State Estimation with a Kalman Filter

- When I drive into a tunnel, my GPS continues to show me moving forward, even though it isn't getting any new position sensing data
  - How does it work?
- A Kalman filter produces estimate of system's next state, given
  - noisy sensor data
  - control commands with uncertain effects
  - model of system's (possibly stochastic) dynamics
  - estimate of system's current state
- In our case, given
  - a blimp with (approximately known) dynamics
  - noisy sensor data
  - control commands
  - our current estimate of blimp's state
- How should we predict the blimp's next state?

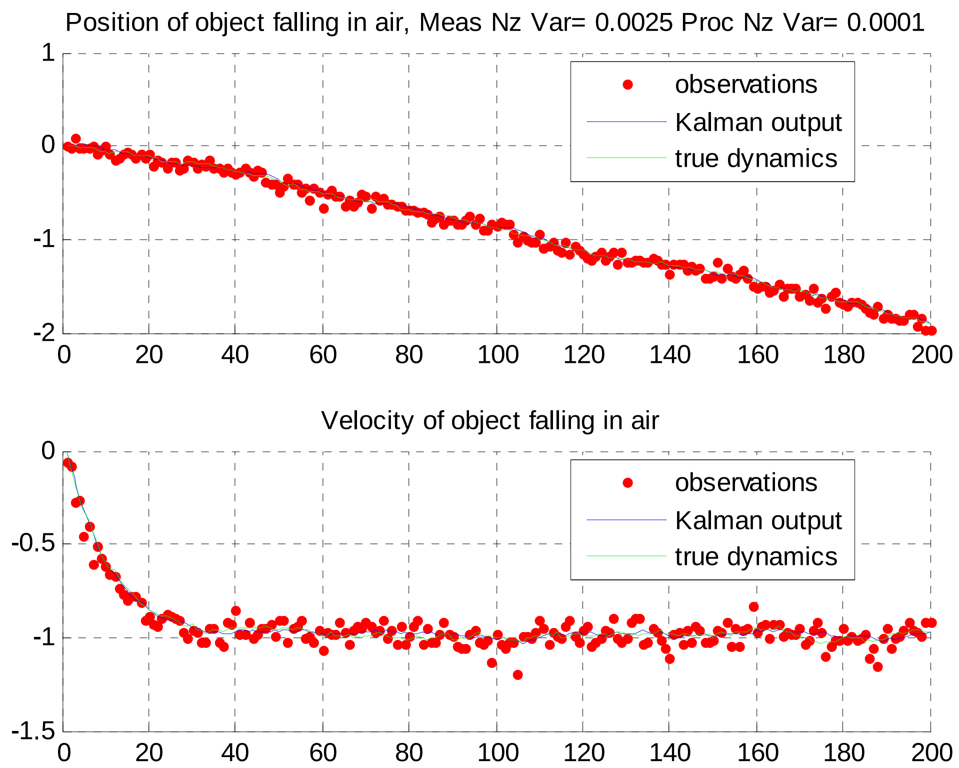
  ➔ How should we control blimp?

Kalman Filter

- Bayesian estimator, computes beliefs about state, assuming everything is linear and Gaussian
  - Gaussian is unimodal ➔ only one hypothesis
  - Example of a Bayes filter
- "Recursive filter," since current state depends on previous state, which depends on state before that, and so on
- Two phases: prediction (not modified by data) and correction (data dependent)
- Produces estimate with minimum mean-squared error
- Even though the current estimate only looks at the previous estimate, as long as the actual data matches the assumptions (Gaussian etc), you can't do any better, even if you looked at all the data in batch!
- Very practical and relatively easy to use technique!

Example

- Object falling in air
- We know the dynamics
  - Related to blimp dynamics, since drag and inertial forces are both significant
  - Dynamics same as driving blimp forward with const fan speed
- We get noisy measurements of the state (position and velocity)
- We will see how to use a Kalman filter to track it



Linear 1D Newtonian dynamics example

Object falling in air

State is (x, v)

$$\text{where } v = \frac{dx}{dt}$$

Linear 1D Newtonian dynamics example

Object falling in air

$$f = ma = m\frac{dv}{dt}$$

$$f_a = -kv \text{ force due to drag}$$

$$\text{(ideally we'd use } v^2 \text{ instead of } v\text{)}$$

$$f_g = -gm$$

$$f = f_a + f_g = -kv - mg$$

$$m\frac{dv}{dt} = -kv - mg$$

$$\frac{dv}{dt} = -\frac{kv}{m} - g$$

Confused by the signs?
If obj is falling down, v is a negative #

g (the gravitational const.) is
a pos. number, but the direction
of gravity is down, so the gravitational force
is -mg (a neg. number).
The frictional force –kv is a positive
number (since v is negative).
Even though we wrote –kv and –mg, these
two forces are in opposite directions when
the object is falling

How to solve the differential equation on the computer?

$$\frac{dx}{dt} = v$$

$$\frac{\Delta x}{\Delta t} = v$$

$$\frac{x_t - x_{t-1}}{\Delta t} = v$$

$$x_t = x_{t-1} + v\Delta t$$

$$\frac{dv}{dt} = -\frac{kv}{m} - g$$

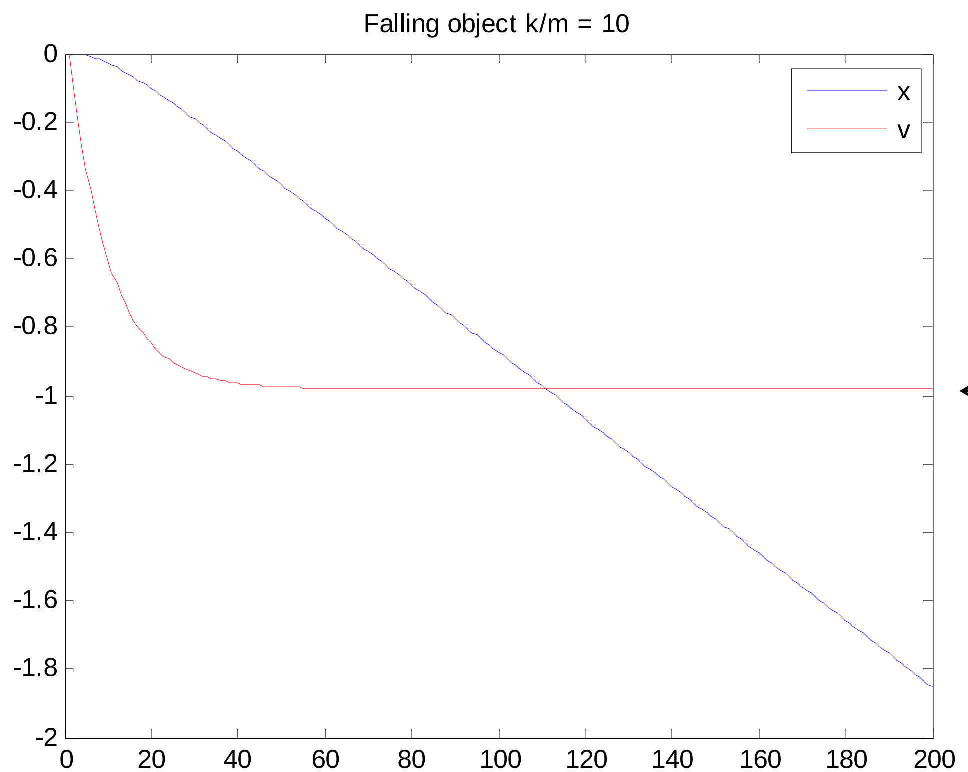
$$\frac{\Delta v}{\Delta t} = -\frac{kv}{m} - g$$

