```
clear; close all; clc
set(0,'DefaultFigureWindowStyle','docked')
```

Table of Contents

Simulation parameters 1
System model 1
  Continous time motion model 1
  Discrete time motion model 1
  Measurement model 2
Run the simulation 3
Run the filter 4
  Standard Kalman filter equations 4
  Evaluate the performance 6

Simulation parameters

```
dt = 1;             % sample time [sec]
T = 50;             % duration [sec]
t = 0:dt:T-dt;      % time vector
N = length(t);      % number of steps
```

System model

Constant velocity motion in 2 dimensions

State vector: $\mathbf{x} = [x, \dot{x}, y, \dot{y}]$

Create a block matrix easily with the Kronecker product:

```
A = kron(eye(2),[0,1;0,0]);
```

Continous time motion model

$\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$

$$\begin{bmatrix} \dot{x}(t) \\ \ddot{x}(t) \\ \dot{y}(t) \\ \ddot{y}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ \dot{x}(t) \\ y(t) \\ \dot{y}(t) \end{bmatrix}$$

Discrete time motion model

$$\mathbf{x}_{k+1} = F\mathbf{x}_k$$

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ y_{k+1} \\ \dot{y}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \end{bmatrix}$$

Convert the continous time system matrix to discrete time with the matrix exponential: $F = \mathrm{e}^{Adt}$

```
F = expm(A*dt);
```

Instead of the exact CV we will use the Nearly Constant Velocity (NCV) model to introduce some uncertainty. The noise is implemented as an Additive White Gaussian Noise (AWGN) acceleration:

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + \Gamma w_k$$

w_k is the acceleration random vector: $w_k = [w_x, w_y]^\top$. This is the amount of change in velocity during time interval dt.

The matrix Γ describes how the noise acts on the state vector:

$$\Gamma = \begin{bmatrix} dt^2/2 & 0 \\ dt & 0 \\ 0 & dt^2/2 \\ 0 & dt \end{bmatrix}$$

```
GAM = [dt^2/2, 0;
         dt, 0;
         0, dt^2/2;
         0, dt];
```

w_k is a two dimensional Gaussian random vector with zero mean. The covariance martix is diagonal, meaning that the noise in the x and y direction is independent hence uncorrelated.

```
q = diag([1,1]) * 2;
```

The matrix Γ transforms the covariance matrix q into

```
Q = GAM * q * GAM';
```

Measurement model

Measuring only the position coordinates

$$\mathbf{z} = H\mathbf{x}$$

$$\begin{bmatrix} z_x \\ z_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}$$

```
H = [1 0 0 0; 0 0 1 0];
```

Modeling sensor noise

$\mathbf{z}_k = H\mathbf{x}_k + v_k$

v_k is a two dimensional gaussian random vector with zero mean and covariance matrix R:

```
R = diag([1,1]) * 100;
```

Run the simulation

```
% Initial velocity
vx0 = 10;
vy0 = 10;

% Starting position
x0 = 100;
y0 = 0;

% Initial state vector
X = zeros(4,N);
X(:,1) = [x0; vx0; y0; vy0];

% Run the simulation
for k = 2:N
   X(:,k) = F * X(:,k-1) + mvnrnd([0;0;0;0],Q)';
end

% Generate measurements
Z = zeros(2,N);
for k = 1:N
    % H*X([1,3],k) + mvnrnd([0;0],R)';
    % mvnrnd(H*X([1,3],k),R)';
    Z(:,k) = X([1,3],k) + mvnrnd([0;0],R)';
end

% Plot trajectory and measurement
fig = figure(1);
hold on; box on; axis equal
plot(0,0,'ro')
plot(X(1,:),X(3,:),'b-')
plot(Z(1,:),Z(2,:),'go')
```

Run the filter

```
% initialize state vector and covariance matrx
X_hat = zeros(4,N);
X_hat(:,1) = [0;0;0;0];

% If P is huge it means we are very uncertain about the state, the filter will give it a small
P = diag([1,1,1,1]) * 1e10;
```

