Harrison Durbin
SID 26951511

HW 2: Model Learning for Smart Home Thermal Management

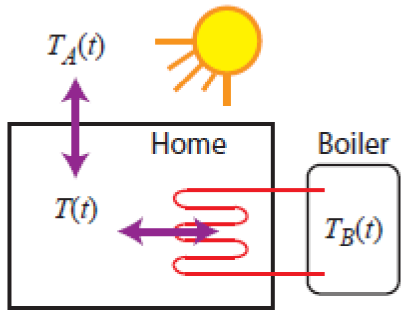
Problem 1: Reading

Each energy storage technology has its respective advantages and disadvantages as outlined in the table below:

Energy Storage Tech	**Pros**	**Cons**
Pumped Hydro Storage (PHS)	• Have a simple design that is the most prominent energy storage tech to date. • Have very low cost maintenance costs. • Able to quickly reach full load in one minute from standstill.	• Only suitable for mountainous geographic regions (e.g. the Alps in Switzerland) that limit unit siting. • Have high capital costs. • Have criticism due to impact on ecosystems. • Requires a lot of space.
Compressed Air Energy Storage (CAES)	• Shared technology with standard combustion engines turbines. • More than 80% of the US has suitable geologic formations for this type. • Have a relatively low cost.	• Must have certain geologic formations for air to be stored (e.g. salt caverns from mining, depleted oil, porous rock, etc). • Some emissions are produced from combustion, causing environmental concern. • Have low energy and power densities.
Flywheels	• There is no negative environmental impacts because the materials are benign and compact. • Are able to achieve high efficiencies for short term storage. • Have relatively simple designs.	• Magnetic bearings are complex systems that require care for operation and maintenance. • Have a high occurrence of self-discharge caused by frictional losses. • Relatively high initial costs. • Poor energy capacity (although excellent power capacity).
Electro-chemical Capacitors	• One of the biggest advantages is the ability to charge and discharge more quickly than batteries. • The lifetime of these devices is no impacted by cycling because no chemical reactions take place as in batteries, which results in a longer service life. • Have excellent power density.	• Have potential for damage if placed with a higher-than-rated voltage. • To achieve higher voltage, hundreds of these cells must be connected in series and if one cell fails, the entire system may fail, causing an issue of reliability. • Have low energy density.

Super-conducting Magnetic Energy Storage (SMES)	• Are extremely efficient. • Have fast response. • Can be scaled to large sizes while having a small footprint. • Superb power density. • Environmentally benign.	• Are very costly, requiring cryogenic cooling to maintain superconductive materials • Have parasitic losses from cooling. • Have poor energy density. • There is uncertainty of the potential effects of non-ionizing radiation.
Lead Acid Batteries	• Have a low cost while being relatively high efficiency. • Have moderate energy capacity. • In larger systems can be connected in series/parallel combinations on racks.	• Contain large quantities of toxic chemicals. • Have a low-cycle lifetime due to chemical reactions. • Are not suitable for extreme temperatures. • May have thermal runaway. • Have low specific energy.
Lithium-ion Batteries	• Have extremely high-efficiency compared to other batteries. • Have high energy density, power density, and cell voltage compare to other battery systems. • Are less chemically reactive resulting in a longer service life.	• Require a very high capital cost. • Required careful management. • Lithium may cause fire when exposed to moisture. • The electrolyte may be toxic.
Flow Batteries	• Have independent power and energy capacity, meaning a single system can be designed to meet many specifications. • Are durable and have a long cycle life.	• Have specific energy/power. • Require plumbing and pumping system.

