Optimal Estimation Methods

(Lecture 20 – Advanced Topics)

Dr. John L. Crassidis

University at Buffalo – State University of New York
Department of Mechanical & Aerospace Engineering
Amherst, NY 14260-4400
johnc@buffalo.edu
http://www.buffalo.edu/~johnc

University at Buffalo The State University of New York

Multiple-Model Adaptive Estimation

Filter Tuning Issues

- Gaussian assumption for measurement noise
  - Usually a pretty good assumption
  - Can use a colored-noise filter if needed
- Gaussian assumption for process noise
  - Usually used to handle model errors
    - Weights "trust" between measurements and model
  - Even if Gaussian, getting good results requires a lot of tuning
- Adaptive filtering methods for tuning
  - Bayesian and maximum likelihood methods $\rightarrow$ suited for multi-model approach, but require large computational loads
  - Covariance matching $\rightarrow$ match covariance of residuals, but may produce biased results
  - Other approaches
    - Neural nets, fuzzy membership functions, etc.
    - Each has its own advantages/disadvantages

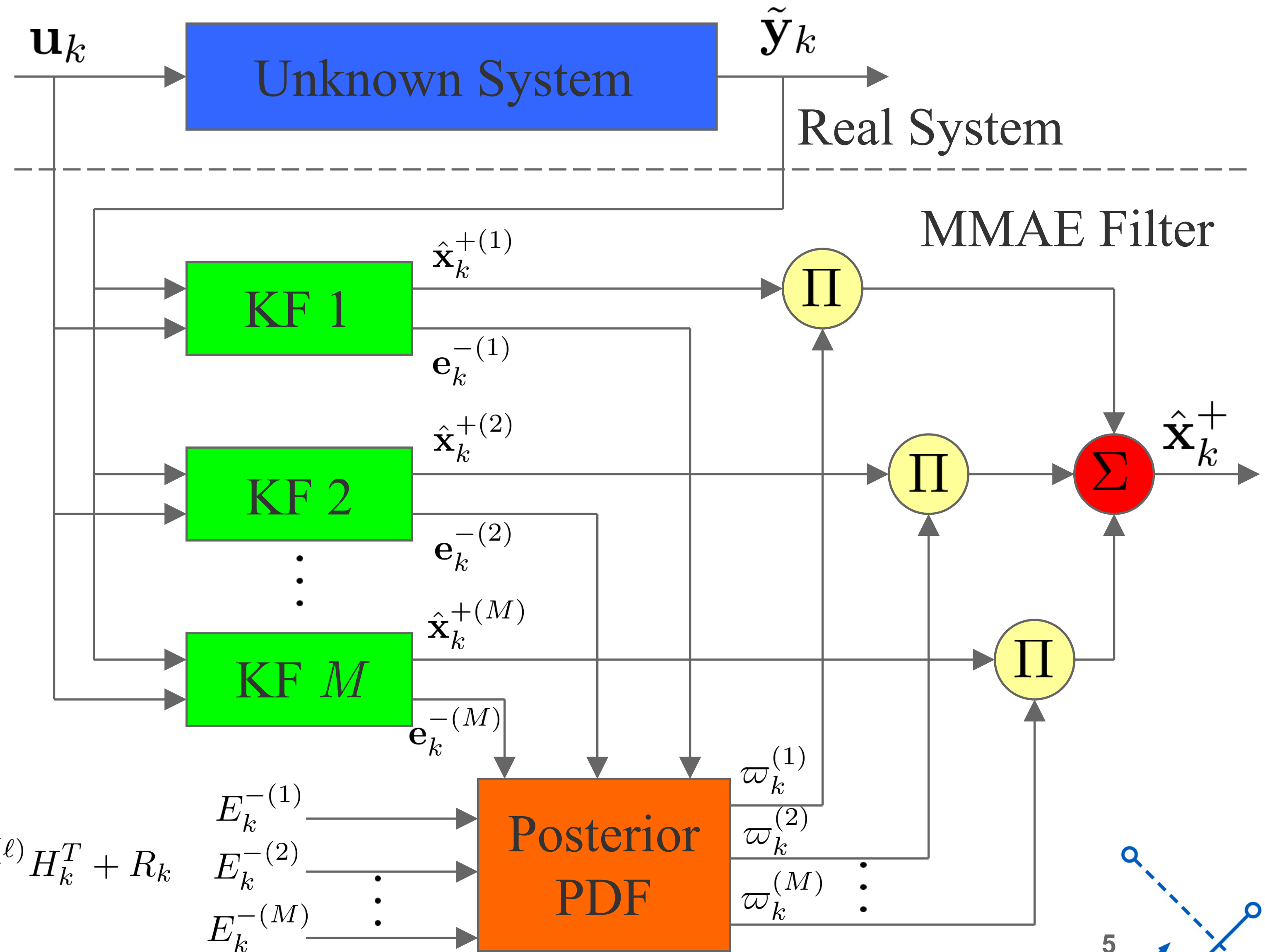
MMAE for Filter Tuning

- Standard Multiple-Model Adaptive Estimation (MMAE) algorithm
  - Uses a bank of parallel filters to provide multiple estimates
    - Each filter uses a different value for the process noise covariance
  - State estimate $\rightarrow$ weighted sum of each filter's estimate
    - Uses likelihood of unknown elements conditioned on current-time measurement-minus-estimate residual to test hypothesis
  - Can work with nonlinear systems
    - Small noise assumption made for output, which is usually a good assumption
  - Uses posterior likelihood function to determine weights
  - Weights are the probabilities that the model is the correct one

MMAE Approach

Weights are the probabilities that the model is the correct one!

Useful to assess which model is most accurate

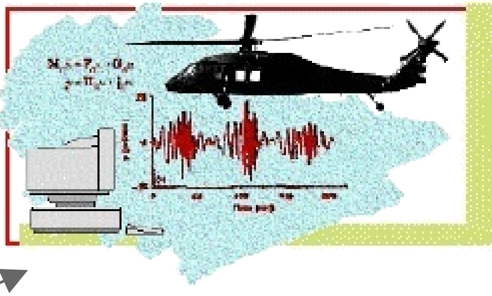


$$E_k^{-(\ell)} \equiv H_k P_k^{-(\ell)} H_k^T + R_k$$

MMAE Applications

- Robust Target Tracking
- Adaptive Signal Processing
- Navigation Applications

- Adaptive Control
- Communication Systems
- Tactical Assessment

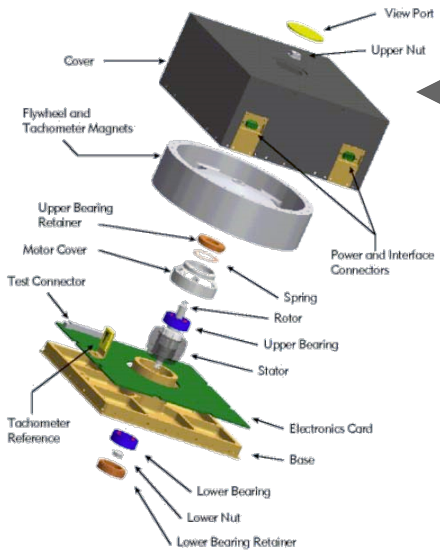


Filter Tuning

Parameter Identification

MMAE

Health Monitoring and Fault Detection



- Sensor and Actuator Degradation Faults
- Structural and Mechanical Health Monitoring
- Reconfigurable (intelligent) Systems
- Higher Level Fusion Applications

Innovation Covariance

- Likelihood requires covariance of the innovation

$$\mathbf{e}_k^- \equiv \tilde{\mathbf{y}}_k - \hat{\mathbf{y}}_k^-$$

  - Substituting measurement and estimate models gives

$$\mathbf{e}_k^- = H_k \mathbf{x}_k + \mathbf{v}_k - H_k \hat{\mathbf{x}}_k^- = H_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^-) + \mathbf{v}_k$$

  - Then we have

$$E\{\mathbf{e}_k^- \, \mathbf{e}_k^{-T}\} = E\left\{\cancelto{P_k^-}{H_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T H_k^T}\right\} + E\left\{\cancelto{0}{H_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)\mathbf{v}_k^T}\right\}$$

$$+ E\left\{\cancelto{0}{\mathbf{v}_k(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T H_k^T}\right\} + E\left\{\cancelto{R_k}{\mathbf{v}_k \mathbf{v}_k^T}\right\}$$

  - So the covariance is given by

$$\boxed{E_k^- \equiv E\{\mathbf{e}_k^- \, \mathbf{e}_k^{-T}\} = H_k P_k^- H_k^T + R_k}$$

  - Note that this is always larger than R_k
    - Innovation error always “larger” than measurement error, which uses the truth instead of the estimate

MMAE Derivation (i)

- Set of parameters $\{\mathbf{p}^{(\ell)};\, \ell = 1, \ldots, M\}$ to produce a set of filtered estimates $\{\hat{\mathbf{x}}_k^{-(\ell)};\, \ell = 1, \ldots, M\}$
  - The goal of the MMAE process is to determine the conditional pdf of the j^{th} element $\mathbf{p}^{(j)}$ given all the measurements
  - This pdf is not easily obtained, but Bayes' rule from can be used to give a recursive formula

$$p(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_k) = \frac{p(\tilde{\mathbf{Y}}_k|\mathbf{p}^{(j)})\, p(\mathbf{p}^{(j)})}{p(\tilde{\mathbf{Y}}_k)} = \frac{p(\tilde{\mathbf{Y}}_k|\mathbf{p}^{(j)})\, p(\mathbf{p}^{(j)})}{\displaystyle\sum_{j=1}^{M} p(\tilde{\mathbf{Y}}_k|\mathbf{p}^{(j)})\, p(\mathbf{p}^{(j)})}$$

where $\tilde{\mathbf{Y}}_k$ denotes the sequence $\{\tilde{\mathbf{y}}_0, \tilde{\mathbf{y}}_1, \ldots, \tilde{\mathbf{y}}_k\}$

