Delft University of Technology

Delft Center for Systems and Control

Overleaf template

Subtitle

Author

Jesper Kreuk 4445015

August 31, 2020

Contents

1 Writing text

1.1 General remarks

Note: the table of contents is automatically generated.
The numbering and references of things like equations and citations are also automatically adjusted.

1.2 How to make use of automatic referencing

Referencing things like figures, equations or sections can be done with \ref{}, this is often done in section 3.
If you want to give credit to our an author use \cite{} [1].
Watch our tips and tricks video on the DSA Kalman website to find out how you can find the sources in BibTeX format easily.

1.3 Making lists

Here is a list of items, the spacing is adjusted for compact notation:

- item 1
- item 2

1.4 Creating tables

It is possible to create tables, as shown in Table 1.

	col1	col2
row1	entry 1	entry 2
row2	entry 3	entry 4

Table 1: Example table

$\sin(x) + \cos(x) + \tan(x)$

2 Figures

LaTeX allows the user to input figures, in the code you can find how to input a figure, 2 figures side by side and rotate figures.

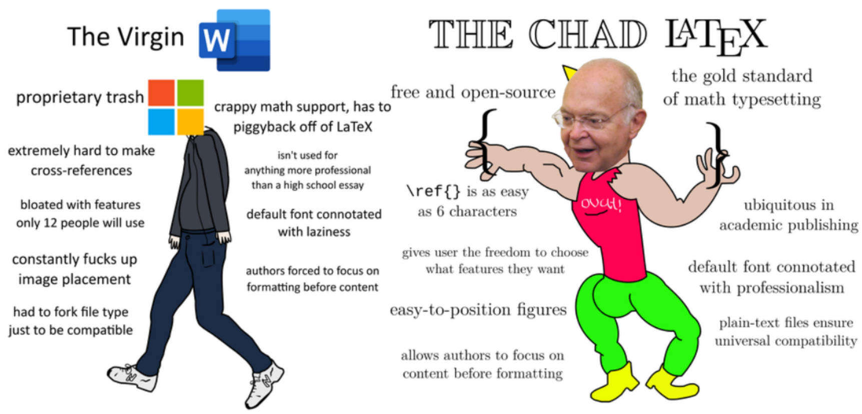

Figure 2: Single figure showing the difference between Word and LaTeX

(a) Picture 1 (b) Picture 2

Figure 3: 2 figures side by side.

Overleaf supports many image formats, the most useful one for us will be the eps file format which is infinitely sharp. These images can be created by MATLAB, an example is shown in Figure 4.

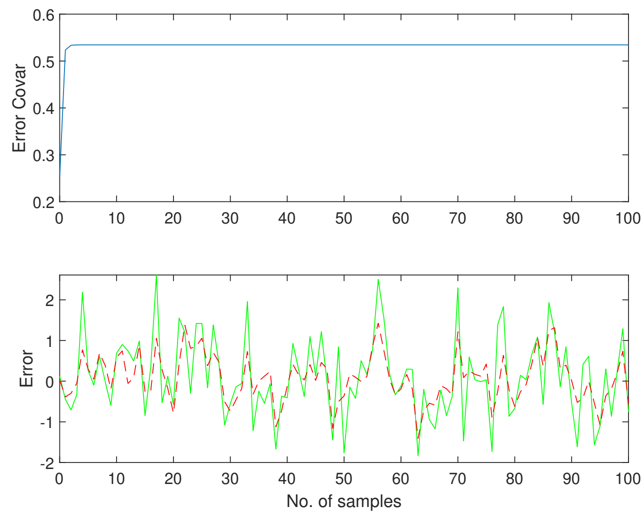

Figure 4: This figure is infinitely sharp due to its vector format.

3 Equations

You can write in math mode in between two $ signs, like this: $y = c_1 \sin(\omega t) + c_2 \cos(\omega t)$.

Equation 1 and 2 both have separate labels and are aligned using the & operator.

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \tag{1}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2}$$

Equation 3 has only 1 label for 2 equations.