$$\frac{v_t - v_{t-1}}{\Delta t} = -\frac{kv}{m} - g$$

$$v_t - v_{t-1} = \left(-\frac{kv}{m} - g \right) \Delta t$$

$$v_t = v_{t-1} - \left(\frac{kv}{m} + g \right) \Delta t$$

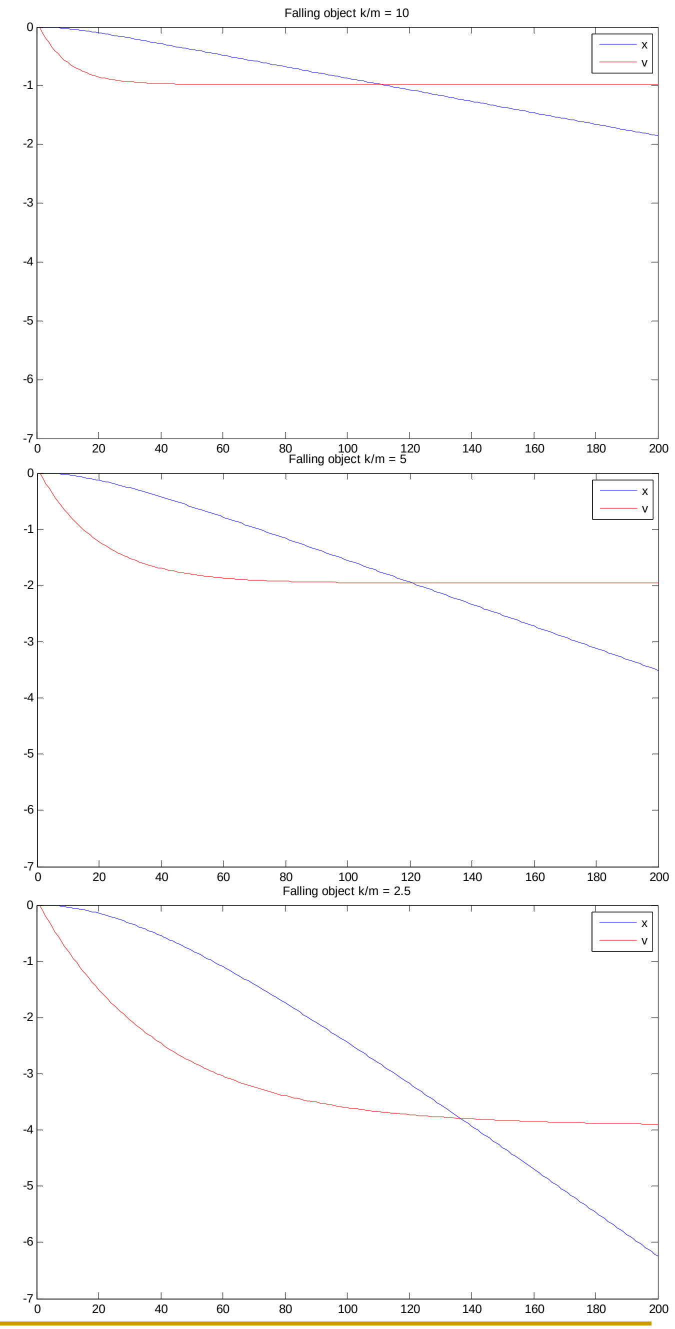
Object falling in air



At terminal velocity

Produced by iterating the difference equations on previous slide

In air, heavier objects do fall faster!



And they take longer to reach their terminal velocity

(Without air, all objects accelerate at the same rate)

Matlab code for modeling object falling in air

```
% falling.m
x0 = 0.0;
v0 = 0.0;
TMAX = 200;
x = zeros(1,TMAX);
V = zeros(1,TMAX);
g=9.8;
m=1.0;
k=10.0;
x(1) = x0;
v(1) = v0;
dt=0.01;
for t=2:TMAX,
    x(t) = x(t-1)+(v(t-1))*dt;
    v(t) = v(t-1)+(-(k/m)*(v(t-1))-g)*dt;
end
figure();
plot(x,'b'); hold on;
title(['Falling object k/m = ' num2str(k/m)]);
plot(v,'r')
legend('x','v'); hold off
```

Multi-dimensional Gaussians

$$P(x \mid m, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-(x-m)^2 / 2\sigma^2]$$

One-dimensional (scalar) Gaussian

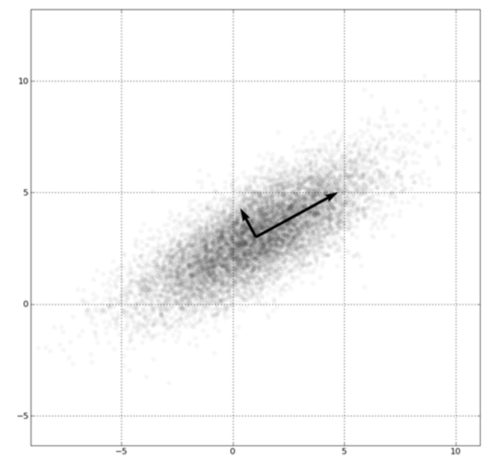
$$P(\mathbf{x} \mid \mathbf{m}, \mathbf{R}) = \frac{1}{Z(\mathbf{R})} \exp[-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T \mathbf{R}(\mathbf{x}-\mathbf{m})]$$

Vector Gaussian

where

$$Z(\mathbf{R}) = \left(\det(\mathbf{R} / 2\pi)\right)^{-1/2}$$

and **R** is the inverse of the covariance matrix.



Multi-dimensional Gaussians, Covariance Matrices, Ellipses, and all that

In an N dimensional space $\mathbf{x}\mathbf{x}^T = R^2$ is a sphere of radius R

$$\text{Note that } \mathbf{x}\mathbf{x}^T = \langle \mathbf{x} \bullet \mathbf{x} \rangle = x_1^2 + x_2^2 + \ldots + x_N^2 = R^2$$

