SAND2021-14271C

Sandia National Laboratories

Dakota and Pyomo for Closed and Open Box Controller Gain Tuning

Tutorial Session: Open Source Software for Control

60th Conference on Decision and Control

Kyle R. Williams, J. Justin Wilbanks, Rachel Schlossman, David Kozlowski, and Julie Parish

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

SAND2021-13485

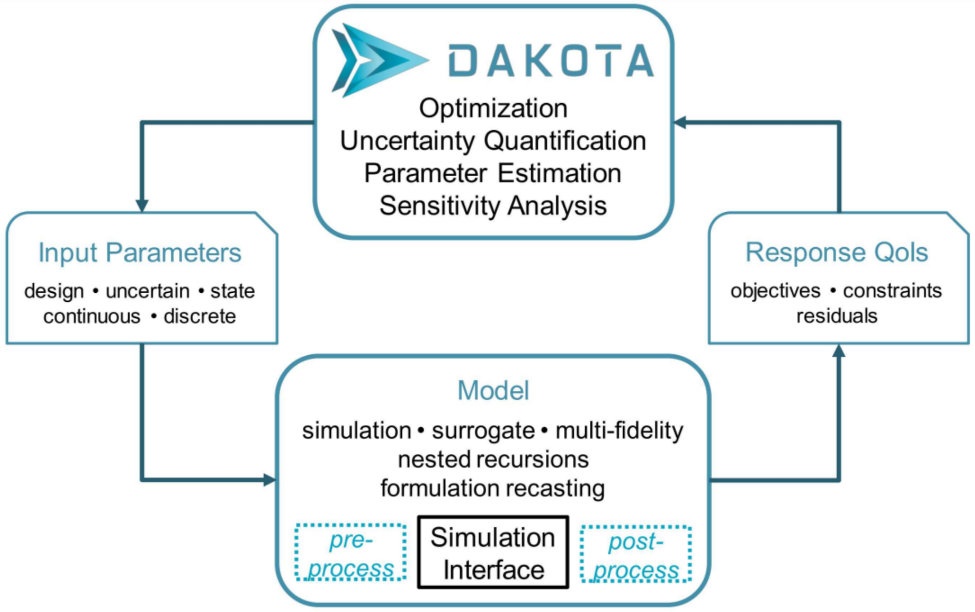
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Introduction

- Control engineering: stabilization of dynamic systems
  - Developing mathematical model
  - Synthesizing a control law
  - **Tuning the parameters**
- Many toolboxes exist
  - Tool suites for linear time invariant systems
  - Robust control
  - Multi-objective software
- Dakota and Pyomo for control system tuning
  - Open source, developed at Sandia
  - Dakota: written in C++, can operated in “closed box” form via direct interaction with an input/output model
  - Pyomo: written in Python, requires transparent “open box” model

Background

- Dakota
  - Complex optimization problems
  - Closed box interface: only needs I/O
  - Can interact with MATLAB, Simulink, GNU Octave, Python, etc.
  - Implements a variety of optimization algorithms (genetic algorithms, gradient based)
- Pyomo (**Py**thon **O**ptimization **Mo**deling)
  - Open box: needs modeling equations
  - Supports a wide range of optimization problems (LP, QP, NLP, MIP, SP)
  - Supports differential algebraic equations (DAEs)
  - Transparent parallelization of subproblems using Python parallel communication libraries



