Velocity estimates from sampled position data

It is often the case that position information has been sampled from continuous process and it is desired to estimate the velocity and acceleration. Example would include data from a quadrature encoder, an analogue to digital converter, a Vicon capture system, a series of ultrasound measurements etc.

The problem is that differentiation is noisy at high frequencies and further more there is a danger of oversampling where multiple records are made of the same value, or quantisation, where the measurement rounded up or down to the nearest binary integer that represents the signal.

An alternative is to use a sensor such as an accelerometer and integrate which is more more robust but suffers from drift (low frequency noise).

The solution is to filter the signal and then compute the velocity but since the filters considered here are linear, this process can be done in a single filter. This can be used to reduce the consequences of noise in the signal but care is needed to ensure that the phase lag does not distort the results, that is to say a position peak (or trough) should be represented by the velocity estimate passing through the horizontal axis (0), the danger is that the velocity will pass through the horizontal axis some time after the velocity peak.

We can consider finite impulse response (FIR) and infinite impulse response (IIR) filters.

FIR Velocity filters

Given a signal u sampled at regular intervals T we want to estimate du/dt

The general FIR filter is $y_n = b_0 u_n + b_1 u_{n-1} + b_2 u_{n-2} ...$

The best equal weighted FIR filters are

2	[1 -1]/T
3	[.5 0 -.5]/T
4	[.3 .1 -.1 -.3]/T
5	[.2 .1 0 -.1 -.2/T
6	[1/7 3/35 1/35 -1/35 3/35 1/7]/T
7	[3/28 1/14 1/28 0 -1/28 1/14/3/28]/T
8	[1/12 5/84 1/28 1/84 -1/84 -1/28 -5/84 -1/12]/T

It can be derived from the Maxima code as follows

```
matrix( [1,1,1,1], [0,T,2*T,3*T] );
transpose(%).invert(%.transpose(%));
```

As the length of the filter increases the phase lag increases.

Note on quantisation

At high sampling speeds where the signal is changing slowly the output of a difference filter will appear as a series of impulses. Check the data and consider resampling your position at a lower frequency (for example keep only every 10th measurement)

Using a mean filter followed by a differentiator

One crude way of filtering a signal is to use a mean filter with a span p that is to say

$$y_n = \frac{1}{p}(u_n + u_{n-1} + u_{n-2}..u_{n-p+1})$$

This can then be followed by the difference filter

$$y_n = \frac{1}{T}(u_n - u_{n-1}$$

It is relatively easy to show that this is simply the filter

$$y_n = \frac{1}{Tp}(u_n - u_{n-p+1})$$

The problem with this approach is that the filter has a large phase lag because equal weight is given to the latest sample and the sample p time steps back.

IIR Velocity filters

An IIR filter is possible by considering

$$\frac{N}{1 + N/s}$$

as a differentiator followed by a first order filter. This can be written as

$$\frac{s}{s/N + 1}$$

which is a closed loop system with gain N and an integrator in the feedback path.

One digital integrator is

$$Y/X = \frac{T}{2}\frac{(1 + z^{-1})}{(1 - z^{-1})} \approx \frac{1}{s}$$

Where the new integrated value is the old value plus the average of the current and the new input. This is essentially a Padé approximation of

$$z = e^{sT} \approx \frac{1 + sT/2}{1 - sT/2}$$

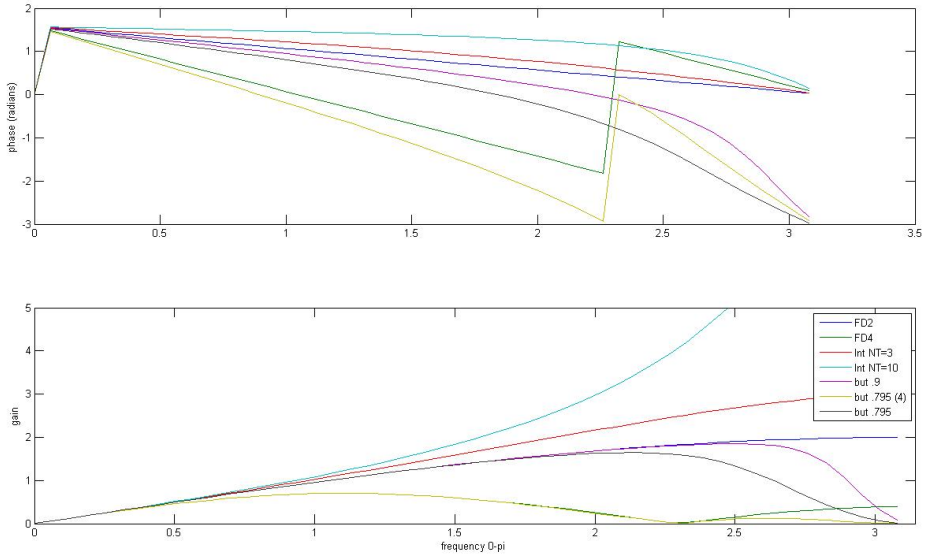

Figure 1: Frequency response of FIR and IIR velocity filters. Pure differentiation has gain= ω phase= $\pi/2$