Can write it more generally by inserting identity matrix

$$\mathbf{x}\mathbf{x}^T = \mathbf{x}\mathbf{I}\mathbf{x}^T = R^2$$

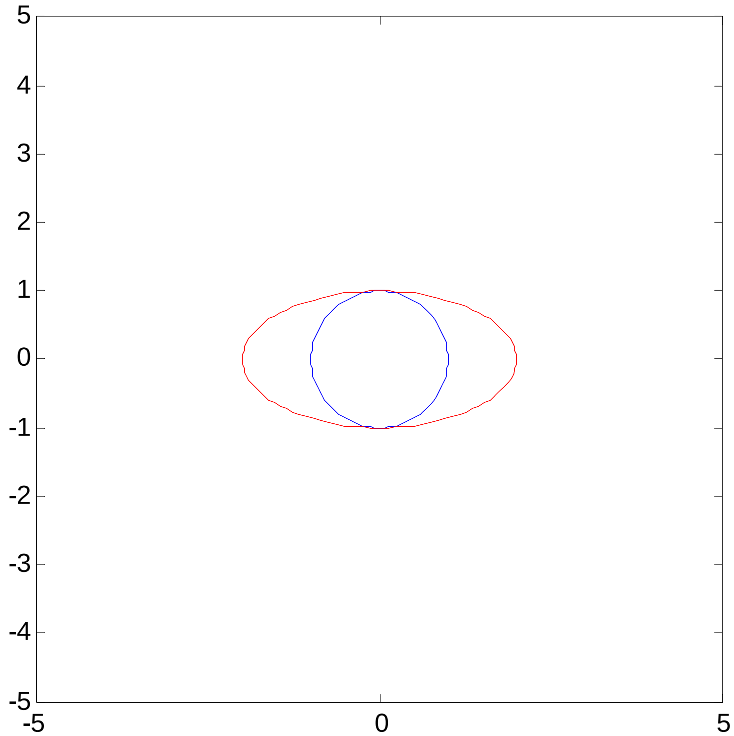
If we replace $\mathbf{I}$ by a more general matrix $\mathbf{M}$, it will distort the sphere: for $\mathbf{M}$ diagonal, it will scale each axis differently, producing an axis-aligned ellipsoid

We could also apply rotation matrices to a diagonal $\mathbf{M}$ to produce a general, non-axis aligned ellipsoid

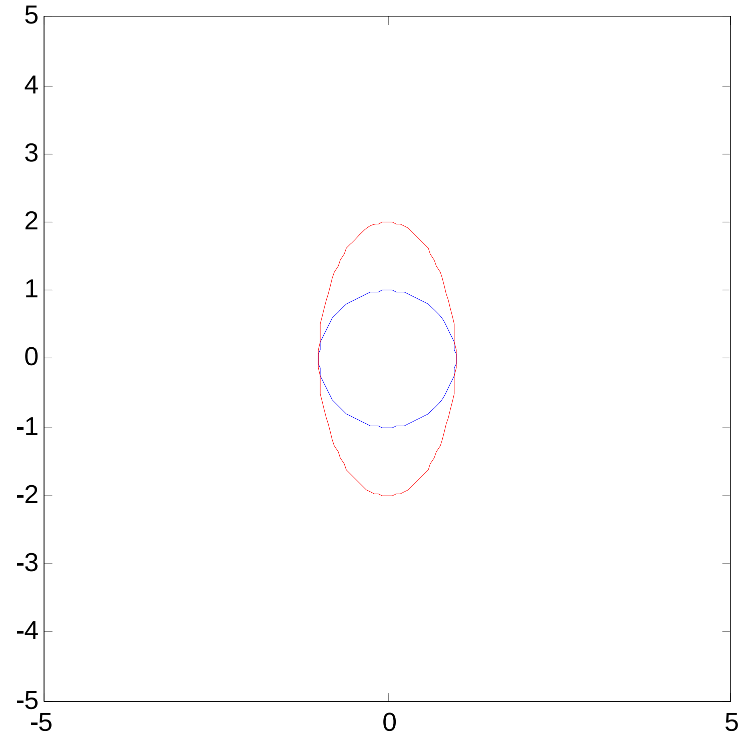
➔ The uncertainty “ball” of a multi-D Gaussian [e.g. the 1 std iso-surface] actually IS an ellipsoid!

Example

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$



$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

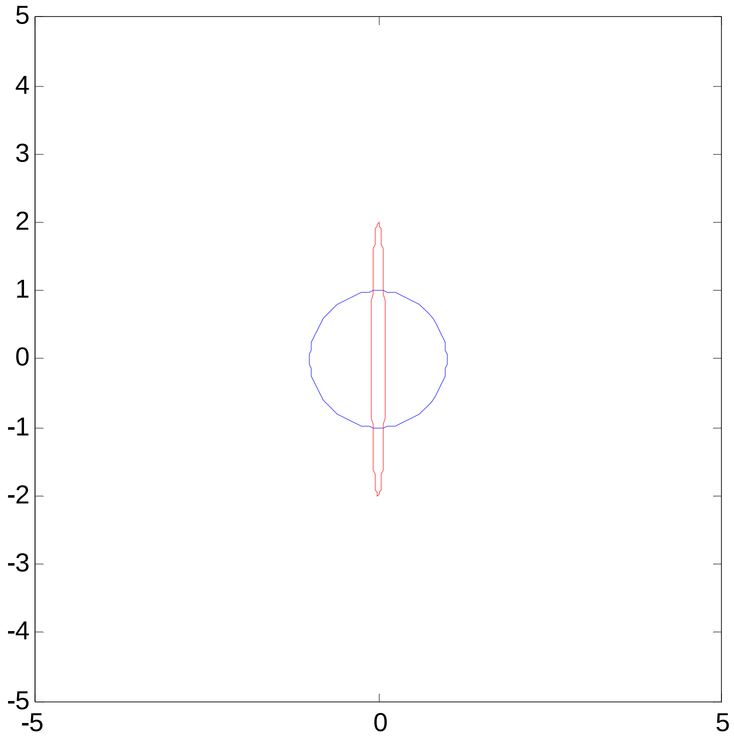


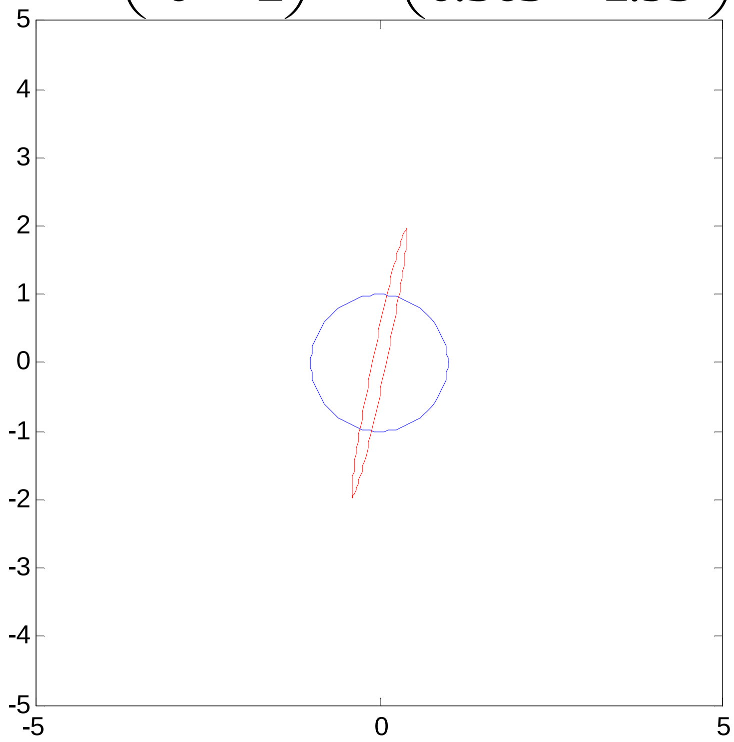
Blue: circle of radius 1 (e.g. 1 std iso-surface of uncorrelated uniform Gaussian noise)
Red: circle after transformation by A

