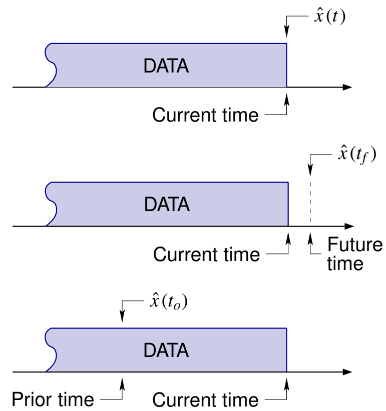
Three Kalman-filtering objectives

- There are three main Kalman-filtering objectives:
  1. We have concentrated on the *"filtering problem"*.
     - Use data up to and including the current time to provide an estimate for the *current* time.
  2. The *"prediction problem"*
     - Use data up to and including the current time to provide an estimate for a *future* time.
  3. The *"smoothing problem"* (offline, post-analysis)
     - Use data up to and including the current time to provide an estimate for a *past* time.



Kalman-filter prediction

- This lesson focuses on Kalman-filter prediction.
- Prediction estimates the system state at a time m beyond the data interval. That is (where $m > k$),

$$\hat{x}^-_{m|k} = \mathbb{E}[x_m \mid \mathbb{Z}_k].$$

- There are three different prediction scenarios:
  - Fixed-point prediction: Find $\hat{x}^-_{m|k}$ where m is fixed, but k is changing as more data become available;
  - Fixed-lead prediction: Find $\hat{x}^-_{k+M|k}$ where M is a fixed lead time;
  - Fixed-interval prediction: Find $\hat{x}^-_{m|k}$ where k is fixed, but m can take on multiple future values.
- The desired predictions can be extrapolated from the standard Kalman filter state and estimates.

Predicting the future state

- The basic approach is to use the relationship:

$$x_m = A^{m-k}x_k + \sum_{i=k}^{m-1} A^{m-i-1}Bu_i + \sum_{i=k}^{m-1} A^{m-i-1}w_i,$$

(where $m > k$) in the relationship $\hat{x}^-_{m|k} = \mathbb{E}[x_m \mid \mathbb{Z}_k]$, with the additional knowledge that $\hat{x}^+_k = \mathbb{E}[x_k \mid \mathbb{Z}_k]$ from a standard Kalman filter. That is,

$$\begin{aligned}\hat{x}^-_{m|k} &= \mathbb{E}[x_m \mid \mathbb{Z}_k] \\ &= \mathbb{E}\left[A^{m-k}x_k \mid \mathbb{Z}_k\right] + \mathbb{E}\left[\sum_{i=k}^{m-1} A^{m-i-1}B_iu_i \mid \mathbb{Z}_k\right] + \mathbb{E}\left[\sum_{i=k}^{m-1} A^{m-i-1}w_i \mid \mathbb{Z}_k\right] \\ &= A^{m-k}\hat{x}^+_k + \sum_{i=k}^{m-1} A^{m-i-1}B\mathbb{E}[u_i \mid \mathbb{Z}_k].\end{aligned}$$

- Note that we often assume that $\mathbb{E}[u_k] = 0$, so $\hat{x}^-_{m|k} = A^{m-k}\hat{x}^+_k$.

The prediction-error covariance

- The covariance of the prediction is:

$$\begin{aligned}\Sigma_{\tilde{x},m|k}^{-} &= \mathbb{E}[(x_m - \hat{x}_{m|k}^{-})(x_m - \hat{x}_{m|k}^{-})^T \mid \mathbb{Z}_k] \\ &= A^{m-k}\Sigma_{\tilde{x},k}^{+}\left(A^{m-k}\right)^T + \sum_{j=1}^{m-k} A^j \Sigma_{\widetilde{w}} \left(A^j\right)^T.\end{aligned}$$

- And that's all! Now, we're ready to implement a fixed-lead-time predictor in Octave.

