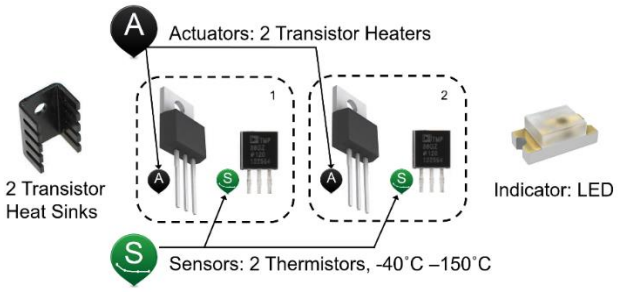
Temperature Control Lab E: Semi-Empirical Moving Horizon Estimation

Design a Moving Horizon Estimator (MHE) for the temperature control lab to estimate the two temperatures and any necessary parameters by combining elements of fundamental and empirical modeling approaches. A recommended approach is shown below but you can create any model that will accurately predict the temperature and can be used in a model predictive controller. This is a final estimation exercise with the TCLab and will be the model that you will use for linear or nonlinear Model Predictive Control.



It is recommended to start with the transient model and parameters identified from Lab C or Lab D as an initial point for the MHE. You can combine elements of the fundamental or empirical models to create a best and final model. Step the heaters to various levels during a 10 minute data collection period that includes rapid as well as slow asynchronous (staggered) steps of the heaters with varying magnitude and direction. Like with the parameter estimation exercise, the estimator may not be able to determine all of the unknown or uncertain parameters in the energy balance because several of the parameters are co-linear. *Report the final sum of absolute errors between the two predicted temperatures and the two measured temperatures. Comment on the parameter variability during the 10 minute collection period.*

Quantity	**Default Starting Values (SI Units) or Use Values from Lab C (Preferred)**
Initial temperature (T_0)	296.15 K
Ambient temperature (T_∞)	296.15 K
Heater Factor (α_1)	0.0100 W/%
Heater Output (Q_1)	0 to 100%
Heater Factor (α_2)	0.0075 W/%
Heater Output (Q_2)	0 to 100%
Heat Capacity (C_p)	500 J/kg-K
Surface Area Not Between Heat Sinks (A)	1×10^{-3} m^2
Surface Area Between Heat Sinks (A_s)	2×10^{-4} m^2
Mass (m)	0.004 kg
Overall Heat Transfer Coefficient (U)	10 W/m^2-K
Emissivity (ε)	0.9
Stefan Boltzmann Constant (σ)	5.67×10^{-8} W/m^2-K^4

Suggested parameters for optimization are α_1, α_2, U, and τ. Other parameters may be able to be updated directly such as using the measured initial temperature (when cool) to update the ambient temperature.

$$mC_p\frac{dT_{h1}}{dt} = UA\,(T_\infty - T_{h1}) + \epsilon\,\sigma\,A\left(T_\infty^4 - T_{h1}^4\right) + UA_s\,(T_{h2} - T_{h1}) + \epsilon\,\sigma\,A_s\left(T_{h2}^4 - T_{h1}^4\right) + \alpha_1 Q_1$$

$$mC_p\frac{dT_{h2}}{dt} = UA\,(T_\infty - T_{h2}) + \epsilon\,\sigma\,A\left(T_\infty^4 - T_{h2}^4\right) + UA_s\,(T_{h1} - T_{h2}) + \epsilon\,\sigma\,A_s\left(T_{h1}^4 - T_{h2}^4\right) + \alpha_2 Q_2$$

The additional empirical equations are to model the temperature delay between the heater and the temperature sensor as:

$$\tau\frac{dT_{c1}}{dt} = -T_{c1} + T_{h1} \qquad \text{and} \qquad \tau\frac{dT_{c2}}{dt} = -T_{c2} + T_{h2}$$

where T_c is the temperature of the sensor and T_h is the temperature of the heater. The additional number 1 or 2 refers to which heater and sensor. T_{c1} is the temperature of sensor 1 and T_{c2} is the temperature of sensor 2. The heater temperatures, T_{h1} and T_{h2}, are predicted from the equations but are not directly measured.

Questions for consideration:

1. How do you decide on a best estimator frequency (update every 1 sec, every 5 sec)?

2. How do you decide on a best estimator horizon (time steps looking backward to include in the objective function)?

3. How well does the estimator hold to consistent parameter values? When do the parameter values seem to change? If they change, do they return to the prior levels?