With $R = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$ and $\theta = -\frac{\pi}{16}$,

$$A = \begin{pmatrix} 0.1 & 0 \\ 0 & 2 \end{pmatrix}$$

$$A = R^T \begin{pmatrix} 0.1 & 0 \\ 0 & 2 \end{pmatrix} R = \begin{pmatrix} 0.172 & 0.363 \\ 0.363 & 1.93 \end{pmatrix}$$





Kalman filter variables

x: state vector

z: observation vector

u: control vector

A: state transition matrix --- dynamics

B: input matrix (maps control commands onto state changes)

P: covariance of state vector estimate

Q: process noise covariance

R: measurement noise covariance

H: observation matrix

Kalman filter algorithm

Prediction for state vector and covariance:

$$\overline{x} = Ax + Bu$$

$$\overline{P} = APA^T + Q$$

Kalman gain factor:

$$K = \overline{P}H^T(H\overline{P}H^T + R)^{-1}$$

Correction based on observation:

$$x = \overline{x} + K(z - H\overline{x})$$

$$P = \overline{P} - KH\overline{P}$$

x: state vector

z: observation vector

u: control vector

A: state transition matrix --- dynamics

B: control commands --> state changes

P: covariance of state vector estimate

Q: process noise covariance

R: measurement noise covariance

H: observation matrix

Need dynamics in matrix form

$$x_t = x_{t-1} + v_{t-1}dt;$$

$$v_t = v_{t-1} - \left(\tfrac{k}{m} v_{t-1} + g\right)dt;$$

Want **A** s.t.

$$\begin{pmatrix} x_t \\ v_t \end{pmatrix} = \mathbf{A} \begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix}$$

Try

$$\mathbf{A} \begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 - \tfrac{k}{m}\Delta t \end{pmatrix} \begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} = \begin{pmatrix} x_{t-1} + v_{t-1}\Delta t \\ v_{t-1} - \tfrac{k}{m} v_{t-1}\Delta t \end{pmatrix} = \begin{pmatrix} x_t \\ v_t \end{pmatrix}$$

Hmm, close but gravity is missing…we can't put it into A because it will be multiplied by v_t. Let's stick it into B!

Need dynamics in matrix form

Want

$$\begin{aligned} x_t &= x_{t-1} + v_{t-1}\text{dt}; \\ v_t &= v_{t-1} - \left(\tfrac{k}{m} v_{t-1} + g\right)\text{dt}; \end{aligned}$$

$$\text{Try } Bu = \begin{pmatrix} 1 & 0 \\ 0 & -g\Delta t \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -g\Delta t \end{pmatrix}$$

$$\begin{pmatrix} x_t \\ v_t \end{pmatrix} = \mathbf{A}\begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} + B\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \Delta t \\ 0 & 1 - \frac{k}{m}\Delta t \end{pmatrix}\begin{pmatrix} x_{t-1} \\ v_{t-1} \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -g\Delta t \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{t-1} + v_{t-1}\Delta t \\ v_{t-1} - (\frac{k}{m} v_{t-1} + g)\Delta t \end{pmatrix}$$

The command to turn on gravity!

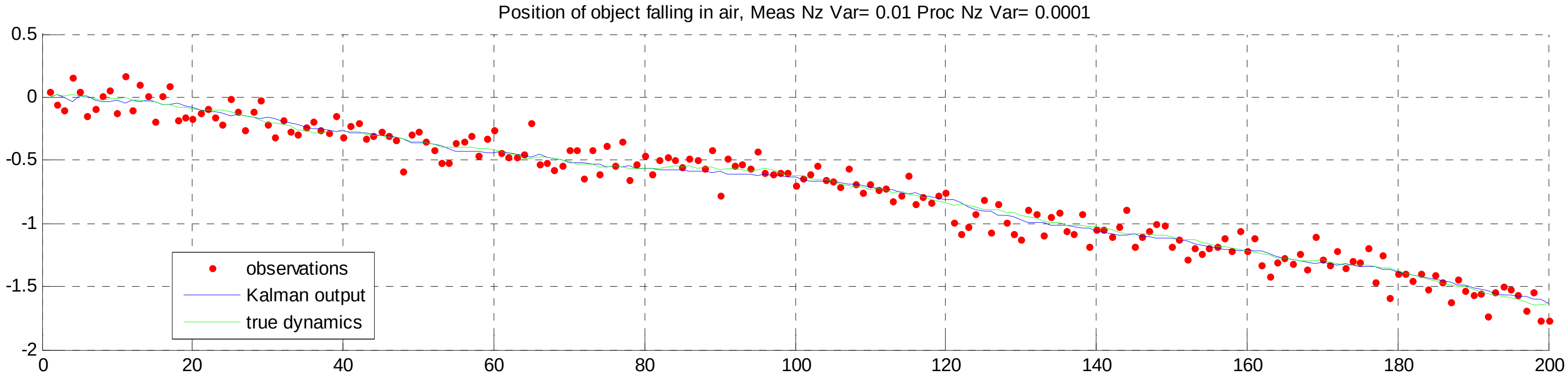
Now gravity is in there…we treated it as a control input.

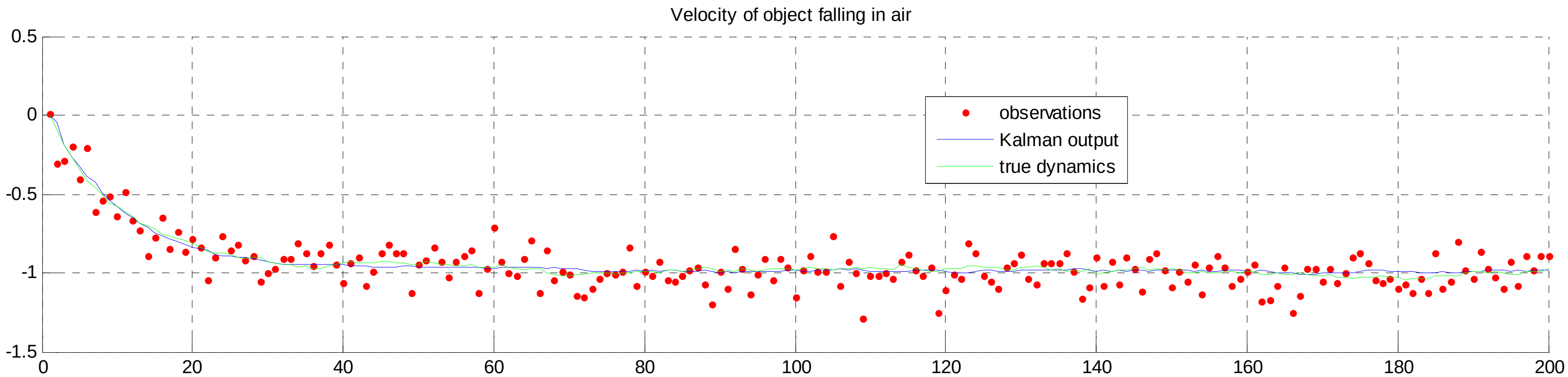
Matlab for dynamics in matrix form

```
% falling_matrix.m: model of object falling in air, w/ matrix notation
x0 = 0.0; v0 = 0.0;
TMAX = 200;
x = zeros(2,TMAX);
g=9.8;
m=1.0;
k=10.0;
x(1,1) = x0; x(2,1) = v0;
dt=0.01;
u=[0 1]';
for t=2:TMAX,
    A=[[1      dt         ]; ...
       [0 (1.0-(k/m)*dt) ]];
    B=[[1    0   ]; ...
       [0  -g*dt ]];
    x(:,t) = A*x(:,t-1) + B*u;
end
% plotting
```