Problem 2: Parametric Modeling



2a) Reformulate into the linear-in-the-parameters form $z(t) = \theta^T\varphi(t)$

The parameters are C, R1, and R2.
The signals that play a role in $\varphi(t)$ are T, T_A, and T_B. Likewise, these signals play a role in z(t) as z(t) is a function of $\varphi(t)$.

$$C * \frac{d}{dt} * T(t) = \frac{1}{R1} * [T_A - T] + \frac{1}{R2} * [T_B - T]$$

$$\frac{d}{dt} * T(t) = \frac{1}{C * R1} * [T_A - T] + \frac{1}{C * R2} * [T_B - T]$$

$$z(t) = \frac{1}{C * R1} * [T_A - T] + \frac{1}{C * R2} * [T_B - T]$$

$$z(t) = \theta^T \varphi$$

With a 2D parameter vector,

$$\theta_{2d} = \begin{bmatrix} \frac{1}{C * R1} \\ \frac{1}{C * R2} \end{bmatrix}$$

$$\varphi_{2d} = \begin{bmatrix} T_A - T \\ T_B - T \end{bmatrix}$$

$$z(t) = \begin{bmatrix} \frac{1}{C * R1} & \frac{1}{C * R2} \end{bmatrix} \begin{bmatrix} T_A - T \\ T_B - T \end{bmatrix}$$

With a 3D parameter vector,

$$\theta_{3d} = \begin{bmatrix} \frac{1}{C*R1} \\ \frac{1}{C*R2} \\ -\frac{1}{C*R1} - \frac{1}{C*R2} \end{bmatrix}$$

$$\varphi_{3d} = \begin{bmatrix} T_A \\ T_B \\ T \end{bmatrix}$$

$$z(t) = \begin{bmatrix} \frac{1}{C*R1} & \frac{1}{C*R2} & -\frac{1}{C*R1} - \frac{1}{C*R2} \end{bmatrix} \begin{bmatrix} T_A \\ T_B \\ T \end{bmatrix}$$

Note: Parameter vector θ can be two-dimensional or three-dimensional.
This is because the equation can be re-written equivalently in multiple forms and one has the option to write it in the optimal form for system identification.

If $\theta \in R^3$, then is it possible to recover the individual three original parameters C, R1, R2? What if $\theta \in R^2$?

If $\theta \in R^3$, it is possible to recover the three original parameters C, R1, and R2 individually, because we will be able to create three equations, having three unknowns, which is solvable. See equations below for more detail:

$\frac{1}{C*R1} = \theta_1$ …where θ_1 is known.

$\frac{1}{C*R2} = \theta_2$ …where θ_2 is known.

$-\frac{1}{C*R1} - \frac{1}{C*R2} = \theta_3$ ...where θ_3 is known.

On the other hand, if $\theta \in R^2$, it is not possible to recover the three original parameters C, R1, and R2 individually, because we will be able to create only two equations having three unknowns, which is unsolvable.

2b) Persistence of Excitation (PE)

The _2D version of the parametric model is easier to identify than the 3D version_, as determined by calculating the PE using the equation:

$$\lambda_{\min} \left\{ \int_0^t \phi(\tau)\phi^T(\tau)d\tau \right\} > 0$$

The minimum eigenvalue was larger using the 2d version of φ, which means the estimate of θ (denoted $\hat{\theta}$), converges faster to true value of θ. The min eigenvalue was 0.0221 for the 2D version and 0.0081 for the 3D version.

