# Module IN3031 / INM378 Digital Signal Processing and Audio Programming

Tillman Weyde t.e.weyde@city.ac.uk

# Revision

# Audio Signal Processing Summary

# Acoustics Basics

- Frequency unit **Hertz** means **1/s**
- **Wavelength** (l or λ), frequency (f), **period** (p), **speed** (c) of sound (~340m/s)
  - Example: f = 680 Hz, p = 1 / 680Hz ~ 0.0015 s,
    l = c/f = 340 (m/s) / 680Hz = .5 ms/s = .5m
  - What is the frequency of a sound with period 10ms
- **Energy** increases as **square of Amplitude** (e.g. 5-fold amplitude means 25-fold energy)
- **Harmonic** sounds have components with frequencies that are integer multiples of $f_0$, the lowest frequency.

# Hearing

- ***Amplitude** - Loudness; **Frequency** – Pitch;* ***Spectrum** - Timbre*
- ***Decibels**: a* is *x* **dB** greater than *b*, means $x = 10 \log_{10} (a/b)$
- Threshold of hearing (roughly): 0 dB **SPL** = $10^{-12}$ Watt/$m^2$ and the threshold of pain 130 dB SPL
- In the inner ear, the **Cochlea** with Basilar membrane transforms **sound waves into neural signals**

# Digitizing Audio

- **Sampling**: regularly measuring air pressure

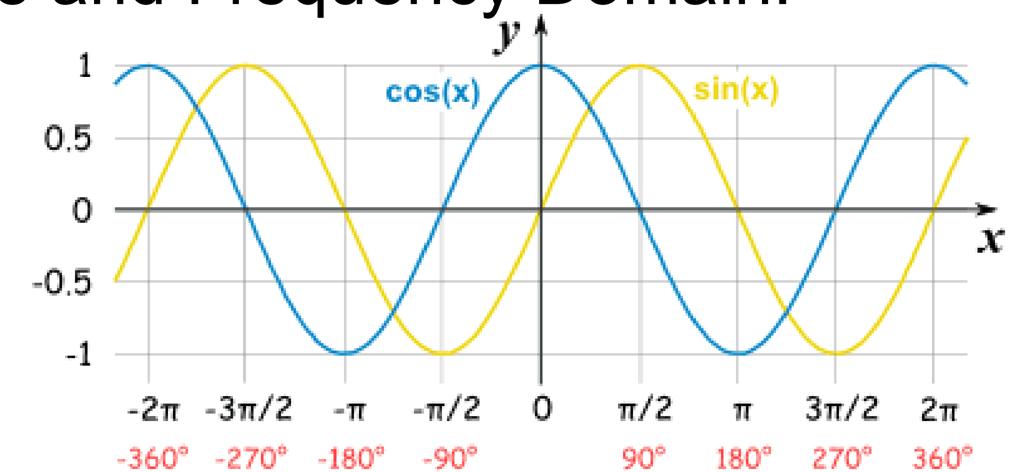
(or voltage, or any other quantity)

- **Sample rate**: how many measurements (per sec)
- **Sample depth: resolution** (# of bits) per sample value
- **Storage space** for audio recordings:
  channels * time * sample rate * sample depth

# Time and Frequency Domains

- Audio signals vary over time: **Time Domain**
- The spectrum decomposes the signal into frequency components: **Frequency Domain** (sin and cos)
- The **Fourier Transformation** (and its inverse) transform between Time and Frequency Domain:

$$x(t) \circ\!\!-\!\!\bullet X(f)$$



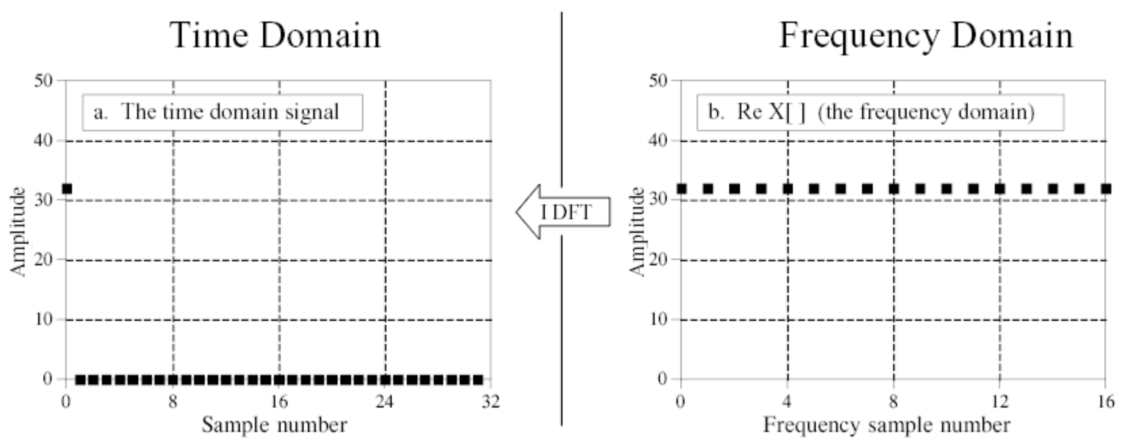
# Spectrum and Time

- The **spectrum** as calculated by the (fast) FT is **as long as the signal** we are analysing
  - longer signal → higher frequency resolution
- Speeding up the signal means stretching the spectrum (and inverse for k<1)