Thus the filter becomes

$$Y/X = \frac{N}{(1+NT/2)} \frac{(1-z^{-1})}{(1+\frac{NT-2}{NT+2}z^{-1})}$$

When NT=2 then the filter imitates a 2 element difference filter. High values of NT implies noise at high frequencies but a constant phase lead of $pi/2$ i.e. correct integration. Low values of NT implies phase large phase lag

Second order Butterworth filter

Another possible velocity is a 2nd order low pass filter following the differentiator.

The 2nd order Butterworth filter is of the form $y_n = b_0 u_n + b_1 u_{n-1} + b_2 u_{n-2} - a_1 y_{n-1} - a_2 y_{n-2}$

Can choose a cut off frequency $0 < wn < 1$ where sampling frequency =2;

The filter coefficients are then derived from $\Omega = \tan(\omega_n \pi/2)$

$den = 1 + \sqrt{2}\Omega + \Omega^2$

$b = [1\ 2\ 1]\Omega^2/den$

$a = [2(\Omega^2 - 1)\ (1 - \sqrt{2}\Omega + \Omega^2)]/den$

Compute your own filter

A simple filter that might do better with you particular data is to consider how to estimate the filter data giving greater weight to recent data.

For example a finite difference filter could difference the current value and the average of the previous two values. This would now be considered to happen over a time period of $1.5T$ that is to say

$y_n = (x_n - 0.5(x_{n-1} + x_{n-2}))/1.5T$ This can be rewritten as $y_n = (2x_n - x_{n-1} + x_{n-2})/3T$

Or alternately use a difference over the last two samples and average this estimate to the previous estimate. If we assume $c + d = 1$ a range of filters are available of the form $y_n = cyn - 1 + \frac{d}{T}(x_n - x_{n-1})$

Matlab code to generate the graph

```
function response2(x)
% some response functions
% for velocity filters based on Butterworth and finite difference.
%
T=1;
% difference filters
BF2=[1 -1]/T;
BF3=[.5 0 -.5]/T;
BF4=[3 1 -1 -3]/(10*T);
BF6=[5 3 1 -1 -3 -5]/(35*T);
BF8=[7 5 3 1 -1 -3 -5 -7]/(84*T);

N=3/T; %red
AC5=[1 (N*T-2)/(N*T+2)];
BC5=[1 -1]*N/(1+N*T/2);

N10=10/T; %cambridge
AC10=[1 (N10*T-2)/(N10*T+2)];
BC10=[1 -1]*N10/(1+N10*T/2);

[Abut5,Bbut]=butter(.5); %butterworth filter at .5
Bbut5=conv(Bbut,BF2);
Bbut5_4=conv(Bbut,BF4);

[Abut795,Bbut]=butter(.795); %butterworth filter at .795
Bbut795=conv(Bbut,BF2);
Bbut795_4=conv(Bbut,BF4);

[Abut9,Bbut]=butter(.9); %butterworth filter at .5
Bbut9=conv(Bbut,BF2);

% Now do the frequency responses
Npts=50;
[h1,w1]=freqz(BF2,1,Npts);
[h2,w2]=freqz(BF4,1,Npts);
[hfb1,wfb1]=freqz(BC5,AC5,Npts);
[hfb2,wfb2]=freqz(BC10,AC10,Npts);
[hb1,wb1]=freqz(Bbut9,Abut9,Npts);
[hb2,wb2]=freqz(Bbut795_4,Abut795,Npts);
```

```
[hb4,wb4]=freqz(Bbut795,Abut795,Npts);

subplot(2,1,1)
plot(w1,angle(h1),w2,angle(h2),wfb1,angle(hfb1),wfb2,angle(hfb2),wb1,angle(hb1),wb2,angl

ylabel('phase (radians)')
subplot(2,1,2)
plot(w1,abs(h1),w2,abs(h2),wfb1,abs(hfb1),wfb2,abs(hfb2),wb1,abs(hb1),wb2,abs(hb2),wb4,a
axis([0 pi 0 5/T])
ylabel('gain')
xlabel('frequency 0-pi')
legend('FD2','FD4','Int NT=3','Int NT=10','but .9','but .795 (4)','but .795');shg
end
function [Abut,Bbut]=butter(wn)
% butterworth
Om=tan(wn*pi/2);
den=1+sqrt(2)*Om+Om^2;
Bbut=[1 2 1]*Om^2/den;
%Abut=[1 2*(Om^2-1) 1-sqrt(2)*Om+Om^2]/den;
Abuta=[2*(Om^2-1) 1-sqrt(2)*Om+Om^2]/den;
Abut=[1 Abuta];
end
```