Matlab for Kalman Filter

```
function s = kalmanf(s)
 s.x = s.A*s.x + s.B*s.u;
   s.P = s.A * s.P * s.A' + s.Q;
   % Compute Kalman gain factor:
   K = s.P * s.H' * inv(s.H * s.P * s.H' + s.R);
   % Correction based on observation:
   s.x = s.x + K*(s.z - s.H *s.x);
   s.P = s.P - K*s.H*s.P;
end
return
```


Position of object falling in air, Meas Nz Var= 0.01 Proc Nz Var= 0.0001
0.5
0
-0.5
-1
-1.5
-2
0
20
40
60
80
100
120
140
160
180
200
observations
Kalman output
true dynamics


Velocity of object falling in air
0.5
0
-0.5
-1
-1.5
0
20
40
60
80
100
120
140
160
180
200
observations
Kalman output
true dynamics

Calling the Kalman Filter (init)

```
x0 = 0.0; v0 = 0.0;
TMAX = 200;
g=9.8;
m=1.0; k=10.0;
dt=0.01;

clear s % Dynamics modeled by A
s.A = [[1      dt       ]; ...
       [0 (1.0-(k/m)*dt)]];
```

Calling the Kalman Filter (init)

```
% Measurement noise variance
MNstd = 0.4;
MNV = MNstd*MNstd;
% Process noise variance
PNstd = 0.02;
PNV = PNstd*PNstd;
% Process noise covariance matrix
s.Q = eye(2)*PNV;
% Define measurement function to return the state
s.H = eye(2);
% Define a measurement error
s.R = eye(2)*MNV; % variance
```

Calling the Kalman Filter (init)

```
% Use control to include gravity
s.B = eye(2); % Control matrix
s.u = [0 -g*m*dt]'; % Gravitational acceleration
% Initial state:
s.x = [x0 v0]';
s.P = eye(2)*MNV;
s.detP = det(s.P); % Let's keep track of the noise by
keeping detP
s.z = zeros(2,1);
```

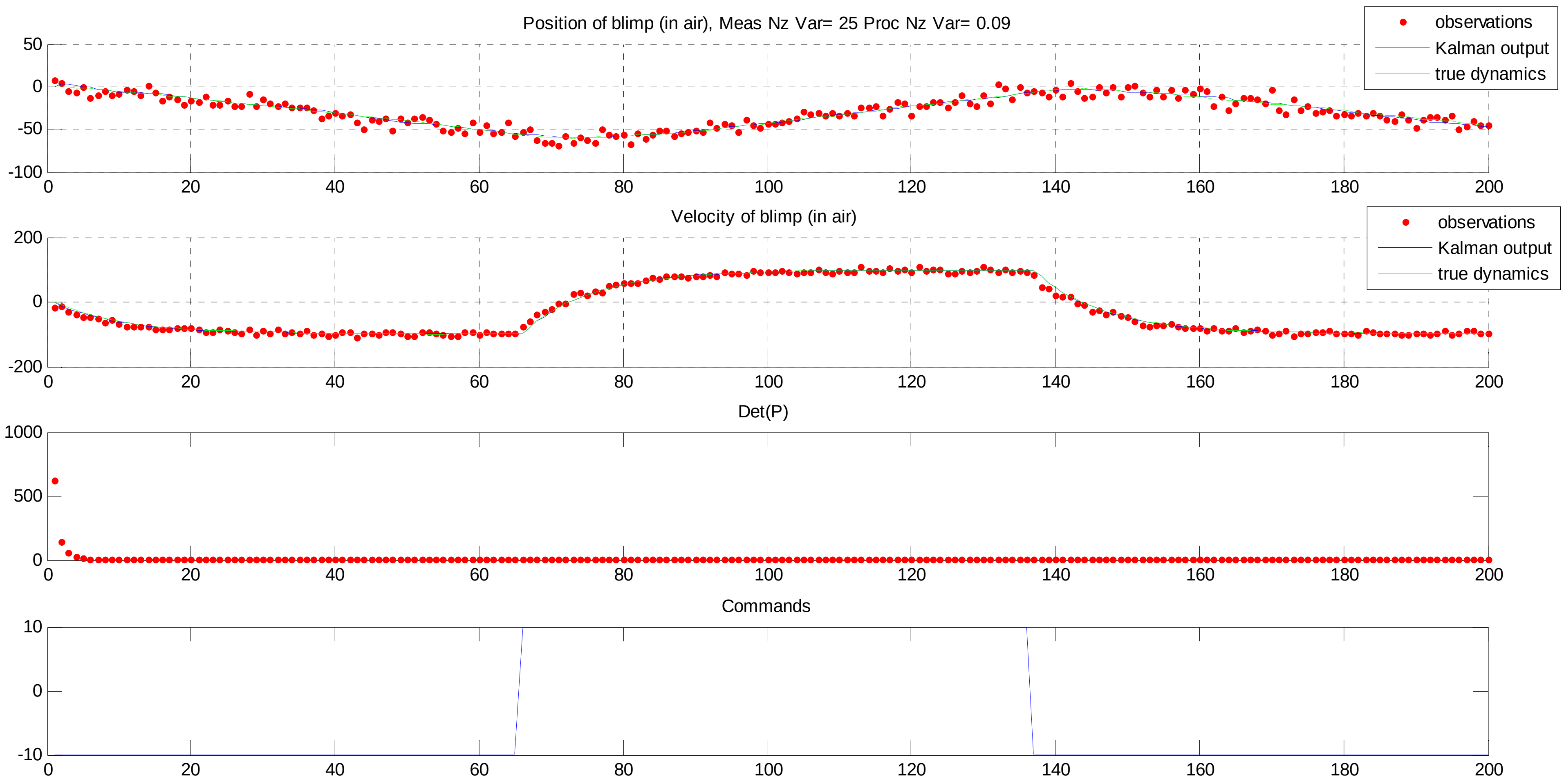
Calling the Kalman Filter

```
% Simulate falling in air, and watch the filter track it
tru=zeros(TMAX,2); % true dynamics
tru(1,:)=[x0 v0];
detP(1,:)=s.detP;
for t=2:TMAX
   tru(t,:)=s(t-1).A*tru(t-1,:)'+ s(t-1).B*s(t-1).u+PNstd *randn(2,1);
   s(t-1).z = s(t-1).H * tru(t,:)' + MNstd*randn(2,1); % create a meas.
   s(t)=kalmanf(s(t-1)); % perform a Kalman filter iteration
   detP(t)=s(t).detP; % keep track of "net" uncertainty
end
```

The variable s is an object whose members are all the important data structures (A, x, B, u, H, z, etc)