MMAE Derivation (ii)

- We wish to develop an update law that only is a function of the current measurement
- To accomplish this task, the conditional probability equality and Bayes' rule are used to yield

$$
\begin{aligned}
p(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_k) &= \frac{p(\tilde{\mathbf{y}}_k,\, \tilde{\mathbf{Y}}_{k-1},\, \mathbf{p}^{(j)})}{p(\tilde{\mathbf{y}}_k,\, \tilde{\mathbf{Y}}_{k-1})} \\
&= \frac{p(\tilde{\mathbf{y}}_k,\, \mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1})\, p(\tilde{\mathbf{Y}}_{k-1})}{p(\tilde{\mathbf{y}}_k|\tilde{\mathbf{Y}}_{k-1})\, p(\tilde{\mathbf{Y}}_{k-1})} \\
&= \frac{p(\tilde{\mathbf{y}}_k,\, \mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1})}{p(\tilde{\mathbf{y}}_k|\tilde{\mathbf{Y}}_{k-1})} \\
&= \frac{p(\tilde{\mathbf{y}}_k|\tilde{\mathbf{Y}}_{k-1},\, \mathbf{p}^{(j)})\, p(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1})}{\displaystyle\sum_{j=1}^{M} p(\tilde{\mathbf{y}}_k|\tilde{\mathbf{Y}}_{k-1},\, \mathbf{p}^{(j)})\, p(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1})} \qquad (1)
\end{aligned}
$$

MMAE Derivation (iii)

- For each $\mathbf{p}^{(j)}$ a set of state estimates is provided, denoted by $\hat{\mathbf{x}}_k^-(\mathbf{p}^{(j)}) \equiv \hat{\mathbf{x}}_k^{-(j)}$ through the filter banks
- Then $p(\tilde{\mathbf{y}}_k|\tilde{\mathbf{Y}}_{k-1},\, \mathbf{p}^{(j)})$ is given by $p\left(\tilde{\mathbf{y}}_k|\hat{\mathbf{x}}_k^{-(j)}\right)$ because $\hat{\mathbf{x}}_k^{-(j)}$ uses all the measurements up to time point $k-1$, and it is a function of $\mathbf{p}^{(j)}$
- Therefore, Eq. (1) becomes

$$p\left(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_k\right) = \frac{p\left(\tilde{\mathbf{y}}_k|\hat{\mathbf{x}}_k^{-(j)}\right) p\left(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1}\right)}{\sum_{j=1}^{M} p\left(\tilde{\mathbf{y}}_k|\hat{\mathbf{x}}_k^{-(j)}\right) p\left(\mathbf{p}^{(j)}|\tilde{\mathbf{Y}}_{k-1}\right)} \tag{2}$$

- Note that the denominator is just a normalizing factor to ensure that it is a pdf

MMAE Derivation (iv)

- Defining $\varpi_k^{(j)} \equiv p\left(\mathbf{p}^{(j)} | \tilde{\mathbf{Y}}_k\right)$ allows us to rewrite Eq. (2) as

$$\boxed{\begin{aligned} \varpi_k^{(j)} &= \varpi_{k-1}^{(j)} p\left(\tilde{\mathbf{y}}_k | \hat{\mathbf{x}}_k^{-(j)}\right) \\ \varpi_k^{(j)} &\leftarrow \frac{\varpi_k^{(j)}}{\displaystyle\sum_{j=1}^{M} \varpi_k^{(j)}} \end{aligned}}$$

where $\leftarrow$ denotes replacement

- Note that only the current time measurement is needed to update the weights
- The weights at time t_0 are initialized to

$$\varpi_0^{(j)} = 1/M \quad \text{for} \quad j = 1, 2, \ldots, M$$

MMAE Algorithm

- Weights

$$\varpi_k^{(\ell)} = \varpi_{k-1}^{(\ell)} p\left(\tilde{\mathbf{y}}_k | \hat{\mathbf{x}}_k^{-(\ell)}\right), \quad \varpi_k^{(\ell)} \leftarrow \varpi_k^{(\ell)} / \sum_{j=1}^{M} \varpi_k^{(j)}$$

where $\varpi_k^{(\ell)} \equiv p\left(\mathbf{p}^{(\ell)} | \tilde{\mathbf{y}}_k\right), \;\; \varpi_0^{(\ell)} = 1/M$

- Likelihood Function: $p\left(\tilde{\mathbf{y}}_k | \hat{\mathbf{x}}_k^{-(\ell)}\right) = L_k^{(\ell)}, \;\; \mathbf{e}_k^{-(\ell)} \equiv \tilde{\mathbf{y}}_k - \hat{\mathbf{y}}_k^{-(\ell)}$

$$L_k^{(\ell)} = \frac{1}{\left\{\det\left[2\pi \left(H_k P_k^{-(\ell)} H_k^T + R_k\right)\right]\right\}^{1/2}} \exp\left[-\frac{1}{2} \mathbf{e}_k^{-(\ell)T} \left(H_k P_k^{-(\ell)} H_k^T + R_k\right)^{-1} \mathbf{e}_k^{-(\ell)}\right]$$

- Estimate and covariance for states

$$\hat{\mathbf{x}}_k^{+} = \sum_{j=1}^{M} \varpi_k^{(j)} \hat{\mathbf{x}}_k^{+(j)}, \quad P_k^{+} = \sum_{j=1}^{M} \varpi_k^{(j)} \left[\left(\hat{\mathbf{x}}_k^{+(j)} - \hat{\mathbf{x}}_k^{+}\right)\left(\hat{\mathbf{x}}_k^{+(j)} - \hat{\mathbf{x}}_k^{+}\right)^T + P_k^{+(j)}\right]$$

- Similar equations for parameter estimate and covariance

$$\hat{\mathbf{p}}_k = \sum_{j=1}^{M} w_k^{(j)} \mathbf{p}^{(j)}, \quad \mathcal{P}_k = \sum_{j=1}^{M} w_k^{(j)} \left(\mathbf{p}^{(j)} - \hat{\mathbf{p}}_k\right)\left(\mathbf{p}^{(j)} - \hat{\mathbf{p}}_k\right)^T$$

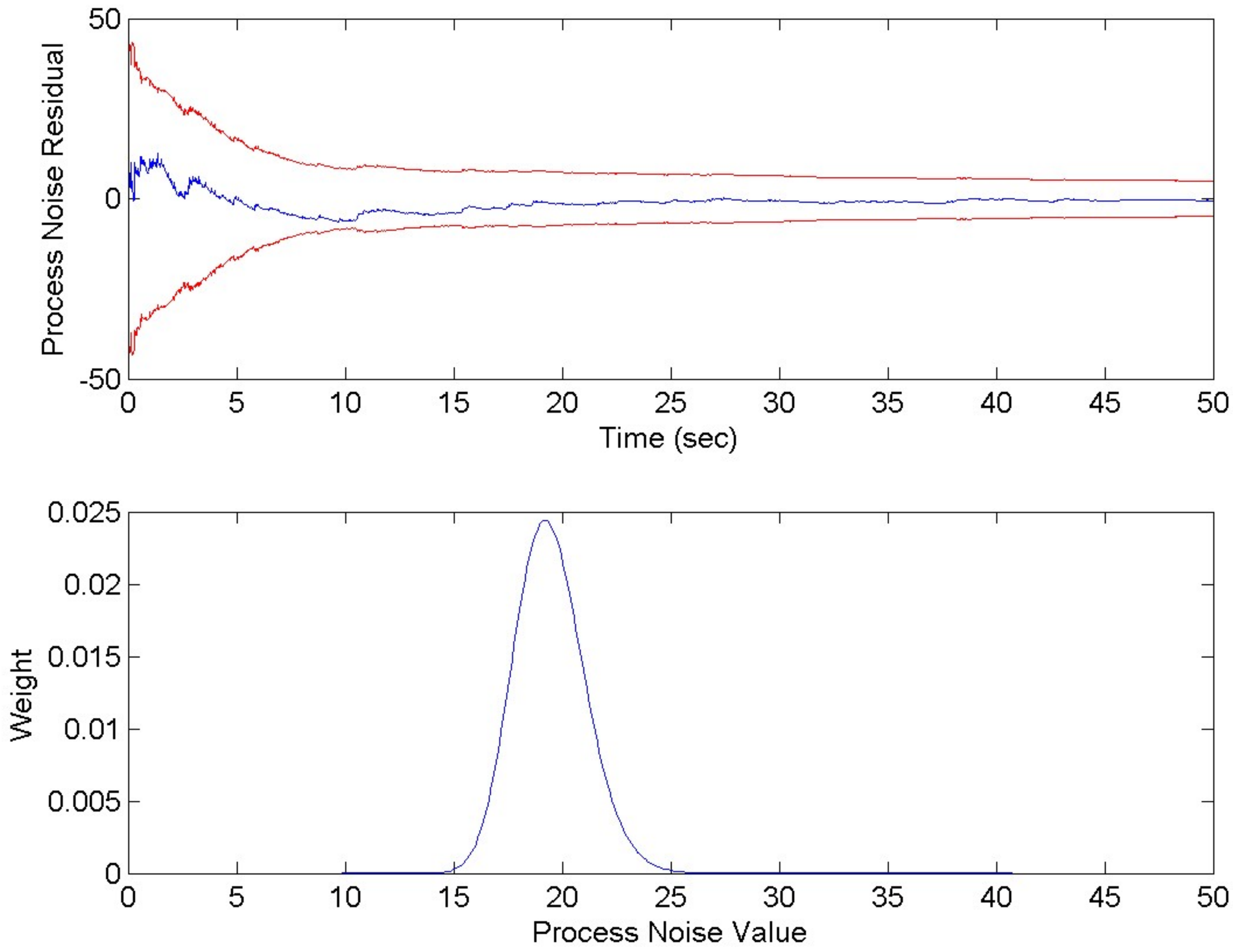
Example

- Simple example

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \mathbf{w}_k \qquad , \qquad Q = q \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^2/2 & \Delta t \end{bmatrix}$$
$$\tilde{y}_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}_k + \mathbf{v}_k$$

