GEKKO Optimization Suite

CAST Webinar

8/23/18

Logan Beal

Quick Start Guide

- Install GEKKO with **pip**

```
pip install gekko
```

- 18 Examples: https://goo.gl/guA2a6

Overview

- Free AML in Python for LP, QP, NLP, MILP, and MINLP
- Easy to install, intuitive syntax
- Integrated model formulation, solution, and visualization

Overview

- Specialization in dynamic optimization
- Full-featured MHE and MPC for industrial, online applications
- Model reduction and initialization strategies
- Warm-start initialization from time-shifting of solutions

Python

Why Python

- Easy, flexible, pretty
- Popular
  - IEEE #1
  - TIOBE #4
- Large community
  - ~9% new stack overflow questions
- Many powerful packages
  - matplotlib, numpy/scipy, pandas, openOPC/pymodbus
- pip install

AML Features

Standard AML Problem

$$\min_{x} \; J(x)$$
$$0 = f(x)$$
$$0 \leq g(x)$$

- AMPL, GAMS, Pyomo, JuMP, CasADi, etc all address this problem
- Automatic Differentiation
- Fast Fortran back-end
- Bundled solvers

HS71

Problem Statement

$$\min_x \; x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

$$\text{s.t.} \;\; 0 = x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40$$

$$0 \le x_1 x_2 x_3 x_4 - 25$$

$$1 \le x_1, x_2, x_3, x_4 \le 5$$

$$x_0 = (1,5,5,1)$$

GEKKO Code

```
from gekko import GEKKO
m = GEKKO() # Initialize gekko
# Initialize variables
x1 = m.Var(1, lb = 1, ub = 5)
x2 = m.Var(5, lb = 1, ub = 5)
x3 = m.Var(5, lb = 1, ub = 5)
x4 = m.Var(1, lb = 1, ub = 5)
# Equations
m.Equation(x1 * x2 * x3 * x4 >= 25)
m.Equation(x1**2 + x2**2 + x3**2 + x4**2 == 40)
m.Obj(x1 * x4 * (x1 + x2 + x3) + x3)
# Objective
m.options.IMODE = 3# Steady state optimization
m.solve()# Solve
print('Results')
print('x1: ' + str(x1.value))
print('x2: ' + str(x2.value))
print('x3: ' + str(x3.value))
print('x4: ' + str(x4.value))
```

Dynamic Optimization

ODE Discretization

$$\min_x J\left(\frac{dx}{dt}, x\right)$$
$$0 = f\left(\frac{dx}{dt}, x\right)$$
$$0 \leq g\left(\frac{dx}{dt}, x\right)$$

- Extend AML general problem to natively handle differential equations
- Orthogonal collocation on finite elements
- Provide model time discretization:

```
m = GEKKO()
m.time = [0,1,2,3]
```

Simultaneous vs Sequential

- Sequential
  - Feasible solutions
  - Easy to implement
  - Computationally inefficient
- Simultaneous
  - Faster for large problems
  - Easy to implement variable bounds
  - Difficult to implement correctly
  - Infeasible solutions are worthless

Common Implementations

- Model Predictive Control
  - IMODE = 6
- Moving Horizon Estimation
  - IMODE = 5
- Dynamic Real Time Optimization
- Economic Model Predictive Control

Custom Variable Types

- Fixed Variable
- Manipulated Variable
- State Variable
- Controlled Variable

Fixed Variable (FV)

- Fixed throughout the horizon
- Built-in tuning

```
m = GEKKO()
x = m.FV()
x.STATUS = 1
x.DMAX = 1
```

Manipulated Variable (MV)

- 0- or 1-order hold, like DCS
- Tuning options:
  - DCOST (Move suppression)
  - DMAX (Move constraint)

```python
m = GEKKO()
x = m.MV()
x.STATUS = 1
x.DCOST = 1
m.options.MV_TYPE = 0
```

State Variable

- Added attributes for inspection

```
m = GEKKO()
x = m.SV()
# ...
# model
# ...
m.solve()
print(x.PRED)
```

Controlled Variable

- Build objectives based on mode:
  - Minimize error to setpoint
  - Minimize error between model/measurements
- Tuning
  - TAU (Time constant)

```
m = GEKKO()
x = m.CV()
x.STATUS = 1 #build objective
m.options.CV_TYPE = 2 #squared error
x.SP = 5
# or
m.options.CV_TYPE = 1 #absolute error
x.SPHI = 6
x.SPLO = 4
```

Intermediates

- Reduce size of matrix math
- Ensure feasible solution to given equations
- Must be defined explicitly

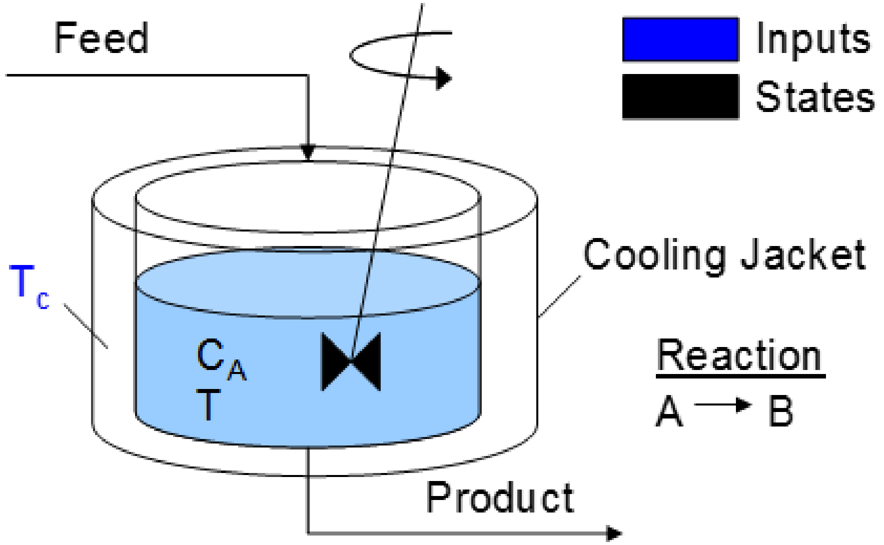
Model Reduction

```
#Rate equations
k = m.Var()
m.Equation(k == k0 * m.exp(-E/(R*T)))
m.Equation(rate == k * Ca)

# OR

k = m.Intermediate(k0 * m.exp(-E/(R*T)))
m.Equation(rate == k * Ca)
```

Examples

CSTR Model

- Model
- RTO
- NMPC
- Tune NMPC



Initialize

```
from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt

#%% NMPC model
m = GEKKO(remote=False)
m.time = np.linspace(0,5,51)
```

Variables

```
#Model Parameters
q = m.Param(value=100)
V = m.Param(value=100)
rho = m.Param(value=1000)
Cp = m.Param(value=0.239)
mdelH = m.Param(value=50000)
ER = m.Param(value=8750)
k0 = m.Param(value=7.2*10**10)
UA = m.Param(value=5*10**4)
Ca0 = m.Param(value=1)
T0 = m.Param(value=350)
#Variables
k = m.Var()
rate = m.Var()
T = m.Var(value=325,lb=250,ub=500)
#MV
Tc = m.MV(value=300)
#CV
Ca = m.CV(value=.7, ub=1, lb=0)
```

Equations

```
#Equations
m.Equation(k==k0*m.exp(-ER/T))
m.Equation(rate==k*Ca)
m.Equation(V* Ca.dt() == q*(Ca0-Ca)-V*rate)
m.Equation(rho*Cp*V* T.dt() == q*rho*Cp*(T0-T) + V*mdelH*rate + UA*(Tc-T))
```

