Scalable Modeling of Infinite-Dimensional Nonlinear Programs with InfiniteExaModels.jl

Joshua L. Pulsiphera\* and Sungho Shinb

aDepartment of Chemical Engineering, University of Waterloo, 200 University Ave. W., Waterloo, ON N2L 3G1, Canada

bMathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Lemont, IL 60439, USA

pulsipher@uwaterloo.ca

Abstract

In this work, we present InfiniteExaModels.jl as a modeling framework to efficiently solve nonlinear infinite-dimensional optimization problems. This framework seamlessly integrates InfiniteOpt.jl, a modeling platform tailored for infinite-dimensional optimization models, with ExaModels.jl, an algebraic modeling and automatic differentiation system for nonlinear optimization problems exhibiting repetitive patterns. Our primary focus lies in recognizing that the discretization of infinite-dimensional optimization problems often reveals repetitive patterns. Leveraging this structure allows us to significantly enhance the efficiency of derivative evaluations—a common computational bottleneck in nonlinear optimization procedures. By harnessing the efficient derivative evaluation capabilities of ExaModels.jl, we achieve substantial speed improvements in the evaluation of derivatives for objective and constraint equations. To illustrate the effectiveness of InfiniteExaModels.jl, we present numerical examples involving quadrotor optimal control and stochastic alternating current (AC) optimal power flow. Our findings demonstrate that by using InfiniteExaModels.jl, one can achieve a minimum fourfold acceleration in derivative evaluation performance compared to state-of-the-art algebraic modeling and automatic differentiation tools.

**Keywords**: infinite-dimensional optimization, nonlinear programming, algebraic modeling

* 1. Introduction

Nonlinear infinite-dimensional optimization (InfiniteOpt) problems appear in various application areas which include optimal control, stochastic optimization, and partial differential equation (PDE)-constrained optimization (Pulsipher et al., 2022). The infinite-dimensional nature of these problems typically mandates the use of discretization schemes (e.g., orthogonal collocation over finite elements) to solve these problems by numerical means (Biegler, 2010). Relying on a manual approach, given the intricate nature of the discretization procedures, is susceptible to errors and often falls short of employing the most effective discretization strategies (Nicholson et. al., 2018). Hence, modeling environments such as pyomo.dae (Nicholson et al., 2018), InfiniteOpt.jl (Pulsipher et al., 2022), and GEKKO (Beal et al., 2018) enable the use of sophisticated discretization schemes for a variety of InfiniteOpt problem types. Building on the popular algebraic modeling language JuMP.jl, InfiniteOpt.jl is uniquely able to model InfiniteOpt formulations with both stochastic modeling elements (e.g., risk measures) and differential & algebraic equations (DAEs).

The discretized formulations produced by these frameworks are often cast as large-scale nonlinear programs, potentially involving millions of variables which can surpass the capabilities of existing algorithms and software tools. Discretized InfiniteOpt problems typically exhibit a highly recurrent structure which is not exploited by most existing modeling environments. For example, a reaction kinetics equality constraint can be enforced over thousands of random scenarios (Chen et al., 2016). Leveraging this inherent structure has great potential to enable efficient (parallelizable) solution routines for discretized InfiniteOpt problems. For instance, exploiting the recurrent structure can significantly reduce the computational burden of expensive automatic-differentiation (AD) routines on complicated nonlinear expressions.

Shin et al. (2023) present a single-instruction, multiple-data (SIMD) abstraction for nonlinear programs (NLPs) which is implemented in the open-source Julia package ExaModels.jl. Contrary to general purpose algebraic modeling languages like JuMP.jl (Lubin et al., 2023), Pyomo (Hart et al., 2017), and Gravity (Hijazi et al., 2018) which do not save repeated structure, ExaModels.jl preserves the parallelizable structure in the model, and in turn, exploits that structure for more efficient computations on multi-threaded CPUs or GPU accelerators. Shin et al. (2023) demonstrate how ExaModels.jl leverages the repeated structure in large-scale AC optimal power flow (ACOPF) problems to speed up AD by 1 to 2 orders-of-magnitude relative to JuMP.jl. Moreover, Shin et al. (2023) obtain an order-of-magnitude speed on the overall solution time for ACOPF problems solved on GPUs using ExaModels.jl and MadNLP.jl in contrast to state-of-the-art CPU solvers. The utility of these modeling/solution approaches has not yet been investigated on discretized InfiniteOpt problems.