  - Measurement covariance set to 0.01
  - Goal is to estimate q using MMAE (true value is 20)
  - Used a final time of 50 seconds with $\Delta t = 0.01$ seconds
  - True initial conditions set to zero
  - Ran 500 parallel filters in the MMAE
    - Each filter is initialized with the true initial values
    - Initial covariance set to $P_0 = 0.001\, I$

Results



50
0
-50
Process Noise Residual
0 5 10 15 20 25 30 35 40 45 50
Time (sec)
0.025
0.02
0.015
0.01
0.005
0
Weight
0 5 10 15 20 25 30 35 40 45 50
Process Noise Value

Code (i)

```
% Process and Measurement Noise Covariances
q=20;r=0.01;

% Time and Models
dt=0.01;tf=50;t=[0:dt:tf]';m=length(t);
phi=[1 dt;0 1];
h=[1 0];n=2;

% Correlated Noise
q_dt=[dt^3/3 dt^2/2;dt^2/2 dt];
qd=q*q_dt;
[v_q,e_q]=eig(qd);
noise_uncorr=kron(diag(e_q)'.^(0.5),ones(m,1)).*randn(m,2);
noise=(v_q*noise_uncorr')';

% Truth and Measurements
x0=[0;0];
y=dlsim(phi,eye(2),h,[0 0],noise,x0);
ym=y+sqrt(r)*randn(m,1);
```

Code (ii)

```
% Process Noise Values and Weights for MMAE
n_part=500;
q_particle=linspace(1,50,500)';
qe=zeros(m,1);qe(1)=mean(q_particle);
w=ones(n_part,1)/n_part;
q_cov=zeros(m,1);
q_diff=q_particle-qe(1);
q_cov(1)=q_diff'*(q_diff.*w);

% Store all KF State Estimates and Covariances
p0=0.001*eye(2);
p=kron(ones(1,n_part),p0);
xe=kron(ones(1,n_part),x0);

p_kf=p0;
xe_kf=zeros(m,2);xe_kf(1,:)=x0(:)';
```

Code (iii)

```
% Main Loop
for i = 1:m-1

% Main Kalman Filter Using Estimated Q
    gain_kf=p_kf*h'/(h*p_kf*h'+r);
    xe_kf(i,:)=xe_kf(i,:)+(gain_kf*(ym(i)-h*xe_kf(i,:)'))';
    p_kf=(eye(2)-gain_kf*h)*p_kf;
    p_kf=phi*p_kf*phi'+qe(i)*q_dt;
    xe_kf(i+1,:)=(phi*xe_kf(i,:)')';

% Update and Propagate all KFs
    gain=reshape(h*p,n,n_part)./kron(((h*reshape(h*p,n,n_part))'+r),ones(1,n))';
    xe=xe+gain.*kron((ym(i)-(h*xe)'),ones(1,n))';
    for j=1:n_part,
        pp=p(1:2,2*j-1:2*j);
        pp=(eye(2)-gain(:,j)*h)*pp;
        p(1:2,2*j-1:2*j)=phi*pp*phi'+q_particle(j)*q_dt;
    end
    xe=phi*xe;
```

Code (iv)

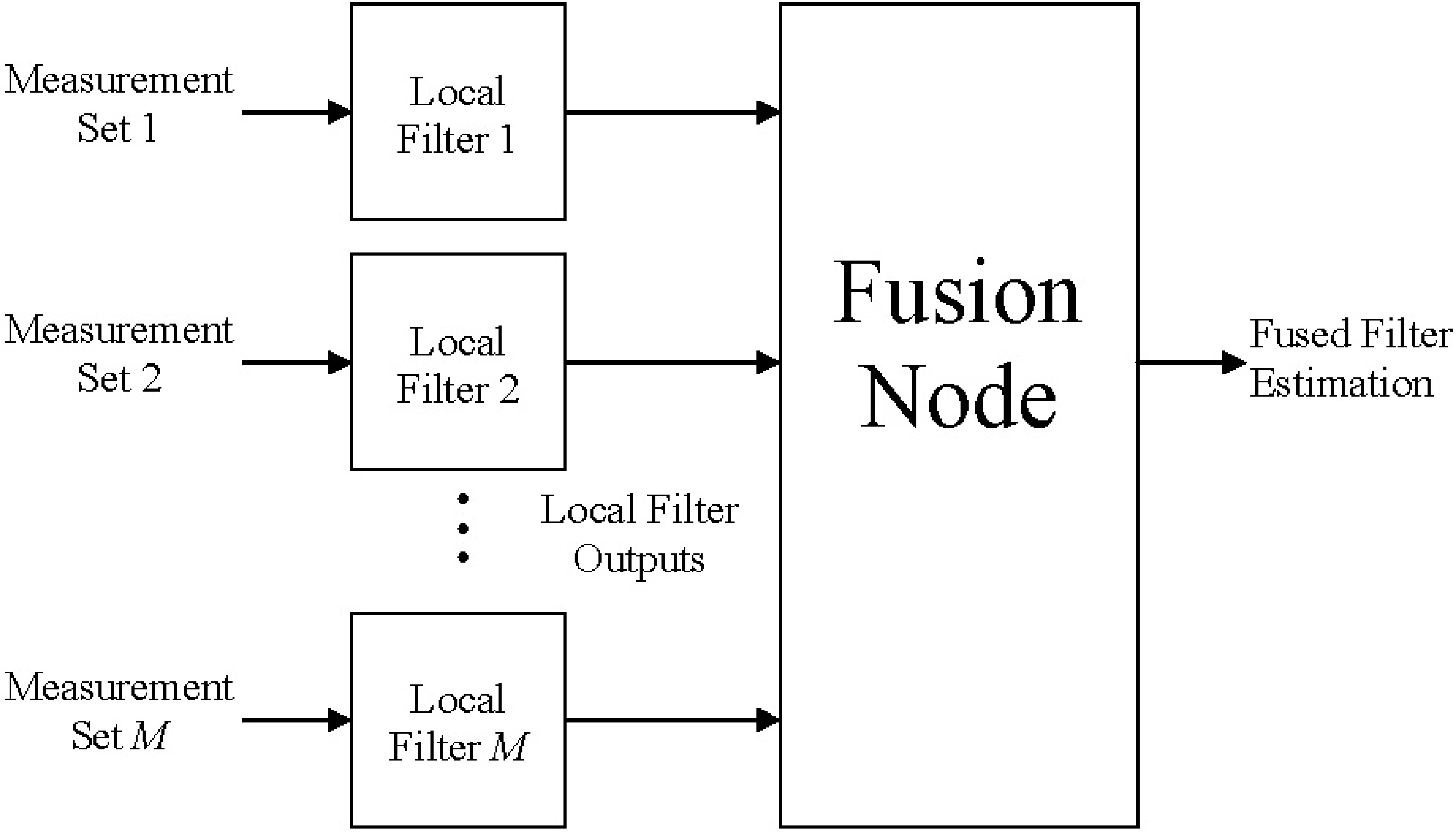
```
% Measurement Minus Estimate Likelihood and Weights
    r_cov=((h*reshape(h*p,n,n_part))'+r);
    w_nonnorm=w.*exp(-(ym(i+1)-(h*xe)').^2./(2*r_cov))./(r_cov.^(0.5));
    w=w_nonnorm/sum(w_nonnorm);
    qe(i+1)=sum(q_particle.*w);

    q_diff=q_particle-qe(i+1);
    q_cov(i+1)=q_diff'*(q_diff.*w);

end

% Plot Results
subplot(211);plot(t,q_cov.^(0.5)*3,'r',t,qe-q,'b',t,-q_cov.^(0.5)*3,'r');set(gca,'fontsize',12)
ylabel('Process Noise Residual');xlabel('Time (sec)')
subplot(212);plot(q_particle,w);set(gca,'fontsize',12)
ylabel('Weight');xlabel('Process Noise Value')

q_estimate=qe(m)
q_true=q
```