Model

```
from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt

#%% NMPC model
m = GEKKO()
m.time = np.linspace(0,5,51)

#Model Parameters
q = m.Param(value=100)
V = m.Param(value=100)
rho = m.Param(value=1000)
Cp = m.Param(value=0.239)
mdelH = m.Param(value=50000)
ER = m.Param(value=8750)
k0 = m.Param(value=7.2*10**10)
UA = m.Param(value=5*10**4)
Ca0 = m.Param(value=1)
T0 = m.Param(value=350)
#Variables
k = m.Var()
rate = m.Var()
T = m.Var(value=325,lb=250,ub=500)
#MV
Tc = m.MV(value=300,lb=250,ub=350)
#CV
Ca = m.CV(value=.7, ub=1, lb=0)

#Equations
m.Equation(k==k0*m.exp(-ER/T))
m.Equation(rate==k*Ca)
m.Equation(V* Ca.dt() == q*(Ca0-Ca)-V*rate)
m.Equation(rho*Cp*V* T.dt() == q*rho*Cp*(T0-T) + V*mdelH*rate + UA*(Tc-T))
```

RTO: Real-Time Optimization

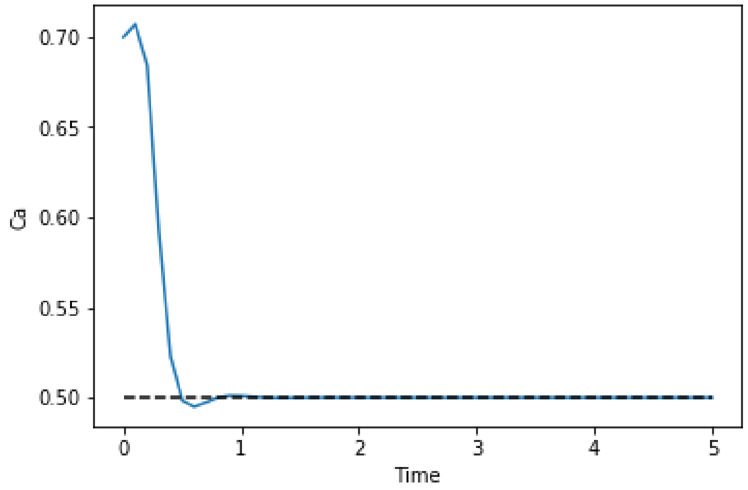
```
#Global options
m.options.IMODE = 3
m.options.CV_TYPE = 2

#MV tuning
Tc.STATUS = 1

#CV Tuning
Ca.STATUS = 1
Ca.SP = .5

m.solve()

print(T.value)
print(Tc.value)
```

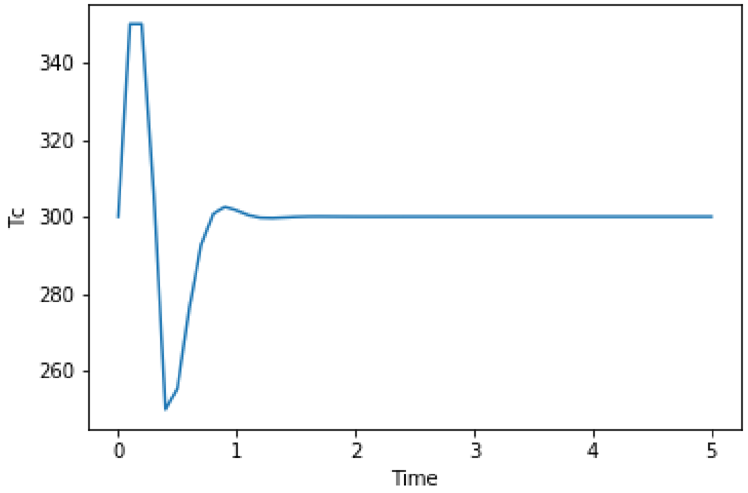
Results:

- [350.1639]
- [299.9514]

NMPC – Nonlinear Model Predictive Control

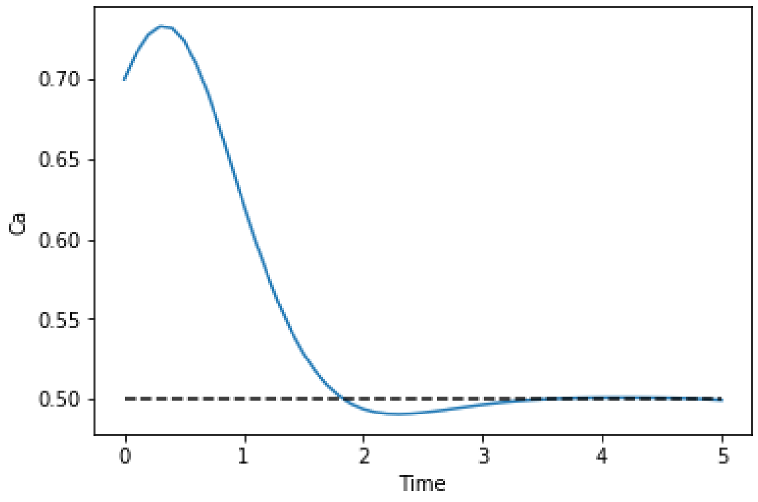
```
#Global options
m.options.IMODE = 6
m.options.CV_TYPE = 2

#MV tuning
Tc.STATUS = 1

#CV Tuning
Ca.STATUS = 1
Ca.SP = .5

m.solve()
```

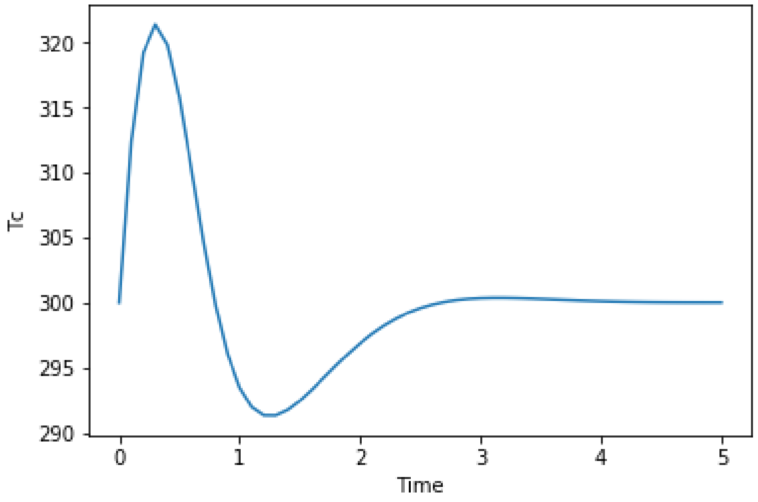




Tuning

```
#Global options
m.options.IMODE = 6
m.options.CV_TYPE = 2

#MV tuning
Tc.STATUS = 1
Tc.DCOST = .01

#CV Tuning
Ca.STATUS = 1
Ca.SP = .5

m.solve()
```





Online Model

- Generic first order model
  - Simulation and control
- Measurements
- Control moves
- Time shifting

