quad d \in \mathcal{D} \\ & Mh(Dy, y(d), z, d) \geq 0. \end{aligned} \quad (24)$$

This abstract form seeks to highlight the different elements of the proposed abstraction (e.g., infinite domains and variables, finite variables, measure operators, differential operators).

2.8. Implementation in InfiniteOpt.jl

We now proceed to describe how the proposed abstraction can facilitate the development of modeling tools. Specifically, the proposed abstraction is used as the backbone of a modeling package that we call InfiniteOpt.jl (<https://github.com/zavalab/InfiniteOpt.jl>). InfiniteOpt.jl is written in the Julia programming language (Bezanson et al., 2017) and builds upon the capabilities of JuMP.jl (Dunning et al., 2017) to intuitively and compactly express InfiniteOpt problems.

Some of the modeling features of InfiniteOpt.jl are illustrated by using the example problem:

$$\min_{y_a(t), y_b(t, \xi), y_c(\xi), z} \int_{t \in \mathcal{D}_t} y_a(t)^2 + 2\mathbb{E}_\xi[y_b(t, \xi)] dt \quad (25a)$$

$$\text{s.t.} \quad \frac{dy_b(t, \xi)}{dt} = y_b(t, \xi)^2 + y_a(t) - z_1, \quad t \in \mathcal{D}_t, \quad \xi \in \mathcal{D}_\xi \quad (25b)$$

$$y_b(t, \xi) \leq y_c(\xi)U, \quad t \in \mathcal{D}_t \quad (25c)$$

$$\mathbb{E}_\xi[y_c(\xi)] \geq \alpha \quad (25d)$$

$$y_a(0) + z_2 = \beta \quad (25e)$$

$$y_a(t), y_b(t, \xi) \in \mathbb{R}_+, y_c(\xi) \in \{0, 1\}, z \in \mathbb{Z}^2, \quad t \in \mathcal{D}_t, \quad \xi \in \mathcal{D}_\xi \quad (25f)$$

Here, $y_a(t)$, $y_b(t, \xi)$, and $y_c(\xi)$ are infinite variables, z are finite variables, $U, \alpha, \beta \in \mathbb{R}$ are constants, $\mathcal{D}_t = [t_0, t_f]$ is the time domain, and \mathcal{D}_ξ is the co-domain of the random parameter $\mathcal{N}(\mu, \Sigma)$.

The corresponding InfiniteOpt.jl syntax for expressing this problem is shown in Code Snippet 1. An InfiniteOpt problem is stored in an InfiniteModel object; Line 4 shows the initialization of the model object model. The model is automatically transcribed into a finite dimensional representation and solved using the KNITRO solver (Nocedal, 2006). More information on how the InfiniteModel is transcribed by InfiniteOpt.jl is provided in Section 3.3. Lines 7 and 8 use @infinite_parameter to define the infinite parameters with their respective infinite domains and indicate that each domain should use 100 finite supports in the transcription. The random parameters ξ can be associated with any probably density function supported by the Julia package Distributions.jl (Besançon et al., 2019). Lines 11–14 define the decision variables and their associated properties in accordance with Eq. (25f) following a symbolic JuMP.jl-like syntax by means of @variable. Line 17 defines the complex objective depicted in Eq. (25a) via @objective. Lines 20–23 define constraints (25b)–(25e) using @constraint. Notice how the differential operator and measure operators (in this case an expectation and an integral) are easily incorporated using Julia syntax. Lines 26 and 27 illustrate how the model model is solved using

optimize! and then how the solution information can be extracted from the model.

Code Snippet 1: Modeling problem (2.25) using `InfiniteOpt.jl`.

```

1 using InfiniteOpt, Distributions, KNITRO
2
3 # Initialize the model
4 model = InfiniteModel(KNITRO.Optimizer)
5
6 # Add the infinite parameters corresponding to the infinite domains
7 @infinite_parameter(model, t ∈ [t0, tf], num_supports = 100)
8 @infinite_parameter(model, ξ[1:10] ~ MvNormal(μ, Σ), num_supports = 100)
9
10 # Add the variables and their domain constraints
11 @variable(model, 0 ≤ ya, Infinite(t))
12 @variable(model, 0 ≤ yb, Infinite(t, ξ))
13 @variable(model, yc, Infinite(ξ), Bin)
14 @variable(model, z[1:2], Int)
15
16 # Define the objective
17 @objective(model, Min, ∫(ya ^ 2 + 2 * E(yb, ξ), t))
18
19 # Add the constraints
20 @constraint(model, ∂(yb, t) == yb ^ 2 + ya - z[1])
21 @constraint(model, yb ≤ yc + U)
22 @constraint(model, E(yc, ξ) ≥ α)
23 @constraint(model, ya(0) + z[2] == β)
24
25 # Solve and retrieve the results
26 optimize!(model)
27 opt_objective = objective_value(model)

```

3. InfiniteOpt transformations

We now discuss how InfiniteOpt problems are solved through the lens of the proposed unifying abstraction. Solution approaches typically rely on transforming the InfiniteOpt problem (24) into a finite-dimensional formulation that can be solved using conventional optimization solvers. There are a large number of possible methods to transform InfiniteOpt problems that are used in different domains such as dynamic, PDE, and stochastic optimization. Our goal here is not to provide an extensive discussion and implementation of all these approaches; instead, we highlight common elements of different approaches and motivate how these can be facilitated by using a unifying abstraction.

3.1. Direct transcription

Our first goal is to obtain a finite representation of an infinite domain \mathcal{D}_ℓ ; direct transcription accomplishes this via a finite set of support points that we represent as $\hat{\mathcal{D}}_\ell = \{\hat{d}_{\ell,i} : \hat{d}_{\ell,i} \in \mathcal{D}_\ell, i \in \mathcal{I}_\ell\}$. The concept of the support set $\hat{\mathcal{D}}_\ell$ used here is general and a variety of methods can be employed to generate it. In stochastic optimization, for instance, a set of MC samples is typically drawn from a probability density function of the infinite parameters (Birge and Louveaux, 2011), while PDE problems commonly use quadrature schemes (Shin and Zavala, 2020). The proposed abstraction seeks to enable porting techniques across fields; for instance, one might envision generating support points for a space-time domain by sampling or one might envision generating support points for a random domain by using quadrature points (as done in sparse grids and Latin hypercube sampling).

The support set for the infiniteOpt problem $\hat{\mathcal{D}}$ is the cartesian product of the individual supports sets:

$$\hat{\mathcal{D}} := \prod_{\ell \in \mathcal{L}} \hat{\mathcal{D}}_\ell. \quad (26)$$

Fig. 11 illustrates how the support set (a finite domain) approximates the infinite domain \mathcal{D} . Note that this definition of $\hat{\mathcal{D}}$ assumes that the individual domains \mathcal{D}_ℓ are independent of one another. This assumption does not hold in some complex applications; for instance, in stochastic dynamic optimization problems, we might

have random parameters that are functions of time (this is discussed further in Section 4.2).

The infinite-dimensional formulation (24) is projected onto the finite support set to yield a finite-dimensional approximation that can be modeled using conventional optimization solvers. We now proceed to discuss how this projection is achieved. Measures are approximated with an appropriate numerical scheme; this can take on a range of forms and may require the incorporation of additional supports and variables. For instance, a common set of measures (e.g., expectations and integrals over space-time domains) are of the form:

$$M_\ell y = \int_{d' \in \mathcal{D}_\ell} y(d') w(d') dd' \quad (27)$$

where $w(\cdot)$ is a weighting function. Such measures can be approximated using support points as:

$$M_\ell y \approx \sum_{i \in \mathcal{I}_\ell} \beta_i y(\hat{d}_{\ell,i}) w(\hat{d}_{\ell,i}). \quad (28)$$

This general form is used in quadrature and sampling schemes; the only difference between these schemes arises in how the supports $\hat{d}_{\ell,i}$ and the coefficients β_i are selected. Fig. 12 depicts a measure approximated via quadrature.

Differential operators appearing in formulation (24) also need to be approximated. Sometimes these operators can be reformulated in integral form; in such a case, one can use the measure approximations previously discussed. However, in some cases, this reformulation is not possible; for instance, a differential operator might be implicitly defined within expression functions (e.g., boundary conditions) and/or within measures. In our framework, we treat differential operators as infinite variables and handle them via *lifting*. To illustrate how this is done, suppose that we have an expression of the form:

$$g\left(\frac{\partial y(d)}{\partial d_\ell}, y(d), z\right) = 0, \quad d \in \mathcal{D}_\ell. \quad (29)$$

Here, we introduce an auxiliary variable $y'(d)$ and reformulate the above constraint as:

$$g(y'(d), y(d), z) = 0, \quad d \in \mathcal{D}_\ell \quad (30a)$$

$$\frac{\partial y(d)}{\partial d_\ell} = y'(d), \quad d \in \mathcal{D}_\ell. \quad (30b)$$

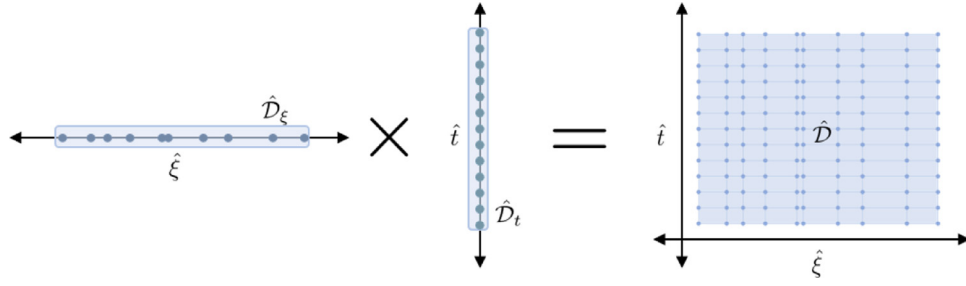


Fig. 11. Finite support set $\hat{\mathcal{D}}$ that approximates infinite domain \mathcal{D} .

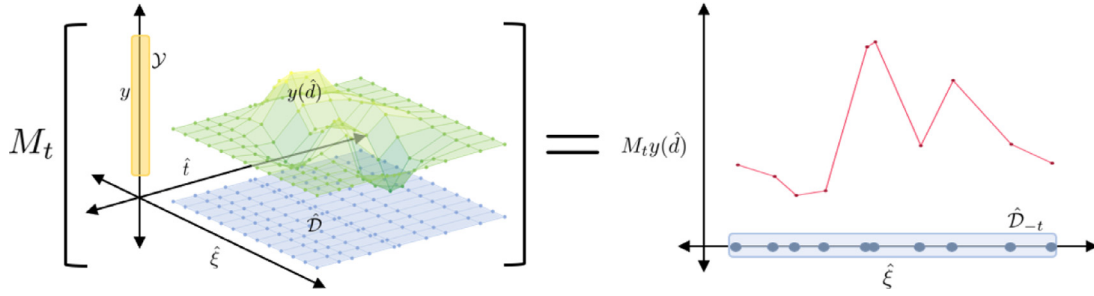


Fig. 12. Depiction of a measure operator M_t approximated via a numerical scheme (quadrature in this case).

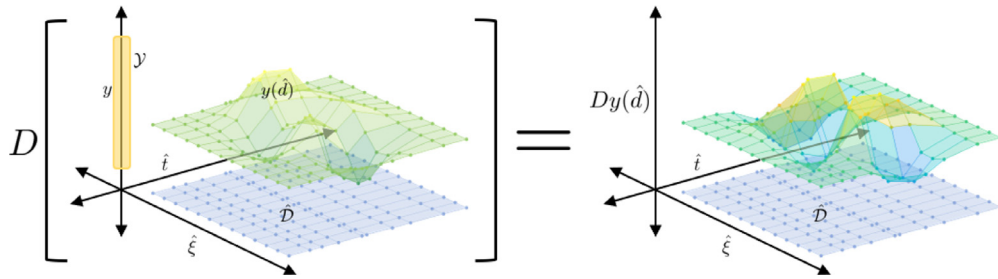


Fig. 13. Depiction of a differential operator D approximated via a numerical scheme (central finite differences in this case) relative to a realization of infinite variable $y(d)$.

The second expression can now be approximated using traditional schemes using support points; for instance, when d denotes time (e.g., in a dynamic optimization problem), one typically uses a backward finite difference:

$$y(\hat{d}_{\ell,i}) = y(\hat{d}_{\ell,i-1}) + (\hat{d}_{\ell,i} - \hat{d}_{\ell,i-1})y'(\hat{d}_{\ell,i}). \quad (31)$$

Fig. 13 illustrates how these techniques approximate differential operators. A lifting approach can be used to handle higher-order and multi-dimensional operators (e.g., Laplacian) via nested recursions. These basic constructs can be used to enable the implementation of direct transcription schemes such as MC sampling, quadrature, finite difference, and orthogonal collocation over finite elements.

Once the measures and derivatives are approximated, the direct transcription procedure is finalized by projecting the remaining constraints with infinite domain dependencies over the finite support set $\hat{\mathcal{D}}$. The transformation incurred by direct transcription is often linear since the typical measure and differential operator approximations are linear transformations of the respective modeling objects (e.g., MC sampling and finite difference). For instance, this means that if the InfiniteOpt problem of interest is an infinite quadratic program (QP), then its transcribed variant will typically be a finite QP.

We note that direct transcription of problems with multiple infinite domains (e.g., PDE-constrained problems) can incur tractability concerns due to the support combinatorics. For such problems, decomposition approaches such as the one proposed in

Na et al. (2020) can be used address these limitations. Moreover, alternative transformations may enhance tractability for certain problem classes as is discussed in Section 3.2.

3.2. Alternative transformations

Direct transcription is a common class of methods for transforming an InfiniteOpt problem into a finite-dimensional representation by using a finite set of support points. A limitation of this approach is that it does not provide a solution in functional form (it only provides a solution defined at the support points). Alternative transformation methods can be envisioned to deliver solutions in functional form. The method of weighted residuals (MWR) is a general class of methods that conducts the transformation by approximating the problem elements using basis expansions. Popular MWR techniques include polynomial chaos expansion (PCE) used in stochastic optimization (Xiu, 2010) and orthogonal collocation used in dynamic optimization (Armaou and Christofides, 2002; Koivu and Pennanen, 2010). For instance, Gnegel et al. recently demonstrated how such basis expansion techniques can enhance the tractability of mixed-integer PDE problems relative to using traditional transcription methods (Gnegel et al., 2021). Such techniques are often behind what are typically referred to as order reduction methods in the PDE community (Armaou and Christofides, 2002).