Code input and output is shown below:

```
%%%% OPTION with 2-D parameter vector
phi = [T_A-T,T_B-T]'; % signals for 2D parametric model
t_end = t(end);
PE_mat = zeros(2);

phi_sq = zeros(2,2,length(t));
for k = 1:length(t)
    phi_sq(:,:,k) = phi(:,k) * phi(:,k)';
end
PE_mat(1,1) = 1/t_end * trapz(t, phi_sq(1,1,:));
PE_mat(2,1) = 1/t_end * trapz(t, phi_sq(2,1,:));
PE_mat(1,2) = 1/t_end * trapz(t, phi_sq(1,2,:));
PE_mat(2,2) = 1/t_end * trapz(t, phi_sq(2,2,:));

PE_lam_min = min(eig(PE_mat)); %  MINIMUM EIGENVALUE OF PE_mat
fprintf(1,'PE Level for 2D Version : %1.4f\n',PE_lam_min);

%%%% OPTION with 3-D parameter vector
phi = [T_A,T_B,T]'; % signals for 3D parametric model
t_end = t(end);
PE_mat = zeros(3);

phi_sq = zeros(3,3,length(t));
for k = 1:length(t)
    phi_sq(:,:,k) = phi(:,k) * phi(:,k)';
end
PE_mat(1,1) = 1/t_end * trapz(t, phi_sq(1,1,:));
PE_mat(2,1) = 1/t_end * trapz(t, phi_sq(2,1,:));
PE_mat(3,1) = 1/t_end * trapz(t, phi_sq(3,1,:));
PE_mat(1,2) = 1/t_end * trapz(t, phi_sq(1,2,:));
PE_mat(2,2) = 1/t_end * trapz(t, phi_sq(2,2,:));
PE_mat(3,2) = 1/t_end * trapz(t, phi_sq(3,2,:));
PE_mat(1,3) = 1/t_end * trapz(t, phi_sq(1,3,:));
PE_mat(2,3) = 1/t_end * trapz(t, phi_sq(2,3,:));
PE_mat(3,3) = 1/t_end * trapz(t, phi_sq(3,3,:));

PE_lam_min = min(eig(PE_mat)); % MINIMUM EIGENVALUE OF PE_mat
fprintf(1,'PE Level for 3D Version : %1.4f\n',PE_lam_min);

output:
PE Level for 2D Version : 0.0221
PE Level for 3D Version : 0.0081
```

Problem 3: Gradient Algorithm

