Introduction to lesson

- In this lesson, you will learn how to implement KF in Octave
  - Recommend you have KF equations nearby as reference
- Code will be demonstrated for simulated system where $\Sigma_{\widetilde{w}} = \Sigma_{\tilde{v}} = 1$ and

$$x_k = x_{k-1} + u_{k-1} + w_{k-1}$$
$$y_k = x_k + v_k$$

- Can be modified in straightforward ways to simulate other systems having other noise sources

Initializing the simulation

- The code begins with some definitions, initializations

```
% Initialize simulation variables
SigmaW = 1; % Process noise covariance
SigmaV = 1; % Sensor noise covariance
A = 1; B = 1; C = 1; D = 0; % Plant definition matrices
maxIter = 40; % Number of iterations to execute simulation

xtrue = 0;  % Initialize true system initial state
xhat = 0;   % Initialize Kalman filter initial estimate
SigmaX = 0; % Initialize Kalman filter covariance
u = 0;      % Unknown initial driving input: assume zero

% Reserve storage for variables we  want to plot/evaluate
xstore = zeros(length(xtrue),maxIter+1); xstore(:,1) = xtrue;
xhatstore = zeros(length(xhat),maxIter);
SigmaXstore = zeros(length(xhat)^2,maxIter);
```


KF step 1

- Code continues with main KF loop, step 1

```
for k = 1:maxIter,
  % KF Step 1a: State estimate time update
  xhat = A*xhat + B*u; % use prior value of "u"

  % KF Step 1b: Error covariance time update
  SigmaX = A*SigmaX*A' + SigmaW;

  % [Implied operation of system in background, with
  % input signal u, and output signal z]
  u = 0.5*randn(1) + cos(k/pi); % for example...
  w = chol(SigmaW)'*randn(length(xtrue));
  v = chol(SigmaV)'*randn(length(C*xtrue));
  ytrue = C*xtrue + D*u + v;  % z is based on present x and u
  xtrue = A*xtrue + B*u + w;  % future x is based on present u

  % KF Step 1c: Estimate system output
  yhat = C*xhat + D*u;
```

KF step 2

- Code continues with main KF loop, step 2

```
  % KF Step 2a: Compute Kalman gain matrix
  SigmaY = C*SigmaX*C' + SigmaV;
  L = SigmaX*C'/SigmaY;

  % KF Step 2b: State estimate measurement update
  xhat = xhat + L*(ytrue - yhat);

  % KF Step 2c: Error covariance measurement update
  SigmaX = SigmaX - L*SigmaY*L';

  % [Store information for evaluation/plotting purposes]
  xstore(:,k+1) = xtrue; xhatstore(:,k) = xhat;
  SigmaXstore(:,k) = SigmaX(:);
end
```


Plot results

- Finally, there is code to plot results

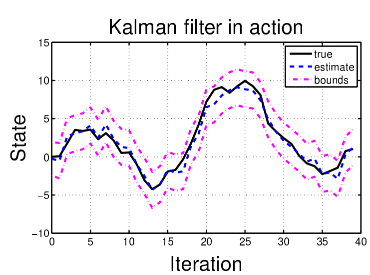
```
figure(1); clf;
plot(0:maxIter-1,xstore(1:maxIter)','k-',...
     0:maxIter-1,xhatstore','b--', ...
     0:maxIter-1,xhatstore'+3*sqrt(SigmaXstore)','m-.',...
     0:maxIter-1,xhatstore'-3*sqrt(SigmaXstore)','m-.'); grid;
legend('true','estimate','bounds');
title('Kalman filter in action');
xlabel('Iteration'); ylabel('State');

figure(2); clf;
plot(0:maxIter-1,xstore(1:maxIter)'-xhatstore','b-',...
     0:maxIter-1,3*sqrt(SigmaXstore)','m--',...
     0:maxIter-1,-3*sqrt(SigmaXstore)','m--'); grid;
legend('Error','bounds',0); title('Error with bounds');
xlabel('Iteration'); ylabel('Estimation Error');
```

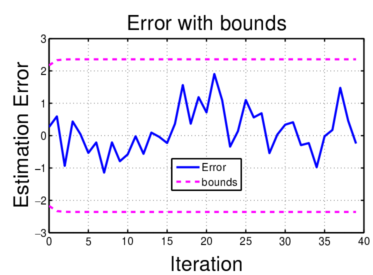

Sample results

- Plots show sample output from KF for $\Sigma_{\widetilde{w}} = \Sigma_{\tilde{v}} = 1$, and

$$x_k = x_{k-1} + u_{k-1} + w_{k-1}$$
$$y_k = x_k + v_k$$

- Error never converges to zero, but stays within predicted bounds (both as expected)





Summary

- You have now learned how to
  - Implement KF code in Octave
  - Cosimulate a system of interest with the KF to validate KF performance
- Code can be used to estimate state of any linear system obeying KF assumptions
- Sample results give intuitive sense for what to expect from KF