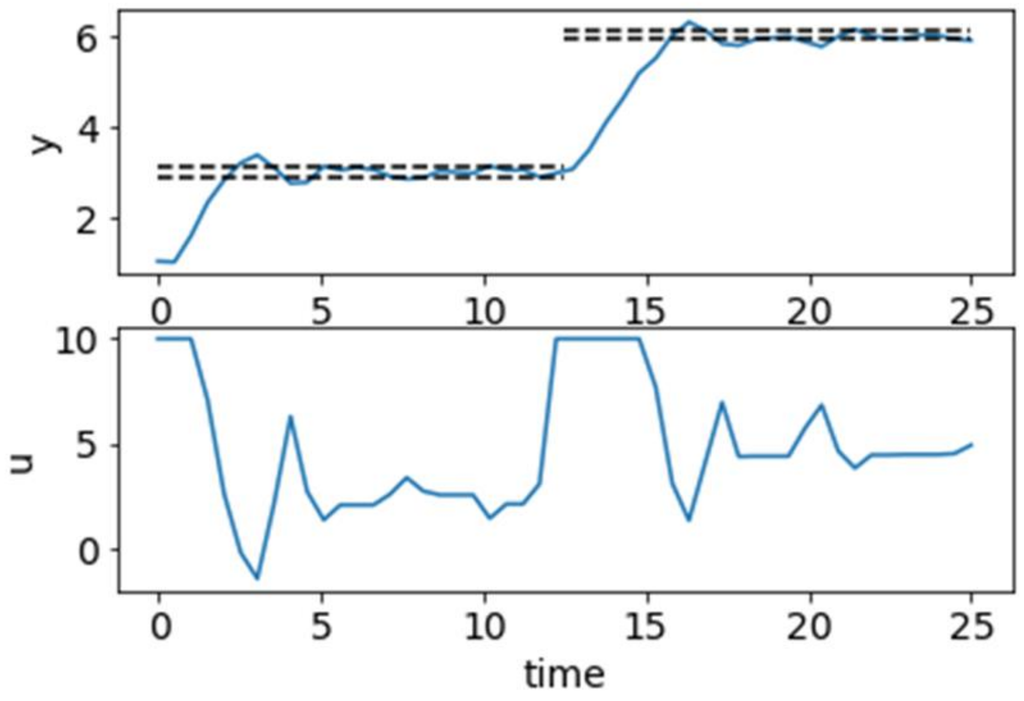
Models

```python
#%%Import packages
import numpy as np
from random import random
import matplotlib.pyplot as plt
from gekko import GEKKO

#%% Process
p = GEKKO()
p.time = [0,.5] #time discretization

#Parameters
p.u = p.MV()
p.K = p.Param(value=1.25) #gain
p.tau = p.Param(value=8) #time constant

#Variable
p.y = p.CV(1) #measurement

#Equations
p.Equation(p.tau * p.y.dt() == -p.y + p.K * p.u)

#options
p.options.IMODE = 4
p.options.NODES = 4

def process_simulator(meas):
    if meas is not None:
        p.u.MEAS = meas
    p.solve(disp=False)
    return p.y.MODEL + (random()-0.5)*0.2
```

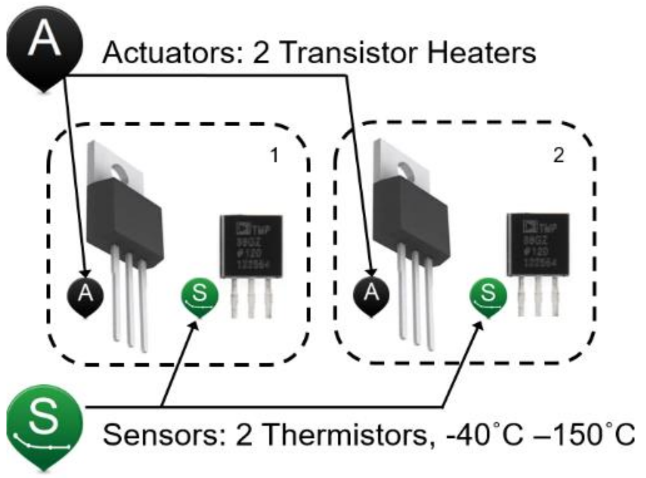
```python
#%% MPC Model
c = GEKKO()
c.time = np.linspace(0,5,11) # discretization (steps of 0.5)

#Parameters
u = c.MV(lb=-10,ub=10) #input
K = c.Param(value=1) #gain
tau = c.Param(value=10) #time constant

#Variables
y = c.CV(1)

#Equations
c.Equation(tau * y.dt() == -y + u * K)

#Options
c.options.IMODE = 6     #MPC
c.options.CV_TYPE = 1   #l1 norm
c.options.NODES = 3

y.STATUS = 1            #write MPC objective
y.FSTATUS = 1           #recieve measurements
y.SPHI = 3.1
y.SPLO = 2.9

u.STATUS = 1            #enable optimization of MV
u.FSTATUS = 0           #no feedback
u.DCOST = 0.05          #discourage unnecessary movement
```