$$x(kt) \circ\!\!-\!\!\bullet \frac{1}{|k|} X\left(\frac{f}{k}\right)$$

# Special Spectra

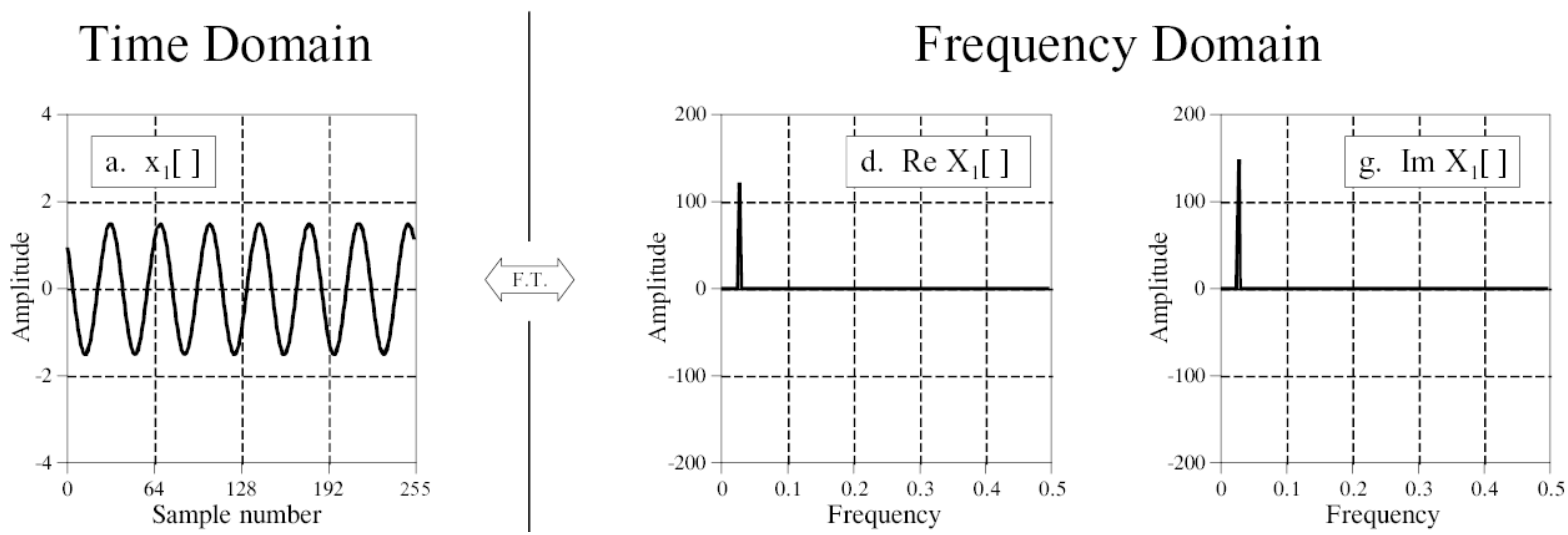
- Spectrum of **a unit impulse** at time 0 consists of **all real 1s** (cos), with 0 imaginary part. Real/imaginary ratio varies according to impulse's position in time.
- The spectrum of a **constant signal** with amplitude 1 is N (=length of the signal) at **frequency 0**, and 0 elsewhere.




This image shows only the lower half, the upper half is the mirror image of the lower.

# Special Spectra 2

- The spectrum of a **single sinusoid** (sine/cosine mixture) with frequency f is **0 in all places but f** (only exact if the signal length is a multiple of the frequency period).
- For a **single sinusoid** with (max.) amplitude 1, the spectrum at f will have magnitude l (length of signal, distribution real/imaginary depends on cosine/sine ratio)




This shows only the lower half of the spectrum, the upper half is a mirror image of the lower.

# Sampling Theorem

- **Sampling** Theorem (**Nyquist Theorem**):
  - When sampling with frequency *fs*, only frequencies less than *fs/*2 (**Nyquist Frequency** Ny) can be recorded accurately.
  - If this is not taken into account, **Aliasing** occurs.

**Aliasing:** a signal component with a frequency greater than Ny is reproduced as a lower frequency:
$f_a = |((f + Ny) \bmod Fs) - Ny|$

# Correlation

**Correlation** is a measure of similarity

$$corr(s1, s2) = \sum_{t=0}^{N2-1} s1[t] s2[t]$$

**Correlation Coefficient** ρ

$$\rho = \frac{corr(x, y)}{\sqrt{\sum x[n]^2 \cdot \sum y[n]^2}}$$

Positive correlation coefficient of two signals means correlated signals (good mono compatibility of stereo signals), 0 means uncorrelated, negative correlation means cancellations in case of mono mixdown.

# Cross-Correlation

**Cross-correlation** is a correlation at a lag

$$xcorr(s1, s2, k) = \sum_{t=0}^{N2-1} s1[t]\, s2[t+k]$$

```
Xcorr([1,2,3],[4,3,2],1)
[ 1 ,2,  3]
    [4,  3,  2]
 1*0+2*4+3*3+0*3 = 17
```

# Other Signal Properties

**Autocorrelation**: Measure of self similarity, cross-correlation of a signal with itself.

$$ac(x,k) = \sum_{t=0}^{N-1} x[t] \cdot x[t+k]$$

Example: If signal x contains an echo at lag *l* autocorrelation *ac(x,l)* will be high. If the echo is inverted (* -1), the autocorrelation with be negative.

# Filtering

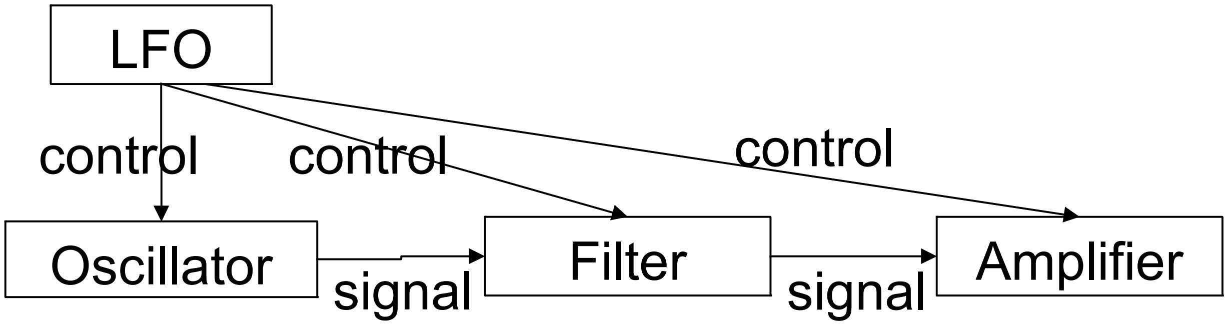
- **Linear Filters** add up samples with different coefficients depending on their 'age' to a new sample value
  - Finite Impulse Response **FIR** filtering applies only to input
  - Infinite Impulse Response **IIR** filtering also uses output samples (feedback loop)
- FIR filtering is *Convolution*
  - **Convolution Theorem** allows calculation and design of frequency response and filter design:
    $x * y \circ\!\!-\!\!\bullet X \cdot Y$