The normalized recursive gradient update law, given by the following equations, was used to develop a function in Matlab:

$$\begin{aligned}\frac{d}{dt}\hat{\theta}(t) &= \Gamma\epsilon(t)\phi(t), \qquad \hat{\theta}(0) = \hat{\theta}_0, \\ \epsilon(t) &= \frac{z(t) - \hat{\theta}^T\phi(t)}{m^2(t)}, \\ m^2(t) &= 1 + \phi^T(t)\phi(t)\end{aligned}$$

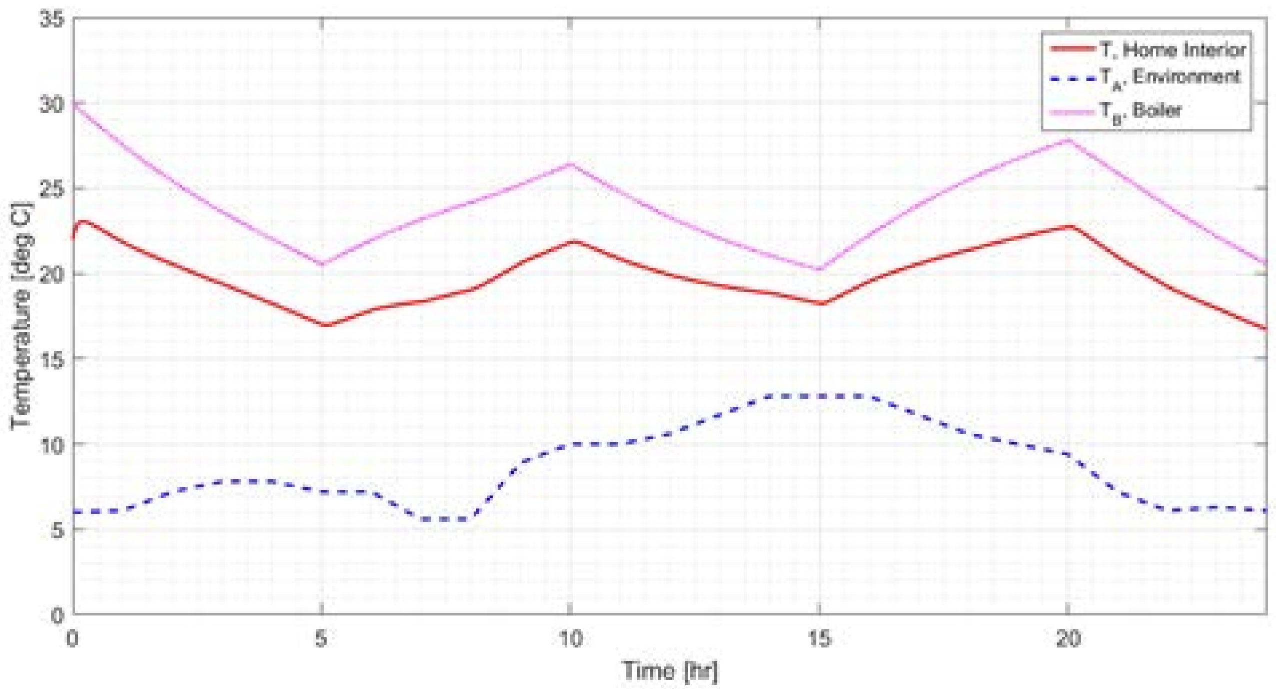
ode_gradient.m

```
% ODEs for the gradient parameter identification algorithm
% t           : time
% theta_h     : parameter estimate
% data        : input-output data used to feed algorithm
% Gam         : Update law gain

function theta_h_dot = ode_gradient(t,theta_h,data,Gam)

%% Parse Input Data
it = data(:,1);      % Time vector
iT = data(:,2);      % Indoor temp. vector
iT_A = data(:,3);    % Ambient temp. vector
iT_B = data(:,4);    % Boiler temp. vector

%% Interpolate data
T = interp1(it,iT,t);
T_A = interp1(it,iT_A,t);
T_B = interp1(it,iT_B,t);

%% Parametric model notation
% Samping time step
dt = 1;

% Compute Room temperature at NEXT time step
T_plus = interp1(it,iT,t+dt);

% Compute \dot{T} using forward difference in time
% z = \dot{T} = (T(t+dt) - T(t))/dt
z = (T_plus-T)/dt ;

% Assemble regressor vector, \phi
phi = [T_A-T,T_B-T]' ;

%% Gradient Update Law
% Normalization signal
msq = 1+(phi'*phi) ;

% Estimation error: \epsilon = (z - \theta_h^T \phi /) msq
epsilon = (z-theta_h'*phi)/msq ;

% Update Law
theta_h_dot = Gam*phi*epsilon ; %Gam*epsilon*phi ;
```

