ModelPredictiveControl.jl: advanced process control made easy using JuMP

Francis Gagnon

Jumine, Quebec City, Canada

Saturday, July 20, 2024

Outline

Outline

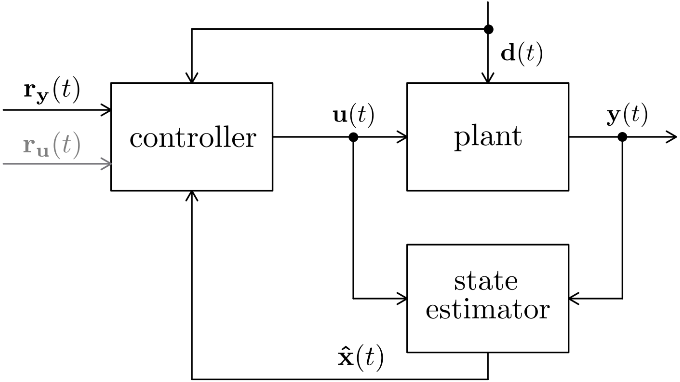
Introduction

Why Julia and JuMP for MPC?

- process control design $\implies$ MATLAB
  - \+ mature, cohesive, well-documented
  - – closed-source, expensive, slow
- model predictive control (MPC)
  - can reduce wastes
  - real-time optimization
  - code generation: two-language problem
- free and open source software
  - accessibility
  - research and development
- Julia and JuMP
  - \+ fast, expressive, math-oriented
  - \+ solver independence
  - – small community and ecosystem

Outline

Modern Control Topology



- $\mathbf{u}$ manipulated inputs
- $\mathbf{y}$ measured outputs
- $\mathbf{d}$ measured disturbances
- $\hat{\mathbf{x}}$ estimated states
- $\mathbf{r_y}$ output setpoints
- $\mathbf{r_u}$ input setpoints

ModelPredictiveControl.jl

`PredictiveController` ← `StateEstimator` ← `SimModel`

`SimModel`

`LinModel` state-space description of the plant (system identification):

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B_u}\mathbf{u}(t) + \mathbf{B_d}\mathbf{d}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D_d}\mathbf{d}(t)\end{aligned}$$

`NonLinModel` nonlinear ODE system:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}\Big(\mathbf{x}(t), \mathbf{u}(t), \mathbf{d}(t)\Big) \\ \mathbf{y}(t) &= \mathbf{h}\Big(\mathbf{x}(t), \mathbf{d}(t)\Big)\end{aligned}$$

StateEstimator

- SteadyKalmanFilter
- KalmanFilter
- Luenberger
- UnscentedKalmanFilter
- ExtendedKalmanFilter
- MovingHorizonEstimator
- InternalModel

`MovingHorizonEstimator`

- analog of MPC but for state estimation
- estimate $\hat{\mathbf{x}}$ using optimization under constrains
- estimation horizon H_e (how many time steps in the past)
- `LinModel` $\rightarrow$ quadratic programming (`OSQP.jl`)
- `NonLinModel` $\rightarrow$ nonlinear programming (`Ipopt.jl`)

PredictiveController

- LinMPC
- ExplicitMPC
- NonLinMPC

LinMPC

- prediction horizon H_p (how many time steps in the future)
- control horizon H_c (how many control moves)
- solve at each time step t:

$$\min_{\Delta\mathbf{U},\epsilon} \underbrace{(\hat{\mathbf{R}}_\mathbf{y} - \hat{\mathbf{Y}})'\mathbf{M}_{H_p}(\hat{\mathbf{R}}_\mathbf{y} - \hat{\mathbf{Y}})}_{\text{output setpoint tracking}} + \underbrace{(\Delta\mathbf{U})'\mathbf{N}_{H_c}(\Delta\mathbf{U})}_{\text{move suppression}} + \underbrace{(\hat{\mathbf{R}}_\mathbf{u} - \mathbf{U})'\mathbf{L}_{H_p}(\hat{\mathbf{R}}_\mathbf{u} - \mathbf{U})}_{\text{input setpoint tracking}} + \underbrace{C\epsilon^2}_{\text{slack}}.$$

- subject to:
  - plant model
  - measured disturbances $\mathbf{d}(t)$ and state estimate $\hat{\mathbf{x}}(t)$
  - inputs $\mathbf{u}$ and outputs $\mathbf{y}$ soft/hard constraints
- quadratic programming (`OSQP.jl`)

