Introduction to human movement analysis

Jason Friedman

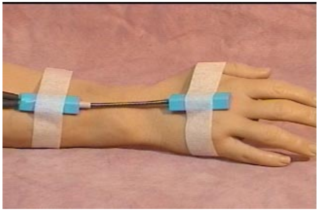
MACCS
Macquarie University

15th October 2010

Types of data that can be collected

- Kinematics
  - Joint angles
    - bend sensors (e.g. cyberglove)

    - goniometers

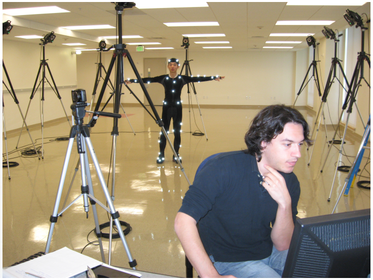


Types of data that can be collected

- Kinematics
  - displacements
    - accelerometers

Types of data that can be collected

- Kinematics
  - Positions of body parts:
    - optical motion capture (active or passive)



Types of data that can be collected

- Kinematics
  - Positions of body parts:
    - magnetic systems

Types of data that can be collected

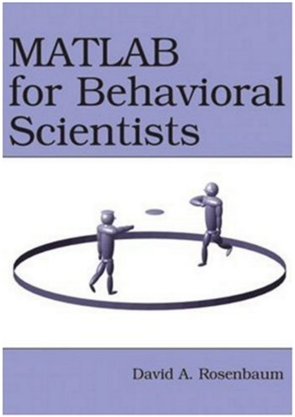
- Dynamics
    - Force sensors - ranging from 1 DOF to 6 DOF
- Neurophysiological
    - electromyography (EMG)

Sampling rates

- Usually limited by the device
- In general, faster is better.
- Smaller parts of the body move faster so need higher sampling rates (e.g. whole person compared to fingertip.)
- Use the Nyquist sampling theorem as a rough guide - sampling rate should be two times the highest frequency in the signal
- Increase the sample rate if you plan on using interpolation to fill in missing values (to be discussed later)
- In practice, 100 - 200Hz is typical for arm movements
- Can get away for less for gait (25Hz), need higher rate if looking at very fine finger movements (e.g. TMS responses)

Matlab

- Matlab is probably the most used tool for movement analysis
- A good book on learning matlab is “Matlab for behavioral scientists” by Rosenbaum (available in the library)



Matlab: load the data

- Raw data usually comes in text files (tab separated or comma separated)
- These can be easily read into Matlab:

Load some data:

```
data = load('optotrak_data.csv');
```

Check that the size is reasonable:

```
size(data)
```

Matlab: give sensible names

Make sure you know what the columns mean (time? x? y? z? stimuli properties?), and put into sensibly named variables. You may also need to trim the data (in this case, the last column indicated the start of the trial)

```
 flags = data(:,8);
 firstsample = find(flags==5);
samplenum = data(firstsample:end,1);
x1 = data(firstsample:end,2);
y1 = data(firstsample:end,3);
z1 = data(firstsample:end,4);
x2 = data(firstsample:end,5);
y2 = data(firstsample:end,6);
z2 = data(firstsample:end,7);
```

Matlab: Check the sampling rate

Check that your sampling rate is really what you think it is:

```
mean(diff(samplenum))
std(diff(samplenum))
```

e.g., In this case, it is optotrak frame number. In this experiment, we sampled at 200Hz, so convert to seconds:

```
times = (samplenum - samplenum(1)) ./200;
```

Matlab: Sanity check

- Make sure you know what your coordinate system is (which direction is which), where the origin is, and what the units are.
- Plot the data and see that is looks reasonable.

