# DynamicOED.jl: A Julia package for solving optimum experimental design problems

Carl Julius Martensen [1*¶], **Christoph Plate** [1*], and **Sebastian Sager** [1]

1 Otto von Guericke University Magdeburg, Germany ¶ Corresponding author * These authors contributed equally.

Summary

Optimum experimental design (OED) problems are typically encountered when unknown or uncertain parameters of mathematical models are to be estimated from an observable, maybe even controllable, process. In this scenario, OED can be used to decide on an experimental setup before collecting the data, i.e., deciding on when to measure and / or how to stimulate a dynamic process in order to maximize the amount of information gathered such that the parameters can be accurately estimated.

Our software package, DynamicOED.jl, facilitates the solution of optimum experimental design problems for dynamical systems. Following ideas presented in Sager (2013), we cast the OED problem into an optimal control problem. This is done by augmenting the user-provided system of ordinary differential equations (ODE) or differential algebraic equations (DAE) with their variational differential (algebraic) equations and the differential equation governing the evolution of the Fisher information matrix (FIM). A suitable criterion based on the FIM is then optimized in the resulting optimal control problem using a direct *first discretize, then optimize* approach.

Statement of need

`DynamicOED.jl` is a Julia (Bezanson et al., 2017) package for solving optimum experimental design problems. Solving OED problems is of interest for several reasons. First, all model-based optimization strategies rely on the knowledge of the accurate values of the model's parameters. Second, computing optimal experimental designs before performing the actual experiments to collect data allows to reduce the number of needed experiments or measurements. This is important in practical applications when measuring quantities of interest is only possible to a limited extent, e.g., due to high costs of performing the measurements.

Our package is designed for high flexibility and ease of use. For formulating the underlying dynamical system, our package is based on the `ODESystem` from `ModelingToolkit.jl` (Ma et al., 2022). This enables researchers and modelers to easily investigate and analyze their models and allows them to collect insightful data for their parameter estimation problems.

To our knowledge, this is the first dedicated package for solving general optimal experimental design problems with dynamical systems written in the Julia programming language. It may therefore be a valuable resource to different communities dealing with experimental data and parameter estimation problems.

Problem statement and usage example

The problem we are interested in solving reads

$$\min_{x,G,F,z,w} \quad \phi(F(t_f))$$

$$\text{s.t.} \quad 0 = f(\dot{x}(t), x(t), u(t), p)$$
$$0 = f_{\dot{x}}(\dot{x}(t), x(t), u(t), p)\dot{G}(t) + f_x(\dot{x}(t), x(t), u(t), p)G(t)$$
$$+ f_p(\dot{x}(t), x(t), u(t), p),$$
$$\dot{F}(t) = \sum_{i=1}^{n_h} w_i(t)(h_x^i(x(t))G(t))^\top (h_x^i(x(t))G(t)),$$
$$\dot{z}(t) = w,$$
$$x(0) = x_0, \quad G(0) = 0, \quad F(0) = 0, \quad z(0) = 0,$$
$$u(t) \in \mathcal{U},$$
$$w(t) \in \mathcal{W},$$
$$z(t_f) - M \leq 0,$$

where $\mathcal{T} = [t_0, t_f]$ is the fixed time horizon and $x : \mathcal{T} \mapsto \mathbb{R}^{n_x}$ are the differential states. The first and second constraint denote the dynamical system and the sensitivities of the solution of the dynamical system with respect to the uncertain parameters, respectively, and are given in an implicit form. Here, $f_{\dot{x}}$, (f_x) denote the partial derivative of f with respect to $\dot{x}$ and (x). The objective $\phi(F(t_f))$ of Bolza type is a suited objective function, e.g., the D-criterion $\phi(F(t_f)) = \det(F^{-1}(t_f))$. The evolution of the symmetric FIM $F : \mathcal{T} \mapsto \mathbb{R}^{n_p \times n_p}$ is governed by the measurement function $h : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_h}$, the sensitivities $G : \mathcal{T} \mapsto \mathbb{R}^{n_x \times n_p}$ and the sampling decisions $w(t) \in \{0,1\}^{n_h}$. The latter are the main optimization variables and represent the decision whether to measure at a given time point or not. In our direct approach, these variables are discretized, hence we write $w(t) \in \{0,1\}^{N_w \times n_h}$, where N_w is the (user-supplied) number of discretization intervals on $\mathcal{T}$. The sampling decisions are then accumulated in the variables z and constrained by $M \in \mathbb{R}_+^{n_h}$. The controls $u \in \mathcal{U}$ can either be fixed or also be viewed as optimization variables after discretization.