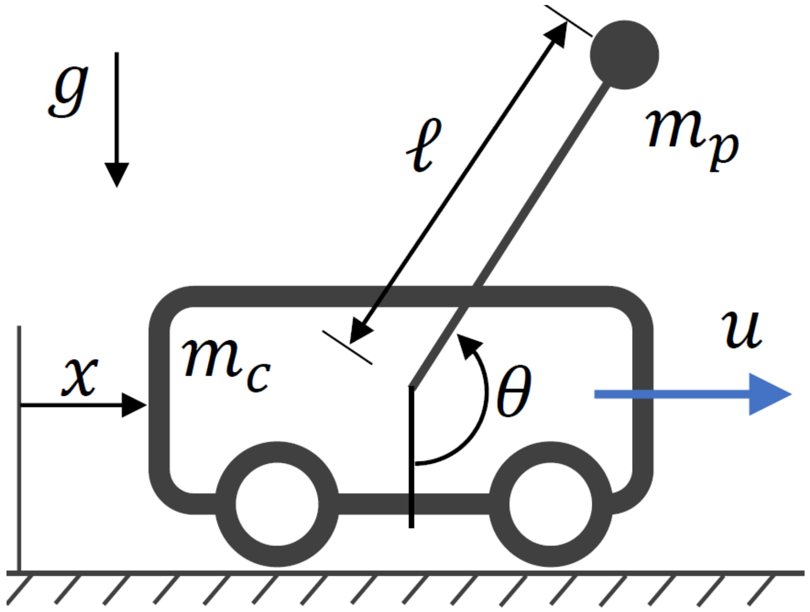
Available at https://dakota.sandia.gov

PYOMO

Available at https://www.pyomo.org

Example problem: cart-pole system

- Nonlinear, underactuated system
- Goal is to balance the pole at the unstable equilibrium (vertical position)
- Controlled in two ways:
  - Linear quadratic regulator (LQR) design
  - Partial feedback linearization (PFL) design



$$\ddot{x} = \frac{1}{m_c + m_p \sin^2\theta}\left[u + m_p \sin\theta(\ell\dot{\theta}^2 + g\cos\theta)\right] \quad (1a)$$

$$\ddot{\theta} = \frac{1}{\ell(m_c + m_p \sin^2\theta)}\left[-u\cos\theta - m_p\ell\dot{\theta}^2\cos\theta\sin\theta - (m_c + m_p)g\sin\theta\right] \quad (1b)$$

LQR optimization: setup

$$\text{minimize} \quad J = \int_0^T \tilde{\mathbf{x}}(t)^T Q \tilde{\mathbf{x}}(t) + \tilde{\mathbf{u}}(t)^T R \tilde{\mathbf{u}}(t) dt$$ ← LQR objective

$$\text{subject to} \quad \text{Nonlinear dynamics given by (1)}$$ ← Full nonlinear dynamics (Pyomo solver taking gradients)

$$\tilde{\mathbf{u}}(t) = -K \tilde{\mathbf{x}}(t)$$ ← Linear control law

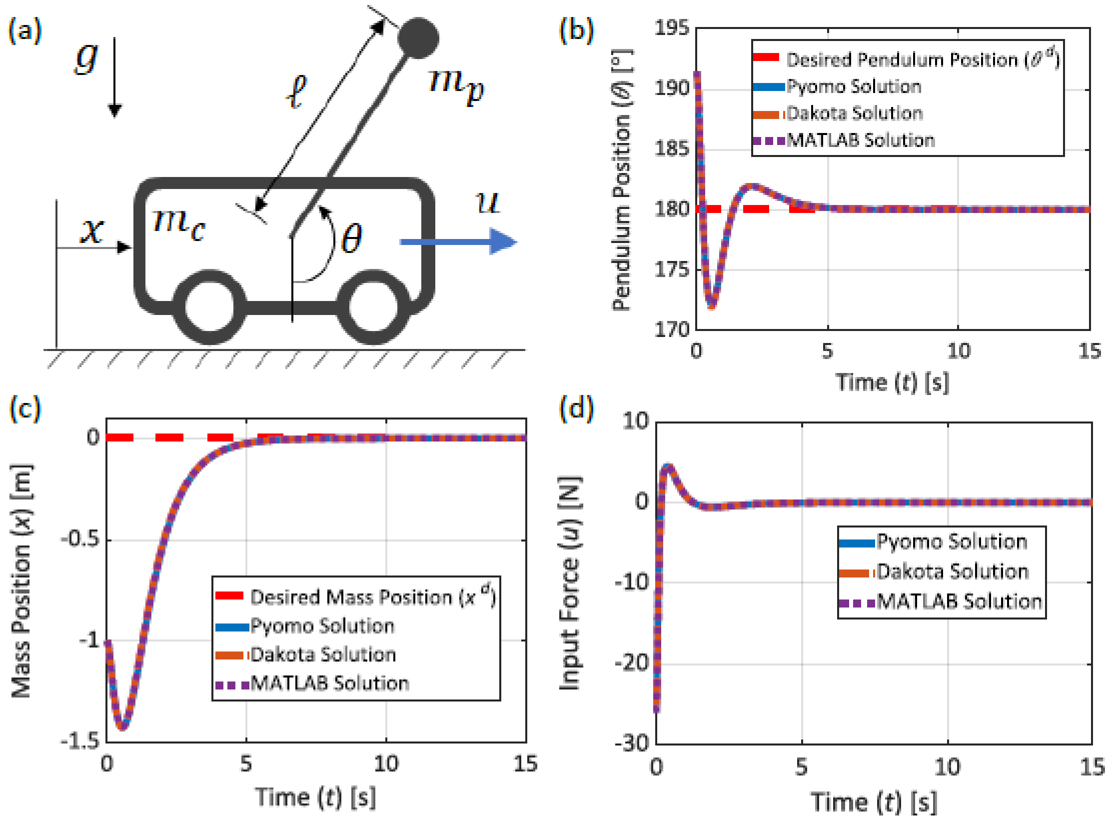
- Pyomo steps
  1. Create Pyomo model: state / control / objective vars, derivative vars, time horizon
  2. Define dynamics constraints "for t in m.T: m.dx1dt[t] = m.x2[t]"
  3. List boundary conditions and initialize guess
  4. Solve: define solver, in our case **IPOPT**