# Short Term Fourier Transform

- **Window width**
  - frequency vs. time resolution
- **Window function** enforces periodicity
  - Hann
- **Crossfading**
  - necessary since signal may be not 0 at boundaries after processing
  - functions should add up to 1 in the overlapping region
  - Triangular or Hann

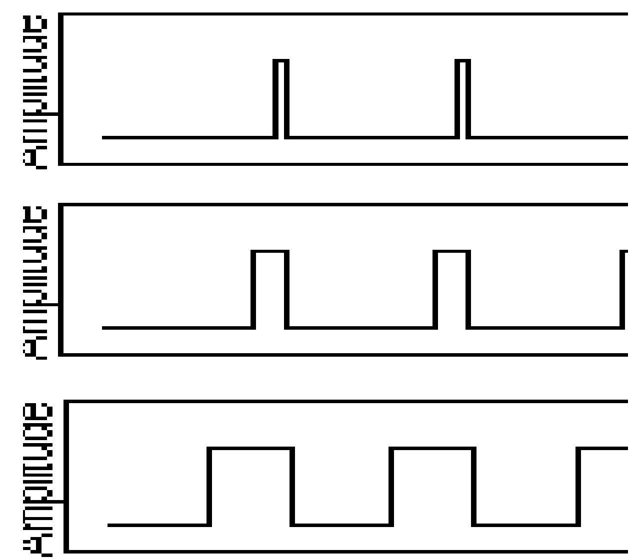
# Sound Synthesis

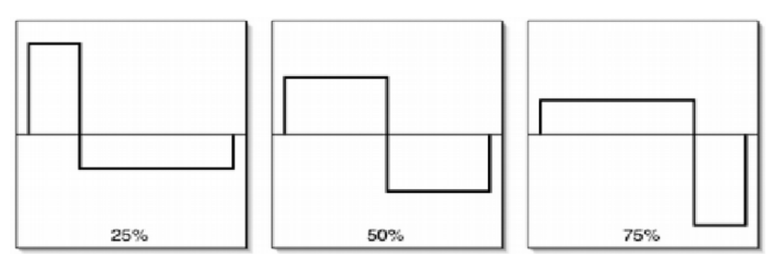
- Subtractive synthesis:



# Sound Synthesis

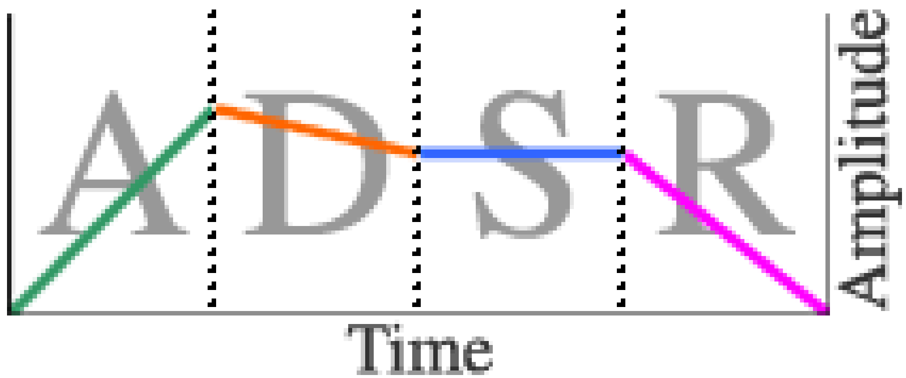
- Waveforms:
  - **Sine**
  - **Sawtooth**
  - **Square**
  - **Pulse Width Modulation**
    - Narrower pulse has more harmonics (high freq)
    - Need to compensate DC component





# Envelop Generators

- ADSR generator.



# Buffers and Latency in Digital Sound Systems

- Use buffers throughout for processing speed
- Buffers cause latency (delays between input and output)
  - Larger buffers
    - + less CPU usage, more stability
    - - higher latency
  - Smaller buffers
    - - more CPU, less stability
    - + lower latency

# Image Processing Summary

# Image Processing Summary (1)

- What is digital image processing?
- Topics in digital image processing
- Digital image acquisition
- Digital image representation
- Elementary image processing operations
- Colour image processing
- Noise in image processing
- Basic image processing in Matlab

# Image Processing Summary (2)

- **Digital image processing** concerns the transformation of an image to a digital format and its processing by computers
- **Topics**: image compression, medical imaging, image restoration, remote sensing, face detection...
- **Image acquisition elements**: energy, optical system, sensor
- The quality of a digital image is largely determined by the number of samples and discrete intensity levels used in **sampling** and **quantization**