Problem 4: Implementation

4a) Plot

The following plot shows an exploratory data analysis of T(t); TA(t); TB(t) versus 24 hours of time.



4b) ODE

The ODE was initialized with values of $\hat{\theta}_0 = [0.1,0,1]^T$.

The value of Γ was iteratively increased on a logarithmic scale to compare various Γ values.

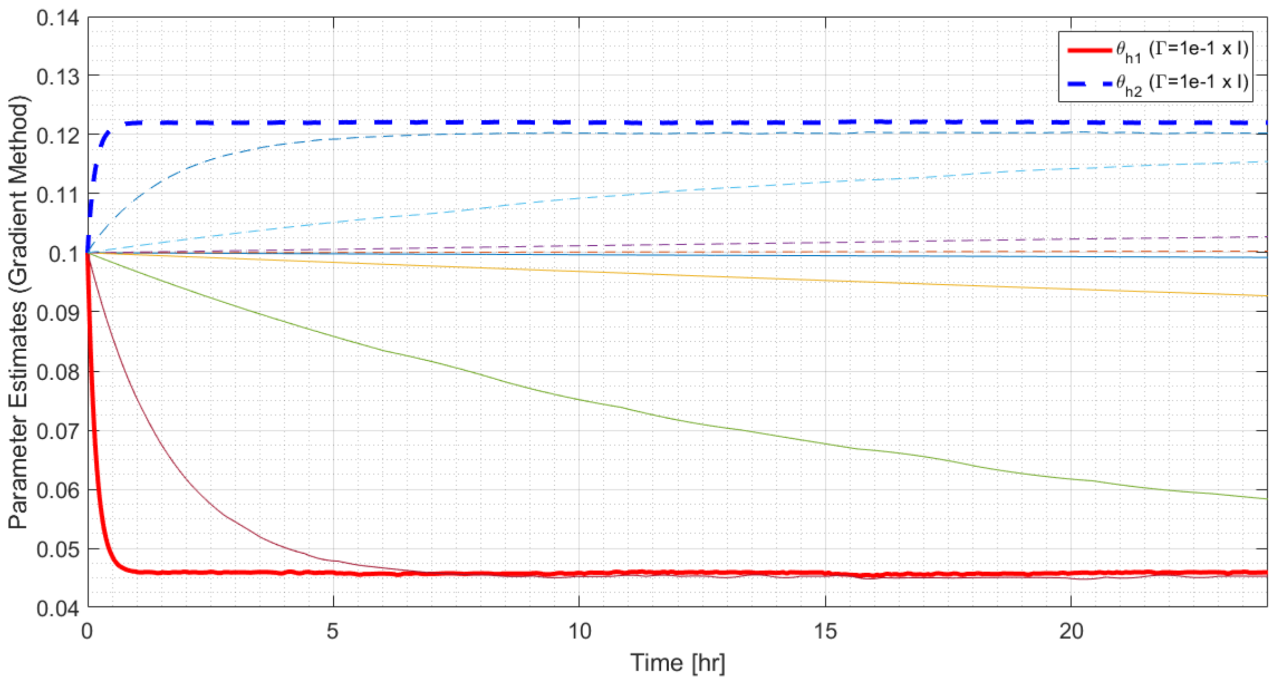
Completed code is provided below:

```
% Assemble Data
data = [t, T, T_A, T_B];

% Initial conditions
theta_hat0 = [0.1,0.1]';

% Update Law Gain
Gam_a = 1e-5*eye(2) ; % increase from super small
Gam_b = 1e-4*eye(2) ;
Gam_c = 1e-3*eye(2) ;
Gam_d = 1e-2*eye(2) ;
Gam_e = 1e-1*eye(2) ;

% Integrate ODEs
[~,y_a] = ode23s(@(t,y_a) ode_gradient(t,y_a,data,Gam_a), t,
theta_hat0); % output y, which is theta_hat
[~,y_b] = ode23s(@(t,y_b) ode_gradient(t,y_b,data,Gam_b), t,
theta_hat0);
[~,y_c] = ode23s(@(t,y_c) ode_gradient(t,y_c,data,Gam_c), t,
theta_hat0);
```

```
[~,y_d] = ode23s(@(t,y_d) ode_gradient(t,y_d,data,Gam_d), t,
theta_hat0);
[~,y_e] = ode23s(@(t,y_e) ode_gradient(t,y_e,data,Gam_e), t,
theta_hat0);

% Parse output
theta_hat_a = y_a;
theta_hat_b = y_b;
theta_hat_c = y_c;
theta_hat_d = y_d;
theta_hat_e = y_e;
```

4c) Estimated Parameters

The following plot shows the parameter estimates, $\hat{\theta}_1$ and $\hat{\theta}_2$, versus time. The two bold lines represent $\hat{\theta}_1$ and $\hat{\theta}_2$ for the final selected Γ value of 1e-1 x I (the 2x2 identity matrix). For comparison plots are shown when using other Γ values. For extremely low values of Γ (1e-5 x I), $\hat{\theta}_1$ and $\hat{\theta}_2$ remain close to the initial conditions, $\hat{\theta}_0 = [0.1,0,1]^T$. However, as Γ increases, $\hat{\theta}_1$ and $\hat{\theta}_2$ diverge more quickly from the initial conditions toward actual values.



The final values are:

- theta_hat1 (Gradient Method): 0.0459
- theta_hat2 (Gradient Method): 0.1220

Assuming C=10, R_1=2, and R_2= 0.75, the true values are:

- theta_true1 : 0.0500
- theta_true2 : 0.1333

Estimates are reasonably close to the true values, with 8% error for both $\hat{\theta}_1$ and $\hat{\theta}_2$.

Harrison Durbin
SID 26951511

Problem 5: Model Validation

5a) State-Space Object

What are the A and B matrices for the LTI system?