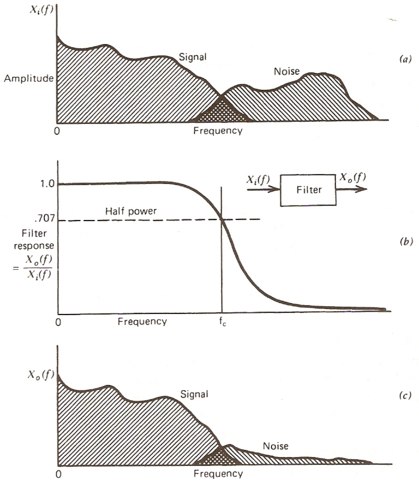
```
figure;
plot(times,x1);
xlabel('time (s)');
ylabel('x (mm)');
figure;
plot(x1,y1);
xlabel('x (mm)');
ylabel('y (mm)');
```

Filtering / smoothing

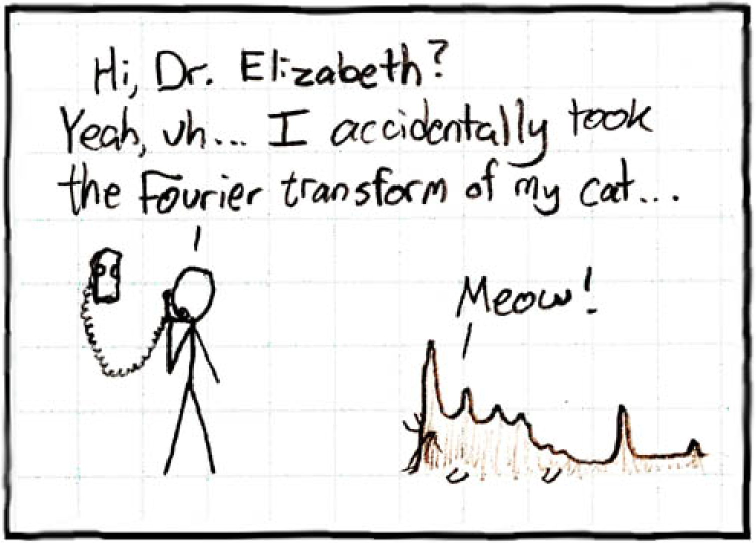
- Why filter?
    - Filtering necessarily changes the data, so you need to ensure appropriate filtering is done.
    - In some cases, this may be no filtering at all.
    - But in general, all measurement devices are noisy, and we ideally want to work with the “actual” movement rather than the recorded movement
- We can filter our data by using different properties of movements compared to noise
- It is never possible to remove all noise from the data, this should be taken into account in the experiment design (i.e., run lots of repetitions)

Filtering

- Filtering is typically performed in the frequency domain
- Noise typically exists at different frequencies to the data, making it easier to separate.



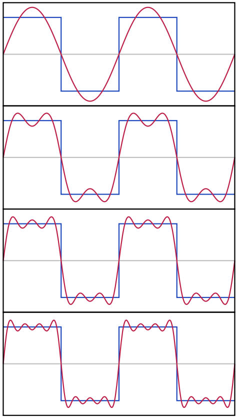
What is the “frequency domain”?


Hi, Dr. Elizabeth?
Yeah, uh... I accidentally took the Fourier transform of my cat...
Meow!

(from `xbcd.com`)

What is the "frequency domain"?

- We can consider trajectories instead as the sum of sin waves:



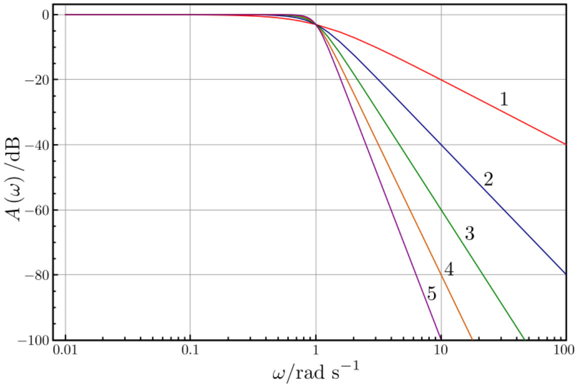
- Note that this decomposition is not perfect and this is one reason why is not possible to just filter noise and not also the data

Typical filter: Butterworth

- Typically use a lowpass “Butterworth” filter.
- Lowpass = Only lets “low” frequencies through (remember that noise in movement data is more common at high frequencies, and arm movements are relatively low frequency)