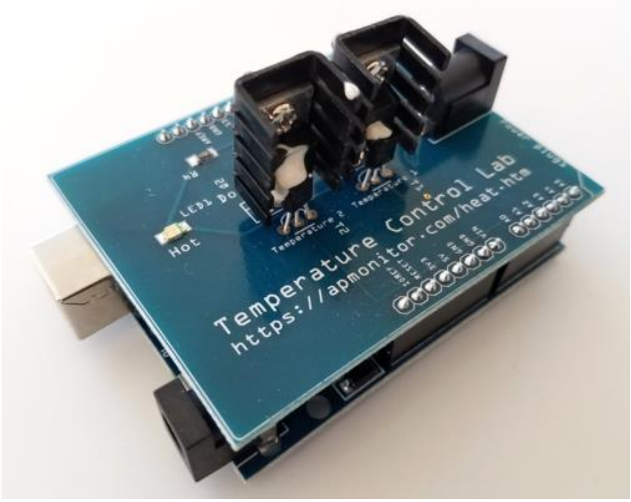
Time Loop

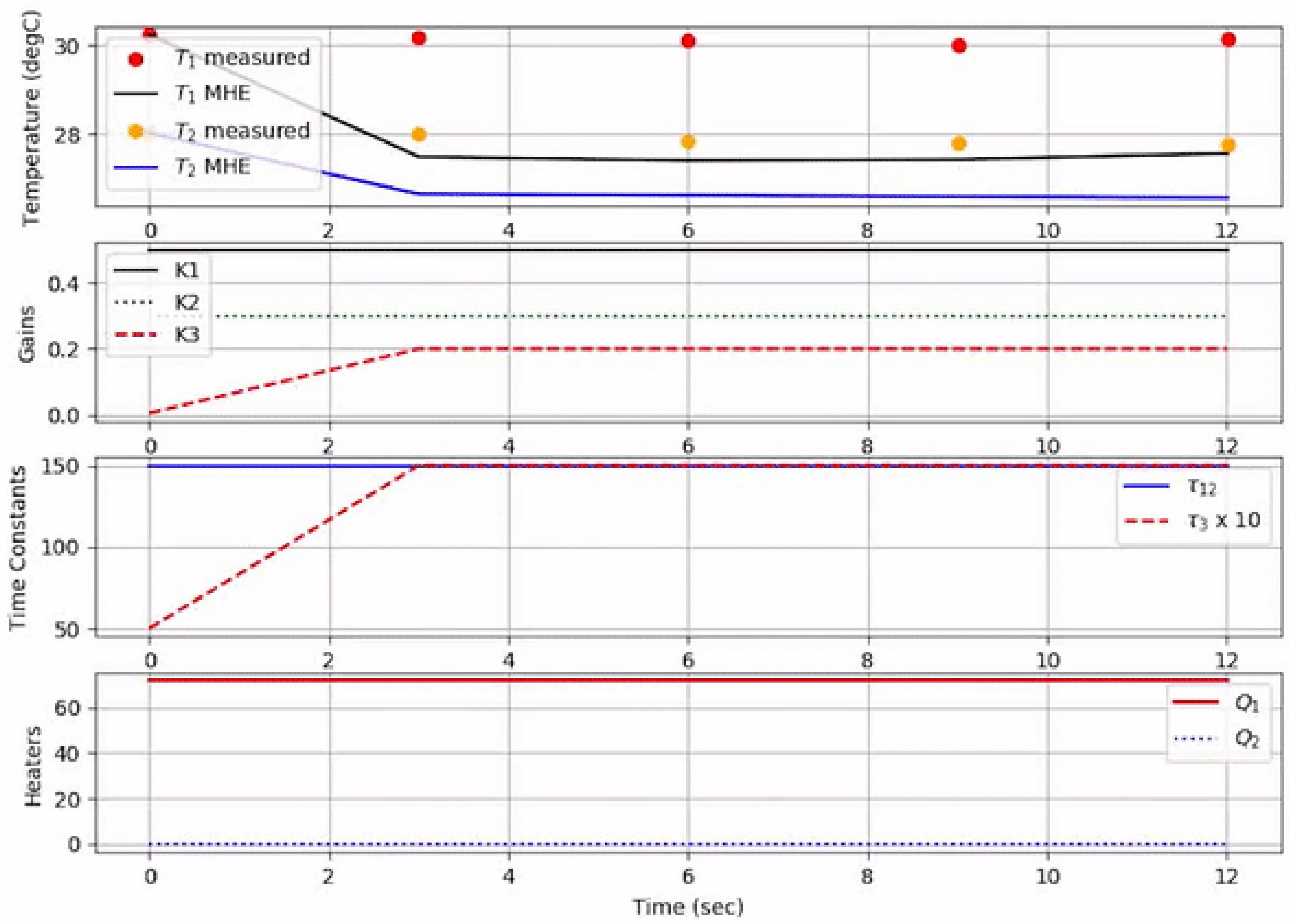
```
#%% Time loop
for i in range(50):
    if i == 24: ##change setpoint
        y.SPHI = 6.1
        y.SPLO = 5.9

    #process
    y.MEAS = process_simulator(u.NEWVAL)
    #controller
    c.solve()
```