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$\dot{T}(t) = \left[-\frac{1}{C*R1} - \frac{1}{C*R2}\right] * T(t) + \left[\frac{1}{C*R1} \quad \frac{-1}{C*R2}\right] * \begin{bmatrix} T_A \\ T_B \end{bmatrix}$$

$\mathrm{A} = \left[-\frac{1}{C*R1} - \frac{1}{C*R2}\right]$ …(note: this is a 1x1 matrix)

$\mathrm{B} = \left[\left[\frac{1}{C*R1} \quad \frac{-1}{C*R2}\right]\right]$ …(note: this is a 1x2 matrix)

$\widehat{\mathrm{A}} = \left[-\hat{\theta}_1 - \hat{\theta}_2\right]$ …(note: this is a 1x1 matrix)

$\widehat{\mathrm{B}} = [\hat{\theta}_1 \quad -\hat{\theta}_2]$ …(note: this is a 1x2 matrix)

What is u(t)?

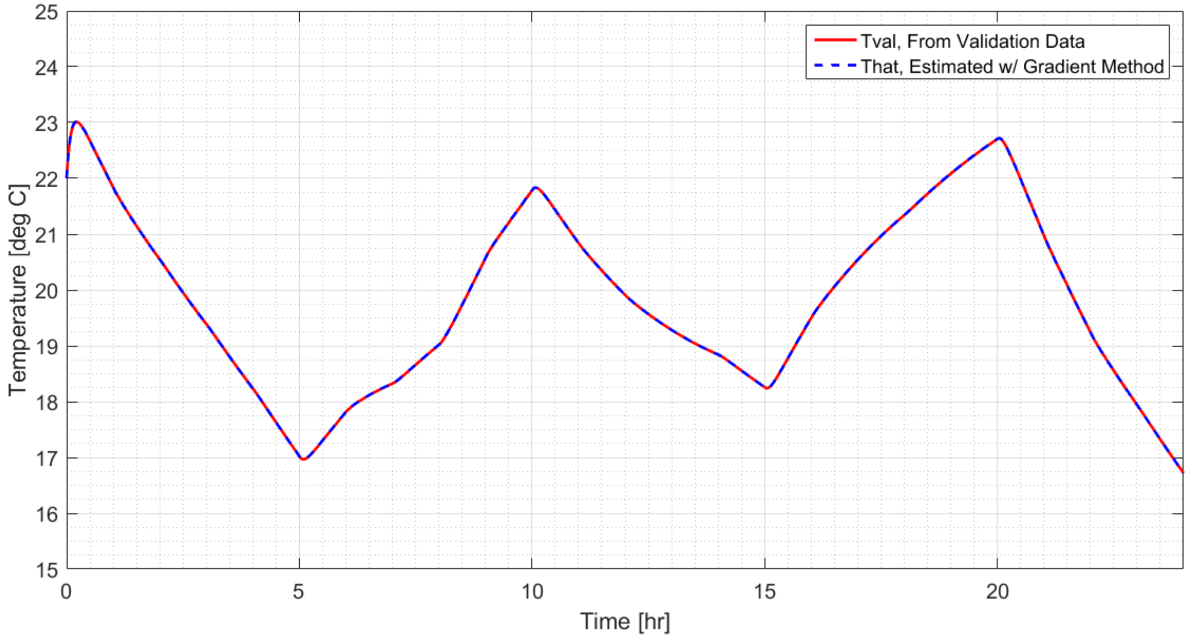
$$u(t) = \begin{bmatrix} T_A \\ T_B \end{bmatrix}$$

See code below.

```
%% Problem 5(a)
% System matrices for identified model
Ahat =  [-theta_hat(end,1)-theta_hat(end,2)]; % must be nxn matrix->1x1
Bhat =  [theta_hat(end,1),theta_hat(end,2)]; % 1x2 matrix

% Output states only (dummy variables, not used later)
C_dummy = 1; % 1x1 matrix
D_dummy = [[0],[0]]; % 2x1 matrix

% State space model
sys_hat = ss(Ahat,Bhat,C_dummy,D_dummy);
```

5b) Simulation

The following flow shows the indoor temperature T(t) predicted by the identified model, and the true indoor temperature T(t) from the validation data set, HW2_ValData.csv.



