Octave code for updating an IMM

- This lesson introduces Octave code "`iterIMM`", which implements one iteration of the IMM.

```
function immState = iterIMM(uold,unew,z,immState,immData)
```

- The inputs to this function comprise the prior system input `uold` $= u_{k-1}$ and the present input `unew` $= u_k$ (both of which are often zero), the present measurement `z` $= z_k$, and the data structures `immState`, and `immData`.
- The `immData` structure variable has fields:
  - `txprob`: This is the state transition matrix p_{ij}.
  - `A`, `B`, `C`, `D`: The system's discrete-time A, B, C, and D matrices, in a three-dimensional array, with the third dimension being the mode.
  - `Sw`, `Sv`: The system's discrete time noise matrices $\Sigma_{\widetilde{w}}$ and $\Sigma_{\tilde{v}}$, in a three-dimensional array, with the third dimension being the mode.

The `immState` structure

- The `immState` structure variable has fields:
  - `mode`: An $M \times 1$ vector containing the pmf describing the likelihood of the system operating in each mode at this point in time. Corresponds to $\mu_{i,k}$ for all values of $i \in \{1 \cdots M\}$ at this time step k.
  - `X`: An $n \times M$ matrix comprising the state estimate of each KF. The jth column of `X` contains the state estimate for the jth KF.
  - `SX`: An $n^2 \times M$ matrix comprising the covariance matrices of each KF. The jth column contains the covariance matrix of the jth KF in columnar form.
  - `xhat`: The output combined state estimate $\hat{x}_k^+$ from the IMM for this time step.
  - `SigmaX`: The output combined covariance $\Sigma_{\tilde{x},k}^+$ from the IMM for this time step.
- The code starts on the next slide; some of it is pretty tricky and bears close examination (especially some very efficient matrix operations that replace loops).

Code for IMM: iterIMM (interaction)

- The `iterIMM` function starts here:

```
function immState = iterIMM(uold,unew,z,immState,immData)
  modes = size(immData.A,3);
  nx = size(immData.A,1);

  % Interaction step: Compute x0 and Sx0 for each filter
  % 1) Compute cbar = sum(p(i,j)*mu(i,k-1))
  cbar = immData.txprob'*immState.mode;
  % 2) Compute mu(i|j,k-1) = 1/cbar * p(i,j)*mu(i,k-1)
  modeij = immData.txprob.*(immState.mode*(cbar.^(-1))');
  modeij(isnan(modeij)) = 0; % take care of impossible final states
  % 3) Compute xhat(mod)
  x0 = immState.X*modeij;
  % 4) Compute Sigma(mod).
  Sx0 = immState.SX; Sx1 = immState.SX; % reserve space for Sx0,Sx1
  for j = 1:modes,
    xk1 = immState.X(:,j); Sx = xk1*xk1'; Sx0(:,j) = Sx(:);
    xk1 = x0(:,j); Sx = xk1*xk1'; Sx1(:,j) = Sx(:);
  end
  Sx0 = (immState.SX + Sx0)*modeij - Sx1; % verified
```

Code for IMM: iterIMM (filtering)

```
  % Filtering step: Run all Kalman filters
  Lambda = zeros(size(cbar)); Sx = zeros(nx);
  for j = 1:modes
    % 0) Set up variables for this filter
    xhat = x0(:,j); Sx(:) = Sx0(:,j);
    A = immData.A(:,:,j); B = immData.B(:,:,j); C = immData.C(:,:,j);
    D = immData.D(:,:,j); Sw = immData.Sw(:,:,j); Sv = immData.Sv(:,:,j);
    % 1a) State estimate time update
    xhat = A*xhat + B*uold;
    % 1b) State covariance time update
    Sx = A*Sx*A' + Sw;
    % 1c) Output estimate
    zhat = C*xhat + D*unew;
    % 2a) Filter gain matrix
    Sz = (C*Sx*C'+Sv); L = Sx*C'/Sz;
    % 2b) State estimate measurement update
    immState.X(:,j) = xhat + L*(z-zhat);
    % 2c) State covariance measurement update
    Sx = Sx - L*Sz*L'; immState.SX(:,j) = Sx(:);
    Lambda(j) = max(1e-9,exp(-0.5 * ((z-zhat)^2/Py)) / sqrt(2*pi*Py));
  end
```


Code for IMM: iterIMM (combination)

```
  % Combination step: Compute overall xhat and SigmaX
  % 1) Compute PMF of being in mode j: mu(j,k)
  immState.mode = Lambda.*cbar;
  immState.mode = immState.mode/sum(immState.mode);
  % 2) Compute composite state estimate
  immState.xhat = immState.X*immState.mode;
  % 3) Compute composite covariance estimate
  modeSx = immState.SX;
  for j=1:modes
    xk1=immState.X(:,j)-immState.xhat;
    Sx=xk1*xk1'; modeSx(:,j)=Sx(:);
  end
  immState.SigmaX(:) = (immState.SX+modeSx)*immState.mode;
end
```

- This concludes the function `iterIMM`.
- Now, we look at how to use this function to run an IMM.