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k \hat{x}_{k-1|k-1} + B_k u_k \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k \end{aligned} \tag{3}$$

Use * to disable the equation labels, this works for many numerating commands (e.g. sections) as well.

$$\sigma = \varepsilon \frac{\partial f(x, \alpha, \beta, \gamma)}{\partial x} \quad \forall\ x \in \mathbb{R}$$

Cases can be made as shown in Equation 4. Note that written text can stand upright with the *text* command.

$$y = \begin{cases} ax + b_1 - \sin(x) & \text{if } x \leq 0 \\ x^2 + b_2 & \text{if } 0 < x \leq 2 \\ (1-x)x^3 + b_3 & \text{if } x > 2 \end{cases} \tag{4}$$

You can use matrices in all kinds of equations as well and fill in whatever you like. Multiple bracket types are possible.

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \neq \begin{bmatrix} \sin(x) & x^2 & x_3 \\ \dot{a} & \bar{b} & \hat{c} \end{bmatrix} \tag{5}$$

When writing repeating matrices you can use dots inside the matrix.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \tag{6}$$

It is also possible to use brackets in equations, as shown in Equation 7.

$$y = \min_x \int_0^2 \left(\underbrace{||(x+2)^3||_2}_{\text{part 1}} + \overbrace{\cos x^2}^{\text{part 2}} \right) dx \tag{7}$$

References

[1] Rudolf Kalman. "On the general theory of control systems". In: *IRE Transactions on Automatic Control* 4.3 (1959), pp. 110–110.

Listing 1: `Kalman filter example code`

```
%% Kalman Filter Design
% This example shows how to perform Kalman filtering. Both a steady state
% filter and a time varying filter are designed and simulated below.

%   Copyright 1986-2012 The MathWorks, Inc.

%%  Problem Description
% Given the following discrete plant
%
% $$ x(n+1) = Ax(n) + Bu(n) + Bw(n) $$
%
% $$ y(n) = Cx(n) + Du(n) $$
%
% where
A = [1.1269   -0.4940    0.1129,
     1.0000         0         0,
          0    1.0000         0];

B = [-0.3832
      0.5919
      0.5191];

C = [1 0 0];

```

```
D = 0;

%%
% design a Kalman filter to estimate the output y based on the
% noisy measurements  yv[n] = C x[n] + v[n]
%

%%  Steady-State Kalman Filter Design
% You can use the function KALMAN to design a steady-state Kalman filter.
% This function determines the optimal
% steady-state filter gain M based on the process noise
% covariance Q and the sensor noise covariance R.
%
% First specify the plant + noise model.
% CAUTION: set the sample time to -1 to mark the plant as discrete.

Plant = ss(A,[B B],C,0,-1,'inputname',{'u' 'w'},'outputname','y');

%%
% Specify the process noise covariance (Q):

Q = 2.3; % A number greater than zero

%%
% Specify the sensor noise covariance (R):

R = 1; % A number greater than zero

%%
% Now design the steady-state Kalman filter with the equations
%
%     Time update:         x[n+1|n] = Ax[n|n-1] + Bu[n] + Bw[n]
%
%     Measurement update:  x[n|n] = x[n|n-1] + M (yv[n] - Cx[n|n-1])
%
%                          where M = optimal innovation gain
% using the KALMAN command:
[kalmf,L,~,M,Z] = kalman(Plant,Q,R);


%%
% The first output of the Kalman filter KALMF is the plant
% output estimate  y_e = Cx[n|n], and the remaining outputs
% are the state estimates. Keep only the first output y_e:

kalmf = kalmf(1,:);

M,   % innovation gain

%%
% To see how this filter works, generate some data and
% compare the filtered response with the true plant response:
%
% <<../kalmdemofigures_01.png>>


%%
% To simulate the system above, you can generate the response of
% each part separately or generate both together.  To
% simulate each separately, first use LSIM with the plant
% and then with the filter. The following example simulates both together.

% First, build a complete plant model with u,w,v as inputs and
% y and yv as outputs:
a = A;
b = [B B 0*B];
c = [C;C];
d = [0 0 0;0 0 1];
```

```
P = ss(a,b,c,d,-1,'inputname',{'u' 'w' 'v'},'outputname',{'y' 'yv'});

%%
% Next, connect the plant model and the Kalman filter in parallel
% by specifying u as a shared input:
sys = parallel(P,kalmf,1,1,[],[]);

%%
% Finally, connect the plant output yv to the filter input yv.
% Note:  yv is the 4th input of SYS and also its 2nd output:
SimModel = feedback(sys,1,4,2,1);
SimModel = SimModel([1 3],[1 2 3]);     % Delete yv form I/O


%%
% The resulting simulation model has w,v,u as inputs and y,y_e as
% outputs:
SimModel.inputname

%%
SimModel.outputname

%%
% You are now ready to simulate the filter behavior.
% Generate a sinusoidal input vector (known):
t = (0:100)';
u = sin(t/5);

%%
% Generate process noise and sensor noise vectors:
rng(10,'twister');
w = sqrt(Q)*randn(length(t),1);
v = sqrt(R)*randn(length(t),1);

%%
% Now simulate the response using LSIM:
out = lsim(SimModel,[w,v,u]);

y = out(:,1);   % true response
ye = out(:,2);  % filtered response
yv = y + v;     % measured response

%%
% Compare the true response with the filtered response:
clf
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Kalman filter response')
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')

%%
% As shown in the second plot, the Kalman filter reduces
% the error y-yv due to measurement noise. To confirm this,
% compare the error covariances:
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);

%%
% Covariance of error before filtering (measurement error):
MeasErrCov

%%
% Covariance of error after filtering (estimation error):
EstErrCov