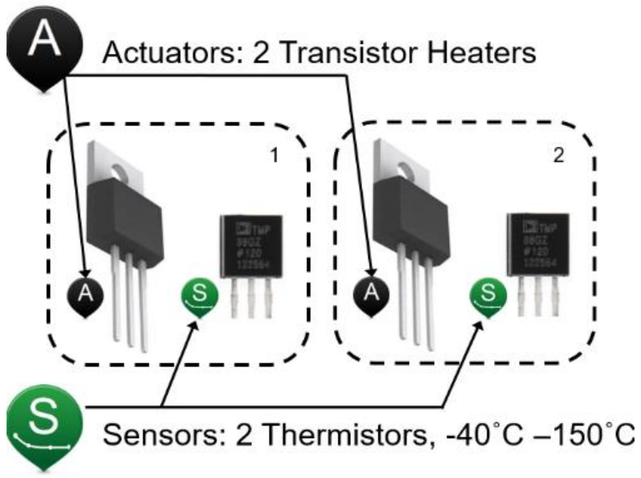
TCLab Estimation

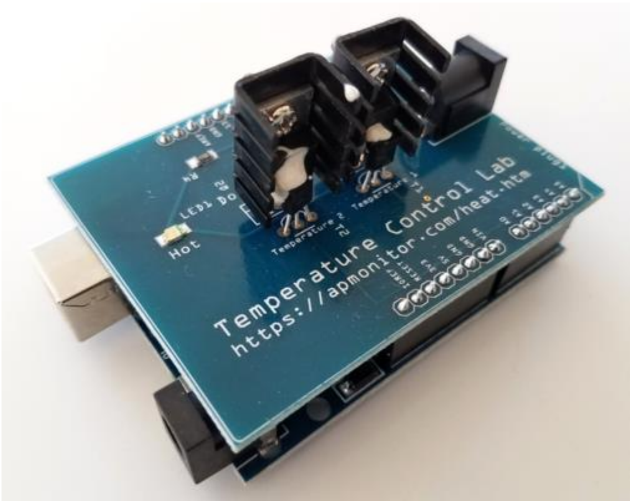


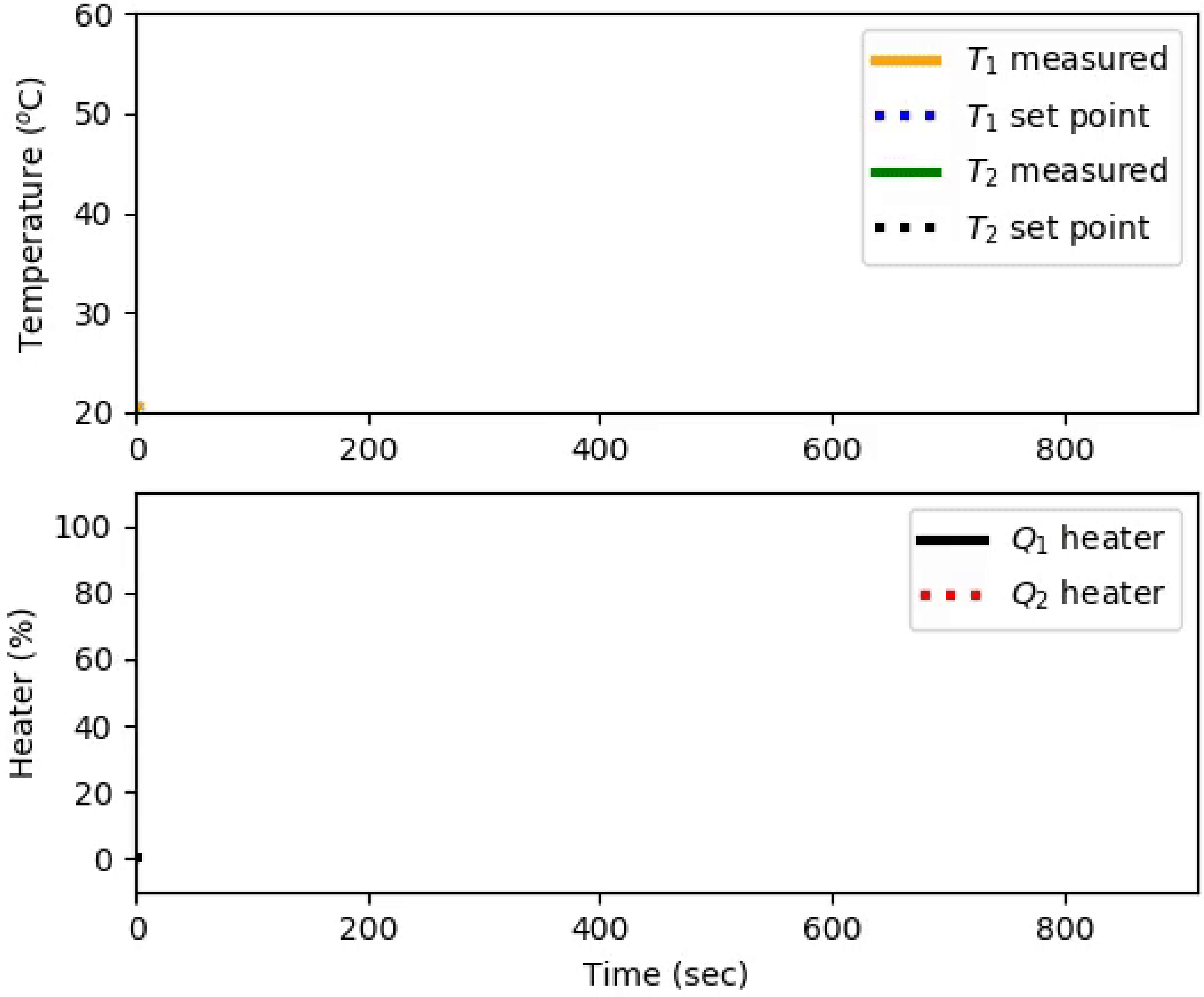




TCLab Predictive Control

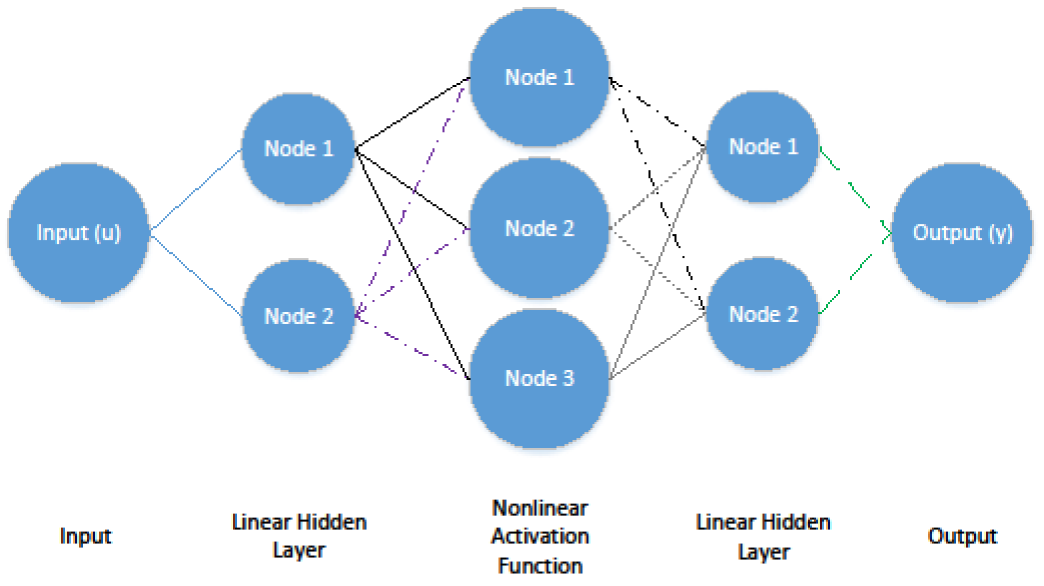


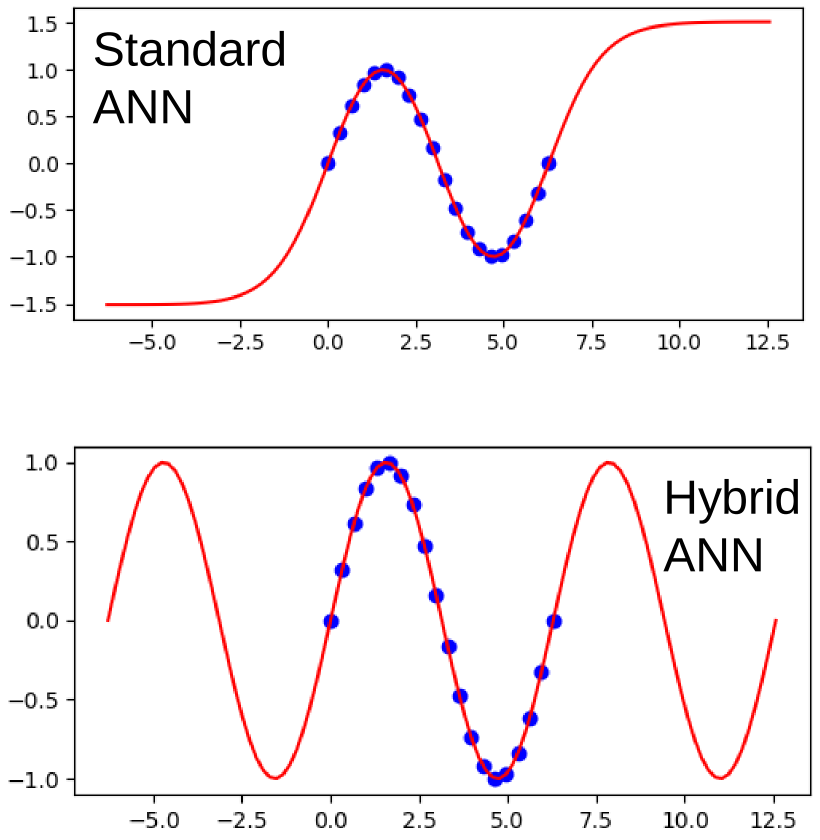




Hybrid Neural Networks

- Combine First Principles and Data Driven Modeling





GUI

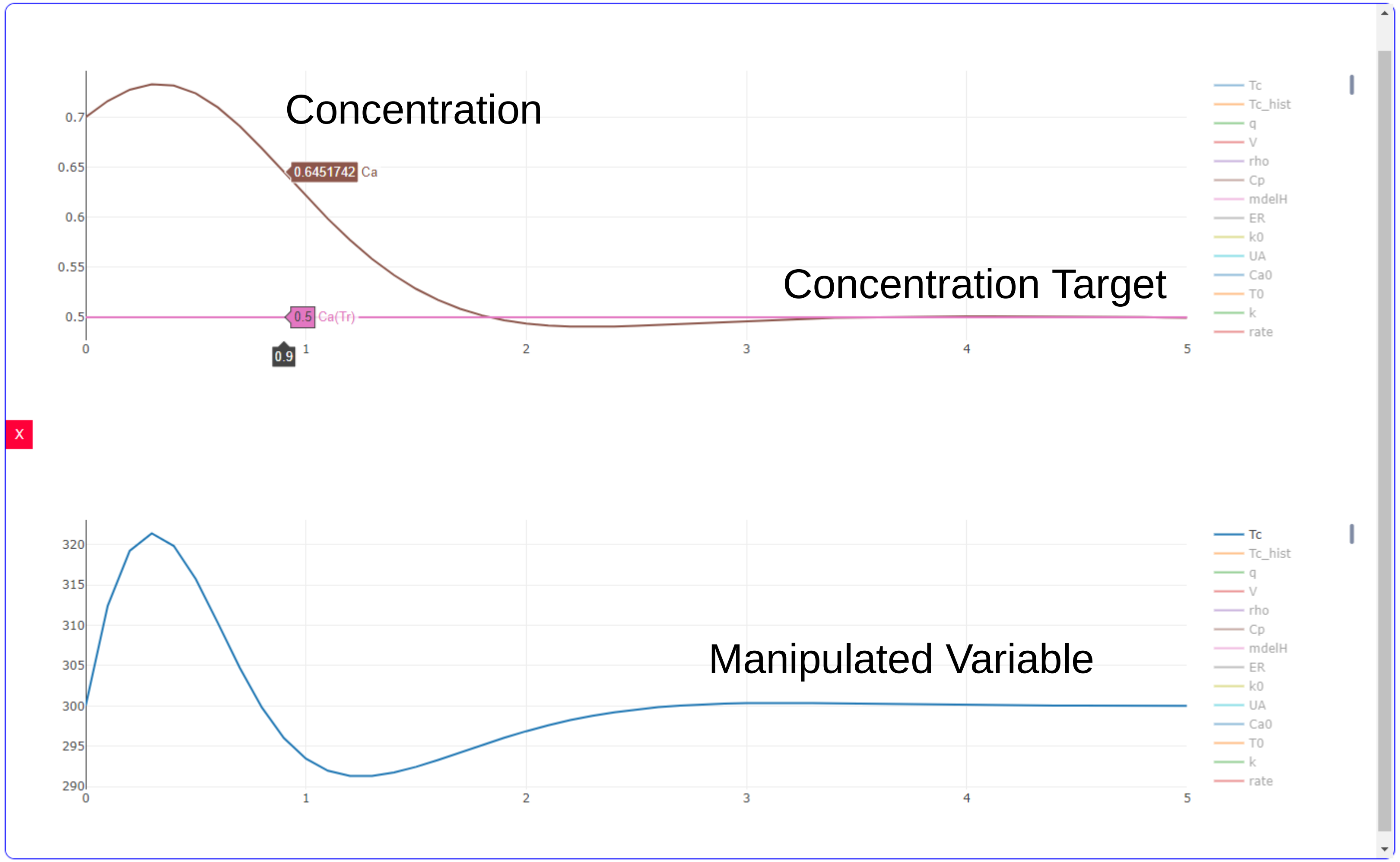
Why GUI

- Rapid viewing
- Online watching
  - Client/server structure

MPC GUI Sample

Model | Variables

Property	Value
APPINFO	0
APPINFOCHG	0
APPSTATUS	1
AUTO_COLD	0
BAD_CYCLES	0
BNDS_CHK	1
COLDSTART	0
CSV_READ	2
CSV_WRITE	1
CTRLMODE	3