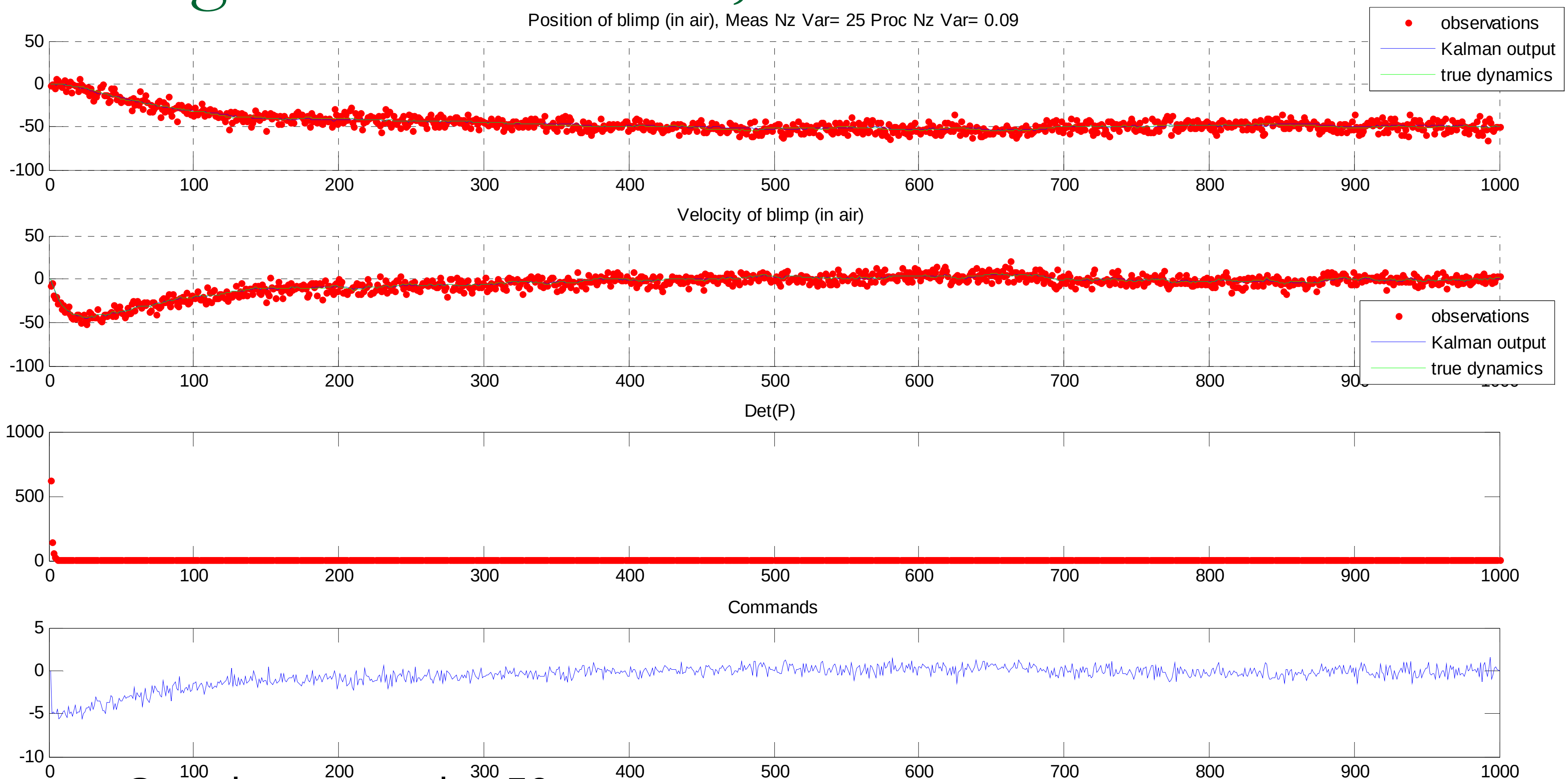
tru: simulation of true dynamics. In the real world, this would be implemented by the actualy physical system

Blimp estimation example



Here we are assuming we can switch gravity back and forth (from -10 to +10 and back)