Decentralized Filtering

Introduction

- To this point all filtering concepts and examples have been assumed to be applied *centrally*
  - Measurement data are processed into a single filter to determine estimates of the state vector
- Decentralized filtering (distributed filtering)
  - Instead sending all measurement information to a central location for processing, multiple filters are executed at each node to develop multiple estimates
  - The estimates (not measurements) are instead sent to a fusion node, which combines estimates in some manner to provide an overall estimate
  - Many types of decentralized filtering concepts
    - Some feed back information to nodes to provide more information to improve local estimates (a *federated filter* is an example)
    - We will investigate the simplest one with no feedback information

Decentralized Filtering (i)



Decentralized Filtering (ii)

- Advantages and Disadvantages
  - Advantages
    - The two main advantages include reliability and flexibility
    - In a decentralized system each filter is providing a local estimate so that the overall system can still function with the loss of a single or multiple nodes, which provide a reliable solution
    - Flexible because local nodes can easily be added or deleted by simply adding or deleting communication links without a significant disruption in the overall architecture
  - Disadvantages
    - Decentralized fused estimate may not be optimal, i.e. it may not be equal to the centralized estimate
    - Redundant information causes problems; naïvely combining estimates may produce 3σ bounds that are *lower* than the optimal one
      - Better to be conservative than have this case happen in practice
    - We will focus on a method that overcomes the second disadvantage (method is called Covariance Intersection)

Covariance Intersection (i)

- Say two pieces of information, A and B, are to be fused to give an output C
  - Both are corrupted with noise, so A and B are random variables denoted by $\mathbf{a}$ and $\mathbf{b}$
  - We assume that the true statistics of these variables are unknown as well
    - But we do have estimates of their statistics, which are denoted by $\{\mathbf{a},\ P_{aa}\}$ and $\{\mathbf{b},\ P_{bb}\}$
  - Define the following true covariances and cross-correlation

$$\bar{P}_{aa} = E\{\tilde{\mathbf{a}}\,\tilde{\mathbf{a}}^T\}, \quad \bar{P}_{bb} = E\{\tilde{\mathbf{b}}\,\tilde{\mathbf{b}}^T\}, \quad \bar{P}_{ab} = E\{\tilde{\mathbf{a}}\,\tilde{\mathbf{b}}^T\}$$

  where $\tilde{\mathbf{a}} \equiv \mathbf{a} - \bar{\mathbf{a}}$ and $\tilde{\mathbf{b}} \equiv \mathbf{b} - \bar{\mathbf{b}}$ are the true errors
  - The only requirement is that

$$P_{aa} - \bar{P}_{aa} \geq 0 \quad \text{and} \quad P_{bb} - \bar{P}_{bb} \geq 0$$

    - This has to do with consistency

Covariance Intersection (ii)

- Objective is to find a linear, unbiased estimate that combines $\mathbf{a}$ and $\mathbf{b}$

$$\mathbf{c} = W_1\mathbf{a} + W_2\mathbf{b}$$

  where $\tilde{\mathbf{c}} \equiv \mathbf{c} - \bar{\mathbf{c}}$
  - For an unbiased estimate we require $E\{\tilde{\mathbf{c}}\} = \mathbf{0}$ which happens if and only if $W_1 + W_2 = I$
  - The covariance $\bar{P}_{cc} = E\{\tilde{\mathbf{c}}\,\tilde{\mathbf{c}}^T\}$ may be unknown, but we want to find its consistent estimate P_{cc}

$$P_{cc} - \bar{P}_{cc} \geq 0$$

  - Note that

$$\bar{P}_{cc} = [W_1 \;\; W_2] \begin{bmatrix} \bar{P}_{aa} & \bar{P}_{ab} \\ \bar{P}_{ab}^T & \bar{P}_{bb} \end{bmatrix} \begin{bmatrix} W_1^T \\ W_2^T \end{bmatrix}$$

Covariance Intersection (iii)

- Suppose that $\bar{P}_{ab} = 0$
  - Then for any given weighting matrices the following estimate can be shown to be consistent ($P_{cc} - \bar{P}_{cc} \geq 0$)

$$P_{cc} = W_1 P_{aa} W_1^T + W_2 P_{bb} W_2^T$$

  - How do we choose the weights? Let's minimize the trace of the above expression, which gives

$$P_{cc} = (P_{aa}^{-1} + P_{bb}^{-1})^{-1}$$
$$W_1 = P_{cc} P_{aa}^{-1} = P_{bb}(P_{aa} + P_{bb})^{-1}$$
$$W_2 = P_{cc} P_{bb}^{-1} = P_{aa}(P_{aa} + P_{bb})^{-1}$$

  - This actually corresponds to the derivation of the Kalman filter
  - Suppose that we have equal covariances (scalar case), then the standard deviation follows

$$\sigma_{cc} = \frac{1}{\sqrt{2}} \sigma_{aa}$$

Classic Result

Covariance Intersection (iv)

- If $\bar{P}_{ab} \neq 0$ but is known, then we have

$$P_{cc} = [W_1 \;\; W_2] \begin{bmatrix} P_{aa} & \bar{P}_{ab} \\ \bar{P}_{ab}^T & P_{bb} \end{bmatrix} \begin{bmatrix} W_1^T \\ W_2^T \end{bmatrix} \equiv W\,P\,W^T$$

  - Set up a constrained optimization problem to determine the weight

$$\min\, J(W) = \mathrm{Tr}(K\,P\,K^T) \quad \text{subject to} \quad K \begin{bmatrix} I \\ I \end{bmatrix} = I$$

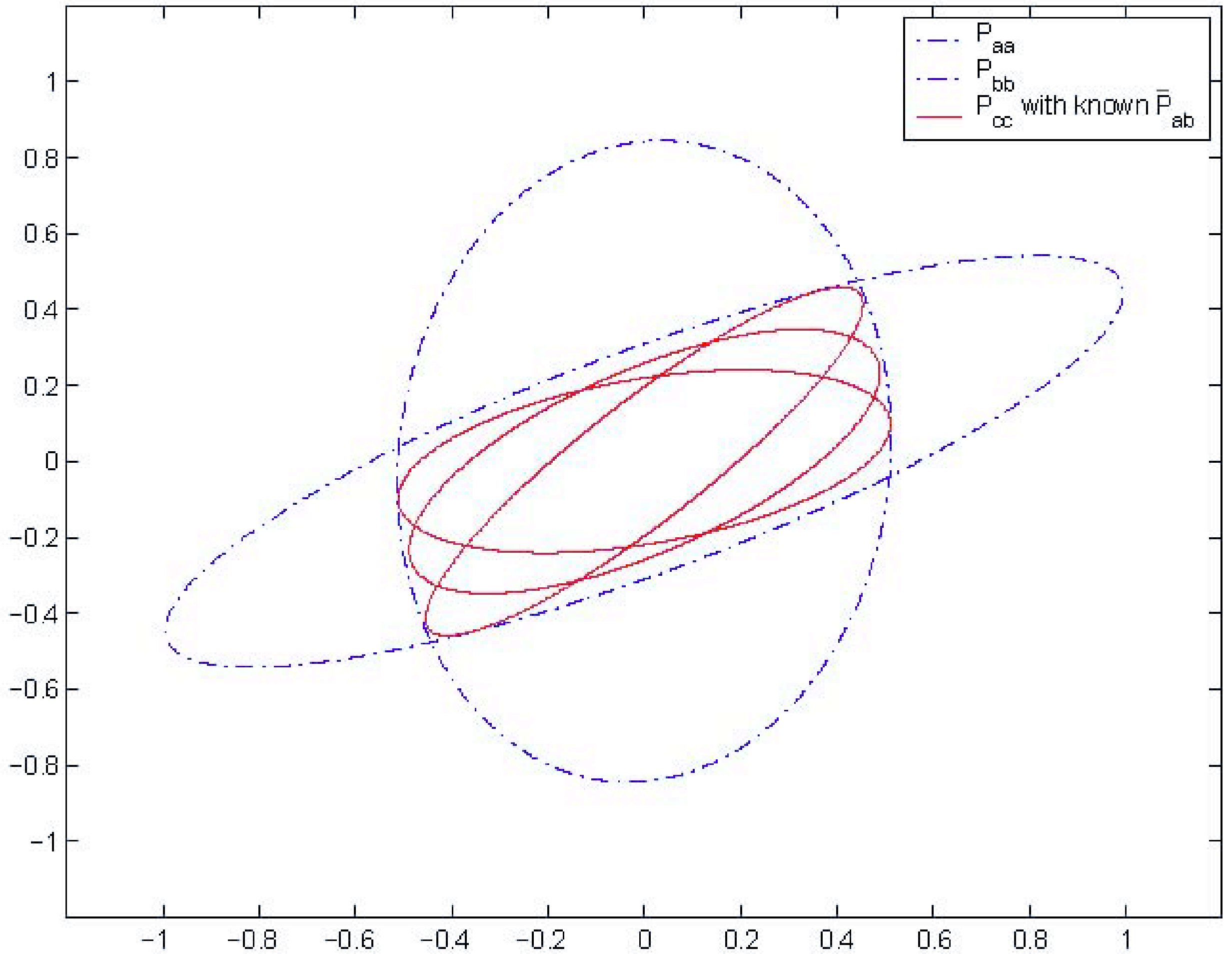
  - Optimal solution is given by

$$\begin{aligned} P_{cc}^{-1} &= [I \;\; I]\,P^{-1} \begin{bmatrix} I \\ I \end{bmatrix} \\ &= P_{aa}^{-1} + (P_{aa}^{-1}\bar{P}_{ab} - I)(P_{bb} - \bar{P}_{ab}^T P_{aa}^{-1} \bar{P}_{ab})^{-1}(\bar{P}_{ab}^T P_{aa}^{-1} - I) \end{aligned}$$