Code for IMM: Driver code

- The following is example driver code for this subroutine:
  - A one-dimensional tracking case is considered
  - The target being tracked operates according to an NCV model for 100 iterations, then according to an NCP model for 100 iterations.

```
% Set up model 1 for mode 1: NCV model, one dimensional, dT=1
% Start with continuous-time; convert to discrete-time
Ac = [0 1; 0 0]; Bc = [0; 1]; Swc = 2;        % continuous-time A/B/Sw
Z = [-Ac Bc*Swc*Bc'; zeros(size(Ac)) Ac']; % convert to discrete time
C = expm(Z*1); A1 = C(3:4,3:4)'; Sw1 = A1*C(1:2,3:4);
B1 = [1^2/2; 1]; C1 = [1 0]; Sv1 = 0.25;

% Set up model 2 for mode 2: NCP model, one dimensional, T=1
% Start with continuous-time; convert to discrete-time
Ac = [0 0; 0 0]; Bc = [1; 0]; Swc = 0.5;     % continuous-time A/B/Sw
Z = [-Ac Bc*Swc*Bc'; zeros(size(Ac)) Ac']; % convert to discrete time
C = expm(Z*1); A2 = C(3:4,3:4)'; Sw2 = A2*C(1:2,3:4);
B2 = [1; 0]; C2 = [1 0 ]; Sv2 = 0.1;
```

Code for IMM: Driver code

```
% Populate the immData structure
immData = [];
immData.txprob    = [0.95 0.05; 0.05 0.95];
immData.A(:,:,1)  = A1;  immData.A(:,:,2)  = A2;
immData.B(:,:,1)  = B1;  immData.B(:,:,2)  = B2;
immData.C(:,:,1)  = C1;  immData.C(:,:,2)  = C2;
immData.D(:,:,1)  = 0;   immData.D(:,:,2)  = 0;
immData.Sw(:,:,1) = Sw1; immData.Sw(:,:,2) = Sw2;
immData.Sv(:,:,1) = Sv1; immData.Sv(:,:,2) = Sv2;

% Populate the immState structure
immState = [];
immState.mode   = [0.8; 0.2];      % a-priori likelihood for each mode
immState.X      = [0 0; 0.3 0.3]; % initialize both modes to same estimate ...
immState.SX     = [0.1 0.1; 0 0; 0 0; 0.1 0.1]; % and columnar covariances
immState.xhat   = [0; 0];          % these two are outputs, not inputs, ...
immState.SigmaX = [0.1 0; 0 0.1]; % but I'll set them anyway
```


Code for IMM: Driver code

```
% Generate the true system data
xtrue = zeros([2,201]);
z = zeros(1,200);
for k = 1:100
  xtrue(:,k+1) = A1*xtrue(:,k) + chol(Sw1,'lower')*randn([2 1]);
  z(k) = C1*xtrue(:,k) + sqrt(Sv1)*randn(1);
end
for k = 101:200
  xtrue(:,k+1) = A2*xtrue(:,k) + sqrt(Sw2)*randn([2 1]); % avoid Cholesky error
  z(k) = C2*xtrue(:,k) + sqrt(Sv2)*randn(1);             % using "sqrt" since Sw2
end                                                      % Sw2 is not pos def

% Run the IMM on these data
xhatstore = zeros([2,200]);  modestore = zeros([2,200]);
sigmastore = zeros([4,200]);
for k = 1:200,
  immState = iterIMM(0,0,z(k),immState,immData);
  xhatstore(:,k)  = immState.xhat;
  sigmastore(:,k) = immState.SigmaX(:);
  modestore(:,k)  = immState.mode;
end
```


Code for IMM: Driver code

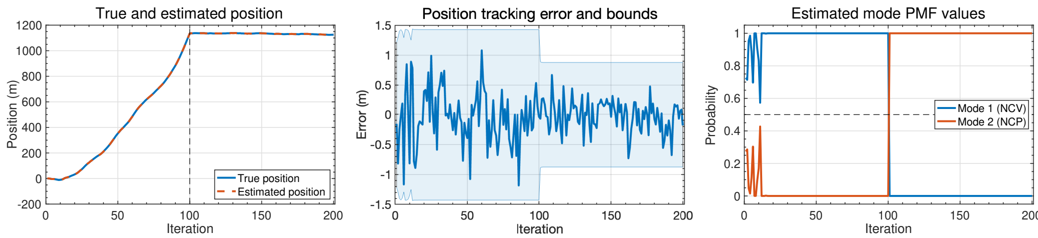
```
% Plot some results
figure(1); clf; plot(xtrue(1,:)); hold on;
hold on; plot(xhatstore(1,:),'r');
title('True and estimated position'); xlabel('Iteration');
ylabel('Position (m)'); a = axis; axis([0 201 a(3) a(4)]);

figure(2); clf; plot(xtrue(1,1:200)-xhatstore(1,:)); hold on;
plot(3*sqrt(sigmastore(1,:)),'k--','linewidth',0.5);
plot(-3*sqrt(sigmastore(1,:)),'k--','linewidth',0.5);
title('Tracking error and bounds');
xlabel('Iteration'); ylabel('Error (m)'); axis([0 201 -1.5 1.5]);

figure(3); clf; plot(modestore');
hold on; plot([0 201],[0.5 0.5],'k--','linewidth',0.5);
title('Estimated mode PMF values'); xlabel('Iteration');
ylabel('Probability'); axis([0 201 -0.05 1.05]);
```

Results from the IMM simulation

- The following graphs show sample output from this code.
- Some comments:
  - The state estimate is always very good;
  - The error bounds improve during NCP (calmer dynamics);
  - The mode tracking is very good: Errors only when NCV is actually stopped, or NCP is actually moving more than a negligible amount.



Summary

- In this lesson, you learned how to write an IMM in Octave.
- The code is divided into an iteration function and a script that sets up and executes the iteration function.
- You learned how to implement and execute an example where a target operates in two modes.
- For this example, the IMM produced excellent position estimates, confidence bounds, and mode estimates.