Typical filter: Butterworth

- Butterworth has a response as flat as possible in the response band, so it doesn't alter the signal in this range.
- The order of the filter determines how quickly it drops off



Filtering

- Usually use a two-way filter (filtfilt in matlab), otherwise there will be a phase shift in the data (i.e., events will occur later in the filtered version).
- Note that when you use a two-way filter, you double the order of the filter (important for reporting)
- ALWAYS plot the filtered vs unfiltered data to make sure it looks reasonable
- When reporting the filter, report the type, cutoff, and order, e.g. a 4th order two-way lowpass Butterworth filter at 20Hz was applied to the data

Matlab:Filtering

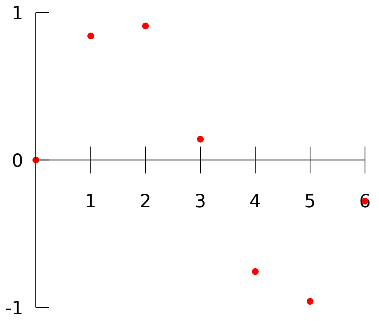
- For this data, we will apply a 4th order, 20Hz lowpass filter
- The matlab command butter takes two arguments:
  1. Order - we will specify 2, because it will be doubled when using a two-way filter
  2. cutoff Wn, where $0 < Wn < 1.0$, where 1.0 is equal to half the sample rate. In our case, 100Hz = half the sample rate, so for a 20Hz cutoff, Wn=0.2

```
cutoff = 20;
samplerate = 200;
Wn = cutoff / (samplerate/2);
[B,A] = butter(2,Wn);
x1_filtered = filtfilt(B,A,x1);
y1_filtered = filtfilt(B,A,y1);
z1_filtered = filtfilt(B,A,z1);
```

Interpolation

- When using optical motion tracking data, samples may be missing because of occlusion
- If there are a lot of missing samples, you should throw away the trial, but if there a small number, you can “guess” what they would have been.
- Other devices may also drop occasional samples which can also be interpolated

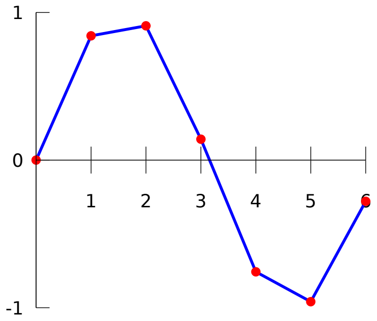
Types of interpolation



- Interpolation is a way of filling in the gaps

Types of interpolation

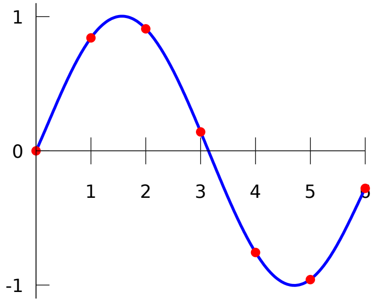
Linear interpolation



- Basically draw a line between the two nearest points

Types of interpolation

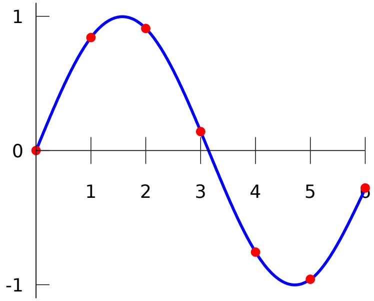
Polynomial interpolation



- Draw a polynomial through all the data points
- Need a polynomial of degree one less than the number of points
- In general is problematic - noisy data can cause very strange results, so usually not used for movement data

Types of interpolation

Spline interpolation



- Use low order polynomials (typically 3rd order - cubic) between each pair of points, and match the gradient at the data points (so it will be smooth)

Matlab: Spline interpolation of missing data

- Missing samples can be estimated using spline interpolation
- e.g., reconstruct samples 200 to 210 from the data

```
% remove the data
x1([200:210]) = NaN;
gooddata = find(~isnan(x1));
% reconstruct it
interpolated_linear = interp1(times(gooddata),...
    x1(gooddata),times);
interpolated_spline = interp1(times(gooddata),...
    x1(gooddata),times,'spline');
% plot it
plot(times(baddata),interpolated_linear,'r');
hold on;
plot(times(baddata),interpolated_spline,'g');
plot(times,x1,'b');
```