- Dakota steps
  1. *.in file: input file which specifies solver type (**COLINY EA**), ranges and initial guesses
  2. *.sh/*.vbs: opens MATLAB in either Windows or Linux
  3. *_Wrapper.m: a MATLAB file used to specifiy the current parameter choices made by Dakota
  4. *.m: additional MATLAB files which contain dynamics and control laws (can call *.slx files); called from *_Wrapper.m

LQR optimization: results

- Validation
  - Pyomo and Dakota produce controller gains which stabilize $x = 0, \theta = \pi$
  - Pyomo compute time: ~2 seconds
  - Dakota compute time: ~30 minutes

  Fundamental tradeoff: upfront setup time for computation time



- Gain comparison

Method	K_x	$K_{\dot{x}}$	K_θ	$K_{\dot{\theta}}$	J
Pyomo	-6.82	-12.45	92.32	28.68	60.92
Dakota	-6.85	-12.42	91.13	28.10	60.86
MATLAB	-7.07	-12.98	94.94	29.48	61.06

Partial feedback linearization

- Why PFL?
  - Linear control law: LQR design only valid near the chosen equilibrium point
  - Nonlinear control law: Partial feedback linearization demonstrates “swing up” capabilities
- Step 1: prescribe desired dynamics: $\ddot{\theta} = v \equiv k_d(\dot{\theta}^d - \dot{\theta}) + k_p(\theta^d - \theta)$
- Step 2: solve for control law

$$u = -\frac{1}{\cos\theta}\left[v\ell(m_c + m_p \sin^2\theta) + m_p \ell \dot{\theta}^2 \cos\theta \sin\theta + (m_c + m_p) g \sin\theta\right]$$

- Assuming a perfect model with perfect cancelation, we get the following transfer function which is stable for $\mathrm{Re}\{s^2 + k_d s + k_p\} < 0$

$$\theta(s) = \underbrace{\frac{k_p + k_d s}{s^2 + k_d s + k_p}}_{T(s)} \theta^d(s)$$

Additional study:
- Apply time delayed control $u(t - \tau)$ signal
- Closed loop transfer function is lost

Partial feedback linearization

$$\text{minimize}\quad J=\int_0^T\left(\theta^d(t)-\theta(t)\right)^2dt+\boxed{w_1k_p+W_{ST}t_s}$$

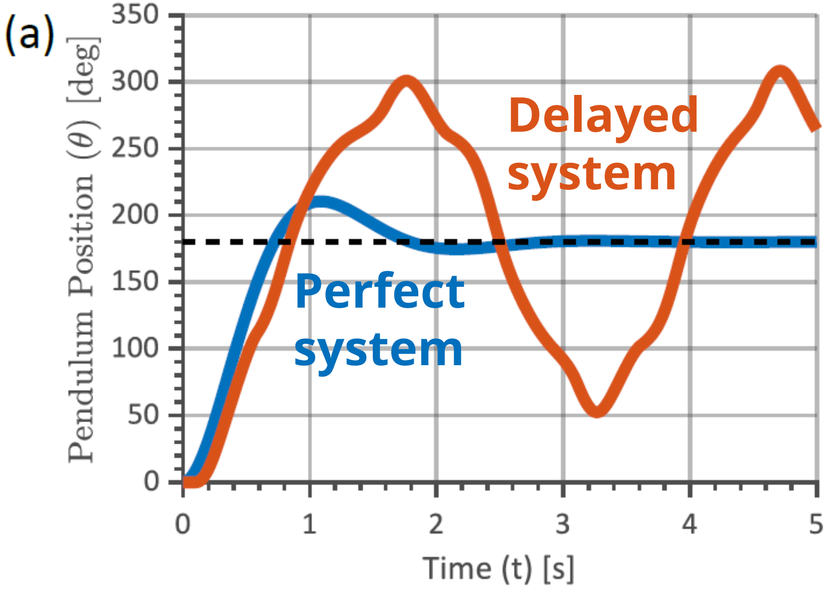
subject to nonlinear dynamics, prev nonlinear control law