Blimp P control
using raw sensor data, no Kalman filter

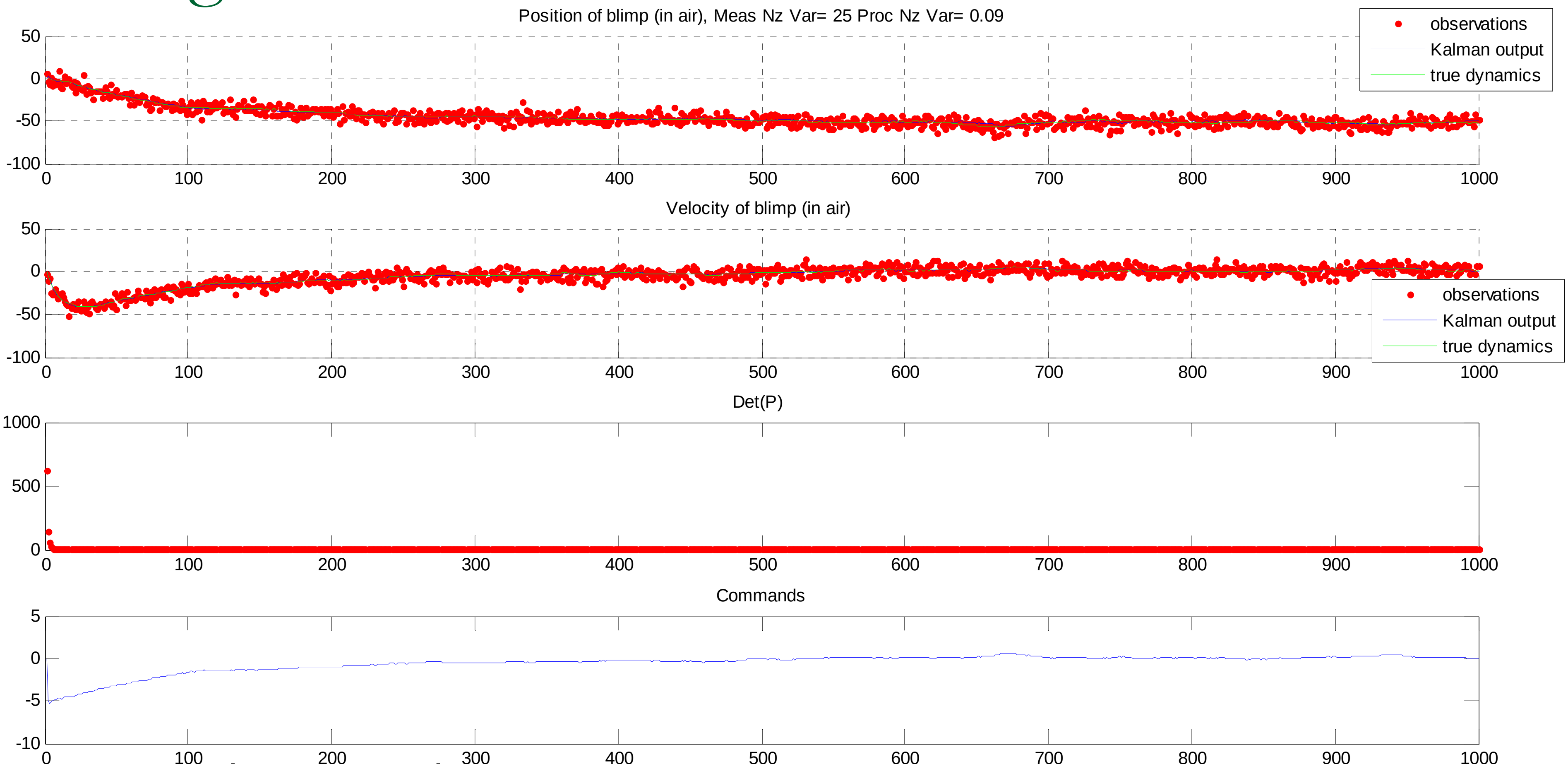


Servoing to setpoint -50

Mean Squared Error: 0.143

RMS: 0.379

Blimp P control using Kalman-filtered state estimate

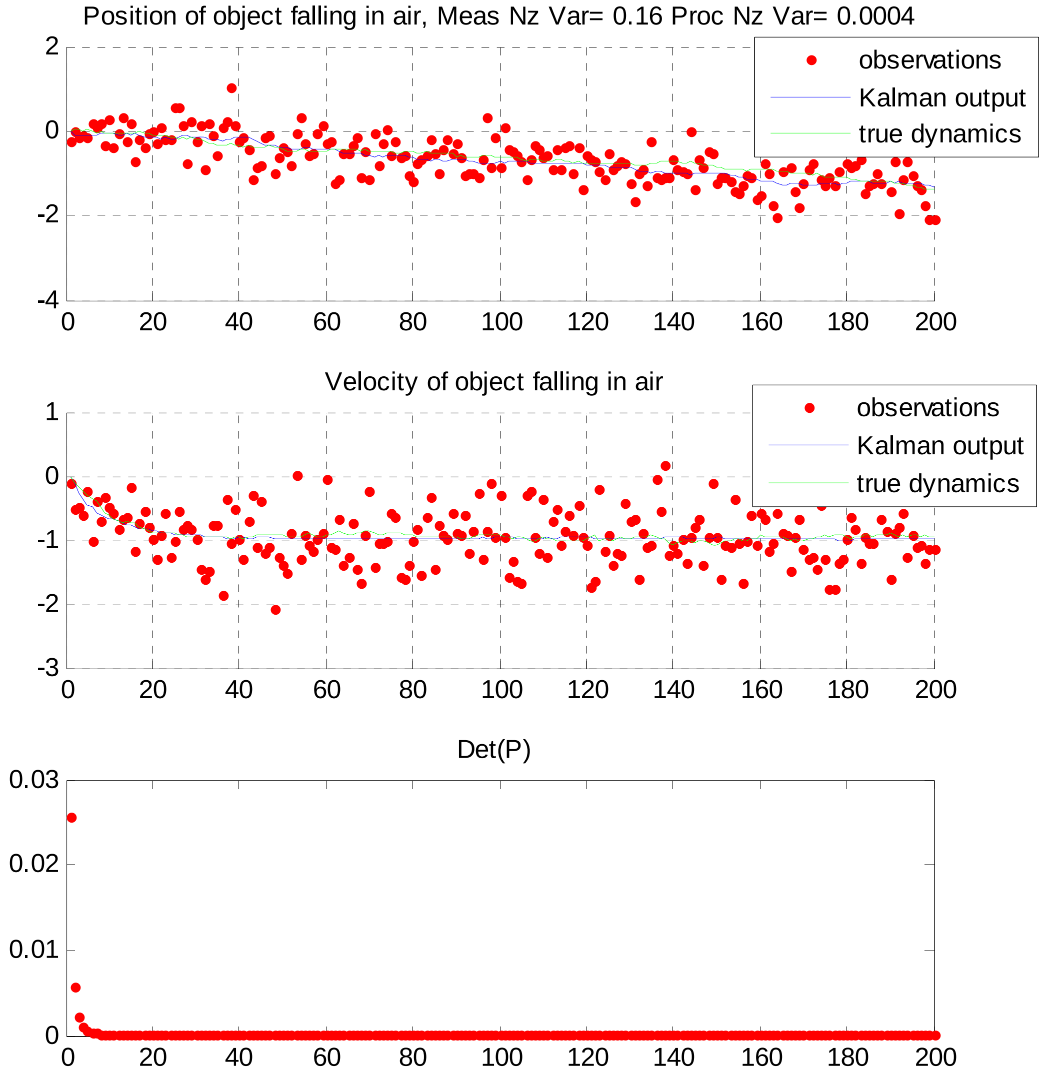


Servoing to setpoint -50
Mean Squared Error: 0.0373
RMS: 0.193

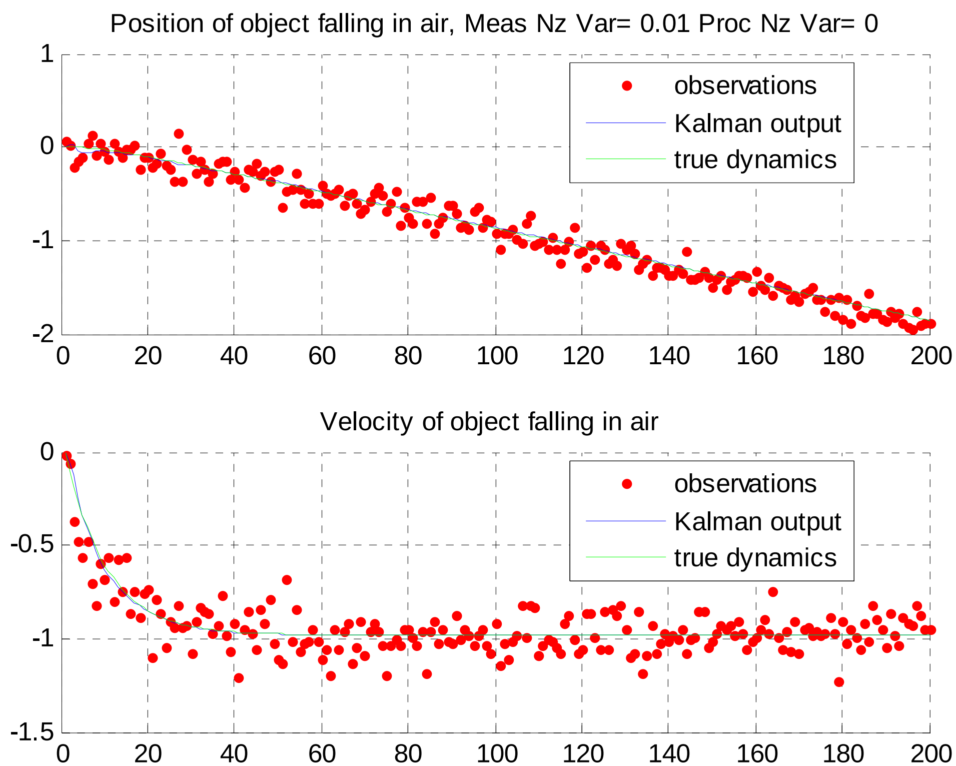
Extensions

- Extended Kalman Filter (EKF)
- Information Filter
- Unscented Kalman Filter (UKF)…the unscented Kalman Filter does not stink!