# Image Processing Summary (3)

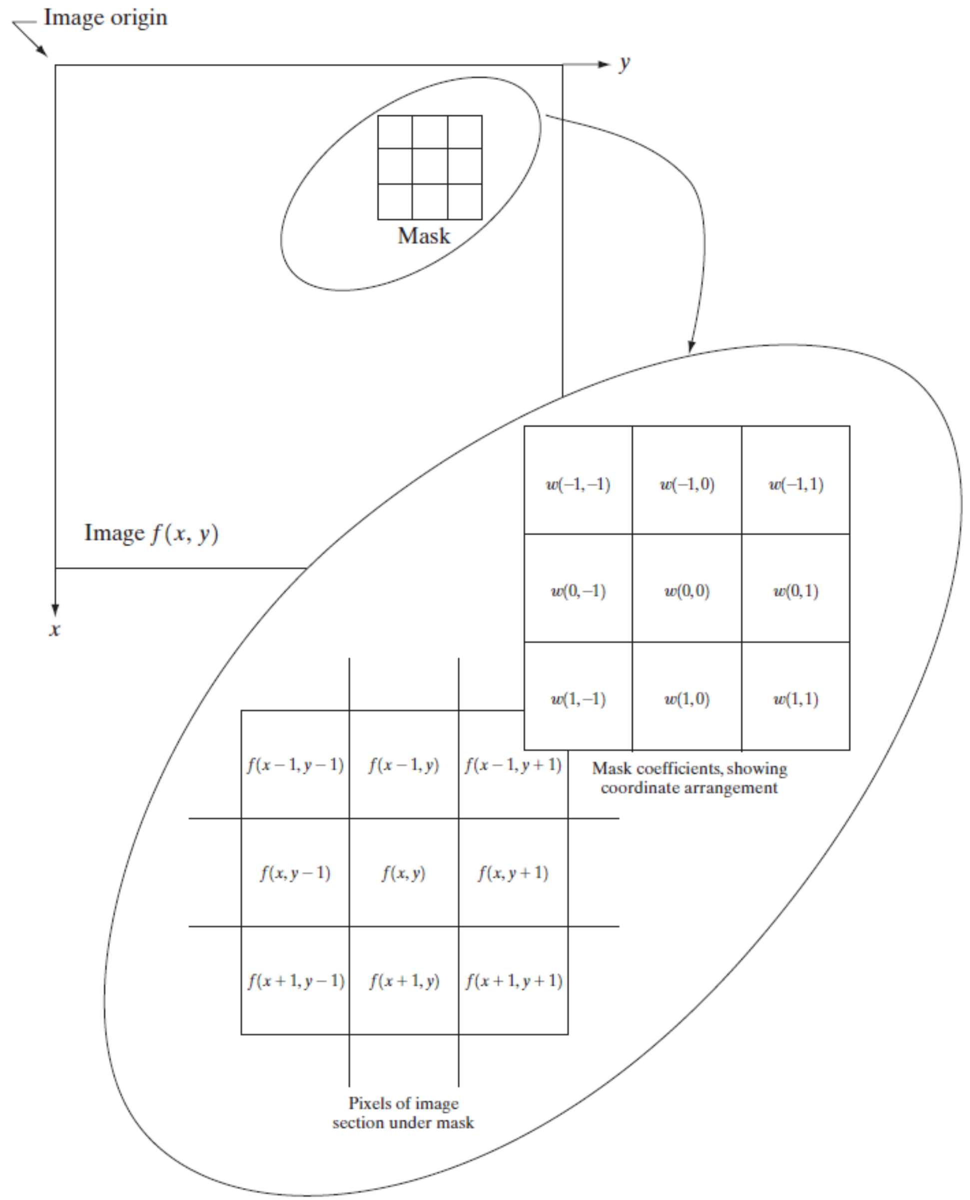
- An image can be represented by a 2-D array
- **Resolution**: spatial, gray levels
- **Basic transformations**: image negatives, gamma correction, addition, subtraction
- **Colour models**: RGB, CMY, HSI
- **Noise** in image processing: film grain, CCD noise...
- **Matlab Image Processing Toolbox**

# Image Processing Summary (4)

- Spatial filtering
- Smoothing/sharpening filters
- 2-D convolution
- 2-D Discrete Fourier Transform (DFT)
- Frequency domain filters

# Image Proc. Summary (5)

- **Mask**: small sub-image used for spatial filtering
- **Smoothing filters** are used for blurring and noise reduction
- **Sharpening filters** highlight fine detail or enhance detail that has been blurred.



# Image Processing Summary (6)

- 2-D convolution: moving a mask from pixel to pixel
- Most filtering takes place in the frequency domain (reason: computational efficiency)

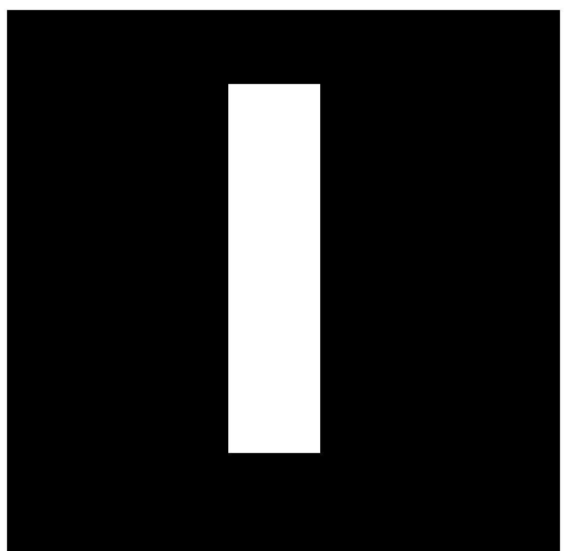
| | | |
|---|---|---|
| 2 | 2 | 2 |
| 2 | 2 | 2 |
| 2 | 2 | 2 |

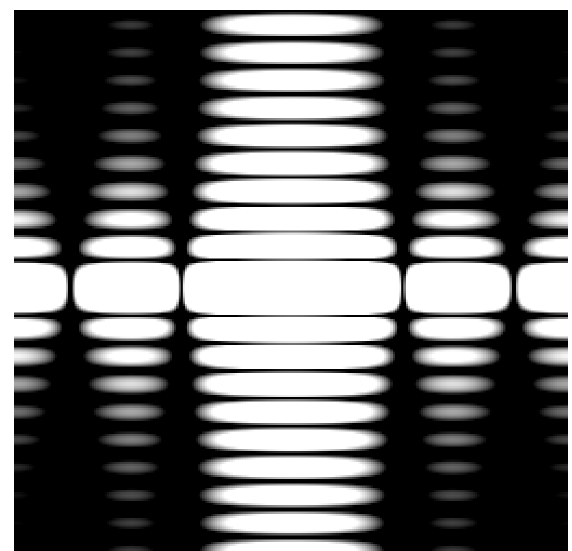
| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 1 | 2 |
| 1 | 2 | 1 |

| | | | | |
|---|---|---|---|---|
| 2 | 6 | 8 | 6 | 2 |
| 6 | 12 | 18 | 12 | 6 |
| 8 | 18 | 26 | 18 | 8 |
| 6 | 12 | 18 | 12 | 6 |
| 2 | 6 | 8 | 6 | 2 |