  - This can be shown to be consistent, so that

$$P_{cc} - \bar{P}_{cc} \geq 0$$

Optimal Solution



Always lies within intersection for known $\bar{P}_{ab}$

Covariance Intersection

- Suppose that $\bar{P}_{ab}$ is not known and is nonzero
  - A consistent estimate is given by

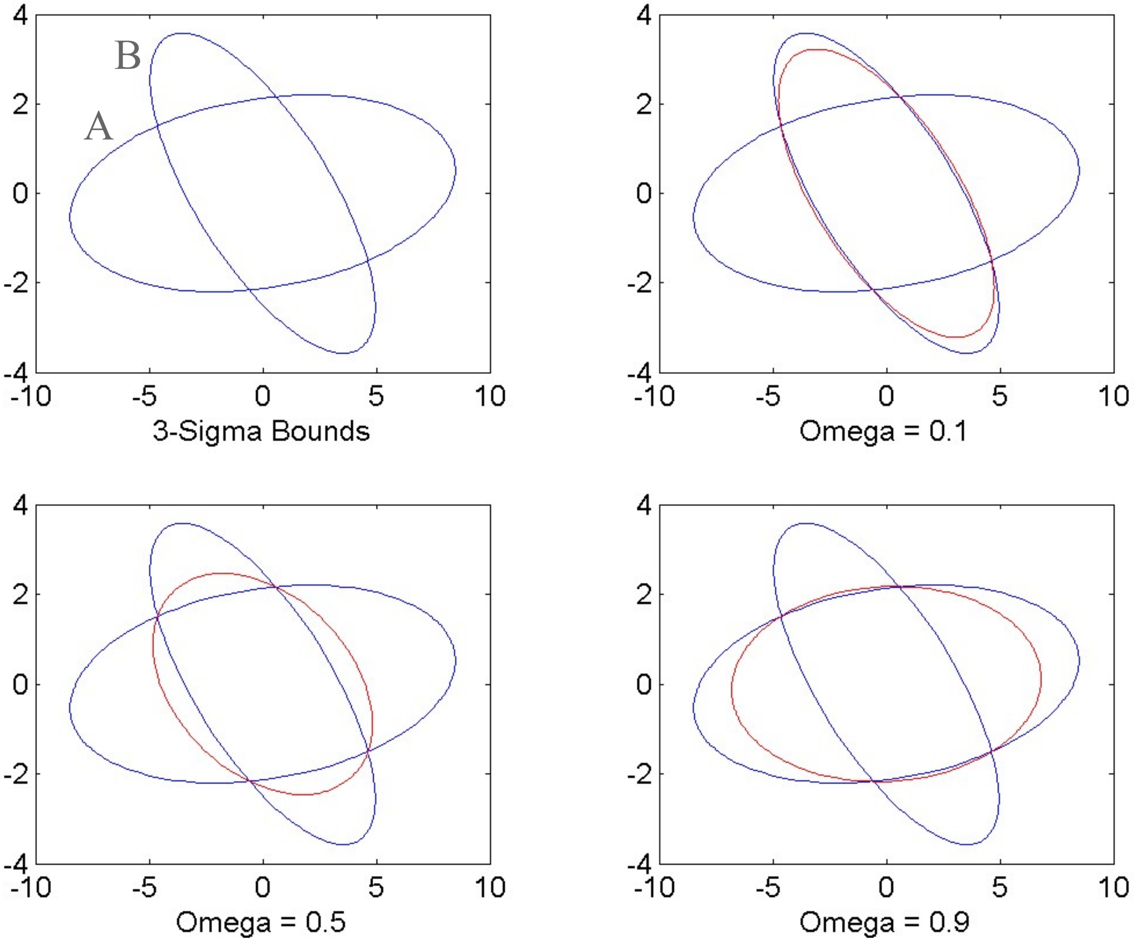
$$\boxed{\begin{aligned} P_{cc}^{-1} &= \omega P_{aa}^{-1} + (1-\omega) P_{bb}^{-1} \\ P_{cc}^{-1}\mathbf{c} &= \omega P_{aa}^{-1}\mathbf{a} + (1-\omega) P_{bb}^{-1}\mathbf{b} \end{aligned}}$$

  where $\omega \in [0,\ \ 1]$
  - Tuning parameter, ω , can be chosen to minimize the trace of the covariance
  - Standard optimization tools can be used
  - Can easily be extended for any number of variables

$$\begin{aligned} P_{cc}^{-1} &= \omega_1 P_{a_1 a_1}^{-1} + \cdots + \omega_n P_{a_n a_n}^{-1} \\ P_{cc}^{-1}\mathbf{c} &= \omega_1 P_{a_1 a_1}^{-1}\mathbf{a}_1 + \cdots + \omega_n P_{a_n a_n}^{-1}\mathbf{a}_n \end{aligned} \qquad \text{with } \sum_{i=1}^{n} \omega_i = 1$$

Julier, S.J., and Uhlmann, J.K., “Non-Divergent Estimation Algorithm in the Presence of Unknown Correlations” *Proceedings of the American Control Conference*, Vol.4, Piscataway, NJ, 1997, pp. 2369–2373.

Effect of Omega



Note the fused covariance always passes through the intersection of point A and B.

When $\omega = 0.5$, updated estimate is Kalman update with covariance inflated by a factor of 2.

Example (i)

- Determine the position of an unknown object using range measurements
  - The true location of the object is given by $x = 5$ and $y = 5$
  - Four sensors are assumed to move around the object with x and y coordinates given by the table below (sensor noise variances are also listed)

j	x_i	y_i	σ_i^2
1	1	t	0.01
2	$2\,t$	2	0.03
3	$-3\,t$	3	0.01
4	3	1	0.01

Time goes from 0 to 10 seconds. Note, fourth sensor is stationary

  - Synthetic range measurements are obtained using

$$\tilde{y}_i = [(x_i - x)^2 + (y_i - y)^2]^{1/2} + v_i, \quad i = 1,\, 2,\, 3$$

  - Measurements are sampled every 0.01 seconds

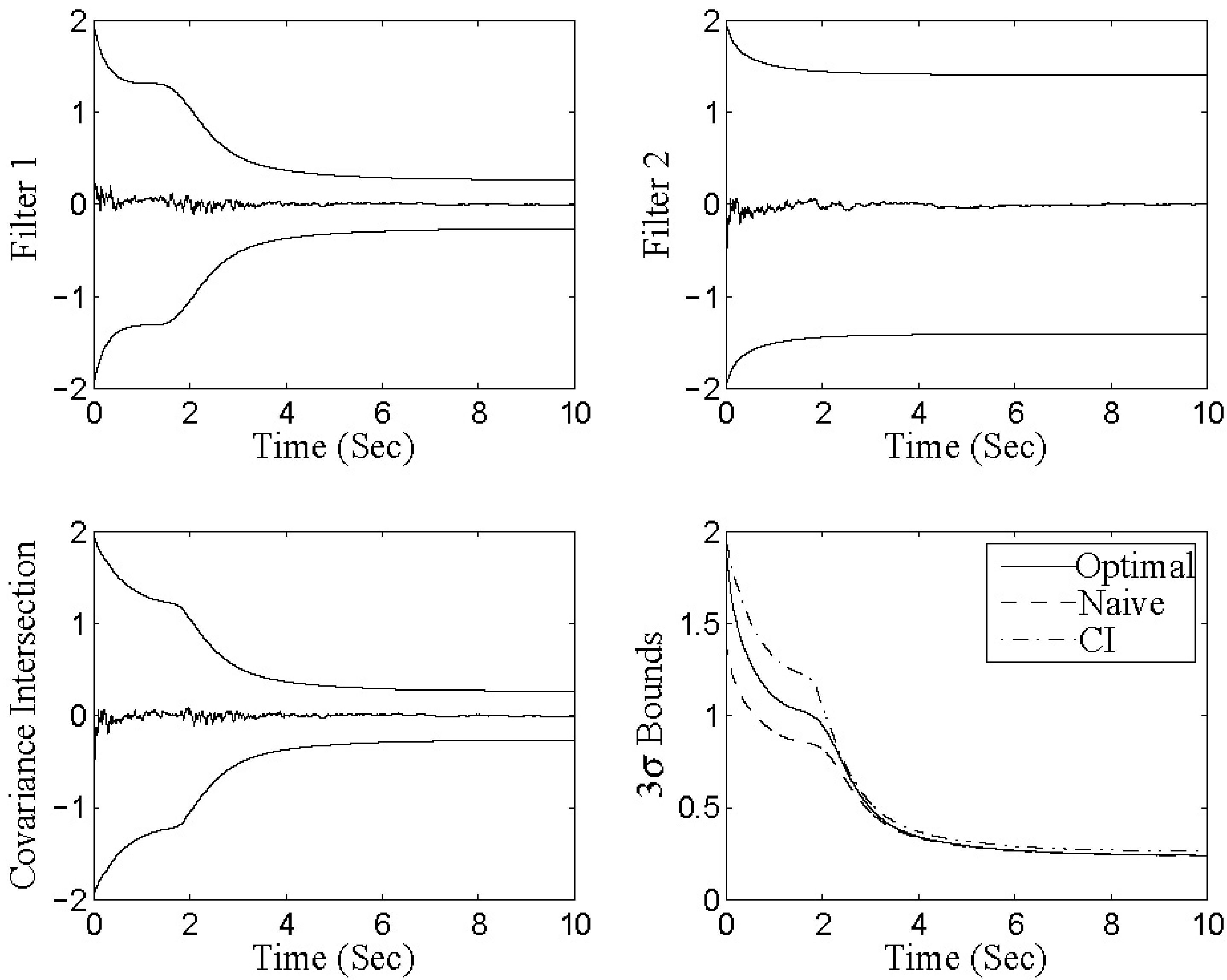
Example (ii)