4. What other tuning values did you change to improve estimator performance? See some possible tuning factors below:

- Application tuning (APM)
  - DIAGLEVEL = diagnostic level (0-10) for solution information
  - EV_TYPE = 1 for l_1-norm and 2 for squared error objective
  - IMODE = 5 or 8 for moving horizon estimation
  - MAX_ITER = maximum iterations
  - MAX_TIME = maximum time before stopping
  - MV_TYPE = Set default MV type with 0=zero-order hold, 1=linear interpolation
  - SOLVER
    - 0=Try all available solvers
    - 1=APOPT (MINLP, Active Set SQP)
    - 2=BPOPT (NLP, Interior Point, SQP)
    - 3=IPOPT (NLP, Interior Point, SQP)
- Fixed Value (FV) - single parameter value over time horizon
  - DMAX = maximum that FV can move each cycle
  - LOWER = lower FV bound
  - FSTATUS = feedback status with 1=measured, 0=off
  - STATUS = turn on (1) or off (0) FV
  - UPPER = upper FV bound
- Manipulated Variable (MV) tuning
  - COST = (+) minimize MV, (-) maximize MV
  - DCOST = penalty for MV movement
  - DMAX = maximum that MV can move each cycle
  - FSTATUS = feedback status with 1=measured, 0=off
  - LOWER = lower MV bound

    - MV_TYPE = MV type with 0=zero-order hold, 1=linear interpolation
    - STATUS = turn on (1) or off (0) MV
    - UPPER = upper MV bound
- Controlled Variable (CV) tuning
    - COST = (+) minimize MV, (-) maximize MV
    - FSTATUS = feedback status with 1=measured, 0=off
    - MEAS_GAP = measurement gap for estimator dead-band

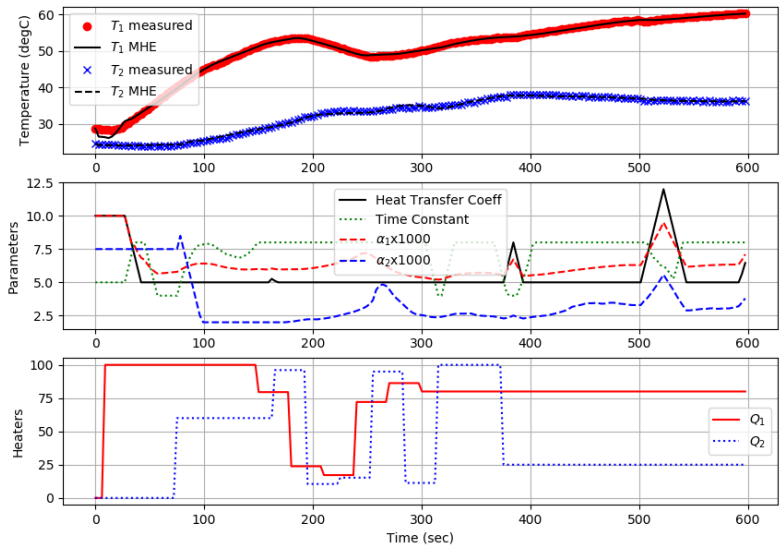
Starting Point in GEKKO (Python)

See Github repository https://github.com/APMonitor/arduino/ (Moving Horizon Estimation directory) for GEKKO MHE source code as a starting point. Install GEKKO (Python) with:

```
pip install gekko    or   pip install gekko --upgrade
```

This script also uses the tclab package that can be pip installed as well.

```
pip install tclab
```




```
import numpy as np
import time
import matplotlib.pyplot as plt
import random
# get gekko package with:
#   pip install gekko
from gekko import GEKKO
# get tclab package with:
#   pip install tclab
from tclab import TCLab

# Connect to Arduino
a = TCLab()

# Final time
tf = 10 # min
# number of data points (1 pt every 3 seconds)
n = tf * 20 + 1
```