Standard Kalman filter equations

Prediction

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathsf{T}} + \mathbf{Q}_k$$

Residual

$$\mathbf{r}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$$

$$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathsf{T}}$$

Gain

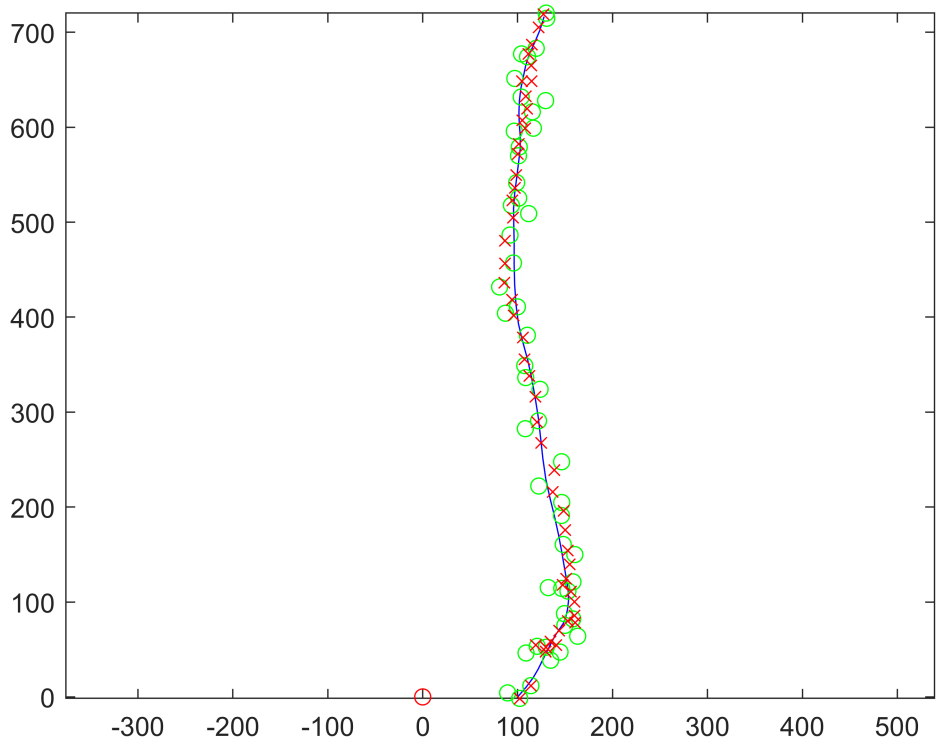
$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathsf{T}} \mathbf{S}_k^{-1}$$

Update

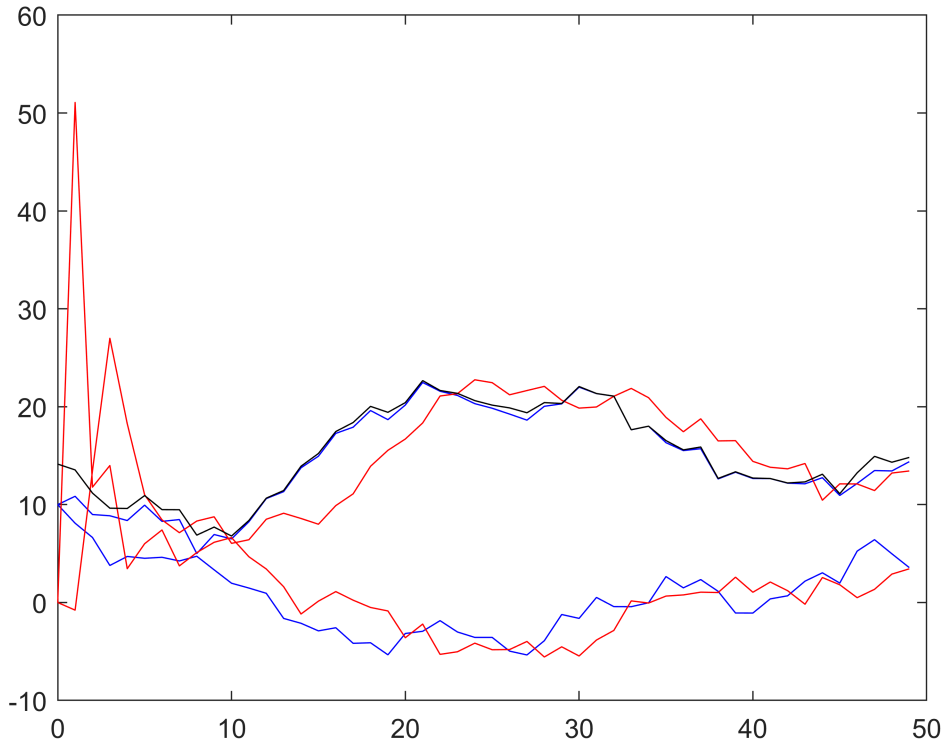
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{r}_k$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$$

```
for k = 2:N
   X_hat(:,k) = F * X_hat(:,k-1);
   P = F * P * F' + Q;

   r = Z(:,k) - H * X_hat(:,k);

   S = H * P * H' + R;
   K = P * H' * inv(S);
   X_hat(:,k) = X_hat(:,k) + K * r;
   P = P - K * H * P;

   plot(X_hat(1,k),X_hat(3,k),'rx')
   %pause(0.01)
end
```




```
% Plot the real and estimated velocity coordinates
figure(2)
hold on; box on
plot(t,X(2,:),'b')
plot(t,X(4,:),'b')
plot(t,X_hat(2,:),'r')
plot(t,X_hat(4,:),'r')
v = sqrt(X(2,:).^2+X(4,:).^2);
plot(t,v,'k-')
```