For more information on optimal experimental design for DAEs and their sensitivity analysis, we refer to Körkel (2002) and Li et al. (2000).

The functionality in this package integrates into Julia's SciML ecosystem. The model is provided in symbolic form as an ODESystem using ModelingToolkit.jl (Ma et al., 2022) with additional frequency information for the observed and control variables. Both ODE or DAE systems can be provided. DynamicOED.jl augments the given system symbolically with its sensitivity equations and the dynamics of the FIM. The resulting system together with a sufficient information criterion defines an OEDProblem, solveable using DifferentialEquations.jl (Rackauckas & Nie, 2017). Here, all sampling and control decisions are discretized in time and can be used to model additional constraints. At last, the OEDProblem can be transformed into an OptimizationProblem as a sufficient input to Optimization.jl (Dixit & Rackauckas, 2023). Here, a variety of optimization solvers for nonlinear programming and mixed-integer nonlinear programming available as additional backends, e.g., Juniper (Kröger et al., 2018) or Ipopt (Wächter & Biegler, 2006). A simple example demonstrates the usage of DynamicOED.jl for the Lotka-Volterra system (Sager, 2013).

Figure 1 shows the solution of the example above including the differential states, sensitivities G and the sampling decisions w. More examples can be found in the documentation.

```julia
using DynamicOED
using ModelingToolkit
using Optimization, OptimizationMOI, Ipopt

@variables t
@variables x(t)=0.5 [description="Biomass Prey"]
@variables y(t)=0.7 [description="Biomass Predator"]
@variables u(t) [description="Control"]
@parameters p[1:2]=[1.0;1.0] [description="Fixed Parameters", tunable=false]
@parameters p_est[1:2]=[1.0;1.0] [description="Tunable Parameters", tunable=true]
D = Differential(t)
@variables obs(t)[1:2] [description = "Observed", measurement_rate=96]
obs = collect(obs)

@named lotka_volterra = ODESystem(
    [
        D(x) ~   p[1]*x - p_est[1]*x*y;
        D(y) ~  -p[2]*y + p_est[2]*x*y
    ], tspan = (0.0, 12.0),
    observed = obs .~ [x; y]
)
@named oed_system = OEDSystem(lotka_volterra)
oed_problem = OEDProblem(structural_simplify(oed_system), DCriterion())

optimization_variables = states(oed_problem)

w1, w2 = keys(optimization_variables.measurements)

constraint_equations = [
    sum(optimization_variables.measurements[w1]) ≤ 32,
    sum(optimization_variables.measurements[w2]) ≤ 32,
]

@named constraint_system = ConstraintsSystem(
    constraint_equations, optimization_variables, Num[]
)

optimization_problem = OptimizationProblem(
    oed_problem, AutoForwardDiff(), constraints = constraint_system,
    integer_constraints = false
)

optimal_design = solve(optimization_problem, Ipopt.Optimizer();
                        hessian_approximation="limited-memory")
```