Penalty on settling time t_s

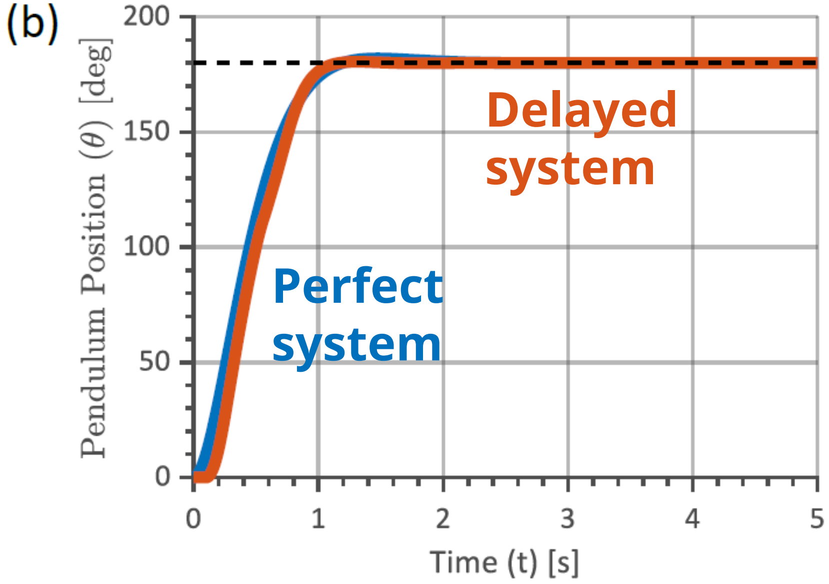
- Pyomo: use w_1=0.02 as surrogate
- Dakota: use t_s directly

Pyomo: with and without 0.1s control input delay

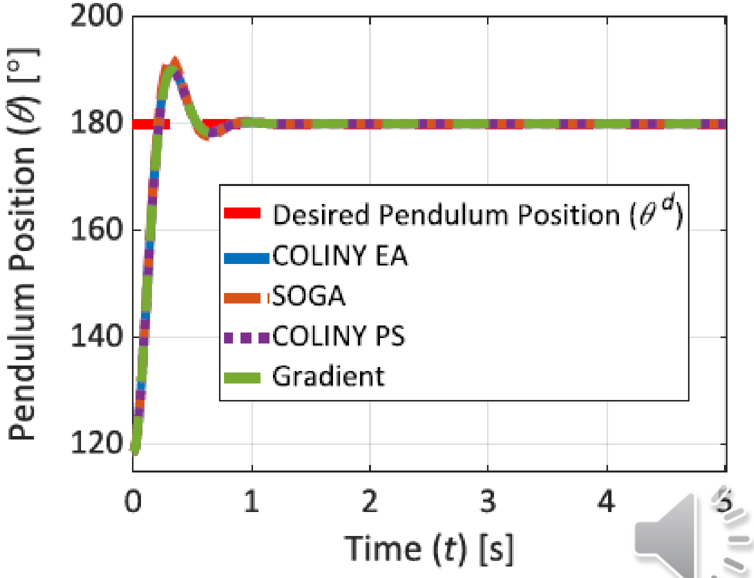
Nominal gains



Improved gains, which consider time delay



Dakota with 0.1s delay



Summary

- Dakota and Pyomo are powerful tools for control design
- Primary tradeoff: setup time vs. optimization time
  - Pyomo enjoys fast computation times, but model setup is non-trivial
  - Dakota computation times can be very long, but enjoys freedom in optimization criteria
- Single input system shown in this work, easily extensible to to multi-input systems (see ref for example)
- Although not shown here, Bayesian Optimization is another optimization framework to be considered
  - In [18], BO is used to tune Q and R matrices of an LQR synthesis to induce some desired behavior