CTRL_HOR	50
CTRL_TIME	0.1
CTRL_UNITS	1
CV_TYPE	2
CV_WGT_SLOPE	0
CV_WGT_START	0
CYCLECOUNT	1
DBS_LEVEL	1
DBS_READ	1
DBS_WRITE	1
DIAGLEVEL	0
EV_TYPE	1
EV_WGT_SLOPE	0
FILTER	1



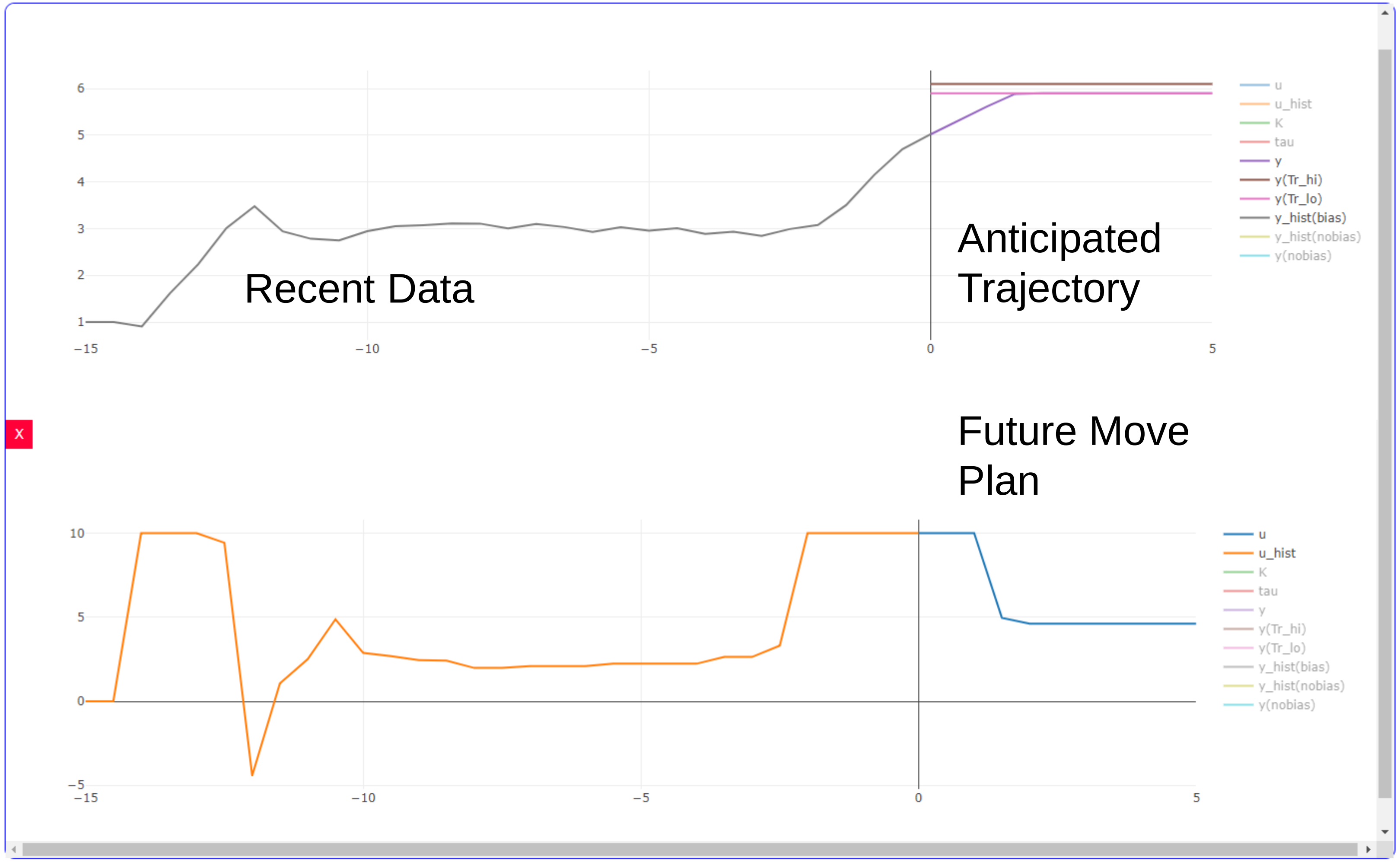
Add Plot | Fullscreen plot

Status: Good

View History and Future Move Plan

Model Variables

Property	Value
APPINFO	0
APPINFOCHG	0
APPSTATUS	1
AUTO_COLD	0
BAD_CYCLES	0
BNDS_CHK	1
COLDSTART	0
CSV_READ	2
CSV_WRITE	1
CTRLMODE	3
CTRL_HOR	10
CTRL_TIME	0.5
CTRL_UNITS	1
CV_TYPE	1
CV_WGT_SLOPE	0
CV_WGT_START	0
CYCLECOUNT	1
DBS_LEVEL	1
DBS_READ	1
DBS_WRITE	1
DIAGLEVEL	0
EV_TYPE	1
EV_WGT_SLOPE	0
FILTER	1