Thus, we present InfiniteExaModels.jl as a new backend for InfiniteOpt.jl to automatically discretize InfiniteOpt problems into the SIMD-NLP representation used by ExaModels.jl. This backend facilitates an intuitive modeling environment for InfiniteOpt problems that gives ExaModels.jl direct access to the repeated structures exhibited by discretized InfiniteOpt models which enable highly efficient AD routines that are parallelizable. This framework is general and can significantly accelerate derivative evaluations on nonlinear InfiniteOpt models across an array of applications.

This paper is structured as follows. Sections 2 discusses the modeling/solution abstractions behind InfiniteOpt.jl and ExaModels.jl. Section 3 describes the InfiniteExaModels.jl interface. Section 4 demonstrates the capabilities of the proposed framework on two real-world case studies. Finally, Section 5 summaries key findings and outlines plans for future work.

* 1. Modeling Abstractions

This section reviews key aspects of InfiniteOpt.jl and ExaModels.jl and their underlying modeling abstractions. We refer the reader to (Pulsipher et al., 2022) and (Shin et al., 2023) for more comprehensive discussions.

* + 1. InfiniteOpt Modeling Abstraction

InfiniteOpt problems involve infinite parameters $d\in D⊆R^{n\_{d}}$ that index infinite decision variables $y : D ↦ Y ⊆ R^{n\_{y}}$ where $D$ is often a continuous domain. Examples include time $t\in D\_{t}=\left[t\_{0},t\_{f}\right]$ and uncertainty $ξ\in D\_{ξ}$ where $ξ$ is randomly distributed. With these we define a general InfiniteOpt formulation with the core abstraction elements:

|  |  |
| --- | --- |
| $$\begin{matrix}min&M\_{d}h\left(Dy\left(d\right),y\left(d\right),z,d\right)& \\s.t.&g\left(Dy\left(d\right),y\left(d\right),z,d\right)\leq 0,&d\in D\end{matrix}$$ | (1) |

where $h\left(⋅\right)$ is the objective function, $g\left(⋅\right)$ are the constraint functions, $M\_{d}:Y↦R^{n\_{y}}$ is a measure operator which summarize infinite variables $y\left(d\right)\in Y$ over the function space $Y$, $D:Y↦Y$ are differential operators that capture the rate of change of infinite variables, and $z\in Z⊆R^{n\_{z}}$ are finite variables. This captures many formulations in stochastic, dynamic, and PDE-constrained optimization.

Transformations are applied to Problem (1) to obtain a formulation that is compatible with standard optimization solvers (e.g., Ipopt). Direct transcription is a common transformation method where the InfiniteOpt problem is projected onto a set of discretization points $\hat{D}≔\{\hat{d}:k\in K\}$. Typically, this leads to the discretized formulation:

|  |  |
| --- | --- |
| $$\begin{matrix}min&\sum\_{k\in K}^{}α\_{k}h\left(Dy\_{k},y\_{k},z,\hat{d}\_{k}\right)& \\s.t.&g\left(Dy\_{k},y\_{k},z,\hat{d}\_{k}\right)\leq 0,&k\in K\\ &q\left(Dy\_{k},y\_{k},\hat{d}\_{k}\right)=0,&k\in K\end{matrix}$$ | (2) |

where $α\_{k}$ are coefficients used to approximate the measure operator $M\_{d}$ and the equality constraints $q\left(⋅\right)=0$ employ a numerical scheme (e.g., orthogonal collocation over finite elements) to approximate the differentiated variables $Dy$ which are treated as auxiliary variables. Note that different choices of discretization scheme and measures can lead to more complex formulations, but these are omitted for concision in presentation. The key observation for this work is that the objective and constraints inherently exhibit a repeated structure over the discretization index $k$.