- Two EKFs are used for the local filters
  - Both assume a first-order state model with no process noise
  - Two cases
    - Case 1: First node uses sensors 1 and 2, second uses 3 and 4 (note they measure the same object and cross correlation information is ignored when a naïve combination is used)
    - Case 2: First node uses sensors 1 and 2, second uses 2, 3 and 4 (measurements are double counted)
  - Optimal (central) EKF solution processes all four measurements
- CI solution is compared with a naïve combination

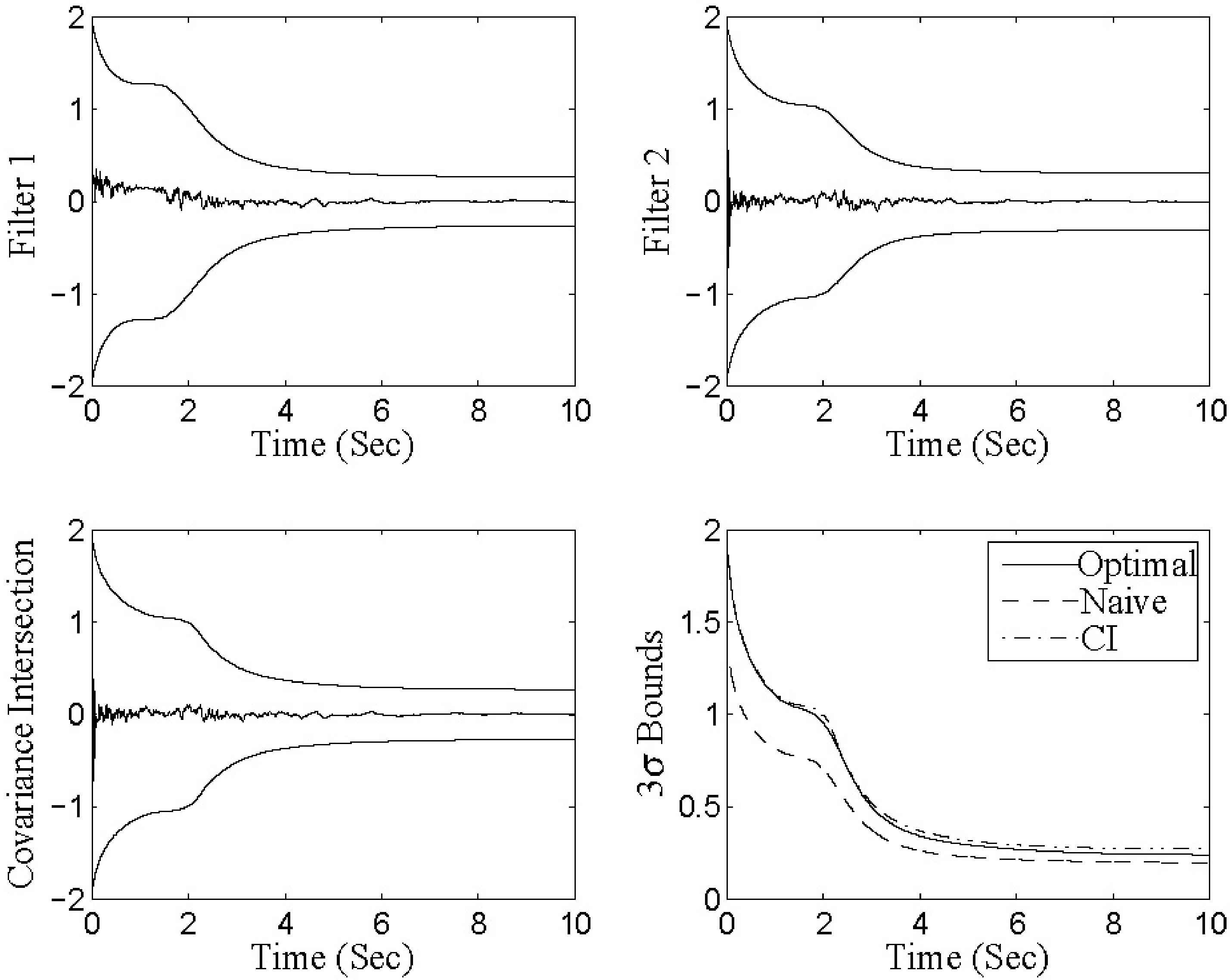
$$P_{cc}^{-1} = P_{aa}^{-1} + P_{bb}^{-1}$$

  - The naïve combination does not "know" about the cross correlation information
  - Naïve combined covariance is actually lower than the optimal one since it's not consistent in both cases (second case is worse than first because measurements are explicitly double counted)

Results – Case 1



Results – Case 2



Code (i)

```
% Set Flag for Case (default is double_count = 0)
% double_count = 0 uses measurements 1,2 in node 1 and measurements 3,4
in node 2
% double_count = 1 uses measurements 1,2 in node 1 and measurements
2,3,4 in node 2
double_count=0;

% Time
t=[0:0.01:10]';m=length(t);

% True Locations
x_loc=5;y_loc=5;
x1=1*ones(m,1);y1=1*t;
x2=2*t;y2=2*ones(m,1);
x3=-3*t;y3=3*ones(m,1);
x4=3*ones(m,1);y4=1*ones(m,1);

% State Transition Matrix
phi=eye(2);
```

Code (ii)

```
% Measurement Covariance
r=diag([0.03 0.01 0.03 0.01]);

% True Outputs
ytrue1=((x1-x_loc).^2+(y1-y_loc).^2).^(0.5);
ytrue2=((x2-x_loc).^2+(y2-y_loc).^2).^(0.5);
ytrue3=((x3-x_loc).^2+(y3-y_loc).^2).^(0.5);
ytrue4=((x4-x_loc).^2+(y4-y_loc).^2).^(0.5);

% Measurements
ym1=ytrue1+sqrt(r(1,1))*randn(m,1);
ym2=ytrue2+sqrt(r(2,2))*randn(m,1);
ym3=ytrue3+sqrt(r(3,3))*randn(m,1);
ym4=ytrue4+sqrt(r(4,4))*randn(m,1);
ym=[ym1 ym2 ym3 ym4];
```

Code (iii)

```
% Initial Condition and Covariance for Optimal Filter
x0=[4;4];p0=(2/3)^2*eye(2);p=p0;p_cov=zeros(m,2);p_cov(1,:)=diag(p)';
xe=zeros(m,2);xe(1,:)=x0';

% Initial Conditions and Covariances for Decentralized Filters
p1=p;p_cov1=zeros(m,2);p_cov1(1,:)=diag(p1)';
p2=p;p_cov2=zeros(m,2);p_cov2(1,:)=diag(p2)';
xe1=zeros(m,2);xe1(1,:)=x0';
xe2=zeros(m,2);xe2(1,:)=x0';

% Initial CI State and Covariance
p_ci=inv(0.5*inv(p1)+0.5*inv(p2));p_cov_ci=zeros(m,2);p_cov_ci(1,:)=diag(p_ci)';
xe_ci=zeros(m,2);xe_ci(1,:)=(0.5*p_ci*inv(p1)*xe1(1,:)'+0.5*p_ci*inv(p2)*xe2(1,:)')';
omega_store=zeros(m,1);omega_store(1)=0.5;

% Naive Covariance Combination
p_naive=inv(inv(p1)+inv(p2));p_cov_naive=zeros(m,2);p_cov_naive(1,:)=diag(p_naive)';
```

Code (iv)

```
% Main Loop
for i = 1:m-1

% Output Estimates for Optimal Filter
 ye1=((x1(i)-xe(i,1))^2+(y1(i)-xe(i,2))^2)^(0.5);
 h1=-[(x1(i)-xe(i,1)) (y1(i)-xe(i,2))]/ye1^3;
 ye2=((x2(i)-xe(i,1))^2+(y2(i)-xe(i,2))^2)^(0.5);
 h2=-[(x2(i)-xe(i,1)) (y2(i)-xe(i,2))]/ye2^3;
 ye3=((x3(i)-xe(i,1))^2+(y3(i)-xe(i,2))^2)^(0.5);
 h3=-[(x3(i)-xe(i,1)) (y3(i)-xe(i,2))]/ye3^3;
 ye4=((x4(i)-xe(i,1))^2+(y4(i)-xe(i,2))^2)^(0.5);
 h4=-[(x4(i)-xe(i,1)) (y4(i)-xe(i,2))]/ye4^3;
 h=[h1;h2;h3;h4];
 ye=[ye1;ye2;ye3;ye4];

% Update for Optimal Filter
 gain=p*h'*inv(h*p*h'+r);
 xe(i,:)=xe(i,:)+(gain*(ym(i,:)'-ye))';
 p=(eye(2)-gain*h)*p;
```