```

```
%% Time-Varying Kalman Filter Design
% Now, design a time-varying Kalman filter to perform the same
% task.  A time-varying Kalman filter can perform well even
% when the noise covariance is not stationary.  However for this
% example, we will use stationary covariance.
%
% The time varying Kalman filter has the following update equations.
%
%     Time update:         x[n+1|n] = Ax[n|n] + Bu[n] + Bw[n]
%
%                          P[n+1|n] = AP[n|n]A' + B*Q*B'
%
%
%     Measurement update:
%                          x[n|n] = x[n|n-1] + M[n](yv[n] - Cx[n|n-1])
%                                                                -1
%                          M[n] = P[n|n-1] C' (CP[n|n-1]C'+R)
%
%                          P[n|n] = (I-M[n]C) P[n|n-1]
%
%
% First, generate the noisy plant response:

sys = ss(A,B,C,D,-1);
y = lsim(sys,u+w);   % w = process noise
yv = y + v;          % v = meas. noise

%%
% Next, implement the filter recursions in a FOR loop:
P=B*Q*B';         % Initial error covariance
x=zeros(3,1);     % Initial condition on the state
ye = zeros(length(t),1);
ycov = zeros(length(t),1);
errcov = zeros(length(t),1);

for i=1:length(t)
  % Measurement update
  Mn = P*C'/(C*P*C'+R);
  x = x + Mn*(yv(i)-C*x);  % x[n|n]
  P = (eye(3)-Mn*C)*P;     % P[n|n]

  ye(i) = C*x;
  errcov(i) = C*P*C';

  % Time update
  x = A*x + B*u(i);        % x[n+1|n]
  P = A*P*A' + B*Q*B';     % P[n+1|n]
end

%%
% Now, compare the true response with the filtered response:
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Response with time-varying Kalman filter')
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')

%%
% The time varying filter also estimates the output covariance
% during the estimation.  Plot the output covariance to see if the filter
% has reached steady state (as we would expect with stationary input
% noise):
subplot(211)
plot(t,errcov), ylabel('Error Covar'),

%%
% From the covariance plot you can see that the output covariance did
% reach a steady state in about 5 samples.  From then on,
```

```
% the time varying filter has the same performance as the steady
% state version.

%%
% Compare covariance errors:
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);

%%
% Covariance of error before filtering (measurement error):
MeasErrCov

%%
% Covariance of error after filtering (estimation error):
EstErrCov

%%
% Verify that the steady-state and final values of the
% Kalman gain matrices coincide:
M,Mn
```