InfiniteOpt.jl is an open-source Julia package that builds upon the modeling capabilities of JuMP.jl to intuitively model InfiniteOpt problems. This modeling environment centers around the InfiniteModel object which stores InfiniteOpt problems using the unifying abstraction discussed in the current section. To enable arbitrary solution approaches for InfiniteModels, InfiniteOpt.jl provides an extendible backend interface which automates the creation and mapping of underlying models that can be optimized (referred to as OptimizerModels) to provide a seamless experience for users. By default, InfiniteOpt.jl employs the TranscriptionOpt backend which uses a suite of direct transcription methods to transform an InfiniteModel into a JuMP.jl model which can be solved using the large collection of solvers supported by JuMP.jl (Lubin et al., 2023). Although this approach handles diverse model structures, one drawback is that JuMP.jl models are unable to leverage the repeated structure inherent in direct transcription formulations to boost computational performance. This motivates the development of InfiniteExaModels.jl as an alternative backend for solving InfiniteModels via direct transcription as discussed in Section 3. To learn more about InfiniteOpt.jl, detailed documentation, tutorials, and examples are available at <https://infiniteopt.github.io/InfiniteOpt.jl/stable/>.

* + 1. SIMD Modeling Abstraction for Nonlinear Programs

The SIMD-NLP modeling abstraction captures such repetitive patterns and facilitates the development of algorithms that exploit these patterns. The repetitive pattern in this abstraction allows for the evaluation of the model and derivative equations with SIMD parallelism. The general SIMD-NLP problem formulation is:

|  |  |
| --- | --- |
| $$\begin{matrix}\min\_{\overline{z}\leq z\leq \overline{z}}&\sum\_{l\in L}^{}\sum\_{i\in I\_{l}}^{}h\_{l}\left(z;p\_{i,l}\right)& \\s.t.&g\_{m}\left(z;q\_{k}\right)+\sum\_{n\in N\_{m}}^{}\sum\_{j\in J\_{n}}^{}f\_{n}\left(z;s\_{j,n}\right)=0,&m\in M,k\in K\_{m}\end{matrix}$$ | (3) |

where $z\in \left[\overline{z},\overline{z}\right]$ are decision variables, $h\_{l}\left(⋅\right)$, $g\_{m}\left(⋅\right),f\_{n}\left(⋅\right)$ represent the repetitive patterns constituting the objective and constraint functions, and $p, q, s$ are data parameters. We note that only the data parameters change with each repetition indexed over $i, k, j$, meaning the algebraic structure in $h\_{l}\left(⋅\right)$, $g\_{m}\left(⋅\right),f\_{n}\left(⋅\right)$ remains constant.

ExaModels.jl implements an algebraic modeling interface to construct SIMD-NLP models where the user specifies the optimization model equations via an iterator, which allows us to capture the repetitive patterns in the model equations and make them available to the AD backend. This enables us to construct derivative evaluation kernels for each computational pattern using a symbolic expression tree and reverse-mode AD. These kernels are compiled and executed over multiple data to numerically evaluate the derivative. This strategy facilitates parallelized sparse AD on GPUs (Shin et al., 2023), providing a remarkable speedup. However, even with serial execution on CPUs, the tailored derivative kernels provide more efficient derivative evaluations relative to conventional AD tools, as we demonstrate in Section 4.