Matlab: Interpolation to resample to a fixed sample size

- Often we are interested in comparing trajectories, or taking their average
- This is a problem if they are all of different durations
- One option is to first resample all the data to be the same length, e.g. 100 samples

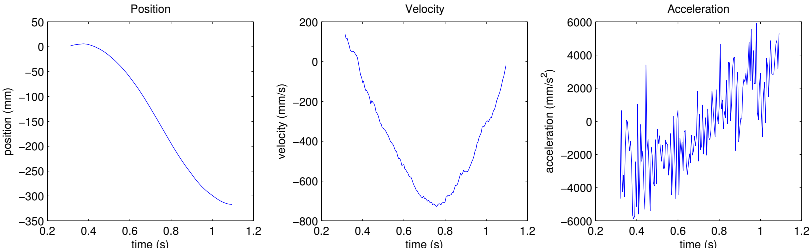
```
x1_cut = x1(76:220);
times_cut = times(76:220);
times_to_sample_at = linspace(times_cut(1),...
    times_cut(end),100);
% resample to 100 samples
x1_resampled = interp1(times_cut,x1_cut,...
    times_to_sample_at);
% Plot to see they are the same
plot(times_cut,x1_cut);
hold on;
plot(times_to_sample_at,x1_resampled,'r.');
```

Side note: cubic smoothing splines

- Splines can be used also to smooth the data
- If less splines are used than there are data points, the splines will not go through every point.
- Can use the Matlab function cpaps from the spline / curve fitting toolbox (but we do not have a license for it) or the function splinefit (free from the Matlab central file exchange)
- Can use this technique to interpolate and smooth data in one step

Derivatives and integration

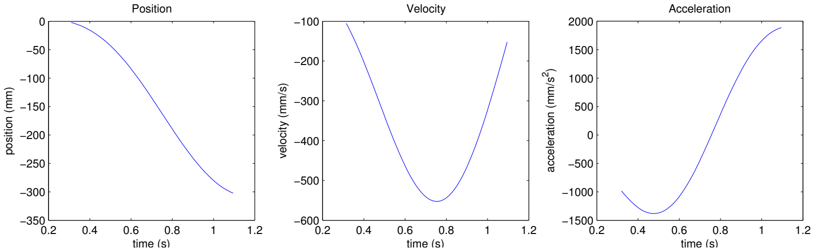
- Often we are interested in taking the derivative or integral of trajectory data
- The derivative of position gives us the velocity, the next derivative acceleration
- If we have accelerometer data, we can integrate once to get velocity or twice to get position data
- If we had perfect data (i.e., no noise), then we could just use finite differences, but with noisy data, the errors are greatly amplified when taking the derivative in this way



Derivatives and integration

- Filters should be applied before taking a derivative
- For best results, different frequency filters should be used depending on which derivative is being calculated (Giakas & Baltzopolous, 1997).
- For noisy data, 5-point derivative can be used (use 2 points before and after a data point):

$$f'(x) \approx \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h}$$



Malab: Taking the derivative

- When taking the derivative using finite differencing, always remember to divide by the time difference between the samples (so that the units will be right)

```
% derivative of unfiltered data
vel_unfiltered = diff(x1) ./ (1/200);
% derivative of filtered data
vel_filtered = diff(x1_filtered) ./ (1/200);
% 5-point derivative
vel_5p = fivepointderivative(x1_filtered ,1/200);
```

Reminders

- Always check that everything is reasonable:
  - The directions of the axes
  - The units (mm? m?)
- Ensure averaging, filtering, interpolation and derivatives are appropriate by plotting the data

Preview of next session

- Overview of kinematic measures and their uses
- Segmenting movement data
- calculation of arc length
- calculation of path offset / “curvature”
- calculation of initial angle
- decomposing movements into submovements
- Anything else?