# Image Processing Summary (7)

- 2-D Discrete Fourier Transform (DFT)
- Lowpass/highpass filters



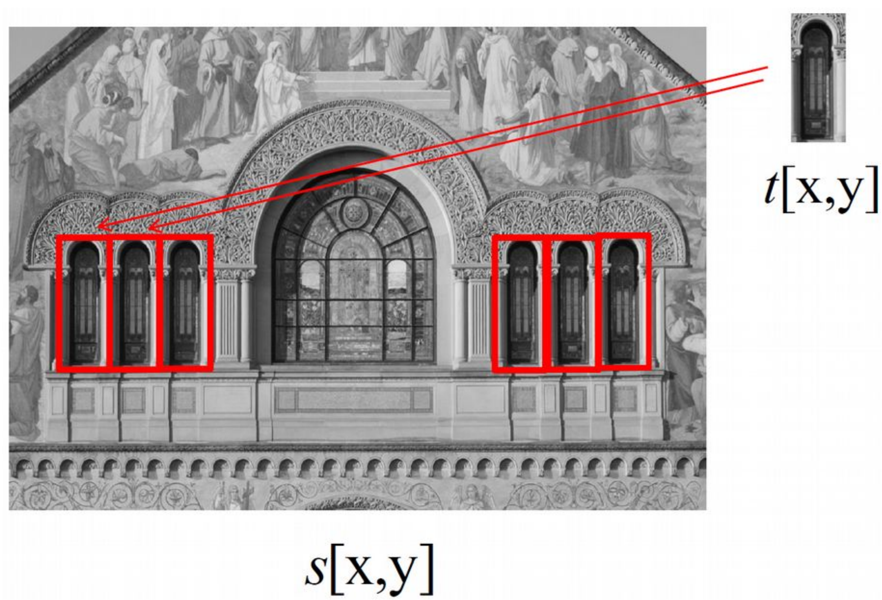


# Image Processing Summary (8)

- **2-D correlation**:

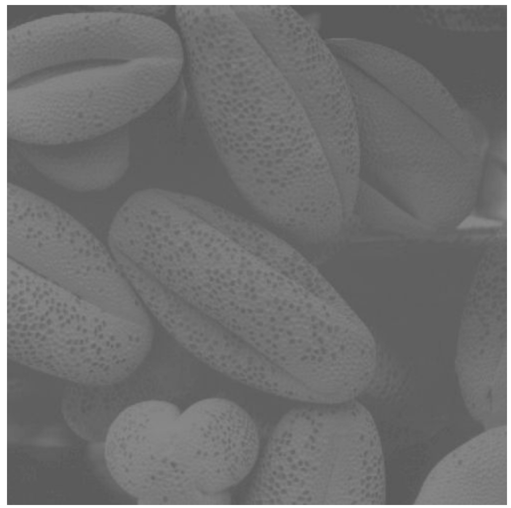
$$x[m,n] \circ y[m,n] = \frac{1}{RS} \sum_{r=0}^{R-1} \sum_{S=0}^{S-1} x^*[r,s]\, y[m+r, n+s]$$

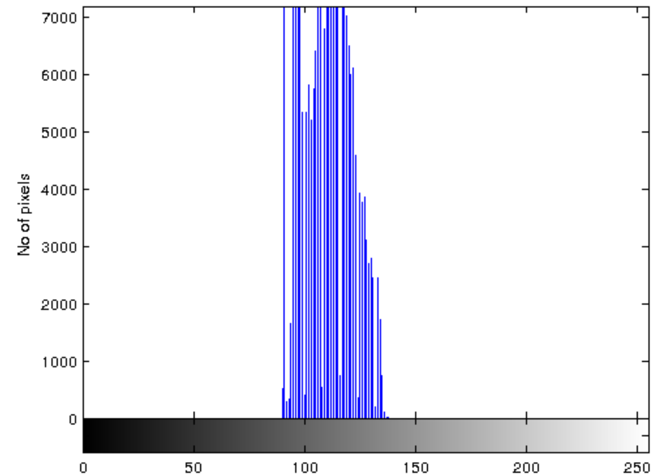
- Used for template matching

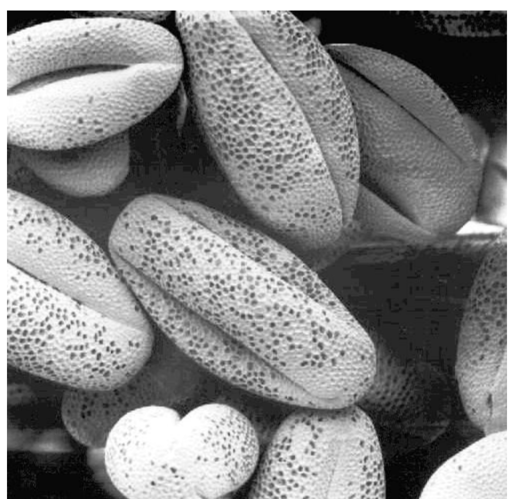


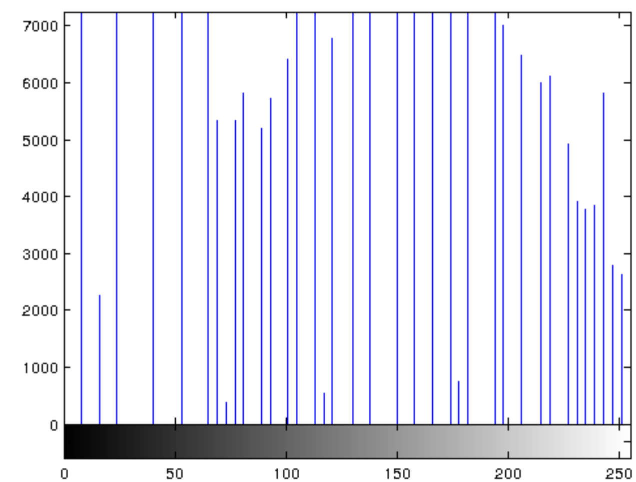
# Image Processing Summary (9)

- **Histogram equalisation**: method for contrast adjustment







# Image Processing Summary (10)

- Implementation:
  - Compute the image histogram ($n$: total number of pixels, $n_k$: number of pixels with gray level $r_k$):

    $$p_r(r_k) = \frac{n_k}{n} \qquad k = 0,1,2,\ldots, L-1$$

    Compute the cumulative histogram:
  - $$s_k = \sum_{j=0}^{k} p_r(r_j)$$

  New gray level values:
  - $$h(r_k) = \frac{s_k - s_{min}}{(MN) - s_{min}} (L-1)$$

# Image Processing Summary (11)

- **Histogram equalisation example**:

| | | | | |
|---|---|---|---|---|
| 52 | 55 | 63 | 67 | 63 |
| 63 | 59 | 55 | 90 | 90 |
| 63 | 59 | 68 | 90 | 68 |
| 63 | 58 | 68 | 55 | 63 |
| 67 | 52 | 68 | 59 | 55 |