* 1. InfiniteExaModels.jl

The repeated discretized structure of InfiniteOpt problems can be translated directly into the SIMD-NLP formulation behind ExaModels.jl shown in Problem (3). This is accomplished by appending all the discretized infinite variables $Dy\_{k},y\_{k} $onto the finite variables $z$, transforming the inequality constraints into equalities via slack variables to obtain $g\_{m}\left(⋅\right)$, refactoring $α\_{k}h\left(⋅\right) $to obtain $h\_{l}\left(⋅\right)$, and setting $f\_{n}\left(⋅\right) = 0$. Inspired by this observation, InfiniteExaModels.jl is a new backend for InfiniteOpt.jl to convert nonlinear InfiniteModels into ExaModels using the same direct transcription methods supported by TranscriptionOpt. Principally it performs this conversion using the exa\_model method which creates an ExaModel along with all the mappings between the model objects such that the optimal values can be queried once the ExaModel has been solved using a solver supported by NLPModels.jl. One current limitation of InfiniteExaModels.jl is the use of metaprogramming to generate generators for ExaModels.jl which induces additional compilation when first calling the AD kernels. However, this will soon be remedied by directly constructing the symbolic expression trees used by ExaModels. We refer the reader to <https://github.com/infiniteopt/InfiniteExaModels.jl> to learn more.

Table 1. Numerical results for quadrotor optimal control

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ndisc | nvar | ncon | **ExaModels** | **JuMP** | **AMPL** |
| deriv. time | total time | deriv. time | total time | deriv. time | total time |
| 2.0k | 36.0k | 36.0k | 0.02 | 0.25 | 0.24 | 0.48 | 0.14 | 0.4 |
| 4.0k | 72.0k | 72.0k | 0.04 | 0.55 | 0.48 | 1.01 | 0.33 | 0.88 |
| 8.0k | 144.0k | 144.0k | 0.16 | 1.35 | 1.04 | 2.24 | 0.71 | 1.99 |
| 16.0k | 288.0k | 288.0k | 0.2 | 2.94 | 1.97 | 4.7 | 1.48 | 4.39 |

Table 2. Numerical results for stochastic optimal power flow

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| nscen | nvar | ncon | **ExaModels** | **JuMP** | **AMPL** |
| deriv. time | total time | deriv. time | total time | deriv. time | total time |
| 2.0k | 68.0k | 68.0k | 0.2 | 1.8 | 1.91 | 3.53 | 0.87 | 2.38 |
| 4.0k | 136.0k | 136.0k | 0.7 | 5.78 | 4.98 | 10.13 | 2.06 | 6.81 |
| 8.0k | 272.0k | 272.0k | 1.08 | 17.84 | 10.45 | 27.08 | 4.51 | 19.93 |
| 16.0k | 544.0k | 544.0k | 2.19 | 59.7 | 28.82 | 86.63 | 8.7 | 60.42 |

* 1. Cases Studies

In this section, we demonstrate the capabilities of InfiniteExaModels.jl with two examples: quadrotor optimal control and stochastic optimal power flow. We formulate each problem using InfiniteOpt.jl and solve the underlying discretized model with three different AD backends: JuMP,jl (i.e., MathOptInterface.Nonlinear.ReverseAD), AMPL (via AmplNLWriter.jl), and ExaModels.jl (via InfiniteExaModels.jl). Each is solved using Ipopt and we compare each AD backend based on the derivative evaluation times. The same solution is obtained for all different modeling tools. The source code and implementation details to reproduce the numerical results are available at <https://github.com/infiniteopt/InfiniteExaModels.jl/tree/main/examples>. All case studies were run on a server computer with two Intel(R) Xeon(R) Gold 6140 CPUs.

* + 1. Quadrotor Optimal Control

We consider the quadrotor optimal control problem presented by Hehn and D’Andrea (2011) and use a sinusoidal setpoint trajectory. We formulate the optimal control problems with a 60 second time horizon and discretize it using a varied number of uniformly spaced discretization points (ndisc). The numerical results are shown in Table 1 and show that the InfiniteExaModels.jl backend consistently outperforms the other conventional backends. On average, AD on ExaModels.jl is 8.7 times faster than JuMP.jl, and 5.6 times faster than AMPL. This substantially improves the solution time since the AD comprises nearly half of solution time using JuMP.jl.