How do they compare?
The estimated temperature (That), appears to perfectly match the actual temperature in the validation data set (Tval); however, on close inspection when zooming into the plot, there is some error to a hundredth of a degree.

Problem 6: Least Squares Algorithm with Forgetting Factor

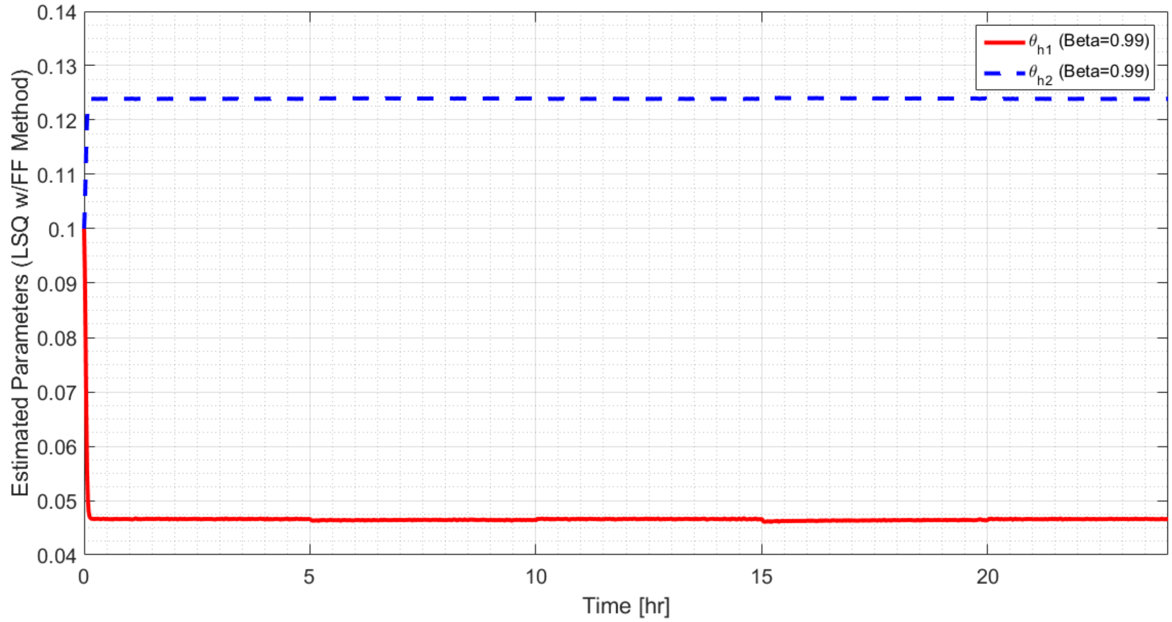
$$\begin{aligned}
\frac{d}{dt}\hat{\theta}(t) &= P(t)\epsilon(t)\phi(t), \qquad \hat{\theta}(0) = \hat{\theta}_0, \\
\frac{d}{dt}P(t) &= \beta P(t) - P(t)\frac{\phi(t)\phi^T(t)}{m^2(t)}P(t), \qquad P(0) = P_0 = Q_0^{-1}, \\
\epsilon(t) &= \frac{z(t) - \hat{\theta}^T\phi(t)}{m^2(t)}, \qquad m^2(t) = 1 + \phi^T(t)\phi(t)
\end{aligned}$$

The equations above were used to create a least squares function, shown in the following code.