In MWR, a set of trial/basis functions $\Phi = \{\phi_i(d) : i \in \mathcal{I}\}$ is defined over an infinite domain \mathcal{D} and linear combinations of these

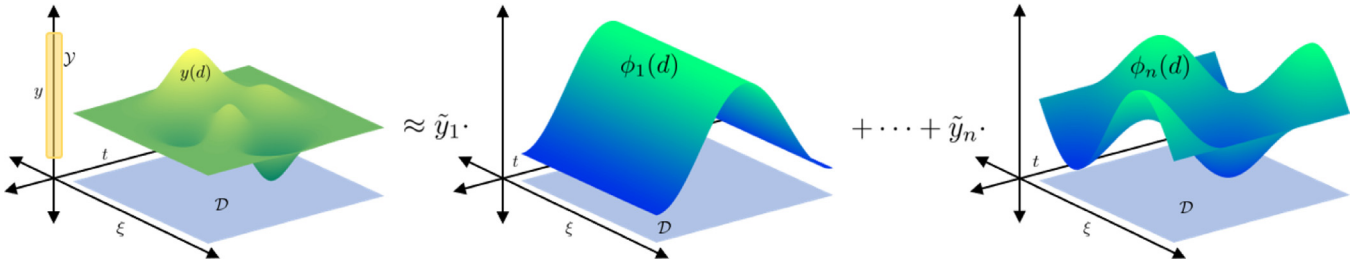


Fig. 14. Depiction of how an infinite variable $y(d)$ can be approximated as a linear combination of basis functions $\phi_i(\cdot)$.

functions are used to approximate the infinite variables:

$$y(d) \approx \sum_{i \in \mathcal{I}} \tilde{y}_i \phi_i(d) \quad (32)$$

where $\tilde{y}_i \in \mathbb{R}$ are the basis function coefficients. An illustration of this approximation is given in Fig. 14; here, we require that the basis functions $\phi_i(d)$ and the infinite variables $y(d)$ both reside in a common space such that this approximation becomes exact when the set Φ is an orthogonal set of basis functions and $|\Phi| \rightarrow \infty$ (Graham and Rawlings, 2013). Since the basis functions are known, this representation allows us to represent the infinite variables $y(d)$ in terms of the coefficients \tilde{y}_i (which are finite variables). As such, this approach effectively transforms infinite variables into finite variables. The goal is now to project the formulation (24) onto a set of basis functions so as to obtain a finite formulation that solely depends on the finite variables \tilde{y}_i and z . This is done by expressing differential and measure operators by using the basis expansion of the infinite variables (i.e., with operators applied to the basis functions). In certain cases, the expansion coefficients can be useful in evaluating certain measure types; for example, the coefficients will correspond to the statistical moments of the infinite variables when PCE is applied to a stochastic formulation with a basis that is orthogonal to the probability density function and these moments are often used to express expectations and risk measures (Zymler et al., 2013).

After basis expansion representations are incorporated, the problem is fully defined in terms of the finite-dimensional variables \tilde{y}_i and z . However, this representation is not yet tractable, since it still contains infinite-dimensional objects (e.g., basis functions and associated operators). To deal with this, we consider the residual (i.e., finite error) $R(d)$ associated with performing this projection on each constraint and on the objective. Each resulting residual will be made as small as possible by exacting that they be orthogonal to a set of weight functions $\psi_k(d)$, $k \in \mathcal{K}$, $d \in \mathcal{D}$:

$$\langle R, \psi_k \rangle_w = 0, \quad \forall k \in \mathcal{K} \quad (33)$$

where $\langle \cdot, \cdot \rangle_w$ denotes the inner product between functions using the appropriate weighting function $w(d)$ for the given space:

$$\langle R, \psi_k \rangle_w = \int_{d' \in \mathcal{D}} R(d') \psi_k(d') w(d') dd'. \quad (34)$$

The weight functions are typically chosen such that $|\mathcal{K}| = |\mathcal{I}|$. The projection results in a tractable finite-dimensional formulation; what remains is our choice of the weight functions $\psi_k(\cdot)$. This choice gives rise to a range of techniques; if the Galerkin method is applied then we choose $\phi_k(d) = \psi_k(d)$ and have that $\mathcal{I} = \mathcal{K}$. This induces the first $|\mathcal{I}|$ terms of the residuals in the trial functions to vanish if the functions are orthogonal (Graham and Rawlings, 2013). Another popular choice is that of orthogonal collocation, where we choose $\psi_k(d) = \delta(d - \hat{d}_k)$; here, the set \hat{d}_k , $k \in \mathcal{K}$ denote collocation points (i.e., particular infinite parameter supports) and $\delta(\cdot)$ is the Dirac delta function. This approach seeks to enforce that the residual is zero at the collocation points. When orthogonal basis functions are chosen and this is applied over a set

of transcription points (i.e., finite elements), we obtain a method known as orthogonal collocation over finite elements. A variety of other methods such as least squares and the method of moments can also be employed to weight the residuals (these are discussed in detail in Finlayson, 2013).

The transformation of (24) to a finite-dimensional form via MWR is, in general, a nonlinear transformation (depending on the choices of the trial functions $\phi_i(\cdot)$, weight functions $\psi_k(\cdot)$, and their corresponding space). However, there exist special cases where the transformation is linear, as is often the case with PCE transformations (Mühlpfordt et al., 2019). Advantages of employing MWR instead of direct transcription is that one obtains functional representations for the infinite variables (as opposed to values at the support points), one can achieve better stability for boundary-valued problems, and one can obtain better accuracy for certain formulations (Devolder et al., 2010). On the other hand, the main disadvantage of MWR is that evaluating differential and measure operators and inner products tends to be cumbersome (especially for nonlinear formulations). Also, basis functions can be difficult to derive for formulations with multivariate infinite domains. In our abstraction, we provide the modeling elements that facilitate the implementation of these transformation techniques.

3.3. Transformation framework in *InfiniteOpt.jl*

In Section 2.8 we discussed how our unifying abstraction is implemented in *InfiniteOpt.jl*; one creates a model as an *InfiniteModel* object. In this section, we discuss a general transformation framework incorporated into *InfiniteOpt.jl* that facilitates the implementation of different transformation approaches (e.g., direct transcription and MWRs). We also outline the efficient direct transcription capabilities that are currently implemented.

The framework centers around applying a transformation to the *InfiniteModel* that converts it to a standard *JuMP.jl* Model object (referred to as an optimizer model in this context). The optimizer model can then be solved using the optimizers implemented in *MathOptInterface.jl* (Legat et al., 2020). Moreover, this framework features a collection of methods to enable a seamless interface between the *InfiniteModel* and its corresponding optimizer model to facilitate capabilities such as information extraction (e.g., solution queries) that do not require direct interrogation of the optimizer model. This framework is summarized in Fig. 15. This software structure distinguishes *InfiniteOpt.jl* from other software tools (e.g., *Pyomo.dae* and *Gekko*) whose implementations center around (and are limited to) direct transcription. Thus, *InfiniteOpt.jl* is unique in providing a flexible API for solving *InfiniteOpt* problems.

Following this framework, a desired solution scheme is incorporated by defining a few prescribed methods (principally `build_optimizer_model!`) to implement the associated transformation. This methodology is implicitly invoked on line 26 of Code Snippet 1 where `optimize!` creates an optimizer model us-

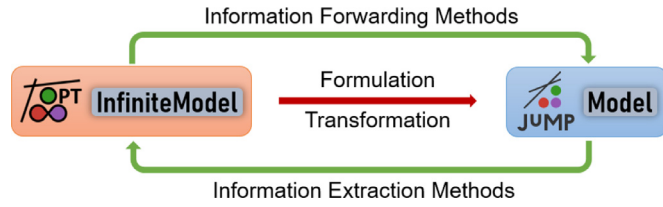


Fig. 15. Transformation framework employed by InfiniteOpt.jl for converting an InfiniteModel into a JuMP.jl Model.

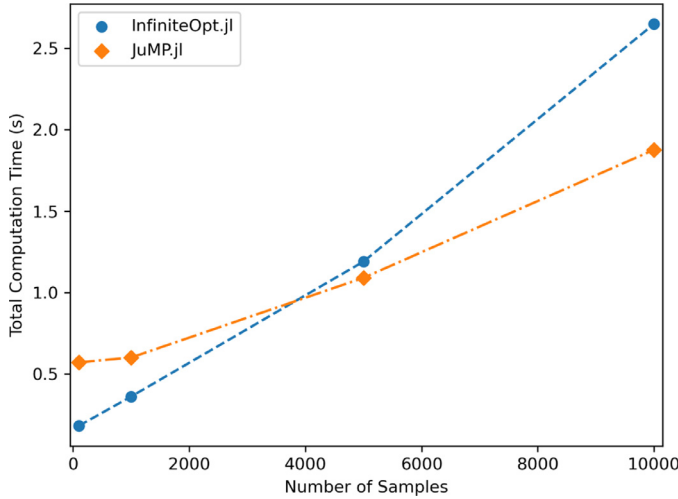


Fig. 16. Juxtaposition of the total computation time used to formulate and solve a stochastic optimization problem (Pulsipher and Zavala, 2019) using MC sampling implemented in InfiniteOpt.jl v0.4.1 and manual transcription in JuMP.jl v0.21.8.

ing the prescribed transformation and then solves it with the desired optimizer. The full technical detail of this API is beyond the scope of this work and is available via the InfiniteOpt.jl documentation.

InfiniteOpt.jl provides an efficient implementation of direct transcription following the procedures described in Section 3.1; this serves as the default transformation technique for InfiniteOpt models. These techniques are implemented in a submodule called TranscriptionOpt that follows the optimizer model framework shown in Fig. 15. The TranscriptionOpt module features a sophisticated finite support generation and management system that enables tackling a wide variety of infinite-dimensional optimization formulations using diverse evaluation techniques for measure and derivative operators. Moreover, its automatic transcription is efficient and compares competitively to manually transcribing a problem and implementing it via JuMP.jl. This incredible behavior is demonstrated in Fig. 16 where a two-stage stochastic optimization problem (the 3-node distribution network example featured in Pulsipher and Zavala, 2019) is solved for a range of MC samples using automatic transcription in InfiniteOpt.jl and manual transcription in JuMP.jl. We note that, contrary to other software implementations, automatic transcription in InfiniteOpt.jl denotes a minor computational expense relative to manual transcription with the benefit of avoiding the errors commonly incurred by transcribing InfiniteOpt formulations manually.

4. Innovations enabled by unifying abstraction

In this section, we discuss innovative modeling approaches that are enabled by the proposed InfiniteOpt abstraction. Specifically, these innovations are facilitated by the ability to transfer model-

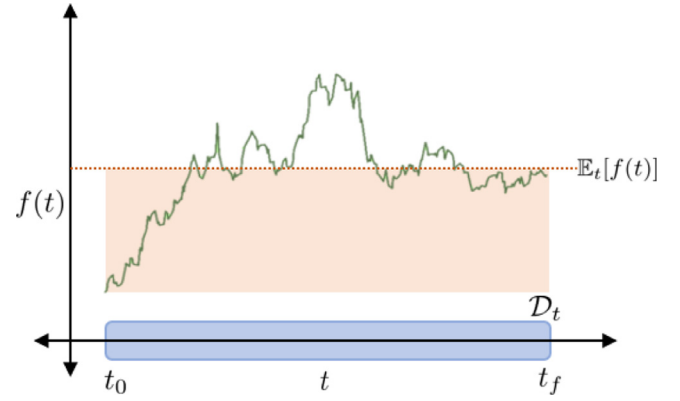


Fig. 17. Visualization of the expectation measure $\mathbb{E}_t[f(t)] = \frac{1}{t_f - t_0} \int_{t_0}^{t_f} f(t) dt$ where the rectangle formed has an area equal to that of the region under $f(t)$.

ing techniques and constructs across disciplines. For instance, in the proposed abstraction, there is no explicit notion of time, space, or random domains (all domains are treated mathematically in the same way); as such, one can easily identify analogues of modeling elements across disciplines.

4.1. Measure operators

Here we highlight the advantages of using an abstraction that focuses on measure operators. We will highlight how InfiniteOpt formulations from different disciplines are connected via analogous mathematical features; we place particular attention to common features arising in dynamic and stochastic optimization problems.

4.1.1. Expectation measures

In Section 2.4, we provide examples of typical measures used to formulate objective functions in InfiniteOpt problems (space-time integrals and expectations). For simplicity in the presentation, we consider a temporal domain \mathcal{D}_t and consider the time integral:

$$\int_{t \in \mathcal{D}_t} f(t) dt \quad (35)$$

where we write $f(t) = f(y(t), t)$ for compactness. Minimizing the measure (35) seeks to shape the cost function (a surface defined in the domain \mathcal{D}_t) in a way that it displaces the entire surface. Analogously, minimizing the expectation measure:

$$\mathbb{E}_\xi[f(\xi)] := \int_{\xi \in \mathcal{D}_\xi} f(\xi) p(\xi) d\xi \quad (36)$$

shapes the cost surface (defined in the domain \mathcal{D}_ξ) in a way that displaces the surface. An obvious difference between time integral and the random expectation is that the random expectation is weighted by a probability density function $p(\cdot)$ and this gives flexibility to put more/less emphasis on different locations of the random domain \mathcal{D}_ξ . Thus, as a simple example on how one can transfer modeling strategies, we can formulate the expectation over the temporal domain using a weighting function as:

$$\mathbb{E}_t[f(t)] := \int_{t \in \mathcal{D}_t} f(t) w(t) dt. \quad (37)$$

Note that the selection of the notation t to denote the infinite domain is arbitrary; one can simply define a general infinite parameter d . If one defines $w(t) = 1/S$ with $S = \int_{t \in \mathcal{D}_t} dt$, we can see that the above measure is the time average of $f(t)$ (with equal weight placed at each location in the domain). If the time domain is $\mathcal{D}_t = [t_0, t_f]$, we have that $S = t_f - t_0$. Fig. 17 provides a geometric interpretation of the time-expectation; here, the area of the rectangle with height $\mathbb{E}_t[f(t)]$ and width S is equivalent to

the area under $f(t)$ (Stewart, 2009). This is just a scaled version of the integral measure (35) and will thus shape $f(t)$ in similar manner. In fact, the integral (35) is just a special case of measure (37) (obtained by setting $w(t) = 1$); also, note that this approach is equivalent to using a weighting function $w(t)$ that corresponds to the probability density function of a uniform random parameter $\xi \sim \mathcal{U}(t_0, t_f)$. As such, one can envision defining weighting functions associated with probability densities of different random parameters (e.g., Gaussian, exponential, Weibull); this would have the effect of inducing interesting prioritization strategies that can be used to shape the cost surface in desirable ways. For instance, a Gaussian weighting function places emphasis on the middle of the time domain (and emphasis decays rapidly as one moves away from the center of the domain), while an exponential weighting function places emphasis at the beginning of the domain (and decays rapidly as one marches in time). This modeling feature can be useful in dynamic optimization and optimal control problems in which it is often desirable to place more/less emphasis on initial or final conditions. For instance, in infinite-horizon problems, $w(\cdot)$ obtained from the probability density function of an exponential density function behaves as a discount factor (Petrik and Scherrer, 2008; Shin et al., 2021).

4.1.2. Risk measures

A large collection of risk measures have been proposed in the stochastic optimization literature to manipulate random cost functions in desirable ways (e.g., to minimize impacts of extreme events) (Ruszczynski and Shapiro, 2006). In this context, risk measures are typically interpreted as measures that aim to shape the tail of the probability densities of cost or constraint functions. A popular risk measure used for this purpose is the conditional value-at-risk (CVaR):

$$\text{CVaR}_\xi(f(\xi); \alpha) := \min_{\hat{f} \in \mathbb{R}} \left\{ \hat{f} + \frac{1}{1-\alpha} \mathbb{E}_\xi(\max(f(\xi) - \hat{f}, 0)) \right\} \quad (38)$$

where $\alpha \in [0, 1)$ is a desired probability level and \hat{f} is an auxiliary variable.