Fixed-lead-time prediction

- The following Octave code initializes a prediction simulation.

```
clearvars
load simOut.mat Ad Bd Cd Dd SigmaV SigmaW dT t u x z

% Initialize simulation variables
M = 6; % how many steps into the future to predict the state

[nx,nt] = size(x); [nz,~] = size(z);
xhat = zeros(nx,1);      % Initialize Kalman filter initial estimate
SigmaX = zeros(nx,nx);   % Initialize Kalman filter covariance (part a)

% Reserve storage for variables we might want to plot/evaluate
xhatstore = zeros(nx,nt);
boundstore = xhatstore;
xhatstore(:,1) = xhat;
xhatPstore = zeros(nx,nt+M);
boundPstore = xhatPstore;
```


Beginning of main loop

```
for k = 2:nt
  % KF Step 1a: State prediction time update
  xhat = Ad*xhat + Bd*u(:,k-1); % use prior value of "u"

  % KF Step 1b: Prediction-error covariance time update
  SigmaX = Ad*SigmaX*Ad' + SigmaW;

  % KF Step 1c: Estimate system output
  zhat = Cd*xhat + Dd*u(k);

  % KF Step 2a: Compute Kalman gain matrix
  L = SigmaX*Cd'/(Cd*SigmaX*Cd' + SigmaV);

  % KF Step 2b: State estimate measurement update
  xhat = xhat + L*(z(k) - zhat);

  % KF Step 2c: Estimation-error covariance measurement update
  SigmaX = SigmaX - L*Cd*SigmaX;
```

Conclusion of main loop

```
  % Store estimate and bounds
  xhatstore(:,k) = xhat;
  boundstore(:,k) = 3*sqrt(diag(SigmaX));

  % Predict state M timesteps into future, along with bounds
  xhatPstore(:,k+M) = Ad^M * xhat;
  useU = 0; % set to 1 if we may use "future" u(k); else set to zero
  for  j = 0:M-1
    xhatPstore(:,k+M) = xhatPstore(:,k+M) + useU*Ad^(M-j-1)*Bd*u(:,min(nt,k+j));
  end

  SigmaXpred = Ad^M*SigmaX*(Ad^M)';
  for  j = 1:M
    SigmaXpred = SigmaXpred + Ad^j * SigmaW * (Ad^j)';
  end
  boundPstore(:,k+M) = 3*sqrt(diag(SigmaXpred));
end
xhatPstore = xhatPstore(:,1:nt);   % truncate at data length
boundPstore = boundPstore(:,1:nt);
```


Plot estimated and predicted states

- Plot states, estimates, and predicted states:

```
CL = lines;
figure(1); clf;
t2 = [t fliplr(t)]; % Prepare for plotting bounds via "fill"
x2 = [xhatstore-boundstore fliplr(xhatstore+boundstore)];
x3 = [xhatPstore-boundPstore fliplr(xhatPstore+boundPstore)];
h1 = fill(t2,x2,CL(1,:),'FaceAlpha',0.15,'LineStyle','none'); hold on; grid on
h3 = fill(t2,x3,CL(2,:),'FaceAlpha',0.20,'LineStyle','none');
set(gca,'ColorOrderIndex',1);
h2 = plot(t,x(1:2,:)',t,xhatstore(1:2,:)','--'); %ylim([-0.15 0.25]);
h4 = plot(t,xhatPstore(1:2,:)','--'); ylim([-0.16 0.26]);
legend([h2;h4;h1(1);h3(1)],{'True posn.','True vel.','Posn. est.',...
  'Vel. est.','Posn. smooth','Vel. smooth','KF bounds','Pred. bounds'},'
    NumColumns',3);
title('KF state estimates (L-step prediction)');
xlabel('Time (s)'); ylabel('State (m or m/s)');
```


Plot estimation and prediction errors

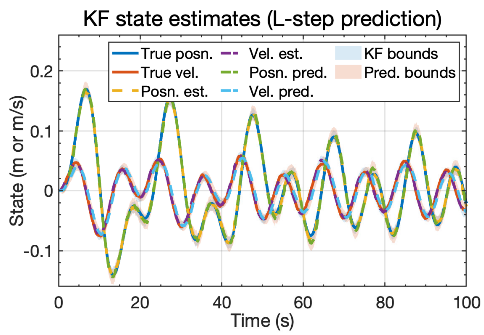
- Plot estimation and prediction errors:

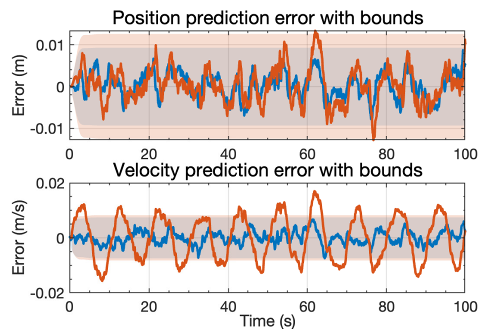
```
figure(2); clf; xerr = x - xhatstore; xPerr = x - xhatPstore;
xPerr(:,1:M) = 0; % zero out prediction error before timestep M
subplot(2,1,1);
fill([t fliplr(t)],[-boundstore(1,:) fliplr(boundstore(1,:))],CL(1,:),...
  'FaceAlpha',0.15,'LineStyle','none'); hold on; grid on;
fill([t fliplr(t)],[-boundPstore(1,:) fliplr(boundPstore(1,:))],CL(2,:),...
  'FaceAlpha',0.20,'LineStyle','none');
plot(t,xerr(1,:),'Color',CL(1,:)); plot(t,xPerr(1,:),'Color',CL(2,:));
title('Position prediction error with bounds'); ylabel('Error (m)');

subplot(2,1,2);
fill([t fliplr(t)],[-boundstore(2,:) fliplr(boundstore(2,:))],CL(1,:),...
  'FaceAlpha',0.15,'LineStyle','none'); hold on; grid on;
fill([t fliplr(t)],[-boundPstore(2,:) fliplr(boundPstore(2,:))],CL(2,:),...
  'FaceAlpha',0.20,'LineStyle','none');
plot(t,xerr(2,:),'Color',CL(1,:)); plot(t,xPerr(2,:),'Color',CL(2,:));
title('Velocity prediction error with bounds');
xlabel('Time (s)'); ylabel('Error (m/s)');
```

Kalman-predictor results for unknown future u_k

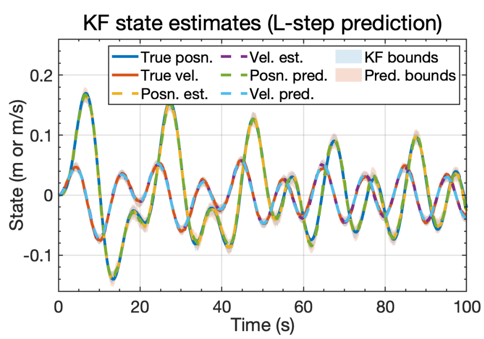
- Figures show results of the fixed-lead predictor for $L = 6$.
- In the error plots, the red lines (and shading) show the prediction error and the blue lines (and shading) show the standard KF estimates.
- Bounds often violated due to incorrectly assuming $u_k = 0$ over prediction interval.

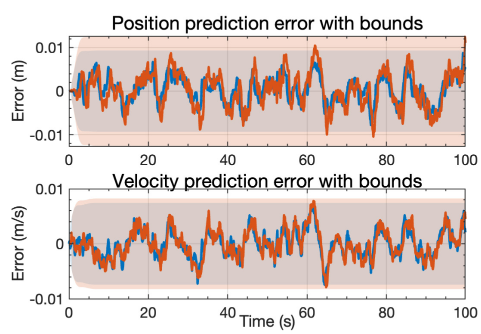




Kalman-predictor results for known future u_k

- Figures show results of the fixed-lead predictor for $L = 6$.
- In this case, we have assumed that future u_k is known.
- The bounds are the same as before, but the predictions are much better and do not violate the bounds.





Summary

- The Kalman filter can be extended to predict a system's state in the future.
- Three common scenarios: Fixed-point prediction, fixed-lead prediction, fixed-interval prediction.
- We focused here on fixed-lead prediction:
  - A standard KF is run and signals are saved.
  - An additional step computes the M-step-ahead prediction and its bounds.
- You learned how to implement a fixed-lead Kalman predictor in Octave code.
- An example showed that errors are larger and bounds wider than simply using a standard Kalman filter due to the additional uncertainty of future inputs and noises.