Add Plot Fullscreen plot

Status: Good

Extras

Connections

- Allow flexibility
  - Optimal Control Problems
  - Periodic constraints

Optimal Control Problem

$$\max_{u(t)} \int_0^{10} \left(E - \frac{c}{x}\right) u\, U_{max}\, dt$$

subject to

$$\frac{dx}{dt} = r\, x(t) \left(1 - \frac{x(t)}{k}\right) - u\, U_{max}$$

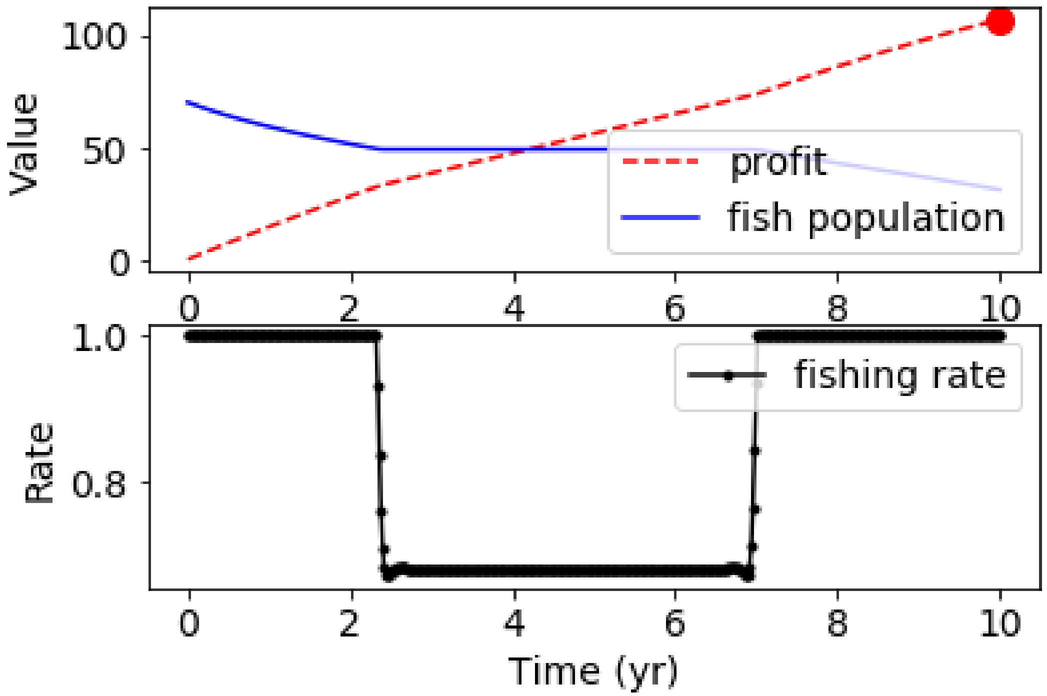
$$x(0) = 70$$

$$0 \leq u(t) \leq 1$$

$$E = 1,\ c = 17.5,\ r = 0.71,\ k = 80.5,\ U_{max} = 20$$

```python
from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt

# create GEKKO model
m = GEKKO()

m.time = np.linspace(0,10,501)

# constants
E = 1
c = 17.5
r = 0.71
k = 80.5
U_max = 20

# fishing rate
u = m.MV(value=1,lb=0,ub=1)
u.STATUS = 1
u.DCOST = 0
```

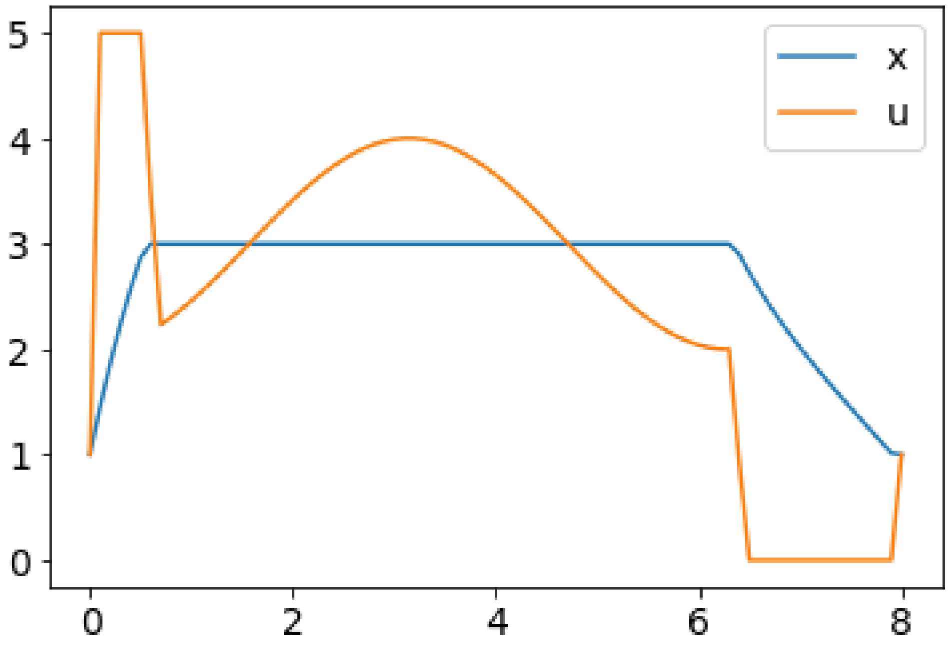
```python
# fish population
x = m.Var(value=70)
# fish population balance
m.Equation(x.dt() == r*x*(1-x/k)-u*U_max)

# objective (profit)
J = m.Var(value=0)
m.Equation(J.dt() == (E-c/x)*u*U_max)
# final objective
Jf = m.FV()
Jf.STATUS = 1
m.Connection(Jf,J,pos2='end')
# maximize profit
m.Obj(-Jf)

# options
m.options.IMODE = 6
m.options.NODES = 3

m.solve()
```



100
50
0
Value
profit
fish population
0
2
4
6
8
10
1.0
0.8
Rate
fishing rate
Time (yr)

Periodic Constraints

```python
#%%Import packages
import numpy as np
from gekko import GEKKO
import matplotlib.pyplot as plt

#%% Build model
m = GEKKO()
m.time = np.linspace(0,8,81)
#Vars
t = m.Var(0)
x = m.Var(1)
u = m.Var(1,0,5)
#periodic constraints
m.periodic(u)
m.periodic(x)
#Equations
m.Equation(t.dt() == 1)
m.Equation(x.dt() == -x + m.cos(t) + u)
#Objective
m.Obj((x-3)**2)
#options
m.options.IMODE = 6
#solve
m.solve()
```