One can show that the value of the auxiliary variable that minimizes the inner function of CVaR is given by $\hat{f}^* = Q_\xi(f(\xi); \alpha)$, which is the α -quantile of $f(\xi)$ (Rockafellar et al., 2000). We recall that the quantile is defined as:

$$Q_\xi(f(\xi); \alpha) := \inf_{\hat{f} \in \mathbb{R}} \left\{ \mathbb{P}_\xi(f(\xi) \leq \hat{f}) \geq \alpha \right\}. \quad (39)$$

The quantile is thus the threshold value for $f(\xi)$ such that the probability of finding this function below the threshold is at least α .

One can also show that CVaR is a conditional expectation of the form:

$$\text{CVaR}_\xi(f(\xi); \alpha) = \mathbb{E}_\xi[f(\xi) \mid f(\xi) \geq Q_\xi(f(\xi); \alpha)] \quad (40)$$

Hence, minimizing CVaR has the effect of minimizing the conditional expectation over the $1 - \alpha$ probability region with the highest cost, thus hedging against extreme events. Moreover, the calculation of CVaR at a probability level α implicitly defines the quantile of $f(\xi)$.

A key property of CVaR is that it is a general measure that captures the expectation $\text{CVaR}_\xi(f(\xi); \alpha) = \mathbb{E}_\xi[f(\xi)]$ as $\alpha \rightarrow 0$ and the worst-case $\text{CVaR}_\xi(f(\xi); \alpha) = \sup_\xi(f(\xi))$ as $\alpha \rightarrow 1$. Fig. 18 shows how these measures are typically interpreted in terms of the probability density function of the cost $f(\xi)$, motivating the α and $1 - \alpha$ probability regions.

Through the perspective of the proposed unifying abstraction, one can interpret CVaR as a measure that captures the excursion of a function (field) from a given threshold. To see this, we define the

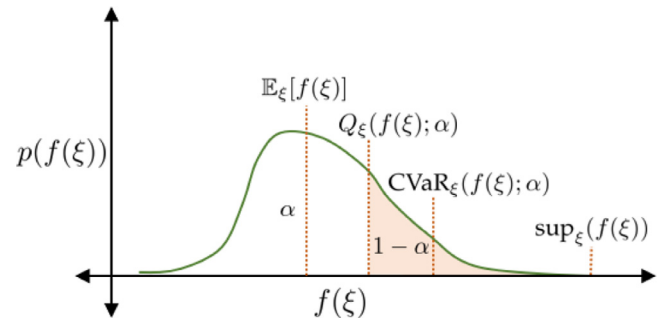


Fig. 18. An illustration of $\text{CVaR}_\xi(f(\xi); \alpha)$ in terms of the probability density function $p(f(\xi))$.

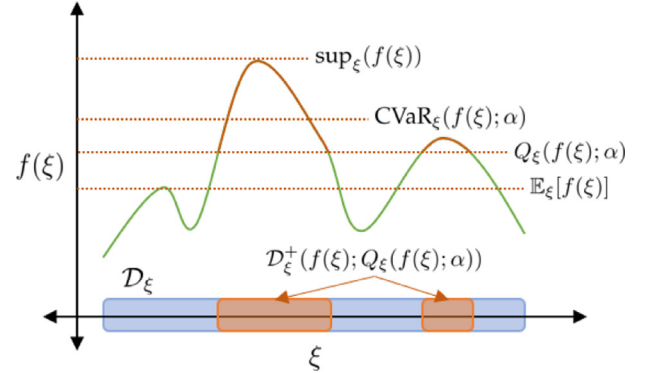


Fig. 19. Illustration of $\text{CVaR}_\xi(f(\xi); \alpha)$ following the representation given in (43). This provides an alternative view of the probabilistic representation shown in Fig. 18.

positive and negative function excursion sets of $f(\xi)$ which denote the range of $f(\xi)$ above and below a threshold \hat{f} , respectively:

$$\begin{aligned} \mathcal{D}_\xi^+(f(\xi); \hat{f}) &:= \{\xi \in \mathcal{D}_\xi : f(\xi) \geq \hat{f}\} \\ \mathcal{D}_\xi^-(f(\xi); \hat{f}) &:= \{\xi \in \mathcal{D}_\xi : f(\xi) \leq \hat{f}\} \end{aligned} \quad (41)$$

where $\mathcal{D}_\xi^+(f(\xi); \hat{f}) \subseteq \mathcal{D}_\xi$ and $\mathcal{D}_\xi^-(f(\xi); \hat{f}) \subseteq \mathcal{D}_\xi$ are the positive and negative function excursion sets, respectively. We simplify the notation for these excursion sets by using $\mathcal{D}_\xi^-(\hat{f})$ and $\mathcal{D}_\xi^+(\hat{f})$. Using these definitions, we can express the quantile function $q_\alpha := Q_\xi(f(\xi); \alpha)$ as:

$$q_\alpha = \inf_{\hat{f} \in \mathbb{R}} \left\{ \int_{\xi \in \mathcal{D}_\xi^-(\hat{f})} p(\xi) d\xi \geq \alpha \right\}. \quad (42)$$

This reveals that CVaR considers the expectation of $f(\xi)$ over the restricted domain $\mathcal{D}_\xi^+(q_\alpha)$, which indexes the $1 - \alpha$ probabilistic region shown in Fig. 18:

$$\text{CVaR}_\xi(f(\xi); \alpha) = \frac{1}{1-\alpha} \int_{\xi \in \mathcal{D}_\xi^+(q_\alpha)} f(\xi) p(\xi) d\xi. \quad (43)$$

Fig. 19 illustrates that $\text{CVaR}_\xi(f(\xi); \alpha)$ using this functional interpretation for a realization of $f(\xi)$ and compares it to the other measures shown in Fig. 18.

The representation of CVaR shown in (43) highlights that this can be seen as a measure operator that can be applied to different types of infinite domains (e.g., space and time). For instance, shaping time-dependent trajectories to minimize extreme values is applicable to dynamic optimization (e.g., to prevent peak costs or extreme operating conditions that can hinder safety). For instance, Risbeck and Rawlings, 2019 recently proposed a model predictive control formulation that has the dual goal of minimizing total ac-

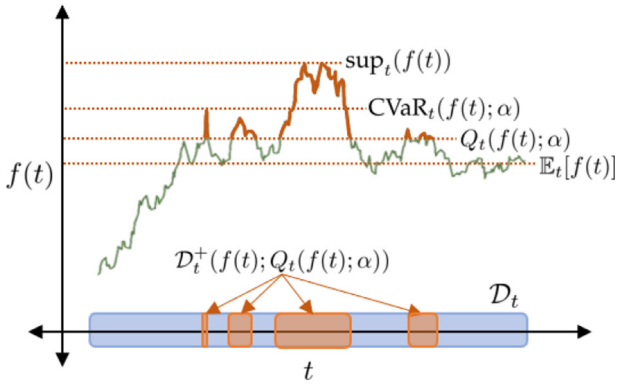


Fig. 20. Illustration of $\text{CVaR}_t(f(t); \alpha)$, as represented in (45).

cumulated cost and peak cost:

$$\int_{t \in \mathcal{D}_t} f(t) dt + \lambda \cdot \max_{t \in \mathcal{D}_t} f(t) \quad (44)$$

where $\lambda \in \mathbb{R}_+$ is a trade-off parameter. More generally, we might want to penalize a subdomain of the time domain that attains the highest costs; this can be achieved by applying a CVaR measure operator to the time-dependent cost $f(t)$:

$$\text{CVaR}_t(f(t); \alpha) = \frac{1}{1-\alpha} \int_{t \in \mathcal{D}_t^+(f(t); \alpha)} f(t) p(t) dt \quad (45)$$

where the density function can be selected as $p(t) = \frac{1}{t_f - t_0}$. Note that this definition implicitly introduces the notion of a quantile in a time domain; this quantile is given by:

$$q_\alpha := Q_t(f(t); \alpha) = \inf_{\tilde{f} \in \mathbb{R}} \left\{ \int_{t \in \mathcal{D}_t^-(f(t); \tilde{f})} p(t) dt \geq \alpha \right\}. \quad (46)$$

Using the properties of CVaR we can conclude that:

$$\begin{aligned} \lim_{\alpha \rightarrow 0} \text{CVaR}_t(f(t); \alpha) &= \int_{t \in \mathcal{D}_t} f(t) p(t) dt \\ \lim_{\alpha \rightarrow 1} \text{CVaR}_t(f(t); \alpha) &= \max_{t \in \mathcal{D}_t} f(t). \end{aligned} \quad (47)$$

This highlights that CVaR provides an intuitive measure that can help shape time-dependent trajectories. Fig. 20 illustrates the application of this measure over the time domain and shows that this is analogous to the application over a random domain shown in Fig. 19. We highlight that, in practice, $\text{CVaR}_t(f(t); \alpha)$ is computed by using (38) (defined for t instead of ξ).

This example illustrates how the CVaR construct used in stochastic optimization can be utilized to motivate new formulations for other optimization disciplines. The measure-centric unifying abstraction facilitates this process by capturing objects via measure operators. The process of transferring ideas through the measure-centric abstraction is also amendable to other risk measures (see Krokhmal et al., 2013 for a review of risk measures). We present a numerical study of using CVaR for an optimal control formulation in Section 5.2.

4.1.3. Event constraints

An interesting modeling paradigm that arises from the proposed abstraction are *event constraints*. These constraints generalize the notion of chance constraints, excursion set conditions, and exceedance probabilities that are used in different scientific disciplines (e.g., stochastic optimization, reliability engineering, and random fields). This unified view also promotes transfer of modeling concepts across disciplines; for instance, we will see that chance constraints in a random domain are analogous to exceedance times in a time domain.

To exemplify the notion of event constraints, consider the so-called *excursion time*; this measure is widely used in reliability analysis of dynamical systems and is defined as the fraction of time that a function $h(t)$, $t \in \mathcal{D}_t$ is above a given threshold (Au and Beck, 2001). Here, we consider a zero threshold value to give the event constraint:

$$\mathbb{P}_t(\{\exists t \in \mathcal{D}_t : h(t) > 0\}) \leq \alpha. \quad (48)$$

where $\alpha \in [0, 1]$. In the context of an InfiniteOpt problem, the function $h(t)$, $t \in \mathcal{D}_t$ can denote a constraint $h(y(t), z, t)$, $t \in \mathcal{D}_t$. The excursion time is expressed as a probability-like measure of the form:

$$\mathbb{P}_t(\{\exists t \in \mathcal{D}_t : h(t) > 0\}) = \int_{t \in \mathcal{D}_t} \mathbb{1}[\{\exists t \in \mathcal{D}_t : h(t) > 0\}] w(t) dt \quad (49)$$

where $w : \mathcal{D}_t \rightarrow [0, 1]$ is a weighting function satisfying $\int_{t \in \mathcal{D}_t} w(t) dt = 1$. The excursion time measure can be interpreted as the *fraction of time* under which the event of interest occurs; for instance, in safety analysis, one might be interested in determining the fraction of time that a constraint is violated and to ensure that this is not greater than some fraction α . Alternatively, we could also search to minimize this measure (by using it as an objective). The excursion time constraint is an event constraint that can help shape a time-dependent trajectory in interesting and non-intuitive ways.

One can generalize the excursion time measure by constructing complex events. For instance, consider that we want to determine the fraction of time that any of the time-varying constraints $h_k(t)$, $k \in \mathcal{K}$ crosses a threshold. This can be expressed as:

$$\mathbb{P}_t\left(\bigcup_{k \in \mathcal{K}} \{\exists t \in \mathcal{D}_t : h_k(t) > 0\}\right). \quad (50)$$

Here, \bigcup is a logical or operator. If we want to determine fraction of time that all constraints are violated then we use:

$$\mathbb{P}_t\left(\bigcap_{k \in \mathcal{K}} \{\exists t \in \mathcal{D}_t : h_k(t) > 0\}\right). \quad (51)$$

Here, \bigcap is a logical and operator.

To highlight transfer across different disciplines, we recognize that the excursion time is directly analogous to a chance constraint (operating in the random space, as opposed to time) and our previous analysis suggests that one can construct chance constraints that capture complex events. For instance, consider the event constraint:

$$\mathbb{P}_\xi\left(\bigcup_{k \in \mathcal{K}} \{\exists \xi \in \mathcal{D}_\xi : h_k(\xi) > 0\}\right) \leq \alpha. \quad (52)$$

Here, we see that the constraint is directly analogous to the event constraint (50). Fig. 21a illustrates the logical event space that constraint (52) shapes. We thus see that events can be defined in a general form over different infinite domains; to highlight this fact, we consider the event constraint:

$$\mathbb{P}_d\left(\bigcup_{k \in \mathcal{K}} \{\exists d \in \mathcal{D} : h_k(d) > 0\}\right) \leq \alpha. \quad (53)$$

The previous event constraint is different to traditional joint-chance constraints used in stochastic optimization:

$$\mathbb{P}_\xi\left(\bigcap_{k \in \mathcal{K}} \{\exists \xi \in \mathcal{D}_\xi : h_k(\xi) \leq 0\}\right) \leq \alpha. \quad (54)$$

This makes it more readily apparent that traditional joint-chance constraints consider the logical event space that is complementary to that of constraint (52). Fig. 21b shows this region which is the logical complement of Fig. 21a.

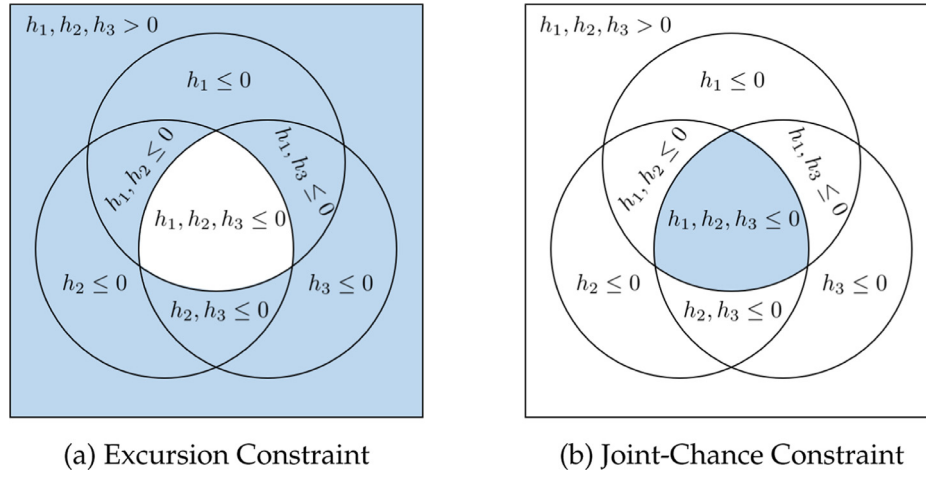


Fig. 21. Logical event regions (shown in blue) constrained by the event constraints (52) and (54). In particular, they constrain the condition $h_1 > 0 \cup h_2 > 0 \cup h_3 > 0$ and the condition $h_1 \leq 0 \cap h_2 \leq 0 \cap h_3 \leq 0$, respectively.