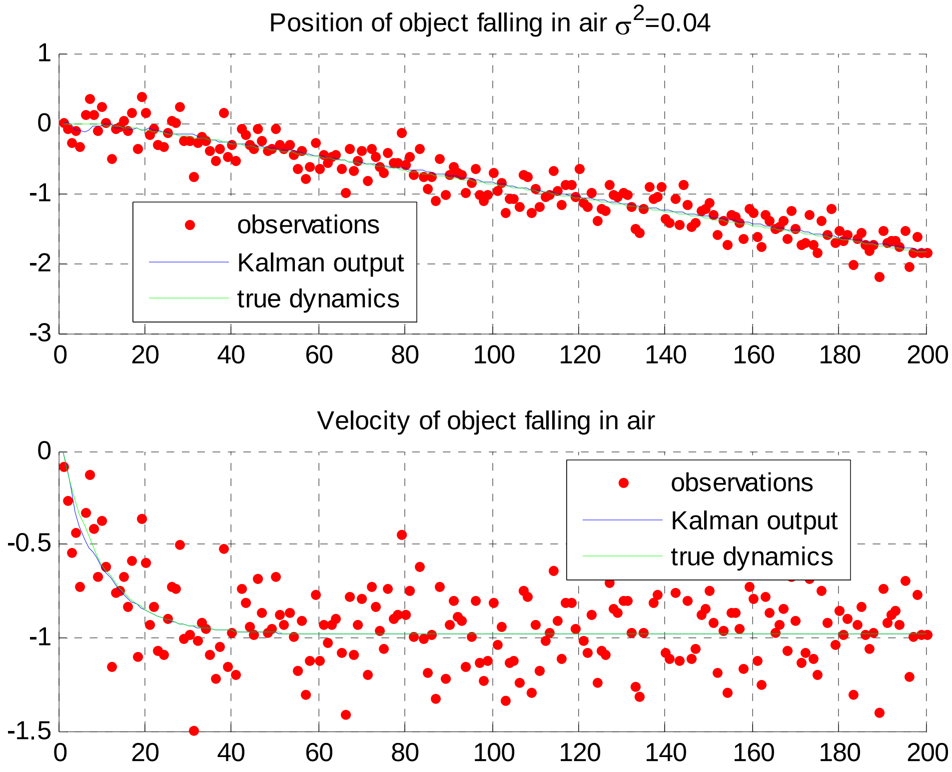
Extra examples…various noise settings



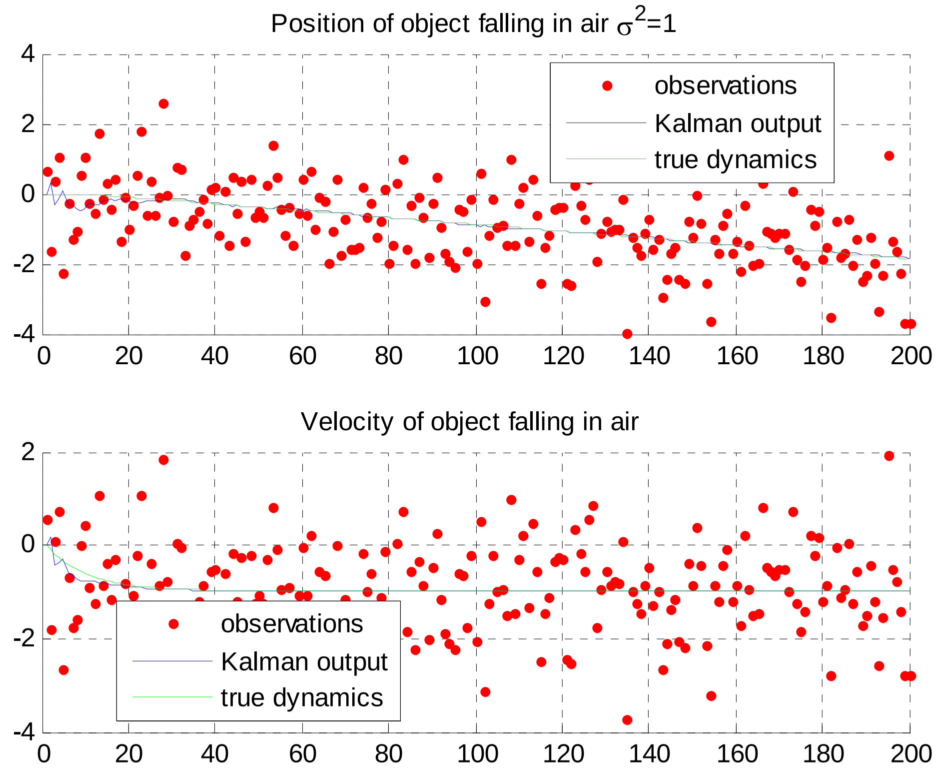
Position of object falling in air, Meas Nz Var= 0.16 Proc Nz Var= 0.0004
observations
Kalman output
true dynamics
Velocity of object falling in air
observations
Kalman output
true dynamics
Det(P)



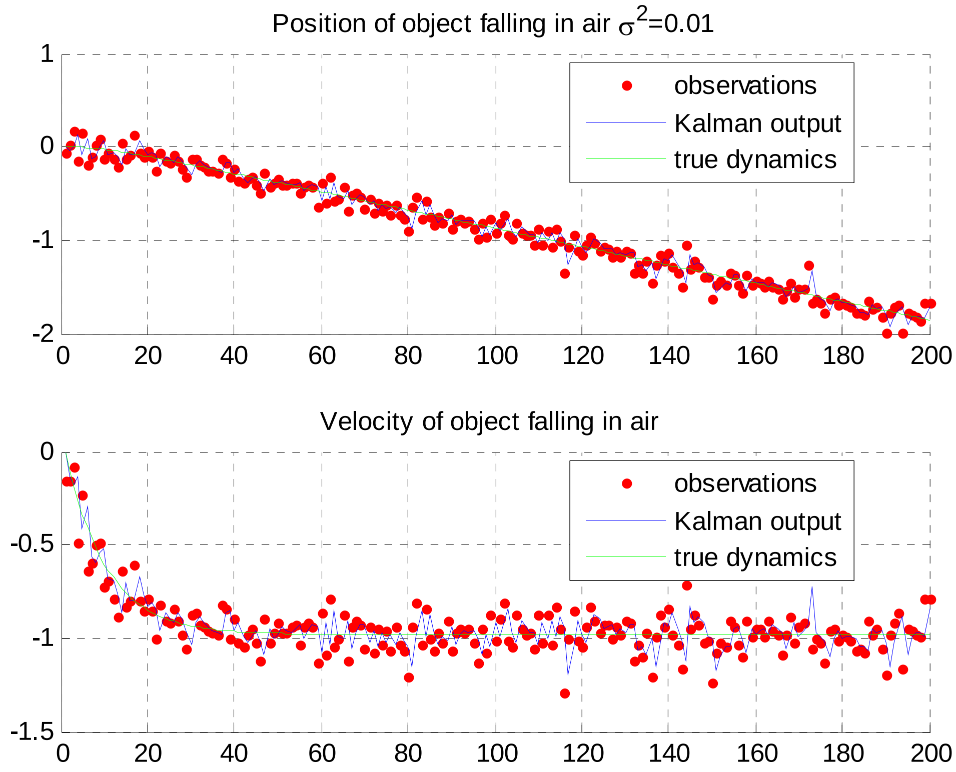
Process noise Q = 0.0



Process noise Q = 0.0



Process noise Q = 0.0



Process noise Q = 0.02