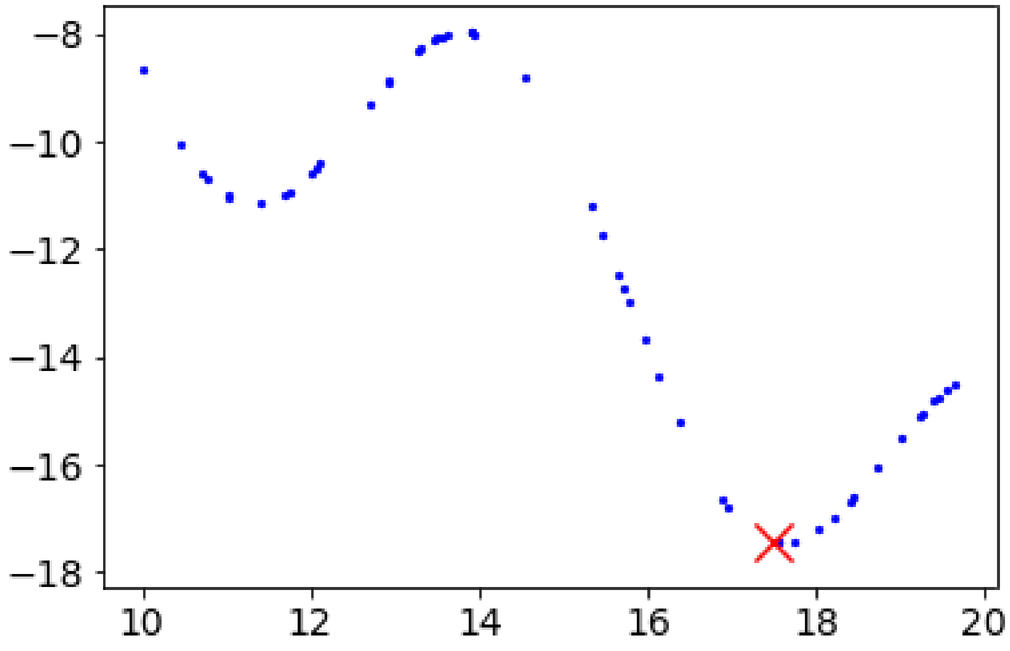


Pre-Built Model Components

- Discrete / Continuous, State Space, Splines
- Other Objects in Development:
  - MPCCs (ABS, MAX, MIN, SIGNUM, PWL)
  - Multi-Component Thermo
  - Object Oriented Flowsheet
    - Pump, Reactor, Mixer, Stream, Vessel
    - Splitter, PID, Flow meter, Flash Column
  - Transfer Function / ARX / FIR
  - Import DMC (MDL), Profit/RMPCT models

Spline Example

```python
from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt

def f(x): # function to generate data for cspline
    return 3*np.sin(x) - (x-3)

x_data = np.random.rand(50)*10+10
y_data = f(x_data)

c = GEKKO()
x = c.Var(value=np.random.rand(1)*10+10)
y = c.Var()
c.cspline(x,y,x_data,y_data,True)
c.Obj(y)
c.solve()

plt.figure()
plt.scatter(x_data,y_data,5,'b')
plt.scatter(x.value,y.value,200,'r','x')
```

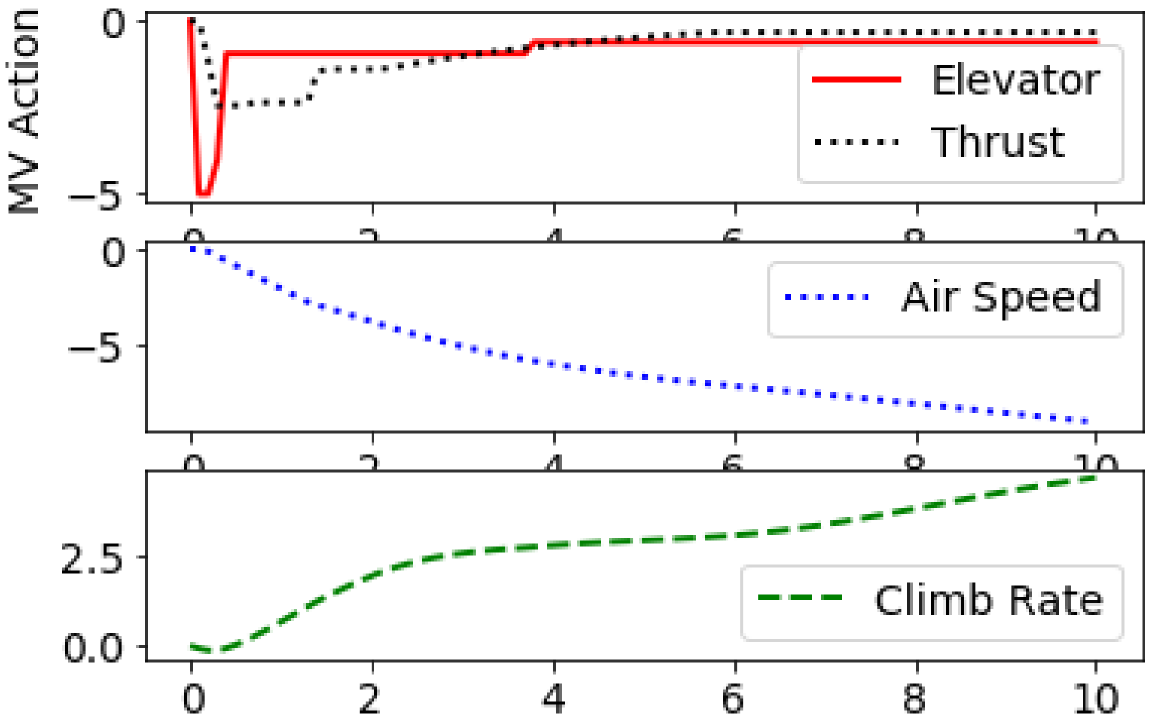


State Space Example

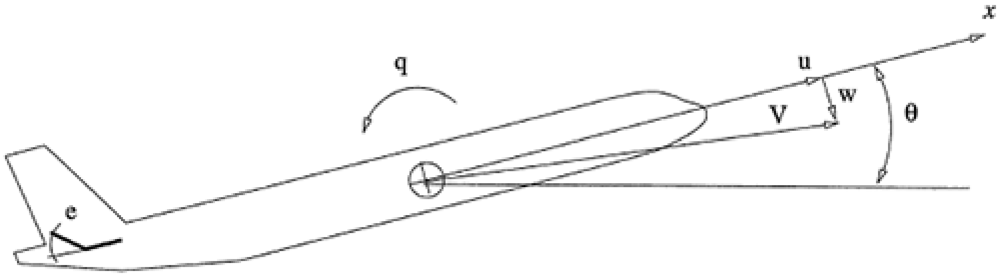
```python
from gekko import GEKKO
import numpy as np

## Linear model of a Boeing 747
A = np.array([[-.003, 0.039, 0, -0.322],
              [-0.065, -0.319, 7.74, 0],
              [0.020, -0.101, -0.429, 0],
              [0, 0, 1, 0]])

B = np.array([[0.01, 1],
              [-0.18, -0.04],
              [-1.16, 0.598],
              [0, 0]])

C = np.array([[1, 0, 0, 0],
              [0, -1, 0, 7.74]])

#%% Build  model
m = GEKKO()

x,y,u = m.state_space(A,B,C)

m.time = [0,0.1,0.2,0.4,1,1.5,2,3,4,5,6,7,8,10,12,15,20]
m.options.IMODE = 6
```

```python
## MV tuning
for i in range(len(u)):
    u[i].lower = -5
    u[i].upper = 5
    u[i].dcost = 1
    u[i].status = 1

## CV tuning
# tau = first order time constant for trajectories
y[0].tau = 3
y[1].tau = 5
# tr_init = 2 (first order traj)
y[0].tr_init = 2
y[1].tr_init = 2
# targets (dead-band needs upper and lower values)
y[0].sphi= -8.5
y[0].splo= -9.5
y[1].sphi= 5.4
y[1].splo= 4.6

y[0].status = 1
y[1].status = 1

m.solve()
```

MPC with State Space Model



$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.003 & 0.039 & 0 & -0.322 \\ -0.065 & -0.319 & 7.74 & 0 \\ 0.020 & -0.101 & -0.429 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u - u_w \\ w - w_w \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.010 & 1 \\ -0.18 & -0.04 \\ -1.16 & 0.598 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ t \end{bmatrix}$$



Additional Resources

- Documentation
  - http://gekko.readthedocs.io
- GitHub Repo
  - https://github.com/BYU-PRISM/GEKKO
- Open Access Paper
  - Beal, L.D.R.; Hill, D.C.; Martin, R.A.; Hedengren, J.D. GEKKO Optimization Suite. *Processes* **2018**, *6*, 106.
- Dynamic Optimization Online Course
  - http://apmonitor.com/do