Code (v)

```
% Propagation for Optimal Filter
 xe(i+1,:)=xe(i,:);
 p=phi*p*phi';
 p_cov(i+1,:)=diag(p)';

% Output Estimates for Decentralized Filter 1
 ye1=((x1(i)-xe1(i,1))^2+(y1(i)-xe1(i,2))^2)^(0.5);
 h1=-[(x1(i)-xe1(i,1)) (y1(i)-xe1(i,2))]/ye1^3;
 ye2=((x2(i)-xe1(i,1))^2+(y2(i)-xe1(i,2))^2)^(0.5);
 h2=-[(x2(i)-xe1(i,1)) (y2(i)-xe1(i,2))]/ye2^3;
 h=[h1;h2];
 ye=[ye1;ye2];
 ym1f=ym(i,1:2);
 r1f=r(1:2,1:2);

% Update for Decentralized Filter 1
 gain=p1*h'*inv(h*p1*h'+r1f);
 xe1(i,:)=xe1(i,:)+(gain*(ym1f'-ye))';
 p1=(eye(2)-gain*h)*p1;
```

Code (vi)

```
% Propagation for Decentralized Filter 1
 xe1(i+1,:)=xe1(i,:);
 p1=phi*p1*phi';
 p_cov1(i+1,:)=diag(p1)';

% Output Estimates for Decentralized Filter 2
 ye2=((x2(i)-xe2(i,1))^2+(y2(i)-xe2(i,2))^2)^(0.5);
 h2=-[(x2(i)-xe2(i,1)) (y2(i)-xe2(i,2))]/ye2^3;
 ye3=((x3(i)-xe2(i,1))^2+(y3(i)-xe2(i,2))^2)^(0.5);
 h3=-[(x3(i)-xe2(i,1)) (y3(i)-xe2(i,2))]/ye3^3;
 ye4=((x4(i)-xe2(i,1))^2+(y4(i)-xe2(i,2))^2)^(0.5);
 h4=-[(x4(i)-xe2(i,1)) (y4(i)-xe2(i,2))]/ye4^3;
 if double_count == 1
  h=[h2;h3;h4];ye=[ye2;ye3;ye4];
  ym2f=ym(i,2:4);r2f=r(2:4,2:4);
 else
  h=[h3;h4];ye=[ye3;ye4];
  ym2f=ym(i,3:4);r2f=r(3:4,3:4);
 end;
```

Code (vii)

```
% Update for Decentralized Filter 2
 gain=p2*h'*inv(h*p2*h'+r2f);
 xe2(i,:)=xe2(i,:)+(gain*(ym2f'-ye))';
 p2=(eye(2)-gain*h)*p2;

% Propagation for Decentralized Filter 2
 xe2(i+1,:)=xe2(i,:);
 p2=phi*p2*phi';
 p_cov2(i+1,:)=diag(p2)';

% CI Solution Using Filters 1 and 2
 omega=fminbnd('ci_fun',0,1,[],p1,p2);
 p_ci=inv(omega*inv(p1)+(1-omega)*inv(p2));p_cov_ci(i+1,:)=diag(p_ci)';
 xe_ci(i+1,:)=(omega*p_ci*inv(p1)*xe1(i+1,:)'+(1-omega)*p_ci*inv(p2)*xe2(i+1,:)')';
 omega_store(i+1)=omega;

% Naive Covariance Combination
 p_naive=inv(inv(p1)+inv(p2));p_cov_naive(i+1,:)=diag(p_naive)';

end
```

Code (viii)

```
% Get 3-Sigma Bounds
sig3=p_cov.^(0.5)*3;
sig31=p_cov1.^(0.5)*3;
sig32=p_cov2.^(0.5)*3;
sig3_ci=p_cov_ci.^(0.5)*3;
sig3_naive=p_cov_naive.^(0.5)*3;

% Plot Results
clf
plot(t,omega_store)
set(gca,'fontsize',12)
set(gca,'Xtick',[0 2 4 6 8 10])
axis([0 10 -2 2])
xlabel('Time (Sec)')
ylabel('Omega')

pause
```

Code (ix)

```
subplot(221)
plot(t,-sig31(:,1),t,xe1(:,1)-x_loc,t,sig31(:,1))
set(gca,'fontsize',12)
axis([0 10 -2 2])
set(gca,'Xtick',[0 2 4 6 8 10])
xlabel('Time (Sec)')
ylabel('Filter 1')

subplot(222)
plot(t,-sig32(:,1),t,xe2(:,1)-x_loc,t,sig32(:,1))
set(gca,'fontsize',12)
axis([0 10 -2 2])
set(gca,'Xtick',[0 2 4 6 8 10])
xlabel('Time (Sec)')
ylabel('Filter 2')
```

Code (x)

```
subplot(223)
plot(t,-sig3_ci(:,1),t,xe_ci(:,1)-x_loc,t,sig3_ci(:,1))
set(gca,'fontsize',12)
axis([0 10 -2 2])
set(gca,'Xtick',[0 2 4 6 8 10])
xlabel('Time (Sec)')
ylabel('Covariance Intersection')

subplot(224)
plot(t,sig3(:,1),t,sig3_naive(:,1),'--',t,sig3_ci(:,1),'-.')
set(gca,'fontsize',12)
legend('Optimal','Naive','CI')
axis([0 10 0 2])
set(gca,'Xtick',[0 2 4 6 8 10])
xlabel('Time (Sec)')
ylabel('Bounds')
```

Code (xi)

```
function f=ci_fun(omega,p1,p2)

f=trace(inv(omega*inv(p1)+(1-omega)*inv(p2)));
```

Ensemble Kalman Filtering

Introduction

- Problems may arise when number of states is very large
  - Both numerical and computational problems
  - An example of a large state system is one that involves a discretization of a partial differential equation model
    - Most issues occur in trying to maintain and use the state covariance matrix in the Kalman filter
  - Classic problems in data assimilation employ large state vectors
    - Plume tracking, weather model, etc.
- Ensemble Kalman Filter
  - Uses a collection of state vectors, i.e. the *ensembles*, to replace the covariance matrix in a Kalman filter with the sample covariance
  - Avoids the computation of propagating the covariance equation, which becomes intractable for large state systems
  - Closely related to sequential Monte Carlo sampling filtering methods

Ensemble Kalman Filter

Model	$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k,\, \mathbf{u}_k,\, k) + \Upsilon_k \mathbf{w}_k,\ \mathbf{w}_k \sim N(\mathbf{0}, Q_k)$ $\tilde{\mathbf{y}}_k = \mathbf{h}(\mathbf{x}_k,\, \mathbf{u}_k,\, k) + \mathbf{v}_k,\ \mathbf{v}_k \sim N(\mathbf{0}, R_k)$
Initialize	$\hat{\mathbf{x}}^{(j)}(t_0) \sim N(\mathbf{x}_0, P_0)$ $P_0 = E\left\{\tilde{\mathbf{x}}(t_0)\, \tilde{\mathbf{x}}^T(t_0)\right\}$
Gain	$K_k = P_k^{e_x e_y}(P_k^{e_y e_y})^{-1}$
Update	$\hat{\mathbf{x}}_k^{+(j)} = \hat{\mathbf{x}}_k^{-(j)} + K_k\left[\tilde{\mathbf{y}}_k + \mathbf{v}_k^{(j)} - \hat{\mathbf{y}}_k^{-(j)}\right],\ \mathbf{v}_k^{(j)} \sim N(\mathbf{0}, R_k)$ $\hat{\mathbf{y}}_k^{-(j)} = \mathbf{h}(\hat{\mathbf{x}}_k^{-(j)},\, \mathbf{u}_k,\, k)$
Propagation	$\hat{\mathbf{x}}_{k+1}^{-(j)} = \mathbf{f}(\hat{\mathbf{x}}_k^{+(j)},\, \mathbf{u}_k,\, k) + \Upsilon_k \mathbf{w}_k^{(j)},\ \mathbf{w}_k^{(j)} \sim N(\mathbf{0}, Q_k)$ $\hat{\mathbf{x}}_k^{-} = \sum_{j=1}^{N} \mathbf{x}_k^{-(j)}, \quad \hat{\mathbf{y}}_k^{-} = \mathbf{h}(\hat{\mathbf{x}}_k^{-},\, \mathbf{u}_k,\, k)$
Covariances	$P_k^{e_x e_y} = \frac{1}{N-1}\sum_{j=1}^{N}[\hat{\mathbf{x}}_k^{-(j)} - \hat{\mathbf{x}}_k^{-}][\hat{\mathbf{y}}_k^{-(j)} - \hat{\mathbf{y}}_k^{-}]^T$ $P_k^{e_y e_y} = \frac{1}{N-1}\sum_{j=1}^{N}[\hat{\mathbf{y}}_k^{-(j)} - \hat{\mathbf{y}}_k^{-}][\hat{\mathbf{y}}_k^{-(j)} - \hat{\mathbf{y}}_k^{-}]^T$

Initial set of ensembles is generated using P_0 and $\mathbf{x}_0$

At each time step new $\mathbf{v}_k^{(j)}$ and $\mathbf{w}_k^{(j)}$ must be generated because the EnKF assumes independence between samples

Example (i)

- Estimate states from a 1D diffusion equation

$$\frac{\partial x(y,\, t)}{\partial t} = \frac{\partial^2 x(y,\, t)}{\partial y^2} + w(y,\, t)$$