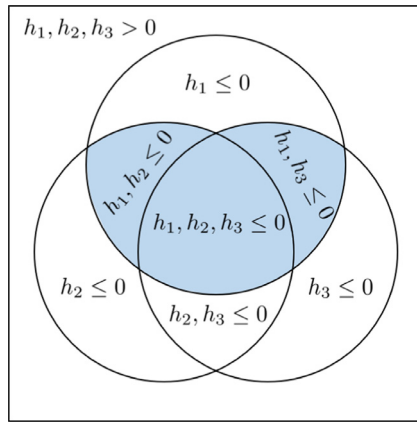


Fig. 22. Illustration of logical event region captured by constraint (55).

Another interesting insight that we draw from event constraints is that logical operators (e.g., \cap and \cup) can be used to model complex decision-making logic. For example, following the constraint system shown in Fig. 21; we might consider the logical event region derived from the condition that $h_1 \leq 0 \cap (h_2 \leq 0 \cup h_3 \leq 0)$ giving the event constraint:

$$\mathbb{P}_{\xi}(\{\forall \xi \in \mathcal{D}_{\xi} : h_1 \leq 0 \cap (h_2 \leq 0 \cup h_3 \leq 0)\}) \geq \alpha. \quad (55)$$

This is depicted in Fig. 22; we note that this event constraint encapsulates a wider probabilistic region relative to that of the traditional joint-chance constraint (54).

In summary, the presented examples illustrate that excursion time constraints and chance constraints as special cases of event constraints. This crossover also led us to representing joint-chance constraints with logical operators which introduce the notion of embedding problem-specific logic to shape the probabilistic region these constraints consider. We illustrate this example further with a stochastic optimal power flow case study in Section 5.1.

4.2. Random fields

We now discuss how the abstraction inspires the incorporation of modeling concepts from random field theory into InfiniteOpt formulations. A random field is a random function with realizations of the form $f(d) : \mathcal{D} \mapsto \mathbb{R}^{n_f}$ (Adler, 2010). For instance, one can think of a continuous-time trajectory that is random (e.g., due

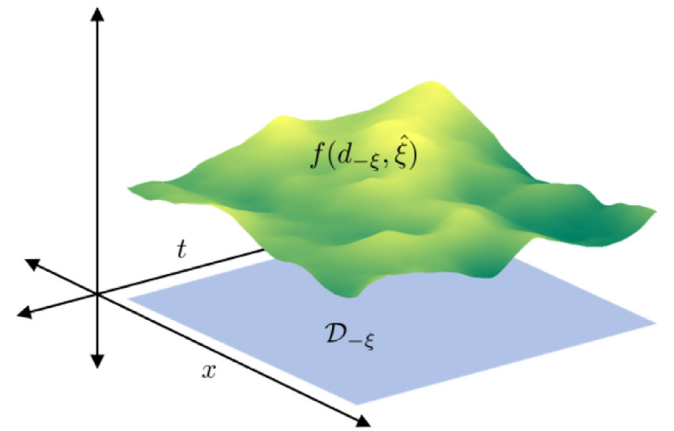


Fig. 23. Sample of a random field defined over space and time.

to uncertainty in a differential equation model that generates it). A random field generalizes the notion of multivariate random variables (e.g., a multivariate Gaussian $\xi \sim \mathcal{N}(\mu, \Sigma)$) that are jointly-distributed to that of infinite jointly-distributed random variables that are indexed over some continuous domain. Another example of a random field is that of a dynamic Gaussian process $\xi(t) \sim \mathcal{GP}(\mu(t), \Sigma(t, t'))$ for $t \in \mathcal{D}_t$. Fig. 23 depicts a realization of a random field.

Modeling concepts from random field theory allows us to incorporate dependencies of a random parameter over other infinite domains (e.g., account for stochastic processes). For instance, in the context of our abstraction, consider the random time-dependent function $y(t, \xi)$. This decision function is random due to the presence of $\xi \in \mathcal{D}_{\xi}$ and is also indexed over $t \in \mathcal{D}_t$; this means that it can be precisely characterized as a random field. From this it follows that all infinite variables $y(d_{-\xi}, \xi)$ are random fields. Hence, optimally choosing $y(d_{-\xi}, \xi)$ in a given formulation amounts to engineering a random field that is in general correlated over the infinite domain $\mathcal{D}_{-\xi}$. This important observation enables us to connect a wide breadth of optimization disciplines to random field theory. For instance, the theory and methods behind random field constructs like excursion sets, excursion probabilities, and expected Euler characteristics are amenable to the measure operators and/or event constraints formalized in our abstraction (see Sections 4.1.2 and 4.1.3) (Adler, 2000).

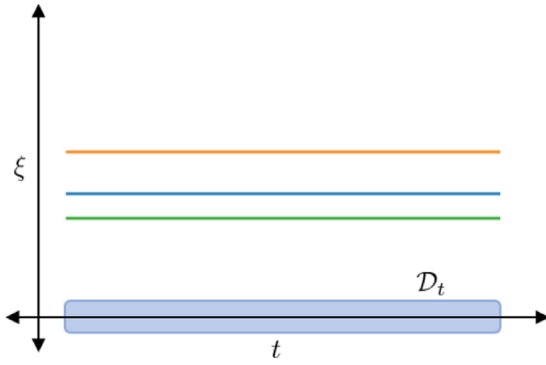


Fig. 24. Realizations of a random parameter $\xi \in \mathcal{D}_\xi$ that are invariant relative to other infinite domains (e.g., time t).

The particular class of infinite variables discussed up to this point only consider a random parameter ξ that is invariant over other domains, as exemplified for a few realizations of ξ in Fig. 24. This means that, although the infinite variables in this class of formulations are general random fields, the input uncertainty ξ is taken to be *static* random parameter that does not capture any spatial or temporal correlation. We can overcome this modeling limitation by extending our abstraction to consider infinite parameter functions (e.g., random parameters that are functions of other infinite parameters). In the context of our current example, this is accomplished by defining the infinite parameter function $\xi(t) \in \mathcal{D}_{\xi(t)}$, $t \in \mathcal{D}_t$ which denotes a known random field whose sample domain $\mathcal{D}_{\xi(t)}$ is a set of temporal functions. With this extension, we can now define the infinite variable $y(t, \xi(t))$, which denotes a random field variable (an infinite variable) where $\xi(t)$ is a known random field that can capture correlations of ξ over time t . Note that $\xi(t)$ is explicitly shown as an input to y so that it is distinguished from deterministic infinite-dimensional variables $y(t)$. Fig. 25 shows a realization of $\xi(t)$ in the case that it exhibits temporal correlation and in the case that ξ approaches no temporal correlation. Note this can also capture the static case shown in Fig. 24.

Random fields have been widely used in disciplines such as robust topology design optimization (Zhao and Wang, 2014; Zhang et al., 2018); however, to the best of our knowledge, the integration of random field theory in optimization represents a new class of optimization formulations. We call this class *random field optimization* and we note that this is a special case of the proposed InfiniteOpt abstraction. This class provides effective means of capturing correlation of uncertainty over other domains an optimization formulation which is enabled by the rich theory that has been developed for random fields (Adler, 2010). Moreover, random field optimization can be seen as a *generalization of stochastic optimization*. For instance, it provides an intriguing alternative to multi-stage (discrete time) stochastic formulations which cannot be readily generalized to continuous time or general infinite domains (e.g., space) (Shapiro, 2003), whereas our proposed formulation class is defined for general infinite domains (e.g., space and time) and can incorporate random field uncertainty models such as Gaussian processes.

Another interesting observation is that extending the proposed abstraction to include infinite parameter functions can also be done for non-random infinite parameters. For instance, we could account for a time-dependent spatial parameter $x(t) \in \mathcal{D}_{x(t)}$ where $\mathcal{D}_{x(t)}$ is some space. We leave a rigorous formalization of such an extension to our abstraction and its implications for random field optimization problems to future work.

4.3. Problem analysis

Here we highlight some of the advantages that arise from characterizing InfiniteOpt problems directly in accordance with formulation (24), in contrast to the standard practice of expressing them solely via finite reformulations. For instance, within the area of dynamic optimization, it is commonplace to abstract and formalize problem classes in discrete time (i.e., in a transcribed form) (Rawlings et al., 2017). This practice tends to make problems more difficult to formulate since they are inherently coupled with the transformation scheme employed (e.g., orthogonal collocation or explicit Euler). Hence, decoupling the problem definition from its transformation helps to ease its formalization. Arguably, this decoupling better defines a particular problem and promotes the use of diverse formulations and transformation techniques. For instance, by operating at a different level of abstraction, one might more easily identify alternative modeling and solution techniques: different measures, non-traditional support schemes, alternative derivative approximations, and/or advanced problem transformations. For instance, analyzing the problem in its infinite-dimensional form is what inspired the case study discussed in Section 5.2; this example shows how to use CVaR as a way to manipulate time-dependent trajectories.

Establishing theoretical properties for InfiniteOpt formulations is also often facilitated when these problems are expressed in their native form. This is exemplified in the recent work of Faulwasser and Grüne (2020), where the authors utilize continuous and discrete time formulations to derive properties of the turnpike phenomenon in the field of optimal control. The turnpike phenomenon refers to the propensity of optimal control trajectories to remain within a certain region for a significant portion of the time horizon until later departing it. Fig. 26 illustrates this for a dynamic variable $y(t)$. The properties discussed by the authors with regard to turnpike behavior are beyond the scope of this work but, interestingly, they observe that a considerable amount of analysis has been done for finite time formulations whereas many conceptual gaps remain for the continuous-time case. This conceptual disparity between the continuous and discrete time cases can at least in part be attributed to the rather standard practice of expressing optimal control formulations in discrete time. This observation is not unique to the optimal control community and there exists much to be explored for InfiniteOpt formulations in their native forms throughout their respective communities in general. Some potential avenues of research for InfiniteOpt formulations include systematic initialization techniques that consider the formulation infinite domain, generalized pre-solving methods (e.g., feasibility checking), and enhanced transformations (e.g., basis function approaches used in Georgakis, 2013 and Gnegel et al., 2021).

The proposed abstraction also encourages a more systematic treatment of infinite-dimensional data and modeling objects. To illustrate this, we consider dynamic parameter estimation formulations and show how lifting a finite-dimensional data representation into an infinite-dimensional form might be beneficial. We consider conducting dynamic experiments $k \in \mathcal{K}$ that collect the set of observations (data) $\{\tilde{y}_k(t) : t \in \hat{\mathcal{D}}_k, k \in \mathcal{K}\}$ over the set of time points $t \in \hat{\mathcal{D}}_k$. A dynamic parameter estimation formulation then seeks the optimal choice of parameters $z \in \mathcal{Z}$ to fit and verify the efficacy of a candidate model $g(y(t), z, t) = 0$ relative to empirical data (Shin et al., 2019; Venturelli et al., 2018). For simplicity in example, we consider a least-squares approach that yields the following canonical discrete-time estimation formulation:

$$\begin{aligned} \min_{y_k(\cdot), z} \quad & \sum_{k \in \mathcal{K}} \sum_{t \in \hat{\mathcal{D}}_k} (y_k(t) - \tilde{y}_k(t))^2 \\ \text{s.t.} \quad & \hat{g}(y(t), z, t) = 0, \quad t \in \hat{\mathcal{D}}_k, k \in \mathcal{K} \\ & z \in \mathcal{Z} \end{aligned} \quad (56)$$

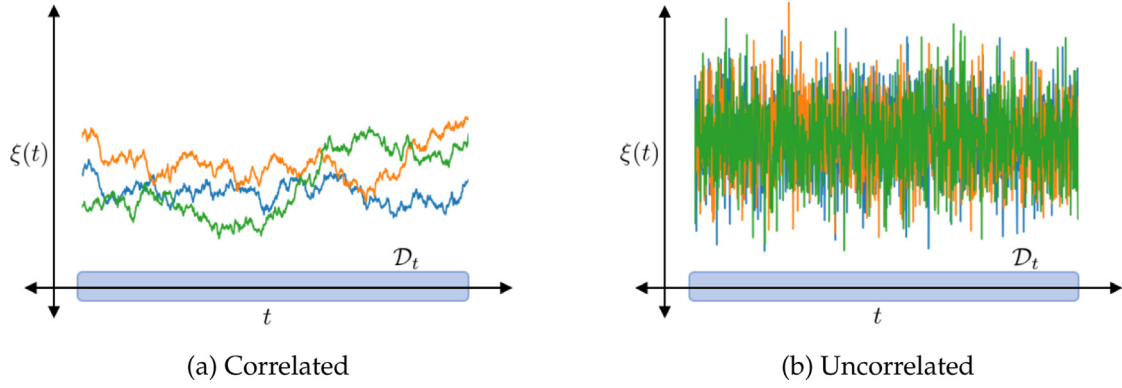


Fig. 25. Realizations of a time-dependent random field $\xi(t)$ with and without time correlation.

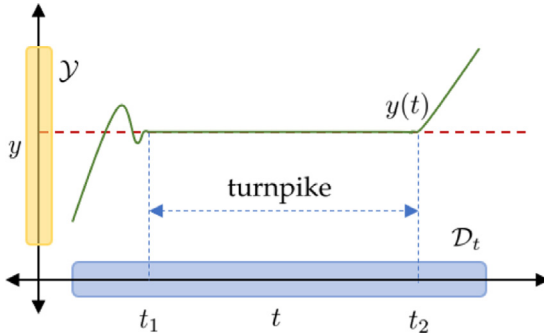


Fig. 26. Illustration of the turnpike phenomenon for a time-dependent trajectory $y(t)$ where the turnpike occurs on the interval $[t_1, t_2]$.

where $\hat{g}(\cdot)$ is the discretized dynamic model and $y_k(\cdot)$ are the model predicted variables. This discrete representation is guided by the nature of the experimental data $\tilde{y}_k(t)$ available, which corresponds to a finite set of time points $t \in \hat{\mathcal{D}}_{t_k}$. This limits the model to a particular transcribed domain; however, we can express formulation (56) in continuous time by representing the experimental data with appropriate infinite-dimensional lifting functions $f_k(t)$, $t \in \mathcal{D}_{t_k}$ such that $f_k(t) = \tilde{y}(t)$ at $t \in \hat{\mathcal{D}}_{t_k}$:

$$\begin{aligned} \min_{y_k(\cdot), z} \quad & \sum_{k \in \mathcal{K}} \left(\int_{t \in \mathcal{D}_{t_k}} (y_k(t) - \tilde{y}_k(t))^2 dt \right) \\ \text{s.t.} \quad & g(y(t), z, t) = 0, \quad t \in \mathcal{D}_{t_k}, k \in \mathcal{K} \\ & \tilde{y}_k(t) = f_k(t), \quad t \in \mathcal{D}_{t_k}, k \in \mathcal{K} \\ & z \in \mathcal{Z}. \end{aligned} \quad (57)$$

We now have a formulation that fits into our unifying InfiniteOpt abstraction; as such, we can begin to consider general modeling elements and transformations. This means, for instance, that we might want to consider alternative time-dependent measures or more accurate derivative approximations (e.g., orthogonal collocation over finite elements) (Tjoa and Biegler, 1991). Fig. 27 demonstrates this principle for a certain derivative Dy_k . This approach also has the potential to alleviate the large computational burden associated with the noisy experimental data that often plague dynamic estimation (Ramsay et al., 2007), since the chosen empirical data functions $f_k(\cdot)$ smooth the empirical domains as a preprocessing step (see Fig. 32 in the case study). The data functions also facilitate the computation of data derivatives; which can be used in estimation techniques such as SINDy. We leave a more rigorous analysis of formulation (57) to future work but we hope that this discussion helps illustrate how lifting can help identify new perspectives to tackle problems that are typically treated as finite-

dimensional. We study this approach further in the biological dynamic parameter estimation case study presented in Section 5.3.