Image

| | | | |
|---|---|---|---|
| 52 | 2 | 2 | 0 |
| 55 | 4 | 6 | 44 |
| 58 | 1 | 7 | 55 |
| 59 | 3 | 10 | 89 |
| 63 | 6 | 16 | 155 |
| 67 | 2 | 18 | 177 |
| 68 | 4 | 22 | 222 |
| 90 | 3 | 25 | 255 |

Pixels, histogram, cumulative histogram, new pixel values

# Image Processing Summary (12)

- **Image restoration**: reconstruct or recover an image that has been degraded
  $g(x, y) = h(x, y) * f(x, y) + \eta(x, y)$

- Noise cannot be predicted but can be approximately described in statistical way using a probability density function (PDF)

- Minimum mean square error filtering (Wiener filtering)

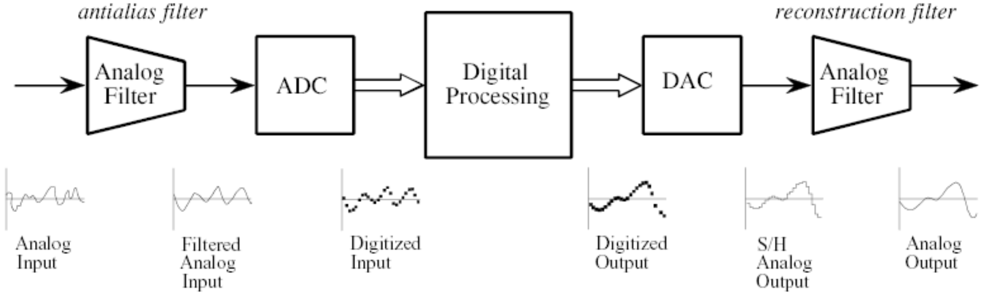
# Games Audio Summary

# Real Time DSP

- DSP algorithms can run in real time on general-purpose computers
- A lot of tasks can actually be quite cheap to run (*O(n)* or *O(n log n)*)
- Applications: DAWs, effects racks, VoIP and video software.

# The Real Time Paradigm

- Real time DSP is done by processing a **buffer** of samples.
- This means there will always be some **latency** (delay) between input and output.
- Increasing the **buffer size** increases the latency, but reduces computational cost.



# Typical Sound Engine Features (e.g. FMOD)

- Support for SW and HW voices and effects
- 3D sound rendering
- Reverb and many other effects
- Dynamic voice management by audibility (Virtual Voices)
- Sound processing **graphs**
- Allow **sound designers** and **audio developers** to work independently

# Directional Hearing and Localisation

- Directional hearing is based mostly on binaural hearing
  - Interaural Intensity Differences (IID)
  - Interaural Time Differences (ITD)
  - IID and ITD vary over frequencies.
- IID and ITD give only information on left-right
  - front-back and high-low are detected through head-related transfer functions (head shape, pinna)
- Room reflection
  - absorbing and reflecting objects give clues for source location

# Intensity and Distance

- Intensity is power (energy per time) per area, measured in Watts/Meter$^2$
- 0 dB (decibels) Sound Pressure Level defined as $10^{-12}$ W/m$^2$ (~threshold of hearing)
- dBs are on a logarithmic scale
- Intensity decreases as the square of distance
- Power increases as the square of amplitude
- Example: 60dB SPL at 1m means 40dB SPL at 10m (10-fold distance -> 100-fold decrease -> -10^2 = -2B = -20dB)

# Doppler Effect

- Doppler Effect changes frequency for moving sources
  - fp = f vs/(vs-vr)
    with fp: perceived frequency ,
    f: frequency, vs: velocity of sound (~340m/s),
    vr: velocity relative to the listener (positive = approaching)
  - Example: car moves with 68 m/s producing a 300 Hz sound. fp for a stationary listener in front of the car is
    300 Hz * 340/340-68 = 300 Hz *5/4 = 375 Hz

# Programming (2D) Sounds with FMOD

- Load a sound:

```
FMOD::Sound *sound;
result = system->createSound(filename,
    FMOD_LOOP_OFF, 0, &sound);
FmodErrorCheck(result);
```

- Create a Channel object and play the sound

```
FMOD::Channel    channel = null;
result = system->playSound(sound, NULL, false,
    &channel);
```

`channel` now has the channel where `sound` is played

```
// set the Volume
result = channel->setVolume(0.8f);
```

# 3D Modelling in FMOD

- FMOD has its own 3D coordinate system and model
- Need to coordinate
  - listener positions and velocities
  - sound source positions and velocities
  - objects in 3D that occlude or obstruct sound

# Programming 3D Sounds with FMOD

- Set up the FMOD systems 3D settings:

  system->set3DSettings(doppler, distFactor, distRolloff);

- Where:
  - doppler scales the intensity of the doppler effect
  - distFactor determines the length of an FMOD unit (1 means 1m – in the OpenGL template, use 10)
  - distRolloff scales the distance roll-off (1 is like real world)

# Custom FMOD DSPs

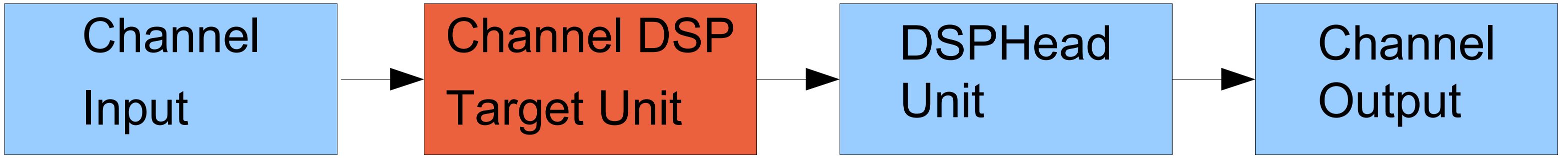
DSP inserts

- For whole system (all channels):