Import packages

Connect to Arduino TCLab

Specify Final Time

```
# Configure heater levels
# Percent Heater (0-100%)
Q1s = np.zeros(n)
Q2s = np.zeros(n)
# Heater random steps every 50 sec
# Alternate steps by Q1 and Q2
Q1s[3:]    = 100.0
Q1s[50:]   = 0.0
Q1s[100:]  = 80.0

Q2s[25:]   = 60.0
Q2s[75:]   = 100.0
Q2s[125:]  = 25.0

# rapid, random changes every 5 cycles between 50 and 100
for i in range(50,100):
    if i%10==0:
        Q1s[i:i+10] = random.random() * 100.0
    if (i+5)%10==0:
        Q2s[i:i+10] = random.random() * 100.0

# Record initial temperatures (degC)
T1m = a.T1 * np.ones(n)
T2m = a.T2 * np.ones(n)
# Store MHE values for plots
Tmhe1 = T1m[0] * np.ones(n)
Tmhe2 = T2m[0] * np.ones(n)
Umhe = 10.0 * np.ones(n)
taumhe = 5.0 * np.ones(n)
amhe1 = 0.01 * np.ones(n)
amhe2 = 0.0075 * np.ones(n)

#############################################################
# Initialize Model as Estimator
#############################################################
m = GEKKO(name='tclab-mhe')
# m.server = 'http://127.0.0.1' # if local server is installed

# 60 second time horizon, 20 steps
m.time = np.linspace(0,60,21)

# Parameters to Estimate
U = m.FV(value=10,name='u')
U.STATUS = 0  # don't estimate initially
U.FSTATUS = 0 # no measurements
U.DMAX = 1
U.LOWER = 5
U.UPPER = 15

tau = m.FV(value=5,name='tau')
tau.STATUS = 0  # don't estimate initially
tau.FSTATUS = 0 # no measurements
tau.DMAX = 1
tau.LOWER = 4
tau.UPPER = 8

alpha1 = m.FV(value=0.01,name='a1')   # W / % heater
alpha1.STATUS = 0  # don't estimate initially
alpha1.FSTATUS = 0 # no measurements
alpha1.DMAX = 0.001
alpha1.LOWER = 0.003
alpha1.UPPER = 0.03

alpha2 = m.FV(value=0.0075,name='a2') # W / % heater
alpha2.STATUS = 0  # don't estimate initially
alpha2.FSTATUS = 0 # no measurements
alpha2.DMAX = 0.001
alpha2.LOWER = 0.002
alpha2.UPPER = 0.02
```

Heater Inputs

Storage for Plots

Build MHE with GEKKO

Parameters to Estimate (FVs)

```
# Measured inputs
Q1 = m.MV(value=0,name='q1')
Q1.STATUS = 0  # don't estimate
Q1.FSTATUS = 1 # receive measurement

Q2 = m.MV(value=0,name='q2')
Q2.STATUS = 0  # don't estimate
Q2.FSTATUS = 1 # receive measurement

# State variables
TH1 = m.SV(value=T1m[0],name='th1')
TH2 = m.SV(value=T2m[0],name='th2')

# Measurements for model alignment
TC1 = m.CV(value=T1m[0],name='tc1')
TC1.STATUS = 1     # minimize error between simulation and measurem
TC1.FSTATUS = 1    # receive measurement
TC1.MEAS_GAP = 0.1 # measurement deadband gap
TC1.LOWER = 0
TC1.UPPER = 200

TC2 = m.CV(value=T2m[0],name='tc2')
TC2.STATUS = 1     # minimize error between simulation and measurem
TC2.FSTATUS = 1    # receive measurement
TC2.MEAS_GAP = 0.1 # measurement deadband gap
TC2.LOWER = 0
TC2.UPPER = 200

Ta = m.Param(value=23.0+273.15)     # K
mass = m.Param(value=4.0/1000.0)    # kg
Cp = m.Param(value=0.5*1000.0)      # J/kg-K
A = m.Param(value=10.0/100.0**2)    # Area not between heaters in m
As = m.Param(value=2.0/100.0**2)    # Area between heaters in m^2
eps = m.Param(value=0.9)            # Emissivity
sigma = m.Const(5.67e-8)            # Stefan-Boltzman

# Heater temperatures
T1 = m.Intermediate(TH1+273.15)
T2 = m.Intermediate(TH2+273.15)

# Heat transfer between two heaters
Q_C12 = m.Intermediate(U*As*(T2-T1)) # Convective
Q_R12 = m.Intermediate(eps*sigma*As*(T2**4-T1**4)) # Radiative

# Semi-fundamental correlations (energy balances)
m.Equation(TH1.dt() == (1.0/(mass*Cp))*(U*A*(Ta-T1) \
                    + eps * sigma * A * (Ta**4 - T1**4) \
                    + Q_C12 + Q_R12 \
                    + alpha1*Q1))

m.Equation(TH2.dt() == (1.0/(mass*Cp))*(U*A*(Ta-T2) \
                    + eps * sigma * A * (Ta**4 - T2**4) \
                    - Q_C12 - Q_R12 \
                    + alpha2*Q2))

# Empirical correlations (lag equations to emulate conduction)
m.Equation(tau * TC1.dt() == -TC1 + TH1)
m.Equation(tau * TC2.dt() == -TC2 + TH2)

# Global Options
m.options.IMODE   = 5 # MHE
m.options.EV_TYPE = 2 # Objective type
m.options.NODES   = 3 # Collocation nodes
m.options.SOLVER  = 3 # IPOPT
m.options.COLDSTART = 1 # COLDSTART on first cycle
```

Parameters to Measure (MVs)

State Variables (SVs)