5. Case studies

In this section, we provide illustrative case studies to demonstrate the concepts discussed. These case studies seek to exemplify how the unifying abstraction captures a wide range of formulation classes and how it drives innovation. These cases are implemented via InfiniteOpt.jl v0.4.1 using Ipopt v3.13.2 for continuous problems and Gurobi v9.1.1 for integer-valued problems on an Intel® Core™ i7-7500U machine running at 2.90 GHz with 4 hardware threads and 16 GB of RAM running Windows 10 Home. All scripts needed to reproduce the results can be found in <https://github.com/zavalab/JuliaBox/tree/master/InfiniteDimensionalCases>.

5.1. Event-constrained optimal power flow

We apply the event constraints featured in Section 4.1.3 to a stochastic optimal power flow (SOPF) formulation (a stochastic optimization problem). We base our SOPF formulation as a variant of the chance-constrained formulation presented in Mühlpfordt et al. (2018). This considers DC power grid networks subject to random power demands $\xi \in \mathcal{D}_\xi \subseteq \mathbb{R}^{n_\xi}$. The optimal policy defines the power generation $y_g(\xi) \in \mathcal{Y}_g \subseteq \mathbb{R}^{n_g}$ and the branch power flow $y_b(\xi) \in \mathcal{Y}_b \subseteq \mathbb{R}^{n_b}$ recourse functions to satisfy the demands where the respective feasible sets denote the engineering limits (i.e., $\mathcal{Y}_g = [0, \bar{y}_g]$ and $\mathcal{Y}_b = [-\bar{y}_b, \bar{y}_b]$). The power model enforces a linear energy balance at each node of the form:

$$Ay_b(\xi) + C_g y_g(\xi) - C_\xi \xi = 0, \quad \xi \in \mathcal{D}_\xi \quad (58)$$

where $A \in \mathbb{R}^{n_n \times n_b}$ is the incidence matrix, $C_g \in \mathbb{R}^{n_n \times n_g}$ maps the generators to the correct nodes, and $C_\xi \in \mathbb{R}^{n_n \times n_\xi}$ maps the demands to the correct nodes.

A traditional joint-chance constraint enforces limits to a certain probabilistic threshold α :

$$\begin{aligned} \mathbb{P}_\xi \left(\left(\bigcap_{i \in \mathcal{I}_g} \{ \forall \xi \in \mathcal{D}_\xi : y_{g,i}(\xi) \leq \bar{y}_{gi} \} \right) \right. \\ \left. \bigcap \left(\bigcap_{i \in \mathcal{I}_b} \{ \forall \xi \in \mathcal{D}_\xi : -\bar{y}_{bi} \leq y_{b,i}(\xi) \leq \bar{y}_{bi} \} \right) \right) \geq \alpha \end{aligned} \quad (59)$$

where \mathcal{I}_g is the set of generator indices and \mathcal{I}_b is the set of branch indices. The non-negativity constraints on the generators are excluded such that they are enforced almost surely. The joint-chance

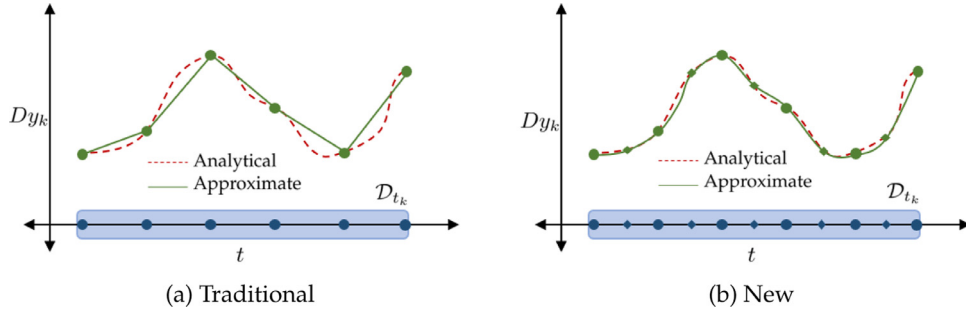


Fig. 27. Comparison between the derivative approximation approaches common to traditional dynamic estimation formulations and higher-order ones possible using our new formulation (e.g., using orthogonal collocation).

constraint thus enforces that all the engineering limits are satisfied to at least a probability α and constraint (59) is equivalent to:

$$\mathbb{E}_{\xi} \left[\mathbb{1} \left[\left(\bigcap_{i \in \mathcal{I}_g} y_{g,i}(\xi) \leq \bar{y}_{gi} \right) \cap \left(\bigcap_{i \in \mathcal{I}_b} -\bar{y}_{bi} \leq y_{b,i}(\xi) \leq \bar{y}_{bi} \right) \right] \right] \geq \alpha. \quad (60)$$

This representation can be reformulated into algebraic constraints by introducing an infinite binary variable $y_w(\xi) \in \{0, 1\}$ and an appropriate upper-bounding constant $U \in \mathbb{R}_+$ (Luedtke and Ahmed, 2008):

$$\begin{aligned} y_{g,i}(\xi) - \bar{y}_{gi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_g \\ -y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ \mathbb{E}_{\xi}[1 - y_w(\xi)] &\geq \alpha. \end{aligned} \quad (61)$$

Similarly, we can apply the excursion probability constraint; this enforces the probability that any engineering limit violation be no more than $1 - \alpha$:

$$\begin{aligned} \mathbb{P}_{\xi} \left(\left(\bigcup_{i \in \mathcal{I}_g} \{ \exists \xi \in \mathcal{D}_{\xi} : y_{g,i}(\xi) > \bar{y}_{gi} \} \right) \right. \\ \left. \bigcup \left(\bigcup_{i \in \mathcal{I}_b} \{ \exists \xi \in \mathcal{D}_{\xi} : |y_{b,i}(\xi)| > \bar{y}_{bi} \} \right) \right) \leq 1 - \alpha. \end{aligned} \quad (62)$$

This constraint is equivalent to a joint-chance constraint; this becomes apparent when we reformulate constraint (62) as:

$$\mathbb{E}_{\xi} \left[\mathbb{1} \left[\left(\bigcup_{i \in \mathcal{I}_g} y_{g,i}(\xi) > \bar{y}_{gi} \right) \cup \left(\bigcup_{i \in \mathcal{I}_b} |y_{b,i}(\xi)| > \bar{y}_{bi} \right) \right] \right] \leq 1 - \alpha. \quad (63)$$

and then use $y_w(\xi)$ and U to obtain:

$$\begin{aligned} y_{g,i}(\xi) - \bar{y}_{gi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_g \\ -y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_w(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ \mathbb{E}_{\xi}[y_w(\xi)] &\leq 1 - \alpha. \end{aligned} \quad (64)$$

These are equivalent to the set of constraints (61) since $\mathbb{E}_{\xi}[1 - y_w(\xi)] \geq \alpha$ implies $\mathbb{E}_{\xi}[y_w(\xi)] \leq 1 - \alpha$.

As an example of leveraging logical operators (e.g., \cap and \cup) to constraint more complex regions, we consider the probability that all the generator limits being satisfied or all the branch limits being satisfied:

$$\mathbb{E}_{\xi} \left[\mathbb{1} \left[\left(\bigcap_{i \in \mathcal{I}_g} y_{g,i}(\xi) \leq \bar{y}_{gi} \right) \cup \left(\bigcap_{i \in \mathcal{I}_b} -\bar{y}_{bi} \leq y_{b,i}(\xi) \leq \bar{y}_{bi} \right) \right] \right] \geq \alpha. \quad (65)$$

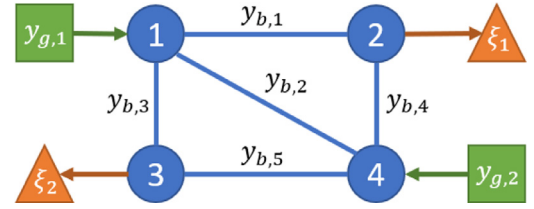


Fig. 28. Sketch of the 4-bus power network topology with its bus nodes (blue circles), branches (blue lines), generators (green squares), and demand loads (orange triangles). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

This encapsulates a wider probabilistic region relative to that of the joint-chance constraint (60). This can be reformulated into a system of algebraic constraints by following the same methodology outlined for the other event constraints; however, we need to use multiple infinite binary variables $y_{w,g}(\xi)$, $y_{w,b}(\xi)$, $y_{w,o}(\xi)$:

$$\begin{aligned} y_{g,i}(\xi) - \bar{y}_{gi} &\leq y_{w,g}(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_g \\ -y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_{w,b}(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ y_{b,i}(\xi) - \bar{y}_{bi} &\leq y_{w,b}(\xi)U, & \xi \in \mathcal{D}_{\xi}, i \in \mathcal{I}_b \\ y_{w,o}(\xi) &\geq y_{w,g}(\xi) + y_{w,b}(\xi) - 1, & \xi \in \mathcal{D}_{\xi} \\ \mathbb{E}_{\xi}[1 - y_{w,o}(\xi)] &\geq \alpha. \end{aligned} \quad (66)$$

We can now define the SOPF formulation; here, we aim to compare the use of constraint (60) with the use of constraint (65):

$$\begin{aligned} \min_{y_g(\xi), y_b(\xi)} \quad & \mathbb{E}_{\xi} [c_g^T y_g(\xi)] \\ \text{s.t.} \quad & A y_b(\xi) + C_g y_g(\xi) - C_{\xi} \xi = 0, & \xi \in \mathcal{D}_{\xi} \\ & y_g(\xi) \geq 0, & \xi \in \mathcal{D}_{\xi} \\ & (5.60) \text{ or } (5.65) \end{aligned} \quad (67)$$

where $c_g \in \mathbb{R}^{n_g}$ are generation unit costs. We apply this SOPF formulation to the 4-bus power system depicted in Fig. 28. We set $\bar{y}_{gi} = 10$, $\bar{y}_{bi} = 4$, $U = 100$, $c_g^T = [1 \ 10]$, and $\xi \sim \mathcal{N}(\mu, \Sigma)$ where

$$\mu = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}. \quad (68)$$

The matrices A , C_g , and C_{ξ} are determined by the topology shown in Fig. 28.

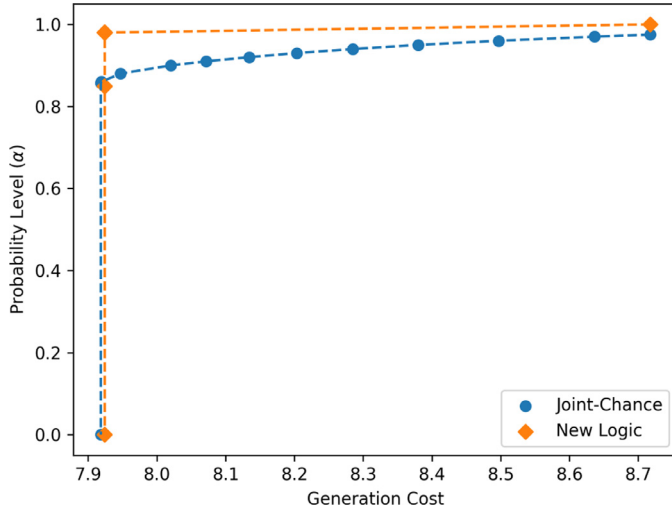


Fig. 29. Pareto frontiers associated formulation (67) where the joint-chance curve refers to using constraint (60) and the new logic curve corresponds to constraint (65).

Code Snippet 2: Formulation (5.67) with constraints (5.61) via `InfiniteOpt.jl` to obtain Pareto solutions.

```

1 using InfiniteOpt, Distributions, Gurobi
2
3 # Initialize the model
4 model = InfiniteModel(Gurobi.Optimizer)
5
6 # Define the parameters
7 @finite_parameter(model, α == 0.95)
8 @infinite_parameter(model, ξ[1:2] ~ MvNormal(μ, Σ), num_supports = 1000)
9
10 # Define the variables
11 @variables(model, begin
12     0 ≤ yg[1:2], Infinite(ξ)
13     yb[1:5], Infinite(ξ)
14     yw, Infinite(ξ), Bin
15 end)
16
17 # Set the objective
18 @objective(model, Min, E(cg' * yg, ξ))
19
20 # Add the constraints
21 @constraint(model, A * yb .+ Cg * yg .- Cξ * ξ .== 0)
22 @constraint(model, yg - yg_lim ≤ yw * U)
23 @constraint(model, -yb - yb_lim ≤ yw * U)
24 @constraint(model, yb - yb_lim ≤ yw * U)
25 @constraint(model, chance, E(1 - yw, ξ) ≥ α)
26
27 # Solve for the Pareto solutions
28 objs = zeros(length(αs))
29 probs = zeros(length(αs))
30 for (i, prob) in enumerate(αs)
31     set_value(α, prob)
32     optimize!(model)
33     objs[i] = objective_value(model)
34     probs[i] = value(chance)
35 end

```

We implement both variants of (67) in `InfiniteOpt.jl` and use direct transcription with 1000 MC samples of ξ to transform the `InfiniteOpt` problem into finite-dimensional form. [Code Snippet 2](#) shows an excerpt of the script used in `InfiniteOpt.jl`. Note that the algebraic reformulations of each event constraint are used (e.g., constraints (61) in place of constraint (60)). Each formulation is solved over a range of α values to obtain Pareto pairs, which are shown in [Fig. 29](#). We observe that the traditional joint-chance constraint requires a higher power generation cost for a given probability α . This makes sense, because the alternate formulation captures a larger probabilistic event region (it is less constraining). This highlights how logic affects the behavior of event-constrained optimization formulations. In summary, in this case study we have used our unifying abstraction to explore the use of event constraints. This is facilitated by studying the analogy between excur-

sion probability constraints and joint-chance constraints through the lens of our abstraction.

5.2. Stochastic optimal pandemic control

We exemplify the discussion in [Section 4.1.2](#) in which we discuss how to use CVaR to shape time-dependent trajectories. We do so by considering a pandemic optimal control problem; here, we seek to combat the spread of a contagion while minimizing the enforced isolation policy $y_u(t) \in [0, 1]$ (i.e., social distancing policy). The majority of other pandemic control studies in the literature ([Lemecha Obsu and Feyissa Balcha, 2020](#); [Area et al., 2017](#); [Tsai et al., 2020](#)) use integral objectives that uniformly penalize the shape of optimal trajectories. We represent traditional formulations using the objective:

$$\min_{y_u(\cdot)} \frac{1}{S} \int_{t \in \mathcal{D}_t} y_u(t) dt \quad (69)$$

with $S = \int_{t \in \mathcal{D}_t} dt$. This objective minimizes the time-average isolation policy. We also consider an alternative objective by incorporating a peak penalty (e.g., (44)) to also control the maximum isolation policy:

$$\min_{y_u(\cdot)} \max_{t \in \mathcal{D}_t} y_u(t). \quad (70)$$

This new objective can be formulated as:

$$\begin{aligned} \min_{y_u(\cdot), z} \quad & z \\ \text{s.t.} \quad & z \geq y_u(t), \quad t \in \mathcal{D}_t \end{aligned} \quad (71)$$

where $z \in \mathbb{R}$ is an auxiliary variable that captures the peak ([Vanderbei, 2020](#)). We will now show that objectives (69) and (70) are special cases of the CVaR objective:

$$\min_{y_u(\cdot)} \text{CVaR}_t(y_u(t); \alpha) \quad (72)$$

where $\alpha \in [0, 1)$. This problem can be reformulated as:

$$\begin{aligned} \min_{y_u(\cdot), y_m(\cdot), z} \quad & z + \frac{1}{1-\alpha} \mathbb{E}_t[y_m(t)] \\ \text{s.t.} \quad & y_m(t) \geq y_u(t) - z, \quad t \in \mathcal{D}_t \\ & y_m(t) \geq 0, \quad t \in \mathcal{D}_t \end{aligned} \quad (73)$$

where $y_m : \mathcal{D}_t \rightarrow \mathbb{R}$ and $z \in \mathbb{R}$ are appropriate infinite and finite auxiliary variables (Dowling et al., 2016).

