SwitchTimeOpt.jl Documentation

Release 0.1

Bartolomeo Stellato

December 06, 2016

Contents

SwitchTimeOpt.jl is a Julia package to easily define and efficiently solve switching time optimization (STO) problems for linear and nonlinear systems. SwitchTimeOpt.jl supports a wide variety of nonlinear solvers through MathProgBase.jl interface such as Ipopt, KNITRO, NLopt.

CHAPTER 1

Installation

You can easily install the package by running

```
Pkg.add("SwitchTimeOpt")
```

This does not automatically install any nonblinear solver. To install, for example, Ipopt, just run

```
Pkg.add("Ipopt")
```

To install other solvers we refer the user to the JuliaOpt page.

CHAPTER 2

Quick Example

Consider the switching time optimization problem in the form

$$\begin{aligned} \underset{\tau}{\text{minimize}} \quad & \int_{t_0}^{t_f} \|x(t)\|_2^2 \, \mathrm{d}t \\ \text{subject to} \quad & \dot{x}(t) = \begin{cases} A_0 x(t) & t < \tau \\ A_1 x(t) & t \geq \tau \end{cases} \\ & x(0) = x_0 \\ & 0 \leq \tau \leq T \end{aligned}$$

with variable $\tau \in \mathbb{R}$, the dynamics defined by matrices $A_0, A_1 \in \mathbb{R}^{n \times n}$ and the initial state $x_0 \in \mathbb{R}^n$.

This problem can be solved by SwitchTimeOpt.jl as follows

```
using SwitchTimeOpt
using Ipopt

# Time Interval
t0 = 0.0; tf = 1.0

# Initial State
x0 = [1.0; 1.0]

# Dynamics
A = Array(Float64, 2, 2, 2)
A[:, :, 1] = randn(2, 2)  # A_0 matrix
A[:, :, 2] = randn(2, 2)  # A_1 matrix

# Create Problem
m = stoproblem(x0, A, t0=t0, tf=tf)

# Solve Problem
solve!(m)

# Get optimal Solution
tauopt = gettau(m)

# Get optimum value
objval = getobjval(m)
```

CHAPTER 3

Optimization

We now describe the interface for defining and solving switching time optimization problems.

3.1 Problem Definition

This package allows us to define and solve problems in the form

$$\begin{array}{ll} \underset{\delta}{\text{minimize}} & \int_{t_0}^{T_\delta} x(t)^\top Q x(t)\ \mathrm{d}t + x(T_\delta)^\top Q x(T_\delta) \\ \text{subject to} & \dot{x}(t) = f_i(x(t)) \quad t \in [\tau_i, \tau_{i+1}) \quad i = 0, \ldots, N \\ & x(0) = x_0 \\ & \delta \in \Delta \end{array}$$

where the decision variable is the vector of $N+1$ intervals $\delta = \begin{bmatrix}\delta_0 & \ldots & \delta_N\end{bmatrix}^\top \in \mathbb{R}^{N+1}$ such that $\delta_i = \tau_{i+1} - \tau_i$. Each interval δ_i defines how long the i-th dynamics are active. The state trajectory is $x(t) \in \mathbb{R}^n$. The value T_δ is defined as the final time when intervals δ are applied, i.e.

$$T_\delta = \sum_{i=0}^{N} \delta_i.$$

The set Δ defines the set of feasible intervals

$$\Delta = \left\{\delta \in \mathbb{R}^{N+1} \;\middle|\; T_\delta = T \wedge 0 \leq lb_i \leq \delta_i \leq ub_i\ \forall i\right\}$$

The variable T defines the desired final time of the interval. The scalars lb_i and ub_i define additional constraints on the interval in case we would like to have a minimum or a maximum time in which the i-th dynamics are active.

3.1.1 Linear Dynamics

In case when the dynamics are linear of the form

$$\dot{x}(t) = A_i x(t), \quad t \in [\tau_i, \tau_{i+1})$$

we can define a 3-dimensional matrix `A` whose slices `A[:,:,i]` represent dynamics A_{i-1}:

```
A = Array(Float64, n, n, N+1)
A[:, :, i] = ...   # Dynamics A_{i-1}
...
```

The switching time optimization problem can be quickle defined as

```
p = stoproblem(x0, A)
```

Where `x0` is the initial state vector x_0 and `A` is the 3-dimensional matrix defining the dynamics.