Evaluate the performance

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N} e_k^2}$$

Error in position

$$e_k = \sqrt{(x_k - \hat{x}_k)^2 + (y_k - \hat{y}_k)^2}$$

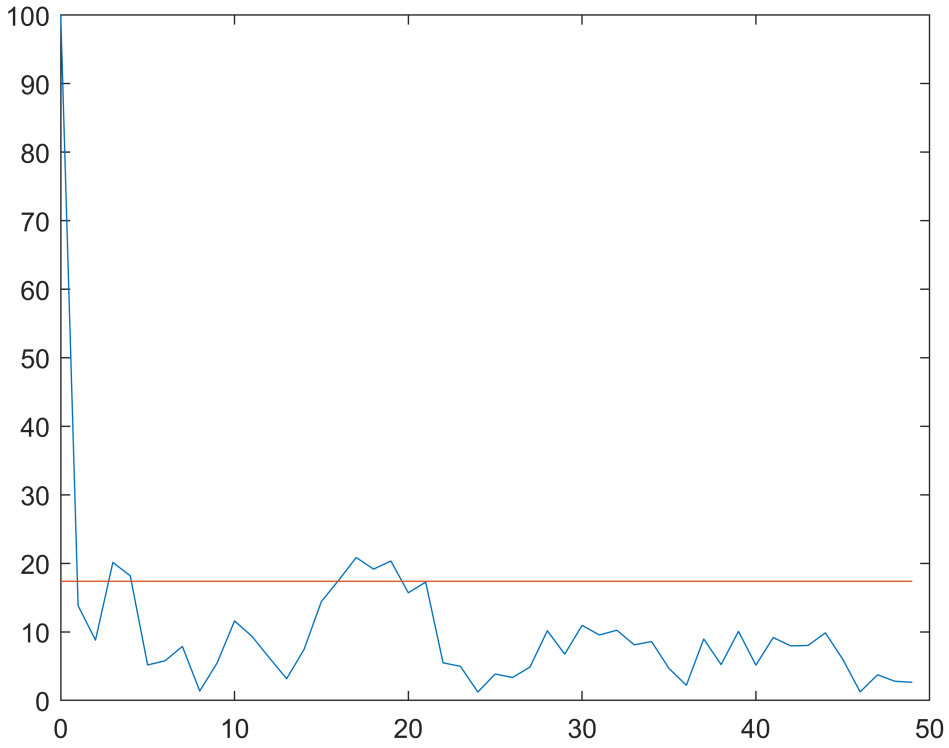
```
e_pos = sqrt((X_hat(1,:) - X(1,:)).^2 + (X_hat(3,:) - X(3,:)).^2);
e_vel = sqrt((X_hat(2,:) - X(2,:)).^2 + (X_hat(4,:) - X(4,:)).^2);

rms_e_pos = rms(e_pos)
```

```
rms_e_pos = 17.3832
```

```
rms_e_vel = rms(e_vel)
```

```
rms_e_vel = 8.3338
```

```
figure(3)
hold on; box on
plot(t,e_pos)
plot(t,repmat(rms_e_pos,N))
```




```
norm(Q)*norm(R)/(norm(Q)+norm(R))
```

```
ans = 2.4390
```

```
% these are the same
%sqrt(mean(e.^2))
%sqrt(e*e'/N)

% rms of the signal and its fft is the same
% divide by N (see definition in help)
E_pos = fft(e_pos)/N;
rms_E_pos = rms(E_pos)
```

```
rms_E_pos = 2.4584
```

```
norm_E_pos = norm(E_pos)
```

```
norm_E_pos = 17.3832
```

```
sqrt(sum(E_pos*E_pos'))
```

```
ans = 17.3832
```

```
sqrt(fft(e_pos)*fft(e_pos)')/N
```

```
ans = 17.3832
```

```


%% Alternatve state vector x = [x, y, theta, v]
```