We model the spread of the contagion through a given population using the SEIR model (Aron and Schwartz, 1984), which considers four population categories:

Susceptible \rightarrow Exposed \rightarrow Infectious \rightarrow Recovered.

We define the fractional populations of individuals susceptible to infection $y_s : \mathcal{D}_t \rightarrow [0, 1]$, exposed individuals that are not yet infectious $y_e : \mathcal{D}_t \rightarrow [0, 1]$, infectious individuals $y_i : \mathcal{D}_t \rightarrow [0, 1]$, and recovered individuals $y_r : \mathcal{D}_t \rightarrow [0, 1]$ (considered immune to future infection). The variables are normalized such that $y_s(t) + y_e(t) + y_i(t) + y_r(t) = 1$. The deterministic SEIR model is formalized as:

$$\begin{aligned} \frac{dy_s(t)}{dt} &= (y_u(t) - 1)\beta y_s(t)y_i(t), & t \in \mathcal{D}_t \\ \frac{dy_e(t)}{dt} &= (1 - y_u(t))\beta y_s(t)y_i(t) - \xi y_e(t), & t \in \mathcal{D}_t \\ \frac{dy_i(t)}{dt} &= \xi y_e(t) - \gamma y_i(t), & t \in \mathcal{D}_t \\ \frac{dy_r(t)}{dt} &= \gamma y_i(t), & t \in \mathcal{D}_t \end{aligned} \quad (74)$$

Code Snippet 3: Formulation (5.75) with objective (5.73) via **InfiniteOpt.jl**.

```
1 using InfiniteOpt, Distributions, Ipopt
2
3 # Initialize the model
4 model = InfiniteModel(Ipopt.Optimizer)
5
6 # Set the infinite parameters
7 @infinite_parameter(model, t ∈ [t0, tf], num_supports = 101)
8 add_supports(t, extra_ts)
9 @infinite_parameter(model, ξ ~ Uniform(ξ_min, ξ_max), num_supports = 20)
10
11 # Set the infinite variables
12 var_inds = [:s, :e, :i, :r]
13 @variable(model, 0 ≤ y[var_inds], Infinite(t, ξ))
14 @variable(model, ysi, Infinite(t, ξ))
15 @variable(model, 0 ≤ yu ≤ 0.8, Infinite(t), start = 0.2)
16
17 # Set the CVaR objective
18 @variable(model, z)
19 @variable(model, 0 ≤ ym, Infinite(t))
20 @objective(model, Min, z + 1 / (1 - α) * E(y_m, t))
21 @constraint(model, ym ≥ yu - z)
22
23 # Define the initial conditions
24 @constraint(model, [v ∈ var_inds], y[v](0, ξ) == y0[v])
25
26 # Define the SEIR equations
27 @constraints(model, begin
28     ∂(y[:s], t) == -(1 - yu) * β * ysi
29     ∂(y[:e], t) == (1 - yu) * β * ysi - ξ * y[:e]
30     ∂(y[:i], t) == ξ * y[:e] - γ * y[:i]
31     ∂(y[:r], t) == γ * y[:i]
32     ysi == y[:s] * y[:i]
33 end)
34
35 # Define the infection limit
36 @constraint(model, y[:i] ≤ i_max)
37
38 # Optimize and get the results
39 optimize!(model)
40 state_opt = value.(y, ndarray = true)
41 control_opt = value(yu)
42 obj_opt = objective_value(model)
43 ts = value(t)
44 ξs = value(ξ)
```

where $\beta, \gamma, \xi \in \mathbb{R}$ are the rates of infection, recovery, and incubation, respectively. For our case study, we consider ξ to be an uncertain parameter $\xi \sim \mathcal{U}(\underline{\xi}, \bar{\xi})$. This introduces the random domain \mathcal{D}_ξ (i.e., the co-domain of $\mathcal{U}(\underline{\xi}, \bar{\xi})$) and gives a stochastic dynamic optimization problem of the form:

$$\begin{aligned} \min \quad & \text{Objective (5.69), (5.71), or (5.73)} \\ \text{s.t.} \quad & \frac{\partial y_s(t, \xi)}{\partial t} = (y_u(t) - 1)\beta y_s(t, \xi)y_i(t, \xi), & t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_e(t, \xi)}{\partial t} = (1 - y_u(t))\beta y_s(t, \xi)y_i(t, \xi) - \xi y_e(t, \xi), & t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_i(t, \xi)}{\partial t} = \xi y_e(t, \xi) - \gamma y_i(t, \xi), & t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & \frac{\partial y_r(t, \xi)}{\partial t} = \gamma y_i(t, \xi), & t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & y_s(0, \xi) = s_0, y_e(0, \xi) = e_0, y_i(0, \xi) = i_0, y_r(0, \xi) = r_0, & \xi \in \mathcal{D}_\xi \\ & y_i(t, \xi) \leq i_{\max}, & t \in \mathcal{D}_t, \xi \in \mathcal{D}_\xi \\ & y_u(t) \in [0, y_u], & t \in \mathcal{D}_t \end{aligned} \quad (75)$$

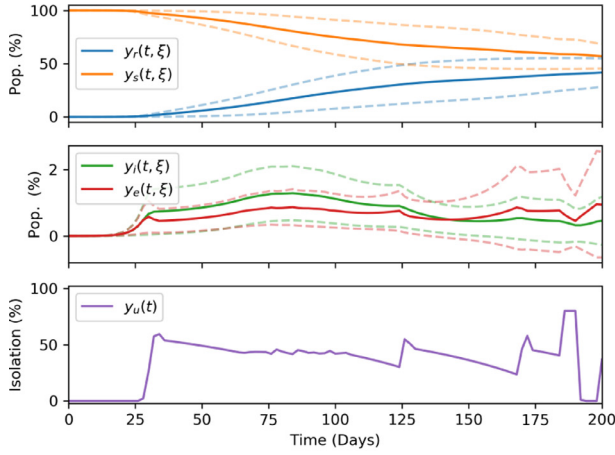
where $s_0, e_0, i_0, r_0 \in \mathbb{R}$ denote the initial population fractions, i_{\max} denotes the maximum allowable fraction of infected individuals $y_i(t)$, and \bar{y}_u denotes the maximum realizable population isolation. The state variables $y_s(\cdot), y_i(\cdot), y_e(\cdot), y_r(\cdot)$ are now infinite variables that are parameterized in the time and random domains, while the control variable y_u is an infinite variable that is only parameterized in the time domain (since we need to decide our control policy before knowing the realizations of ξ).

We solve the InfiniteOpt problem (75) using the parameters defined in Table 1 with $\mathcal{D}_t = [0, 200]$. We transcribe it via InfiniteOpt.jl using 111 supports for \mathcal{D}_t and 20 MC samples for \mathcal{D}_ξ . Code Snippet 3 shows an excerpt of this implementation in InfiniteOpt.jl. The optimal policies corresponding to objectives (69) and (71) are shown in Fig. 30. Penalizing the peak isolation provides a smoother isolation policy $y_u(t)$ relative to the more traditional integral based objective. Moreover, the population of susceptible individuals $y_s(t, \xi)$ associated with penalizing the peak isolation decreases at a lower rate which indicates that penalizing the peak isolation is more effective at mitigating the spread of the contagion in this particular case.

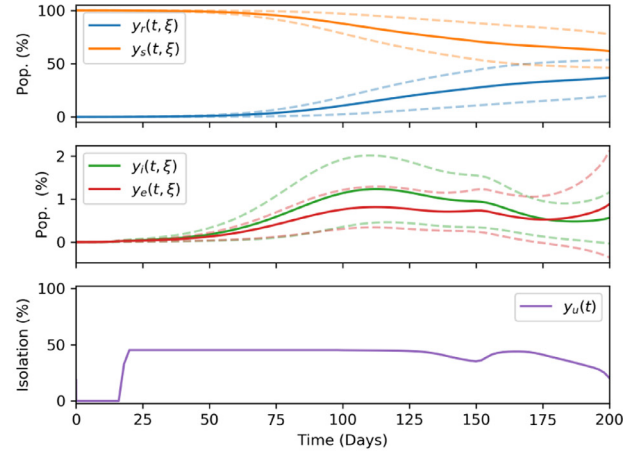
We address formulation (75) by using the proposed CVaR measure. Three solutions are obtained corresponding to $\alpha = \{0, 0.5, 0.99\}$ and are depicted in Fig. 31. Note that we use $\alpha = 0.99$ because a singularity is incurred with $\alpha = 1$. We observe that Fig. 31a and 31b are identical to Fig. 30a and 30b; this illustrates that the CVaR objective (72) is a generalization of objectives (69) and (70). Moreover, CVaR provides a range between

Table 1
Parameter values used in the InfiniteOpt formulation (75).

β	γ	$\underline{\xi}$	$\bar{\xi}$	i_{\max}	\bar{y}_u	s_0	e_0	i_0	r_0
0.727	0.303	0.1	0.6	0.02	0.8	$1 - 10^{-5}$	10^{-5}	0	0



(a) Objective (5.69)



(b) Objective (5.71)

Fig. 30. Optimal trajectories for formulation (75) using traditional dynamic optimization objectives. For the state variables $y_s(t, \xi)$, $y_e(t, \xi)$, $y_r(t, \xi)$, and $y_i(t, \xi)$ the solid lines denote the trajectories averaged over ξ and the dashed lines denote the trajectories that are one standard deviation away from the mean.

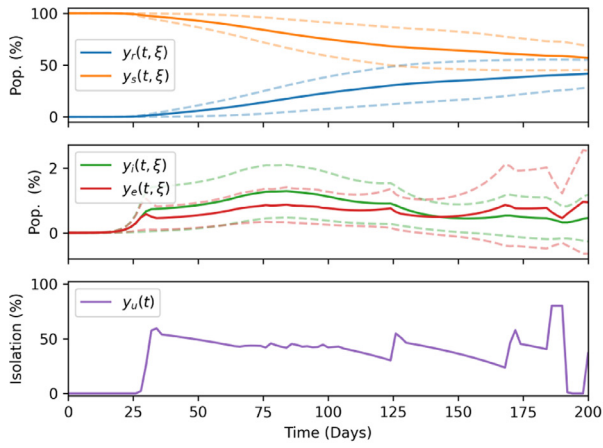
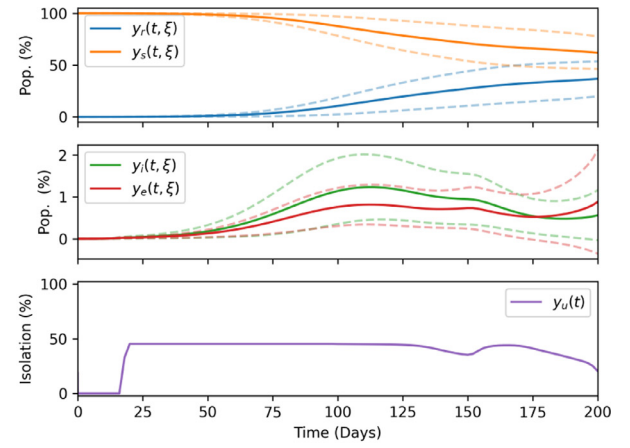
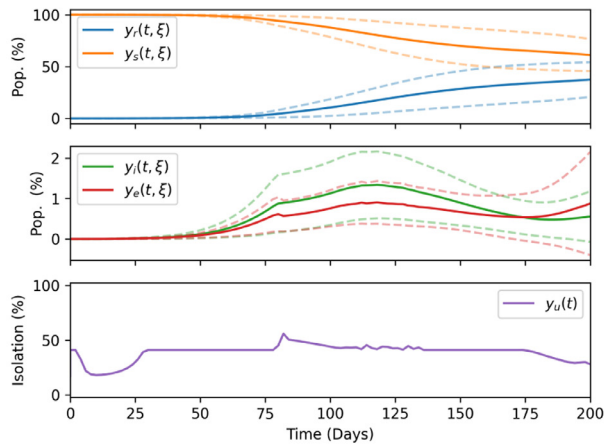
(a) $\alpha = 0$ (b) $\alpha = 0.99$ (c) $\alpha = 0.5$

Fig. 31. Optimal policies for solving Formulation (75) in combination with Objective (73).

Table 2
Species membership of the microbial community.

Species Name	Abbreviation
Blautia hydrogenotrophica	BH
Collinsella aerofaciens	CA
Bacteroides uniformis	BU
Prevotella copri	PC
Bacteroides ovatus	BO
Bacteroides vulgatus	BV
Bacteroides thetaiotaomicron	BT
Eggerthella lenta	EL
Faecalibacterium prausnitzii	FP
Clostridium hiranonis	CH
Desulfovibrio piger	DP
Eubacterium rectale	ER

these cases by varying the value of α . Hence, we can also seek tradeoff solutions such as what is shown in Fig. (31)c with $\alpha = 0.5$. Interestingly, this optimal policy combines aspects of the average and peak solution policies, but also is unique in enforcing isolation policies from the start. This shows that CVaR can be used to shape time-dependent trajectories in unique ways that are difficult to achieve with traditional measures used in dynamic optimization.

5.3. Estimation in dynamic biological systems

We consider a dynamic parameter estimation problem for a biological system model; this study aims to demonstrate how the unifying abstraction inspires new formulation classes by lifting formulations into infinite dimensional spaces. This follows from the discussion in Section 4.3 with regard to formulations (56) and (57).

To juxtapose the utility of formulations (56) and (57), we consider a microbial community consisting of the 12 species described in Table 2 using the generalized Lotka–Volterra (gLTV) model:

$$\frac{dy_{x,i}(t)}{dt} = \left(z_{\mu,i} + \sum_{j \in \mathcal{I}} z_{\alpha,ij} y_{x,j}(t) \right) y_{x,i}(t), \quad i \in \mathcal{I} \quad (76)$$

where \mathcal{I} represents the set of microbial species, $y_{x,i}(t)$ is the estimated absolute abundance of species $i \in \mathcal{I}$, $z_{\mu,i}$ is the growth rate of species i , and $z_{\alpha,ij}$ is a species interaction coefficient which describes the effect of the recipient species i on the growth of the donor species j (Shin et al., 2019). We use the gLV model parameters presented in Venturelli et al. (2018) to generate simulated experimental data with random noise $\epsilon \sim \mathcal{N}(0, 0.01)$ for 12 mono-species and 66 pairwise experiments. This will enhance our assessment of Formulations (56) and (57), since we have an established ground truth for the model parameters.