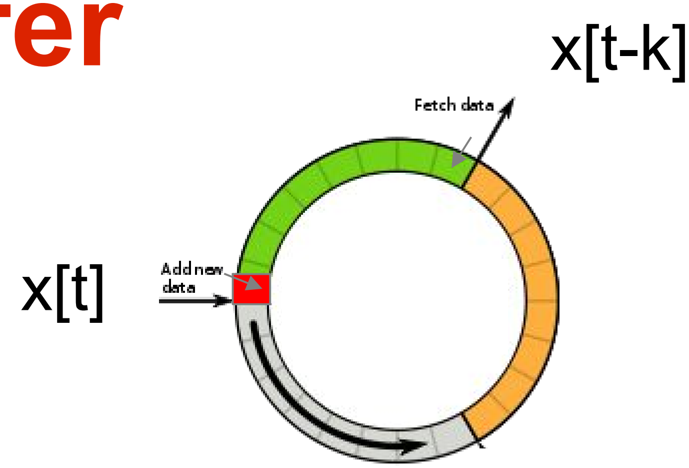
```
system->addDSP()
```

- For specific channel

```
channel->addDSP()
```



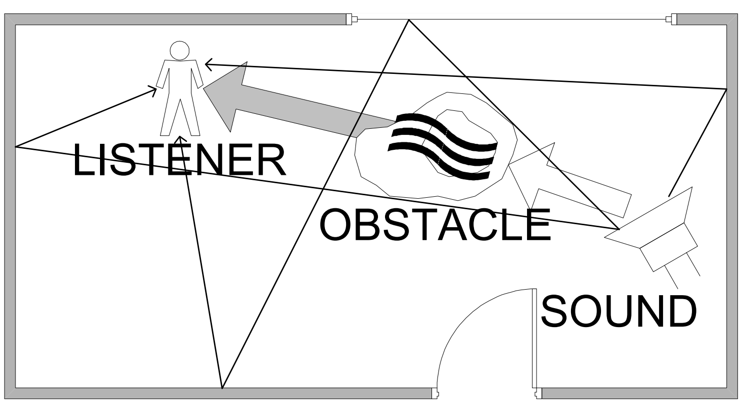
# Circular Buffer

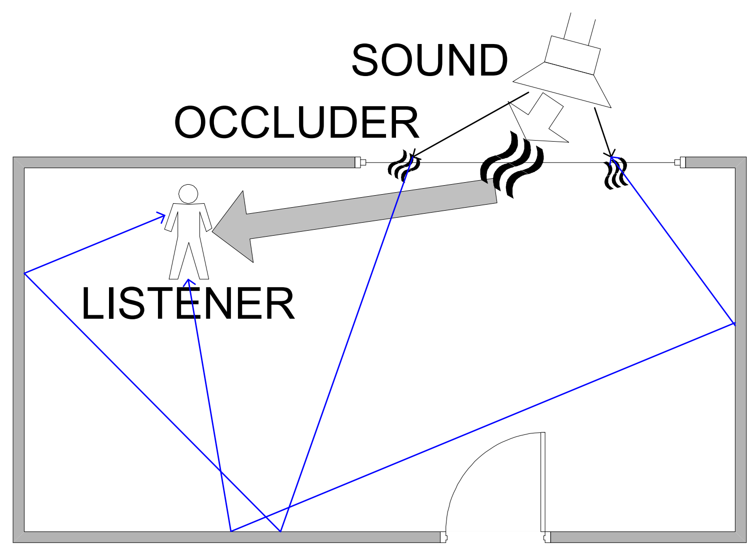



- Avoid high buffer maintenance costs
- Address the buffer for writing:
  `pos % bufferLength`
- Use index `delay` offset
- `(pos - delay)% bufferLength` points to read position

# Occlusion

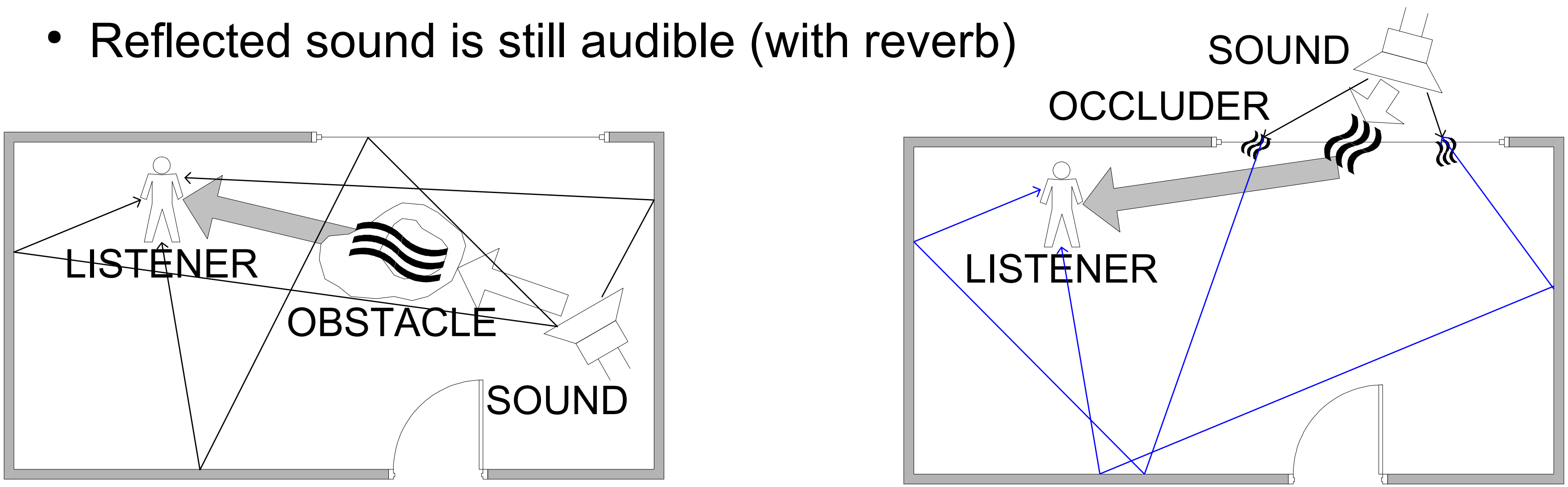
- Obstruction: obstacle blocks direct path between sound source and listener.
- Occlusion: occluder blocks all paths from the source to the listener.