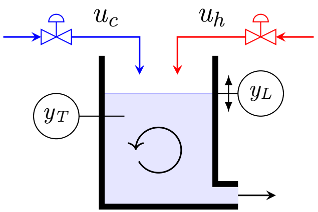
NonLinMPC

- same objective and constraints as LinMPC
- LinModel or NonLinModel
- additional nonlinear term for economical costs $EJ_E(\hat{\mathbf{Y}}, \mathbf{U}, \hat{\mathbf{D}})$
- user-defined J_E function
- nonlinear programming (Ipopt.jl)

Outline

Continuously Stirred-Tank Reactor (CSTR)

Linear MPC (1/2)



$$\mathbf{u} = \begin{bmatrix} u_c & u_h \end{bmatrix}^\mathsf{T}$$

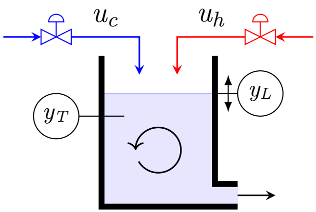
$$\mathbf{y} = \begin{bmatrix} y_L & y_T \end{bmatrix}^\mathsf{T}$$

$$\frac{\mathbf{y}(s)}{\mathbf{u}(s)} = \begin{bmatrix} \frac{1.90}{18s+1} & \frac{1.90}{18s+1} \\ \frac{-0.74}{8s+1} & \frac{0.74}{8s+4} \end{bmatrix}$$

```julia
using ModelPredictiveControl, ControlSystemsBase
G = [ tf(1.90, [18, 1]) tf(1.90, [18, 1])
      tf(-0.74,[8, 1])  tf(0.74, [8, 1]) ]
plant = LinModel(G, 2.0)
mpc = setconstraint!(LinMPC(plant), ymin=[45, -Inf])
function test_mpc(mpc, plant)
    N = 75; ry = [50, 30]; ul = 0
    U, Y, Ry = zeros(2, N), zeros(2, N), zeros(2, N)
    for i = 1:N
        i == 26 && (ry = [48, 35])
        i == 51 && (ul = -10)
        y = plant()
        u = mpc(ry)
        U[:,i], Y[:,i], Ry[:,i] = u, y, ry
        updatestate!(mpc, u, y)
        updatestate!(plant, u+[0,ul])
    end
    return U, Y, Ry
end
U_data, Y_data, Ry_data = test_mpc(mpc, plant)
res = SimResult(mpc, U_data, Y_data; Ry_data)
using Plots; plot(res)
```

Continuously Stirred-Tank Reactor (CSTR)

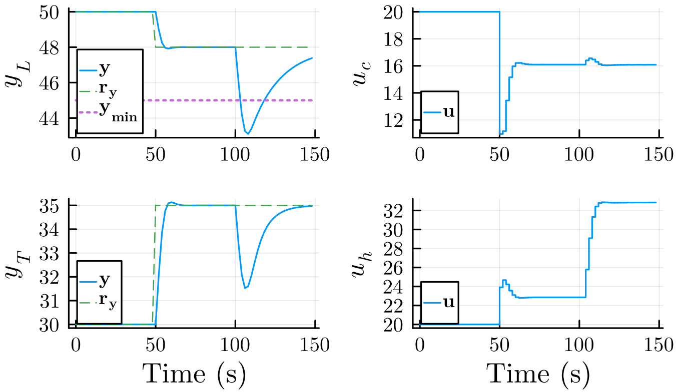
Linear MPC (2/2)



$$\mathbf{u} = \begin{bmatrix} u_c & u_h \end{bmatrix}^\intercal$$

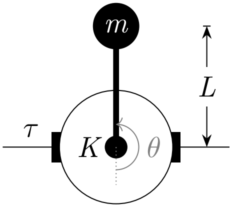
$$\mathbf{y} = \begin{bmatrix} y_L & y_T \end{bmatrix}^\intercal$$

$$\frac{\mathbf{y}(s)}{\mathbf{u}(s)} = \begin{bmatrix} \frac{1.90}{18s+1} & \frac{1.90}{18s+1} \\ \frac{-0.74}{8s+1} & \frac{0.74}{8s+4} \end{bmatrix}$$