Incorporating (76) into Formulation (56) provides us with our discrete dynamic biological community estimation formulation:

$$\begin{aligned} \min \quad & \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \sum_{t \in \hat{\mathcal{D}}_k} (y_{x,ik}(t) - \tilde{y}_{x,ik}(t))^2 \\ \text{s.t.} \quad & \frac{dy_{x,ik}(t)}{dt} = \left(z_{\mu,i} + \sum_{j \in \mathcal{I}} z_{\alpha,ij} y_{x,jk}(t) \right) y_{x,ik}(t), \quad t \in \hat{\mathcal{D}}_k, i \in \mathcal{I}, k \in \mathcal{K} \\ & 0.09 \leq z_{\mu,i} \leq 2.1, \quad i \in \mathcal{I} \\ & -10 \leq z_{\alpha,ii} \leq 0, \quad i \in \mathcal{I} \\ & -10 \leq z_{\alpha,ij} \leq 10, \quad (i, j \neq i) \in \mathcal{I} \times \mathcal{I}. \end{aligned} \quad (77)$$

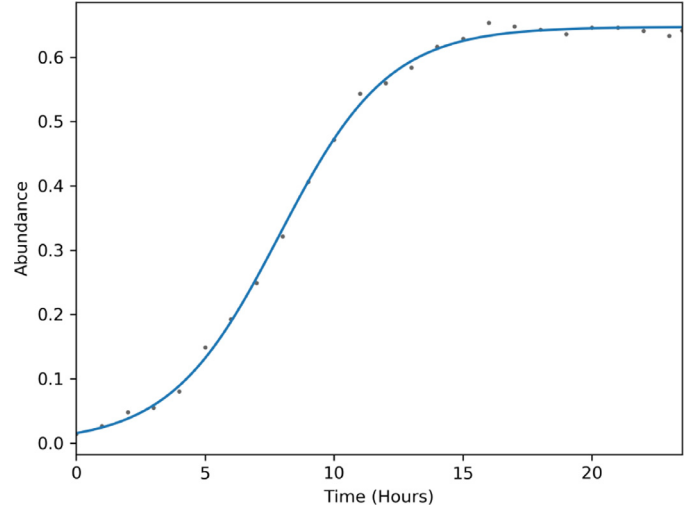


Fig. 32. Empirical fit for a mono-species experiment of *Bacteroides vulgatus* using (78).

Note that we sum over each species i for each experiment k in accordance with (76). Also, we recall that the derivative terms are limited to finite difference approximation schemes that employ the support points in $\hat{\mathcal{D}}_k$.

To derive an explicit form of Formulation (57), we first fit an empirical function $f_{ik}(t_k)$ to each experiment k and each species i . Our goal is not to create a generalizable predictive representation but rather to construct a continuous, infinite-dimensional function that represents the dynamic behavior observed in each experiment. We observe that the experiments appear to exhibit sigmoidal characteristics, and thus we use least-squares to fit each experiment k and species i to an empirical function of the form:

$$f_{ik}(t) := \frac{\beta_{1,ik}}{\beta_{2,ik} + \beta_{3,ik} e^{\beta_{4,ik}(t - \beta_{5,ik})}}, \quad t \in \mathcal{D}_{t_k}, i \in \mathcal{I}, k \in \mathcal{K} \quad (78)$$

where $\beta_{1,ik}$, $\beta_{2,ik}$, $\beta_{3,ik}$, $\beta_{4,ik}$, and $\beta_{5,ik}$ are fitting parameters. Eq. (78) fits our experimental datasets well as demonstrated for the particular experiment shown in Fig. 32 which is indicative of overall fit qualities we observed. Thus, we substitute Eqs. (76) and (78) into Formulation (57) to obtain:

$$\begin{aligned} \min \quad & \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{I}} \left(\int_{t \in \mathcal{D}_{t_k}} (y_{x,ik}(t) - \tilde{y}_{x,ik}(t))^2 dt \right) \\ \text{s.t.} \quad & \frac{dy_{x,ik}(t)}{dt} = \left(z_{\mu,i} + \sum_{j \in \mathcal{I}} z_{\alpha,ij} y_{x,jk}(t) \right) y_{x,ik}(t), \quad t \in \mathcal{D}_{t_k}, i \in \mathcal{I}, k \in \mathcal{K} \\ & \tilde{y}_{x,ik}(t) = \frac{\beta_{1,ik}}{\beta_{2,ik} + \beta_{3,ik} e^{\beta_{4,ik}(t - \beta_{5,ik})}}, \quad t \in \mathcal{D}_{t_k}, i \in \mathcal{I}, k \in \mathcal{K} \\ & 0.09 \leq z_{\mu,i} \leq 2.1, \quad i \in \mathcal{I} \\ & -10 \leq z_{\alpha,ii} \leq 0, \quad i \in \mathcal{I} \\ & -10 \leq z_{\alpha,ij} \leq 10, \quad (i, j \neq i) \in \mathcal{I} \times \mathcal{I}. \end{aligned} \quad (79)$$

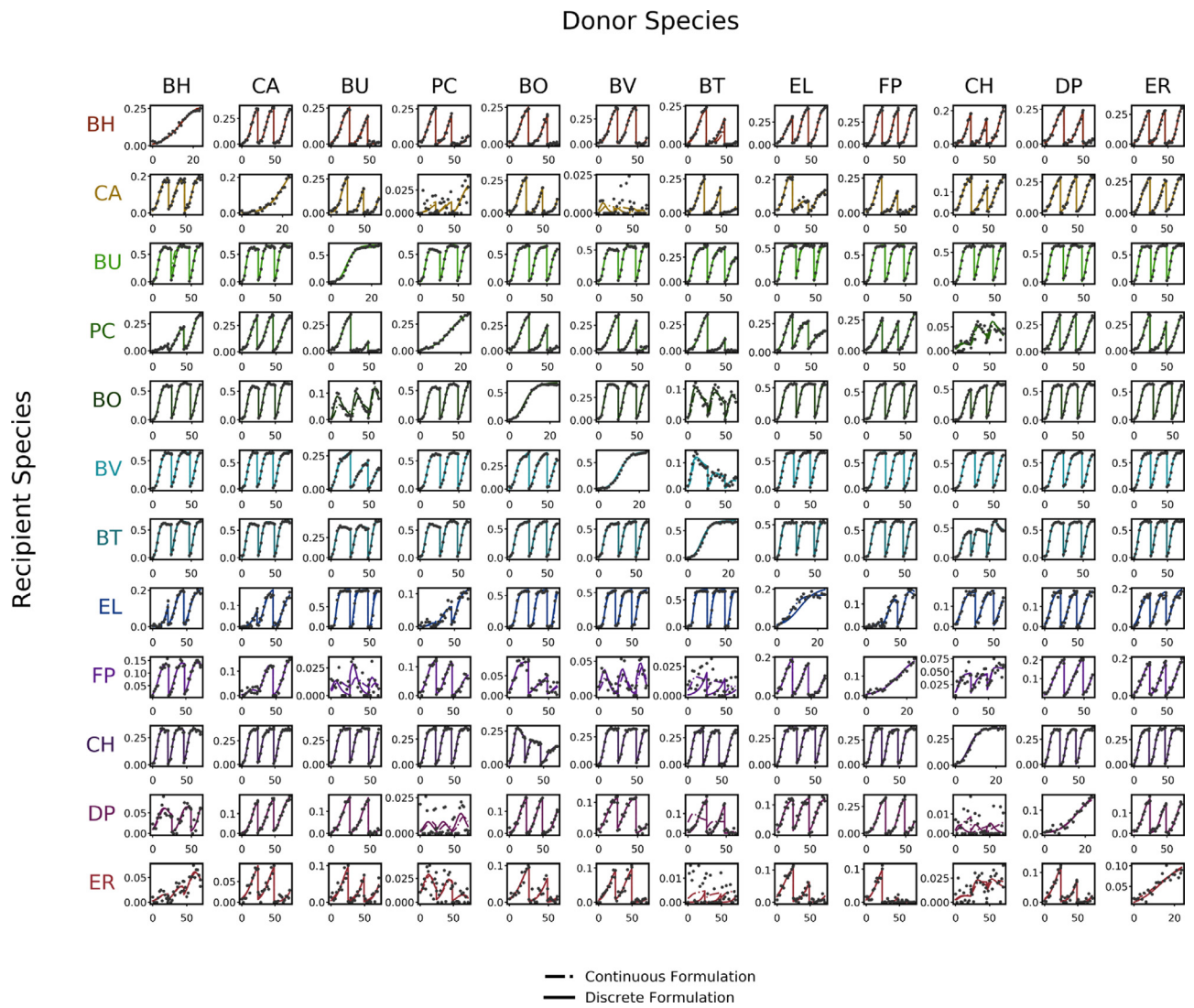


Fig. 33. Optimal trajectories from formulations (77) and (79) using orthogonal collocation over finite elements to approximate the derivatives with two and four points, respectively. All shown in comparison to the experimental data. The x-axis is time in hours, and the y-axis is the absolute abundance of the recipient species in contact with the corresponding donor species. The results for the mono-species experiments are observed on the diagonal with the rest being pairwise.

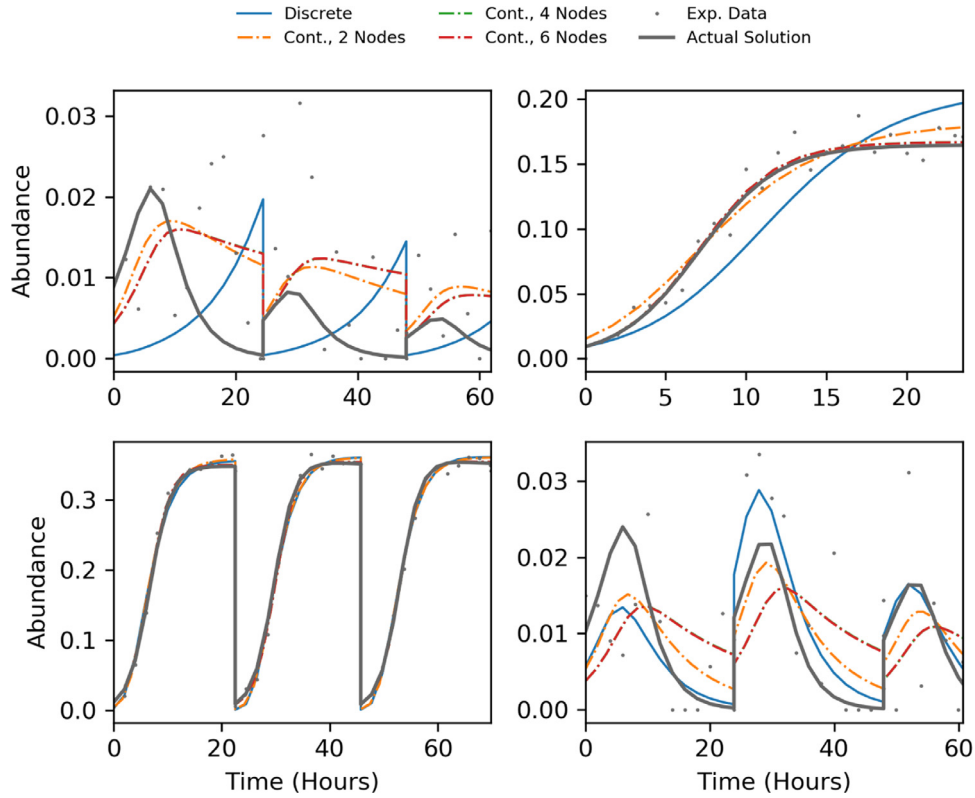


Fig. 34. Optimal profiles for select experiments using different formulations and collocation node amounts (top-left: (FP, BT), top-right: (EL, EL), bottom-left: (CH, PC), bottom-right: (FP, BU)).

Code Snippet 4: Formulation (5.79) using *InfiniteOpt.jl*.

```

1 using InfiniteOpt, Ipopt
2
3 # Initialize the model
4 model = InfiniteModel(Ipopt.Optimizer)
5
6 # Set the infinite parameters
7 @infinite_parameter(model, t[k ∈ K] ∈ [0, T[k]], num_supports = 15, independent = true,
8 derivative_method = OrthogonalCollocation(4))
9
10 # Set the finite variables
11 @variable(model, 0.09 ≤ zμ[i ∈ I] ≤ 2.1)
12 @variable(model, -10 ≤ zα[i ∈ I, j ∈ I] ≤ zα_max[i, j])
13
14 # Set the infinite variables
15 @variable(model, yx[i ∈ I, k ∈ K] ≥ 0, Infinite(t[k]))
16 @variable(model, zayx[i ∈ I, j ∈ I, k ∈ K], Infinite(t[k]))
17
18 # Set the empirical functions using fitted q[i, k](t) functions
19 @parameter_function(model, yx_tilde[i ∈ I, k ∈ K] == q[i, k](t[k]))
20
21 # Set the least-squares objective
22 @objective(model, Min, sum(((yx[i, k] - yx_tilde[i, k]) ^ 2, t[k]) for i ∈ I, k ∈ K))
23
24 # Define the gLV equations
25 @constraint(model, [i ∈ I, k ∈ K],
26 ∂(yx[i, k], t[k]) == (zμ[i] + sum(zayx[i, j, k] for j ∈ I)) * yx[i, k])
27 @constraint(model, [i ∈ I, j ∈ I, k ∈ K], zayx[i, j, k] == zα[i, j] * yx[j, k])
28
29 # Optimize and get the results
30 optimize!(model)
31 zα_opt = value.(zα)
32 zμ_opt = value.(zμ)
33 yx_opt = value.(yx)
34 ts = supports(t)

```

We solve formulations (77) and (79) using *InfiniteOpt.jl* via its automated transcription capabilities. [Code Snippet 4](#) presents an illustrative summary of the syntax for implementing Formulation (79). We employ 15 time supports for each experiment in Formulation (79) and necessarily use the measurement times as supports for formulation (77). We use orthogonal collocation over finite elements to approximate the differential equations; we use two nodes per finite element (necessarily using only the boundary points) in Formulation (77) and a range of

node amounts with formulation (79) to investigate their effect on solution quality. [Fig. 33](#) summarizes the model fits of both solution sets relative to the experimental data. Although both estimation problems are able to choose parameters that characterize the data well, there are significant deviations between the profiles across a few experiments.

[Fig. 34](#) shows a representative subset of experiments to demonstrate how the solutions vary with respect to the estimation for-

Table 3

Sum-squared-errors between the actual and estimated parameters for each formulation and number of collocation nodes per finite element.

Formulation	SSE_{z_α}	SSE_{z_μ}
Discrete, 2 Nodes	1.02×10^2	4.72×10^{-2}
Continuous, 2 Nodes	2.14×10^1	2.46×10^{-2}
Continuous, 4 Nodes	1.59×10^1	9.05×10^{-4}
Continuous, 6 Nodes	1.59×10^1	9.10×10^{-4}

mulation and the number of collocation points. We find that the discrete formulation solution generally deviates from the analytical solution to a greater extent than the solutions procured via the continuous formulation. However, it is difficult to conclude which estimation problem best represents the data when the measurement noise is near the magnitude of the absolute abundance because no model matches the true analytical solution particularly well; the continuous formulation solutions, however, in general seem to better represent the trend of the system. This suggests that Formulation (79) has effectively smoothed over noisy experiments, leading to a better fit. Furthermore, we observe that the ability of Formulation (79) to enable arbitrary collocation nodes has a significant effect on the accuracy of the solutions. This increased accuracy seems to effectively taper off at four collocation nodes in this case.