Controlled Variables (CVs) For Alignment Between Measured and Model Values

Parameters (Fixed)

Intermediates
See Link

Equations

Energy Balance (2)

Empirical (2)

MHE Options

```
#################################################################
# Create plot
plt.figure(figsize=(10,7))
plt.ion()
plt.show()

# Main Loop
start_time = time.time()
prev_time = start_time
tm = np.zeros(n)

try:
    for i in range(1,n):
        # Sleep time
        sleep_max = 3.0
        sleep = sleep_max - (time.time() - prev_time)
        if sleep>=0.01:
            time.sleep(sleep-0.01)
        else:
            time.sleep(0.01)

        # Record time and change in time
        t = time.time()
        dt = t - prev_time
        prev_time = t
        tm[i] = t - start_time

        # Read temperatures in Celsius
        T1m[i] = a.T1
        T2m[i] = a.T2

        # Insert measurements
        TC1.MEAS = T1m[i]
        TC2.MEAS = T2m[i]
        Q1.MEAS = Q1s[i-1]
        Q2.MEAS = Q2s[i-1]

        # Start estimating U after 10 cycles (20 sec)
        if i==10:
            U.STATUS = 1
            tau.STATUS = 1
            alpha1.STATUS = 1
            alpha2.STATUS = 1

        # Predict Parameters and Temperatures with MHE
        # use remote=False for local solve
        m.solve(remote=True)

        if m.options.APPSTATUS == 1:
            # Retrieve new values
            Tmhe1[i] = TC1.MODEL
            Tmhe2[i] = TC2.MODEL
            Umhe[i]  = U.NEWVAL
            taumhe[i] = tau.NEWVAL
            amhe1[i] = alpha1.NEWVAL
            amhe2[i] = alpha2.NEWVAL
        else:
            # Solution failed, copy prior solution
            Tmhe1[i] = Tmhe1[i-1]
            Tmhe2[i] = Tmhe1[i-1]
            Umhe[i]  = Umhe[i-1]
            taumhe[i] = taumhe[i-1]
            amhe1[i] = amhe1[i-1]
            amhe2[i] = amhe2[i-1]

        # Write new heater values (0-100)
        a.Q1(Q1s[i])
        a.Q2(Q2s[i])
```

Create Figure

Timing to Enforce 3 Sec Intervals

Record Temperatures and Heater Values

Turn on Parameter Estimation after 10 Cycles (30 sec)

Solve MHE

Check if Successful and Retrieve Solution

Insert New Heater Values

```
            # Plot
            plt.clf()
            ax=plt.subplot(3,1,1)
            ax.grid()
            plt.plot(tm[0:i],T1m[0:i],'ro',label=r'$T_1$ measured')
            plt.plot(tm[0:i],Tmhe1[0:i],'k-',label=r'$T_1$ MHE')
            plt.plot(tm[0:i],T2m[0:i],'bx',label=r'$T_2$ measured')
            plt.plot(tm[0:i],Tmhe2[0:i],'k--',label=r'$T_2$ MHE')
            plt.ylabel('Temperature (degC)')
            plt.legend(loc=2)
            ax=plt.subplot(3,1,2)
            ax.grid()
            plt.plot(tm[0:i],Umhe[0:i],'k-',label='Heat Transfer Coeff
            plt.plot(tm[0:i],taumhe[0:i],'g:',label='Time Constant')
            plt.plot(tm[0:i],amhe1[0:i]*1000,'r--',label=r'$\alpha_1$x
            plt.plot(tm[0:i],amhe2[0:i]*1000,'b--',label=r'$\alpha_2$x
            plt.ylabel('Parameters')
            plt.legend(loc='best')
            ax=plt.subplot(3,1,3)
            ax.grid()
            plt.plot(tm[0:i],Q1s[0:i],'r-',label=r'$Q_1$')
            plt.plot(tm[0:i],Q2s[0:i],'b:',label=r'$Q_2$')
            plt.ylabel('Heaters')
            plt.xlabel('Time (sec)')
            plt.legend(loc='best')
            plt.draw()
            plt.pause(0.05)

    # Turn off heaters
    a.Q1(0)
    a.Q2(0)
    # Save figure
    plt.savefig('tclab_mhe.png')

# Allow user to end loop with Ctrl-C
except KeyboardInterrupt:
    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Shutting down')
    a.close()
    plt.savefig('tclab_mhe.png')

# Make sure serial connection still closes when there's an error
except:
    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Error: Shutting down')
    a.close()
    plt.savefig('tclab_mhe.png')
    raise
```

Update Figure

Turn off Heaters
Save Plot (PNG)
Catch Ctrl-C or Other Errors