Extensions

Several extensions are planned for the future. First, a multiple shooting approach is planned. Also, other steps to increase the efficiency of our implementation may be considered. For example, in the case of fixed initial values and controls, the integration of x and G need to be done only once and can be decoupled from the numerical integration of F and the subsequent optimization over w.
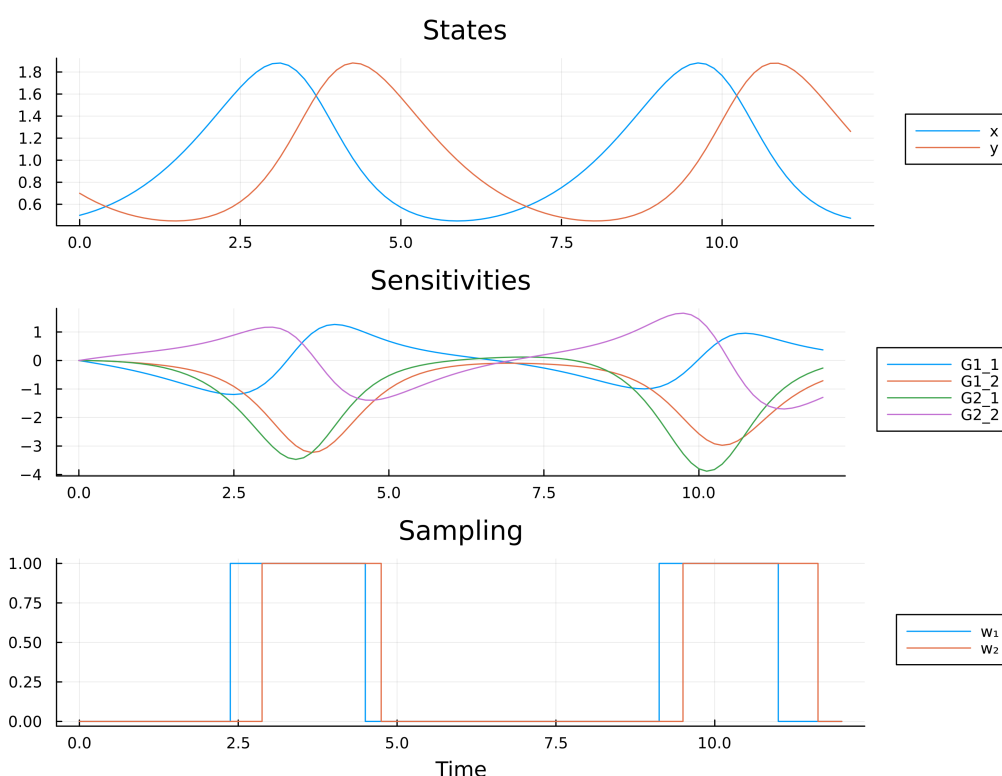
Figure 1: Differential states, sensitivities of the states with respect to the parameters and the optimal sampling design for Lotka-Volterra system.

Acknowledgements

References

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Rev.*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Dixit, V. K., & Rackauckas, C. (2023). *Optimization.jl: A unified optimization package*. Zenodo. https://doi.org/10.5281/zenodo.7738525

Körkel, S. (2002). *Numerische methoden für optimale versuchsplanungsprobleme bei nichtlinearen DAE-Modellen* [PhD thesis, Universität Heidelberg]. https://doi.org/10.11588/heidok.00002980

Kröger, O., Coffrin, C., Hijazi, H., & Nagarajan, H. (2018). Juniper: An open-source nonlinear branch-and-bound solver in Julia. In W.-J. van Hoeve (Ed.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 377–386). Springer International Publishing. https://doi.org/10.1007/978-3-319-93031-2_27

Li, S., Petzold, L., & Zhu, W. (2000). Sensitivity analysis of differential–algebraic equations: A comparison of methods on a special problem. *Applied Numerical Mathematics*, *32*(2), 161–174. https://doi.org/10.1016/S0168-9274(99)00020-3

Ma, Y., Gowda, S., Anantharaman, R., Laughman, C., Shah, V., & Rackauckas, C. (2022).

ModelingToolkit: A composable graph transformation system for equation-based modeling (No. arXiv:2103.05244). arXiv. https://doi.org/10.48550/arXiv.2103.05244

Rackauckas, C., & Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. *The Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.151

Sager, S. (2013). Sampling decisions in optimum experimental design in the light of Pontryagin's maximum principle. *SIAM J. Control Optim.*, *51*(4), 3181–3207. https://doi.org/10.1137/110835098

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y