We seek to substantiate our qualitative observation that the continuous formulation is able to better represent the experimental data by comparing the sum-squared-errors (SSE) between the true parameters used to generate the experimental data to those approximated from formulations (77) and (79). Specifically, we consider the error metrics:

$$SSE_{z_\mu} := \sum_{i \in \mathcal{I}} (z_{\mu,i} - \bar{z}_{\mu,i})^2$$

$$SSE_{z_\alpha} := \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}} (z_{\alpha,ij} - \bar{z}_{\alpha,ij})^2$$

where $\bar{z}_{\mu,i}$ and $\bar{z}_{\alpha,ij}$ denote the actual parameters used in the simulations to generate the experimental data. The results are shown in Table 3 and demonstrate that the continuous formulation solutions yield significantly smaller sum-squared-errors. Moreover, increased collocation nodes for each finite element are able to reduce the overall error by more than an order of magnitude.

6. Conclusions and future work

We have presented a unifying abstraction for representing infinite-dimensional optimization (InfiniteOpt) problems that appear across diverse disciplines. This unifying abstraction introduces the notions of infinite variables (variables parameterized over infinite-dimensional domains). The abstraction also uses measure and differential operators that facilitate the construction of objectives and constraints. The proposed abstraction facilitates knowledge transfer; specifically, it helps identify and transfer modeling constructs across disciplines. For instance, we discussed how chance constraints are analogues of excursion probabilities and how these can be generalized using event constraints; as another example, we show how one can use risk measures to shape time-dependent trajectories in dynamic optimization. The proposed modeling abstraction aims also to decouple the formulation from transformation schemes (e.g., direct transcription), as we believe that this facilitates analysis and implementation. The proposed abstraction serves as the backbone of a Julia-based modeling framework called *InfiniteOpt.jl*.

In future work, we will further investigate theoretical crossovers between disciplines. In particular, we are interested in rigorously analyzing the characteristics of risk measures across general infinite domains. Such measures have the potential to enhance the shaping of optimal trajectories in spatio-temporal domains and provide an intuitive analogy for stochastic optimization problems. Event constraints also present an intriguing general constraint class that warrants further research (e.g., connections with disjunctive programming). We also plan to further develop the theoretical and algorithmic foundations for incorporating random field theory into optimization. Furthermore, we will continue enriching the capabilities of *InfiniteOpt.jl* to support these research endeavors and to make advanced transformation approaches (e.g., MWR approaches) more readily accessible.

Declaration of Competing Interest

None.

Acknowledgments

We acknowledge financial support from the U.S. Department of Energy under grant DE-SC0014114 and from the U.S. National Science Foundation under award 1832208.

References

- Adler, R.J., 2000. On excursion sets, tube formulas and maxima of random fields. *Ann. Appl. Probab.* 1–74.
- Adler, R.J., 2010. *The geometry of random fields*. SIAM.
- Area, I., Ndaïrou, F., Nieto, J. J., Silva, C. J., Torres, D. F., 2017. Ebola model and optimal control with vaccination constraints. *arXiv preprint arXiv:1703.01368*.
- Armaou, A., Christofides, P.D., 2002. Dynamic optimization of dissipative PDE systems using nonlinear order reduction. *Chem. Eng. Sci.* 57 (24), 5083–5114.
- Aron, J.L., Schwartz, I.B., 1984. Seasonality and period-doubling bifurcations in an epidemic model. *J. Theor. Biol.* 110 (4), 665–679.
- Asprey, S., Macchietto, S., 2002. Designing robust optimal dynamic experiments. *J. Process Control* 12 (4), 545–556.
- Asteasuain, M., Tonelli, S.M., Brandolin, A., Bandoni, J.A., 2001. Dynamic simulation and optimisation of tubular polymerisation reactors in Gproms. *Comput. Chem. Eng.* 25 (4–6), 509–515.
- Au, S., Beck, J.L., 2001. First excursion probabilities for linear systems by very efficient importance sampling. *Probab. Eng. Mech.* 16 (3), 193–207.
- Beal, L.D., Hill, D.C., Martin, R.A., Hedengren, J.D., 2018. Gekko optimization suite. *Processes* 6 (8), 106.
- Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P., 1995. *Dynamic Programming and Optimal Control*, 1. Athena scientific Belmont, MA.
- Besaçon, M., Anthoff, D., Arslan, A., Byrne, S., Lin, D., Papamarkou, T., Pearson, J., 2019. Distributions.jl: definition and modeling of probability distributions in the JuliaStats ecosystem. *arXiv preprint arXiv:1907.08611*.
- Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B., 2017. Julia: a fresh approach to numerical computing. *SIAM Rev.* 59 (1), 65–98.
- Biegler, L.T., 2007. An overview of simultaneous strategies for dynamic optimization. *Chem. Eng. Process. Process Intensif.* 46 (11), 1043–1053.
- Biegler, L.T., Ghattas, O., Heinkenschloss, M., van Bloemen Waanders, B., 2003. Large-scale PDE-constrained optimization: an introduction. In: *Large-Scale PDE-Constrained Optimization*. Springer, pp. 3–13.
- Birge, J.R., Louveaux, F., 2011. *Introduction to Stochastic Programming*. Springer Science & Business Media.
- Çakmak, U., Özekici, S., 2006. Portfolio optimization in stochastic markets. *Math. Methods Oper. Res.* 63 (1), 151–168.
- Chen, M., Mehrotra, S., Papp, D., 2015. Scenario generation for stochastic optimization problems via the sparse grid method. *Comput. Optim. Appl.* 62 (3), 669–692.
- Dentcheva, D., Ruszczyński, A., 2006. Portfolio optimization with stochastic dominance constraints. *J. Bank. Financ.* 30 (2), 433–451.
- Devolder, O., Glineur, F., Nesterov, Y., 2010. Solving infinite-dimensional optimization problems by polynomial approximation. In: *Recent Advances in Optimization and its Applications in Engineering*. Springer, pp. 31–40.
- Dowling, A.W., Ruiz-Mercado, G., Zavala, V.M., 2016. A framework for multi-stakeholder decision-making and conflict resolution. *Comput. Chem. Eng.* 90, 136–150.
- Dunning, I., Huchette, J., Lubin, M., 2017. Jump: a modeling language for mathematical optimization. *SIAM Rev.* 59 (2), 295–320.
- Faulwasser, T., Grüne, L., 2020. Turnpike properties in optimal control: an overview of discrete-time and continuous-time results. *arXiv preprint arXiv:2011.13670*.
- Finlayson, B.A., 2013. *The Method of Weighted Residuals and Variational Principles*. SIAM.

- Georgakis, C., 2013. Design of dynamic experiments: a data-driven methodology for the optimization of time-varying processes. *Ind. Eng. Chem. Res.* 52 (35), 12369–12382.
- Gnegel, F., Fügenschuh, A., Hagel, M., Leyffer, S., Stierner, M., 2021. A solution framework for linear PDE-constrained mixed-integer problems. *Math. Program.* 1–34.
- Graham, M.D., Rawlings, J.B., 2013. *Modeling and Analysis Principles for Chemical And Biological Engineers*. Nob Hill Pub..
- Hinze, M., Pinnau, R., Ulbrich, M., Ulbrich, S., 2008. *Optimization with PDE Constraints*, 23. Springer Science & Business Media.
- Houska, B., Ferreau, H.J., Diehl, M., 2011. Acado toolkit - an open-source framework for automatic control and dynamic optimization. *Optim. Control Appl. Methods* 32 (3), 298–312.
- Jalving, J., Cao, Y., Zavala, V.M., 2019. Graph-based modeling and simulation of complex systems. *Comput. Chem. Eng.* 125, 134–154.
- Jalving, J., Shin, S., Zavala, V. M., 2020. A graph-based modeling abstraction for optimization: concepts and implementation in plasmo. *jl. arXiv preprint arXiv:2006.05378*.
- Kleywegt, A.J., Shapiro, A., Homem-de Mello, T., 2002. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optim.* 12 (2), 479–502.
- Koivu, M., Pennanen, T., 2010. Galerkin methods in dynamic stochastic programming. *Optimization* 59 (3), 339–354.
- Krokhmal, P., Zabaranin, M., Uryasev, S., 2013. Modeling and optimization of risk. In: *Handbook of the Fundamentals of Financial Decision Making: Part II*, pp. 555–600.
- Lan, T., Zhou, Z., Huang, G.M., 2018. Modeling and numerical analysis of stochastic optimal transmission switching with DCOPT and ACOPT. *IFAC-PapersOnline* 51 (28), 126–131.
- Legat, B., Dowson, O., Garcia, J. D., Lubin, M., 2020. Mathoptinterface: a data structure for mathematical optimization problems. *arXiv preprint arXiv:2002.03447*.
- Lemecha Obsu, L., Feyissa Balcha, S., 2020. Optimal control strategies for the transmission risk of COVID-19. *J. Biol. Dyn.* 14 (1), 590–607.
- Luedtke, J., Ahmed, S., 2008. A sample approximation approach for optimization with probabilistic constraints. *SIAM J. Optim.* 19 (2), 674–699.
- Mühlfordt, T., Faulwasser, T., Hagenmeyer, V., 2018. A generalized framework for chance-constrained optimal power flow. *Sustain. Energy Grids Netw.* 16, 231–242.
- Mühlfordt, T., Roald, L., Hagenmeyer, V., Faulwasser, T., Misra, S., 2019. Chance-constrained ac optimal power flow: a polynomial chaos approach. *IEEE Trans. Power Syst.* 34 (6), 4806–4816.
- Na, S., Shin, S., Anitescu, M., Zavala, V. M., 2020. Overlapping Schwarz decomposition for nonlinear optimal control. *arXiv preprint arXiv:2005.06674*.
- Nicholson, B., Siirola, J.D., Watson, J.-P., Zavala, V.M., Biegler, L.T., 2018. Pyomo. DAE: a modeling and automatic discretization framework for optimization with differential and algebraic equations. *Math. Program. Comput.* 10 (2), 187–223.
- Nocedal, J., 2006. Knitro: an integrated package for nonlinear optimization. In: *Large-Scale Nonlinear Optimization*. Springer, pp. 35–60.
- Nocedal, J., Wright, S., 2006. *Numerical Optimization*. Springer Science & Business Media.
- Petrik, M., Scherrer, B., 2008. Biasing approximate dynamic programming with a lower discount factor. In: *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems-NIPS 2008*.
- Pulsipher, J.L., Zavala, V.M., 2019. A scalable stochastic programming approach for the design of flexible systems. *Comput. Chem. Eng.* 128, 69–76.
- Pulsipher, J.L., Zavala, V.M., 2020. Measuring and optimizing system reliability: a stochastic programming approach. *TOP* 1–20.
- Qin, S.J., Badgwell, T.A., 2003. A survey of industrial model predictive control technology. *Control Eng. Pract.* 11 (7), 733–764.
- Ramsay, J.O., Hooker, G., Campbell, D., Cao, J., 2007. Parameter estimation for differential equations: a generalized smoothing approach. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* 69 (5), 741–796.
- Rawlings, J.B., 2000. Tutorial overview of model predictive control. *IEEE Control Syst. Mag.* 20 (3), 38–52.
- Rawlings, J.B., Mayne, D.Q., Diehl, M., 2017. *Model Predictive Control: Theory, Computation, and Design*, 2. Nob Hill Publishing Madison, WI.
- Risbeck, M.J., Rawlings, J.B., 2019. Economic model predictive control for time-varying cost and peak demand charge optimization. *IEEE Trans. Autom. Control* 65 (7), 2957–2968. doi:10.1109/TAC.2019.2939633.
- Roald, L., Misra, S., Chertkov, M., Andersson, G., 2015. Optimal power flow with weighted chance constraints and general policies for generation control. In: *Proceedings of the 54th IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 6927–6933.
- Rockafellar, R.T., Uryasev, S., et al., 2000. Optimization of conditional value-at-risk. *J. Risk* 2, 21–42.
- Ruszczyński, A., Shapiro, A., 2006. Optimization of risk measures. In: *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer, pp. 119–157.
- Shapiro, A., 2003. Inference of statistical bounds for multistage stochastic programming problems. *Math. Methods Oper. Res.* 58 (1), 57–68.
- Shin, S., Anitescu, M., Zavala, V. M., 2021. Exponential decay of sensitivity in graph-structured nonlinear programs. *arXiv preprint arXiv:2101.03067*.
- Shin, S., Venturelli, O.S., Zavala, V.M., 2019. Scalable nonlinear programming framework for parameter estimation in dynamic biological system models. *PLoS Comput. Biol.* 15 (3), e1006828.
- Shin, S., Zavala, V. M., 2020. Diffusing-horizon model predictive control. *arXiv preprint arXiv:2002.08556*.
- Stankiewicz, A.J., Moulijn, J.A., et al., 2000. Process intensification: transforming chemical engineering. *Chem. Eng. Prog.* 96 (1), 22–34.
- Stein, O., Still, G., 2003. Solving semi-infinite optimization problems with interior point techniques. *SIAM J. Control Optim.* 42 (3), 769–788.
- Stewart, J., 2009. *Calculus: Concepts and Contexts*. Cengage Learning.
- Straub, D.A., Grossmann, I.E., 1993. Design optimization of stochastic flexibility. *Comput. Chem. Eng.* 17 (4), 339–354.
- Suksuwan, A., Spence, S.M., 2018. Optimization of uncertain structures subject to stochastic wind loads under system-level first excursion constraints: a data-driven approach. *Comput. Struct.* 210, 58–68.
- Tjoa, I.B., Biegler, L.T., 1991. Simultaneous solution and optimization strategies for parameter estimation of differential-algebraic equation systems. *Ind. Eng. Chem. Res.* 30 (2), 376–385.
- Tsay, C., Lejarza, F., Stadtherr, M. A., Baldea, M., 2020. Modeling, state estimation, and optimal control for the us COVID-19 outbreak. *arXiv preprint arXiv:2004.06291*.
- Vanderbei, R.J., 2020. *Linear Programming: Foundations and Extensions*, 285. Springer Nature.
- Venturelli, O.S., Carr, A.V., Fisher, G., Hsu, R.H., Lau, R., Bowen, B.P., Hromada, S., Northen, T., Arkin, A.P., 2018. Deciphering microbial interactions in synthetic human gut microbiome communities. *Mol. Syst. Biol.* 14 (6), e8157.
- Xiu, D., 2010. *Numerical Methods for Stochastic Computations: a Spectral Method Approach*. Princeton University Press.
- Zhang, X., He, J., Takezawa, A., Kang, Z., 2018. Robust topology optimization of phononic crystals with random field uncertainty. *Int. J. Numer. Methods Eng.* 115 (9), 1154–1173.
- Zhao, J., Wang, C., 2014. Robust structural topology optimization under random field loading uncertainty. *Struct. Multidiscip. Optim.* 50 (3), 517–522.
- Zymler, S., Kuhn, D., Rustem, B., 2013. Distributionally robust joint chance constraints with second-order moment information. *Math. Program.* 137 (1–2), 167–198.