* + 1. Stochastic AC Optimal Power Flow

We model a two-stage stochastic AC optimal power flow (ACOPF) problem based on the formulation presented in Coffrin et. al., (2018) augmented with multivariate Gaussian bus loads. We use the pglib\_opf\_case3\_lmbd test case from the pglib-opf library (Babaeinejadsarookolaee et. al., 2019) and vary the number of scenarios (nscen) used to discretize the model. The numerical results are shown in Table 2 and again the InfiniteExaModels.jl backend is the most performant. On average, the ExaModels.jl AD is 11.5 times faster than JuMP.jl, and 4.4 times faster than AMPL.

* 1. Conclusions and Future Outlook

We introduced InfiniteExaModels.jl as an effective tool to efficiently solve nonlinear InfiniteOpt problems. It bridges the intuitive modeling environment provided by InfiniteOpt.jl with the efficient SIMD-NLP modeling abstraction behind ExaModels.jl to accelerate AD performance by exploiting the repetitive structures inherent in discretized InfiniteOpt problems. With serial CPU computations, our numerical experiments in optimal control and stochastic ACOPF demonstrate approximately an order-of-magnitude reduction in NLP derivative evaluation times relative to state-of-the-art tools (i.e, JuMP.jl, AMPL). The SIMD-NLP abstraction also enables GPU acceleration within the solution routines which has potential to further accelerate solution times. In future work, we will investigate how InfiniteExaModels.jl performs using GPU-based AD evaluations in combination with the GPU interior-point solver available with MadNLP.jl.

References

S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris et al., 2019, The power grid library for benchmarking ac optimal power flow algorithms, arXiv preprint arXiv:1908.02788

L. D. Beal, D. C. Hill, R. A. Martin, J. D. Hedengren, 2018, Gekko optimization suite, Processes 6 (8), 106

L. T. Biegler, 2010, Nonlinear programming: concepts, algorithms, and applications to chemical processes, SIAM, Philadelphia, USA

W. Chen, L. T. Biegler, S. G. Muñoz, 2016, An approach for simultaneous estimation of reaction kinetics and curve resolution from process and spectral data, Journal of Chemometrics 30 (9), 506–522

C. Coffrin, R. Bent, K. Sundar, Y. Ng, M. Lubin, 2018, Powermodels. jl: An open-source framework for exploring power flow formulations, In 2018 Power Systems Computation Conference (PSCC), pp. 1-8. IEEE, Dublin, Ireland

W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, J. D. Siirola, et al., 2017, Pyomo-optimization modeling in python, Vol. 67, Springer

M. Hehn and R. D'Andrea, 2011, Quadrocopter trajectory generation and control, IFAC proceedings Volumes 44, no. 1: 1485-1491

H. Hijazi, G. Wang, C. Coffrin, 2018, Gravity: A mathematical modeling language for optimization and machine learning, NIPS 2018 Workshop MLOSS, Montreal, Canada

M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat, J. P. Vielma, 2023, Jump 1.0: recent improvements to a modeling language for mathematical optimization, Mathematical Programming Computation, 1–9

B. Nicholson, J. D. Siirola, J.-P. Watson, V. M. Zavala, L. T. Biegler, 2018, pyomo. dae: A modeling and automatic discretization framework for optimization with differential and algebraic equations, Mathematical Programming Computation 10, 187–223

J. L. Pulsipher, W. Zhang, T. J. Hongisto, V. M. Zavala, 2022, A unifying modeling abstraction for infinite-dimensional optimization, Computers & Chemical Engineering 156, 107567

S. Shin, F. Pacaud, M. Anitescu, 2023, Accelerating optimal power flow with gpus: Simd abstraction of nonlinear programs and condensed-space interior-point methods, arXiv preprint arXiv:2307.16830