Inverted Pendulum

Nonlinear MPC (1/2)



$$\dot{\theta} = \omega$$

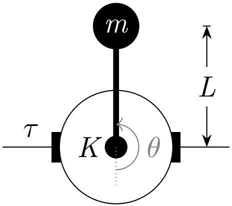
$$\dot{\omega} = -\frac{g}{l}\sin\theta - \frac{K}{m}\omega + \frac{1}{mL^2}\tau$$

```julia
using ModelPredictiveControl
function pendulum(par, x, u)
    g, L, K, m = par
    θ, ω = x[1], x[2]
    τ  = u[1]
    dθ = ω
    dω = -g/L*sin(θ) - K/m*ω + τ/m/L^2
    return [dθ, dω]
end
const par = (9.8, 0.4, 1.2, 0.3)
f(x, u, _ ) = pendulum(par, x, u)
h(x, _ )    = [180/π*x[1]]
nu, nx, ny, Ts = 1, 2, 1, 0.1
plant = NonLinModel(f, h, Ts, nu, nx, ny)
nmpc = NonLinMPC(plant; Hp=20, Hc=2, Mwt=[0.5], Nwt=[2.5])
nmpc = setconstraint!(nmpc; umin=[-1.5], umax=[+1.5])

...
```

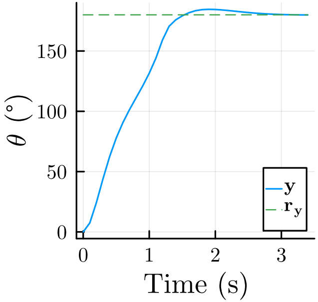
Inverted Pendulum

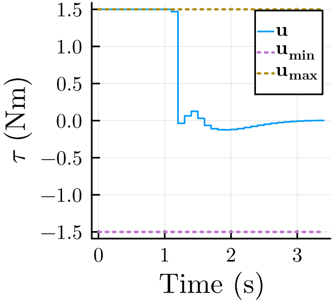
Nonlinear MPC (2/2)



$$\dot{\theta} = \omega$$

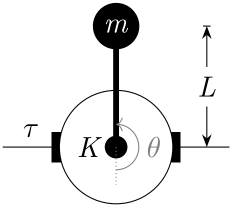
$$\dot{\omega} = -\frac{g}{l}\sin\theta - \frac{K}{m}\omega + \frac{1}{mL^2}\tau$$





Inverted Pendulum

Economic MPC

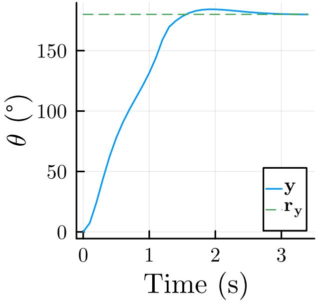


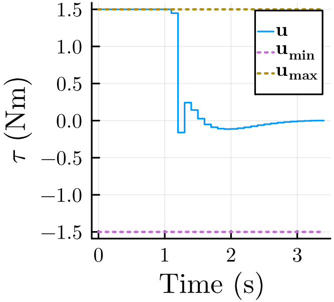
$$\dot{\theta} = \omega$$

$$\dot{\omega} = -\frac{g}{l}\sin\theta - \frac{K}{m}\omega + \frac{1}{mL^2}\tau$$

Work W as economical costs:

$$J_E = W = \int \tau(t)\omega(t)\mathrm{d}t \approx \sum \ldots$$





Energy Consumption

before: $W = 3.92$ J

after: $W = 3.89$ J

Benchmarks

			Median Time (s)	
Plant	Control	Solver	Julia	MATLAB
CSTR	Linear MPC	OS	0.0013	0.0196
CSTR	Linear MPC	AS	0.0044	0.0169
Pendulum	Nonlinear MPC	IP	0.7283	1.3373
Pendulum	Nonlinear MPC	SQ	0.3041	0.6565
Pendulum	Economic MPC	IP	0.7093	1.0852
Pendulum	Economic MPC	SQ	0.3382	0.7558

Outline

Conclusion

- Julia and JuMP are a powerful combination for MPC
- case studies show the simplicity and flexibility of the package
- benchmarks show the great performance of the toolbox
- future control projects at Jumine may use the package

francis.gagnon@jumine.ca