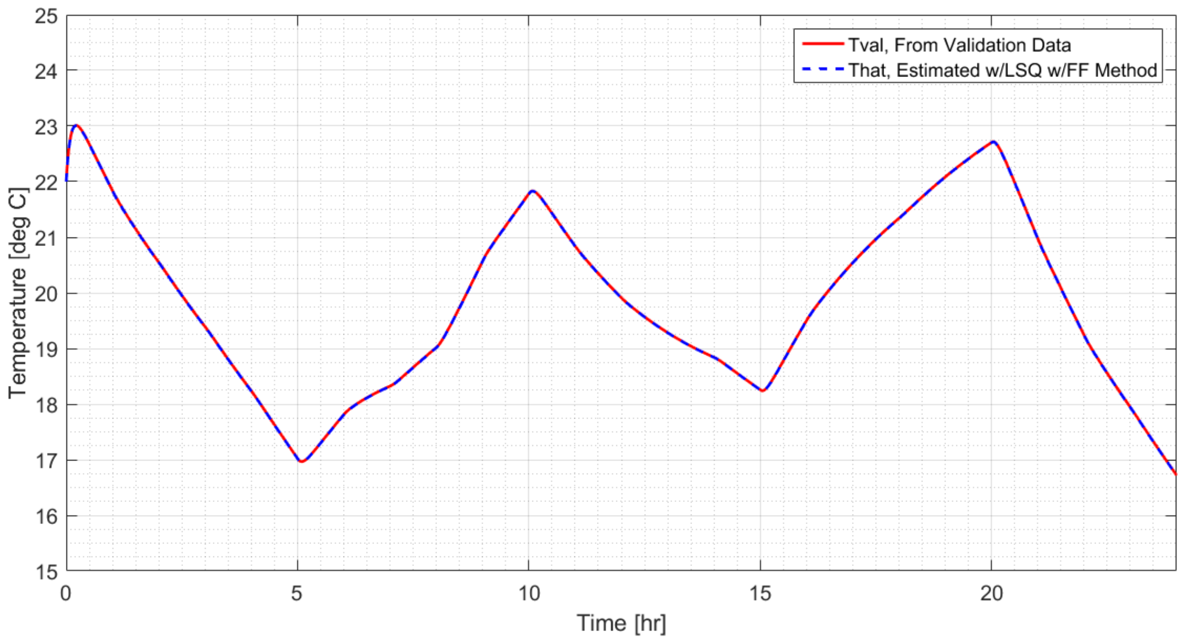
ode_lsq.m

```
% ODEs for the gradient parameter identification algorithm
% t          : time
% y_conc     : the concatenate of theta_h and P
% theta_h    : parameter estimate
% P          : LSQ factor
% data       : input-output data used to feed algorithm
% beta       : forgetting factor

function theta_h_dot__P_dot = ode_lsq(t,y_conc,data,beta)

%% Parse Input Data
it = data(:,1);     % Time vector
it_end=it(end);     % End of time vector
iT = data(:,2);     % Indoor temp. vector
iT_A = data(:,3);   % Ambient temp. vector
iT_B = data(:,4);   % Boiler temp. vector

%% Interpolate data
T = interp1(it,iT,t);
T_A = interp1(it,iT_A,t);
T_B = interp1(it,iT_B,t);

%% Parametric model notation
% Samping time step
dt = 1;
% Compute Room temperature at NEXT time step
T_plus = interp1(it,iT,t+dt);

% Compute \dot{T} using forward difference in time
% z = \dot{T} = (T(t+dt) - T(t))/dt
z = (T_plus-T)/dt ;

% Assemble regressor vector, \phi
phi = [T_A-T,T_B-T]' ;

%% Least Squares Algorithm (LSQ) w/ forgetting factor
theta_h = y_conc(1:2,1);
P=reshape(y_conc(3:6),2,2);

% Normalization signal
msq = 1+(phi'*phi) ;

% Estimation error: \epsilon = z - \theta_h^T \phi
epsilon = (z-(theta_h'*phi))/msq  ;

% Update Law
theta_h_dot = P*epsilon*phi;
P_dot = beta*P-P*(phi'*phi)/msq*P;
P_dot_reshaped = reshape(P_dot, 4, 1);
theta_h_dot__P_dot=[theta_h_dot;P_dot_reshaped];
```

The following plot shows the parameter estimates, $\hat{\theta}_1$ and $\hat{\theta}_2$, versus time for the LSQ with forgetting factor algorithm. A forgetting factor of 0.99 was used.



The following plot shows of That(t) predicted from the model (using least squares) and the true T(t).



Similar to the results for the gradient algorithm, the estimated temperature (That), appears to perfectly match the actual temperature in the validation data set (Tval); however, on close inspection when zooming into the plot, there is some error to a hundredth of a degree.