3.1.2 Noninear Dynamics

Given a nonlinear system defined by dynamics

$$\dot{x}(t) = f(x(t), u(t))$$

where $u(t)$ the input vector assuming integer values u_i between switching instants

$$u(t) = u_i \quad t \in [\tau_i, \tau_{i+1}),$$

we can define our switched nonlinear system as

$$\dot{x}(t) = f(x(t), u_i) = f_i(x(t)) \quad t \in [\tau_i, \tau_{i+1}).$$

To create the optimization problem we need to define the nonlinear dynamics by means of an additional function

```
function nldyn(x, ui)
  ...
end
```

returning the vector of states derivatives. The variable `x` is the state $x(t)$ and `ui` is the input vector u_i. Moreover, we need to define the jacobian of the switched dynamics with respect to the system states

$$J_{f_i} = \frac{\partial f_i(x(t))}{\partial x(t)}$$

by means of an additional function

```
function jac_nldyn(x, ui)
  ...
end
```

Note that the function `jac_nldyn` returns a matrix having in each row the gradient of every component of the function $f_i(x(t))$ with respect to each state component. Last element necessary to construct the matrix `U` having a column each integer input vector `ui`. Then, we can define the switching time optimization problem as:

```
p = stoproblem(x0, nldyn, jac_nldyn, U)
```

Note: The nonliner switched system optimization operates by introducing additional linearization points at an equally spaced linearization grid. To set the number of linearization points to 100 for example, it is just necessary to add an extra argument to the previous function call as follows:

```
p = stoproblem(x0, nldyn, jac_nldyn, U, ngrid = 100)
```

where `ngrid` defines the number of linearization points.

3.1.3 Optional Arguments

There are many additional keyword arguments that can be be passed to the `stoproblem(...)` function to customize the optimization problem.

Parameter	Description	Default value
`t0`	Initial Time t_0	`0.0`
`tf`	Final Time t_0	`1.0`
`Q`	Cost matrix Q	`eye(n)`
`lb`	Vector of lower bounds lb_i	`zeros(N+1)`
`ub`	Vector of lower bounds ub_i	`Inf*ones(N+1)`
`tau0ws`	Warm starting initial switching times	Equally spaced between `t0` and `tf`
`solver`	MathProgbase.jl solver	`IpoptSolver()`

3.2 Problem Solution

Once the problem is defined, it can be solved by simply running

```
solve!(p)
```

3.2.1 Choosing Solver

Any NLP solver supported by JuliaOpt may be used through MathProgBase.jl interface. The default solver is Ipopt. To use KNITRO solver with the linear example, it is just necessary to specify an `AbstractMathProgSolver` object (see here for more details) when the problem is created

```
using KNITRO
p = stoproblem(x0, A, solver = KnitroSolver())
```

All the solver-specific options can be passed when creating the `AbstractMathProgSolver` object: algorithm types (first/second order methods), tolerances, verbosity and so on.

3.2.2 Obtaining Results

The optimal cost function and the optimal switching times and intervals can be obtained as follows:

```
objval = getobjval(p)
tauopt = gettau(p)
deltaopt = getdelta(p)
```

We can get the execution time (including the time for the function calls) and the status of the solver by executing:

```
stat = getstat(p)
soltime = getsoltime(p)
```

3.2.3 Optimizing in a Loop

The toolbox is suited for receeding horizon implementations. To run the optimization in a loop it is just necessary to update the value of the current state `x0` and to update the warm starting point `tau0ws` which is usually chosen as the optimal solution at the previous optimizaton.

To set the initial state at `x0` it is just necessary to return

```
setx0!(m, x0)
```

We can set the warm starting point at `tau0ws` with

```
setwarmstart!(m, tau0ws)
```

CHAPTER 4

Simulation

The system can be simulated with the obtained switching times by running

```
x, xsw, optval, t = simulate(m)
x, xsw, optval, t = simulate(m, tau)       # Specify switching time vector
x, xsw, optval, t = simulate(m, tau, t)    # Specify switching times and time vectors
```

The outputs of the simulation are

- `x` State trajectory. Each $x(t)$ can be obtained as `x[:, i]`
- `xsw` States at each switching time. Each $x(\tau_i)$ can be obtained as `xsw[:, i]`
- `optval` Simulated value function optimum
- `t` Time vector during the simulation

4.1 Additional Functions for Nonlinear Dynamics

In case of nonlinear dynamics it is possible to simulate the system linearized at the linearization points obtained after the optimization

```
x, xsw, optval, t = simulatelinearized(m)
x, xsw, optval, t = simulatelinearized(m, tau, t)  # Specify switching times time vector
```

In addition, it is possible to easily obtain the input vector trajectory at each time instant by running

```
u, t = simulateinput(m)
u, t = simulateinput(m, t)  # Specify time vector
```

Each vector $u(t)$ can be obtained by slicing the output `u[:, i]`.

CHAPTER 5

Citing this package

If you use SwitchTimeOpt.jl for published work, we encourage you to cite the following paper:

```
@article{2016arXiv160808597S,
  author = {{Stellato}, B. and {Ober-Bl{\"o}baum}, S. and {Goulart}, P.~J.},
  title = "{Second-Order Switching Time Optimization for Switched Dynamical Systems}",
  journal = {ArXiv e-prints},
  archivePrefix = "arXiv",
  eprint = {1608.08597},
  primaryClass = "math.OC",
  keywords = {Mathematics - Optimization and Control},
  year = 2016,
  month = aug
}
```