# Occlusion

- Material, size and shape of the object determine how much sound (if any) reaches the listener
- Effects: drop in amplitude, loss of high frequency components (low pass filter)
- Reflected sound is still audible (with reverb)



# FMOD for Game Music

- Supports
  - Loop
  - Synchronisation based on beats and bars
  - Conditional transitions and repetitions
- Integrated with FMOD Event system

# Loops and Meter

- The metrical structure is normally maintained during loop playback.
- Common loop sizes are 4, 8 or 16 bars (although sometimes musical structures have different values, e.g. 'Eleanor Rigby' by the Beatles has a 5 bar structure)

# Loops and Harmony

- Harmony describes the sounding of several (pitched) notes together
- In harmonic contexts, some notes sound consonant, others sound dissonant/inappropriate.
- Layered music loops need common harmonic structure  (not true for sound loops)
- Each layer in the same harmonic pattern ensures they are musically 'compatible'

# MIDI vs Audio

- MIDI representation
  - used mostly in music production
  - used to be applied in Games directly
- MIDI is symbolic representation
  - Advantages:
    - independent tempo and pitch
    - easy to modify for musicians
    - low data volume
  - Disadvantages
    - sound quality (depends on used sound library)

# MIDI vs Audio in Loops

- Audio
  - can have superior quality (e.g. recorded human performance)
  - costly and lossy change of pitch/tempo
  - changing individual notes hardly possible

  → careful planning needed, good for final production
- MIDI
  - very flexible (easy to change tempo, pitch, notes)

  → can experiment, good for developing a soundtrack

# Time Series Summary

# Prediction by Smoothing with Moving Average

- Moving average of span *k* smoothes the data

$$\hat{y}_t = (y_t + y_{t-1} + \ldots + y_{t-k-1})/k$$

  ($\hat{y}$ is the prediction)

- A low pass FIR filter with coefficients 1/k,1/k, … , 1/k

- In Matlab: `filter([.25,.25,.25,.25],[1],Y)`

# Exponentially Weighted Moving Average

- Moving average of infinite span smoothes the data

$$\tilde{y}_t = w\, y_t + (1-w)\, \tilde{y}_{t-1}$$

- A low-pass IIR filter with coefficients *w* and (1-*w*)
  In Matlab: `filter([.25],[1,-(1-.25)],Y)`

- Assumption:
  - Recent values are more important than older ones

# Linear Trend Estimation

- Linear regression: find a straight line to fit the data

$$\hat{y}_t = a_0 + a_1 t$$

- Determine $a_0$ and $a_1$ to minimise the sum or squares error

$$sse = \sum_t (\hat{y}_t - y_t)^2$$

- Solve the system of equations

In Matlab: `coeff = polyfit(t,y,1)`

# Seasonal Average Method

- Seasonal averages = seasonal values total / # of years
- General average = seasonal averages total / # of seasons
- Multiplicative modelling:
  Seasonal index = seasonal average / general average
- Additive modelling:
  Seasonal offset = seasonal average – general average

# Residual Autocorrelation

- After linear modelling and seasonal adjustment we can study the autocorrelation of the residuals $e_t = y_t - \hat{y}_t$

**Correlations**

| | Residual Log Passengers | Lag 1 Residuals | Lag 2 Residuals |
|---|---|---|---|
| Residual Log Passengers | 1.0000 | 0.7896 | 0.6722 |
| Lag 1 Residuals | 0.7896 | 1.0000 | 0.7832 |
| Lag 2 Residuals | 0.6722 | 0.7832 | 1.0000 |

- With linear regression we can improve the prediction based on residuals

$$\hat{e}_t = -0.000153 + 0.7918985 e_{t-1}$$

- This is a form of a generalised (weigthed) moving average

$$y_t = \hat{y} + e_t + \sum_{i=1}^{q} \theta_i e_{t-i}$$

# Harmonic Modelling

- Fourier modelling for residuals or seasonality
  - keep only strong components (assumption: weaker components contain noise)
  - Advantage: more efficient than autocorrelation, calculation with FFT
  - Disadvantage: need to know cycle length, less robust than autocorrelation

# Modelling Caveats

- Overfitting:
  - to many parameters → model learns noise in the data but not the trend
  - need to test on data not used in building the model
  - cross-validation can when data is scarce
- Predictions get less reliable further away from sample data

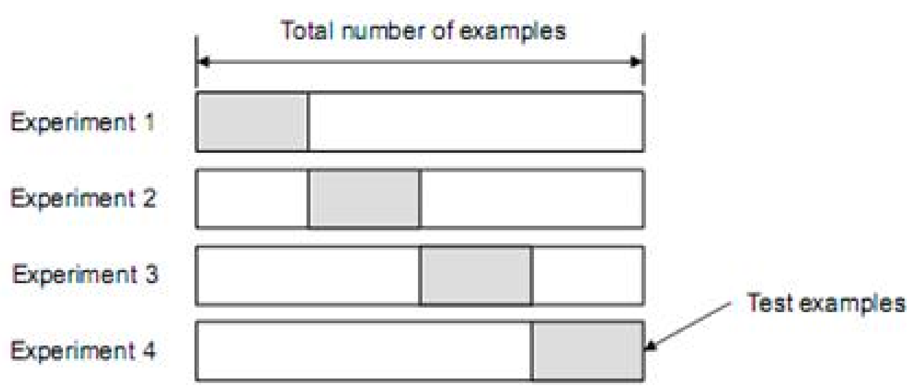
# Maximum Likelihood and Regularisation

- Common approach: Linear models with least squares optimisation
  - Maximise the likelihood of the data given the prediction (assuming normal distribution)

- Regularisation helps avoid overfitting (especially with small datasets)
  - Most popular: keep size of parameters low using a 'penalty term': sum of squares or absolutes of the parameters (*ridge* or *lasso*)
  - Add penalty term to errors and calculate gradient to optimise (use packaged solutions)

# Cross-Validation for Regularisation

- Optimise regularisation and other parameters:
  - Divide the data into *k* equally sized subsets ('folds')
  - Adapt the model to k-1 joint subsets, test on the remaining subset, and iterate through all folds
  - Test a grid of regularisation values (or other parameters) and choose the one with best results on test sets



# More Information

- Mock paper on Moodle for your reference
- If you have any questions, please get in touch

# That's it, we wish you successful exams!