  - A physical system that follows this equation is a heat conduction model of a thin and rigid body of length L, where $x(y,t)$ is the temperature at position y and time t
  - The term $w(y,t)$ is a heat source or sink disturbance, which is modeled using a zero-mean Gaussian noise process
  - Initial conditions are chosen as $\partial x(y,\, t)/\partial t = 0$ at $y = 0$ and $\partial x(y,\, t)/\partial t = 0$ at $y = L$
  - An approximate solution to this partial differential equation is possible by using a spatial discretization approach
  - Consider cutting the body into n slices with increment $\Delta y = L/n$
  - The temperature in each slice is denoted by $x_i(t) \equiv x(y,t)$ for $i = 1, 2, \ldots, n$

Example (ii)

- A central difference can be used to approximate the second derivative

$$\dot{x}_i(t) = \frac{x_{i+1}(t) - 2\,x_i(t) + x_{i-1}(t)}{\Delta y^2} + w_i(t)$$

where $x_{i+1}(t) \equiv x(y+\Delta y, t)$ and $x_{i-1}(t) \equiv x(y-\Delta y, t)$

- Using a difference approximation to the initial boundary conditions yields $x_0(t) = x_1(t)$ and $x_n(t) = x_{n+1}(t)$
- Thus we consider the following state vector

$$\mathbf{x}(t) = [x_1(t) \;\; x_2(t) \;\; \ldots, \;\; x_n(t)]^T$$

with initial conditions $x_i(0) = 1 + iL/n$

- The state space model is then given by

$$\dot{\mathbf{x}}(t) = F\,\mathbf{x}(t) + G\,\mathbf{w}(t)$$

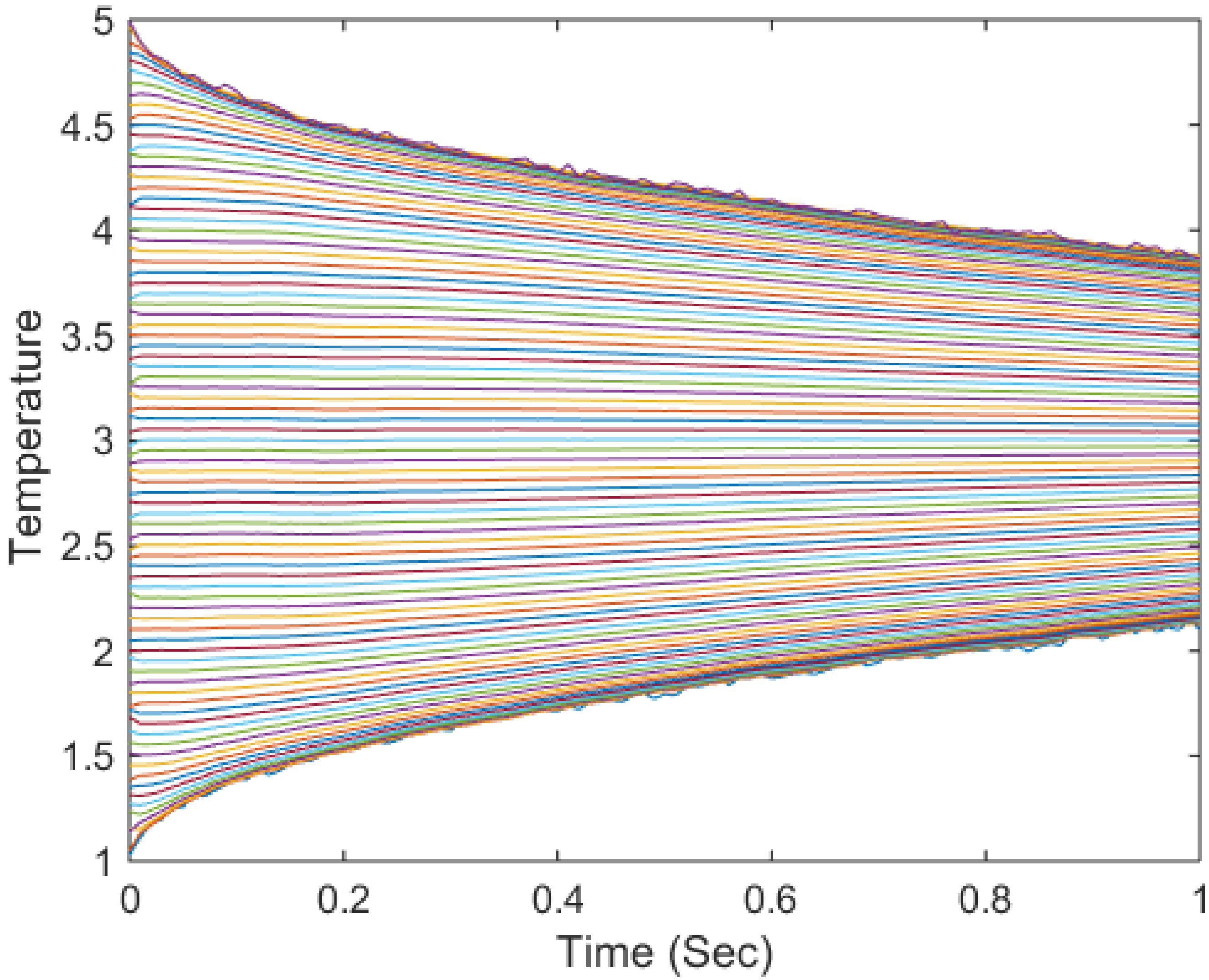
The matrix G distributes the heat source or sink

Example (iii)

$$F = \frac{1}{\Delta y^2} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & -1 \end{bmatrix}$$

- Performance of EnKF tested under the following conditions: $L = 4$ and $\Delta y = 0.005$ with a time increment of 0.01 seconds
- Results in an 801 state vector
- Simulation case with synthetic measurements of the states $x_1(t)$ and $x_2(t)$ using a variance of 0.01 for each measurement
- Process noise is added to the first and final states only using a spectral density of 1 for each state
- The initial states are set to their respective true values and P_0 is chosen to be $0.01\, I$

Example (iv)



Code (i)

```
% Define Parameters for Model and Time
length_y=4;delta_y=0.005;n=length_y/delta_y+1;
dt=0.01;tf=1;t=[0:dt:tf]';m=length(t);

% Get State Matrix
f=zeros(n);f(1,1:2)=[-1 1];f(n,n-1:n)=[1 -1];
for i=2:n-1,
 f(i,i-1:i+1)=[1 -2 1];
end
f=f/(delta_y)^2;

% Discrete-Time State Matrix
phi=c2d(f,zeros(n,1),dt);

% Process Noise Covariance
g=zeros(n,2);g(1,1)=1;g(n,2)=1;q=1*eye(2);qd=dt*q;

% Initial Conditions
x0=1+[1:n]'*length_y/n;
x=zeros(m,n);x(1,:)=x0(:)';
```

Code (ii)

```
% Output Matrix
h=zeros(2,n);h(1,1)=1;h(2,n)=1;

% Measurements
r=0.01*eye(2);
y=zeros(m,2);y(1,:)=(h*x(1,:)')';
ym=zeros(m,2);
ym(1,:)=y(1,:)+[sqrt(r(1,1))*randn(1) sqrt(r(2,2))*randn(1)];

% Number of Ensembles
n_ens=50;

% Initial Covariance and Ensemble Generation
p0=0.1^2*eye(n);
x_samp=kron(diag(p0).^(0.5),ones(1,n_ens)).*randn(n,n_ens)+kron(x0(:),ones(1,n_ens));
x_ens=x_samp;

% Estimates
xe=zeros(m,n);xe(1,:)=mean(x_ens,2)';
p_cov=zeros(m,n);
```

Code (iii)

```
% Main Loop
for i =1:m-1
% Generate Truth
 x(i+1,:)=(phi*x(i,:)'+dt*g*[sqrt(qd(1,1))*randn(1);sqrt(qd(2,2))*randn(1)])';
 y(i+1,:)=(h*x(i+1,:)')';ym(i+1,:)=y(i+1,:)+ [sqrt(r(1,1))*randn(1) sqrt(r(2,2))*randn(1)];

% Ensemble Process Noise and Measurement Noise
 w_samp=kron(diag(qd).^(0.5),ones(1,n_ens)).*randn(length(qd),n_ens);
 v_samp=kron(diag(r).^(0.5),ones(1,n_ens)).*randn(length(r),n_ens);

% Compute Sample Covariances
 e_state=x_ens-kron(xe(i,:)',ones(1,n_ens));e_out=h*e_state;
 p_xy=1/(n_ens-1)*e_state*e_out';
 p_yy=1/(n_ens-1)*e_out*e_out';
 p_cov(i,:)=1/(n_ens-1)*diag(e_state*e_state')';

% Ensemble Kalman Update
 gain_ens=p_xy*inv(p_yy);
 ens_res=kron(ym(i,:)',ones(1,n_ens))+v_samp-h*x_ens;
 x_ens=x_ens+gain_ens*ens_res;
```

Code (iv)

```
 % Ensemble Propagation and Estimate
 x_ens=phi*x_ens+g*w_samp;
 xe(i+1,:)=mean(x_ens,2)';

end

% Covariance at Final Point
e_state=x_ens-kron(xe(i+1,:)',ones(1,n_ens));
p_cov(i+1,:)=1/(n_ens-1)*diag(e_state*e_state')';
sig3=p_cov.^(0.5)*3;

% Plot Results
k_skip=[1:10:801]';
plot(t,xe(:,k_skip))
set(gca,'fontsize',12)
axis([0 1 1 5])
xlabel('Time (Sec)')
ylabel('Temperature')
```