# Module IN3031 / INM378 Digital Signal Processing and Audio Programming

Johan Pauwels
johan.pauwels@city.ac.uk
based on slides by Tillman Weyde

# (Fast) Fourier Transform in Practice

# Summary: DFT

- The **Discrete Fourier Transform** calculates the **spectrum *X*** from a **signal *x*:**

$$X[k]=\sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}nk}$$

- The **inverse** is very similar

$$x[n]=\frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{+i\frac{2\pi}{N}nk}$$

# Summary: FFT

- The **Fast Fourier Transform** (fft,ifft) is **efficient:** **$O(N \log N)$** instead of $O(N^2)$
- Used on **signals** of **length n** being **power of 2**
- Signals can be **zero-padded** to fit

# FFT Normalisation

- The FFT **preserves the energy** contained in a signal, if multiplied by **factor** $\sqrt{N}$
- The **whole FFT / iFFT cycle** produces a **multiplication** by *N*
- Normally **cancelled** only **in the iFFT** for efficiency reasons
- Alternatively, **divide** the **power spectrum** by N.

# Magnitude and Power Spectrum

- The **magnitude** (absolute of complex number) **spectrum** describes the **amplitude** for each frequency used (so called **bins**).
- The **square** of the **absolute** value describes the **energy** of the signal in that **bin**.

# Short-Time FFT
# Windowing
# Convolution & Filtering

# Spectra over Time

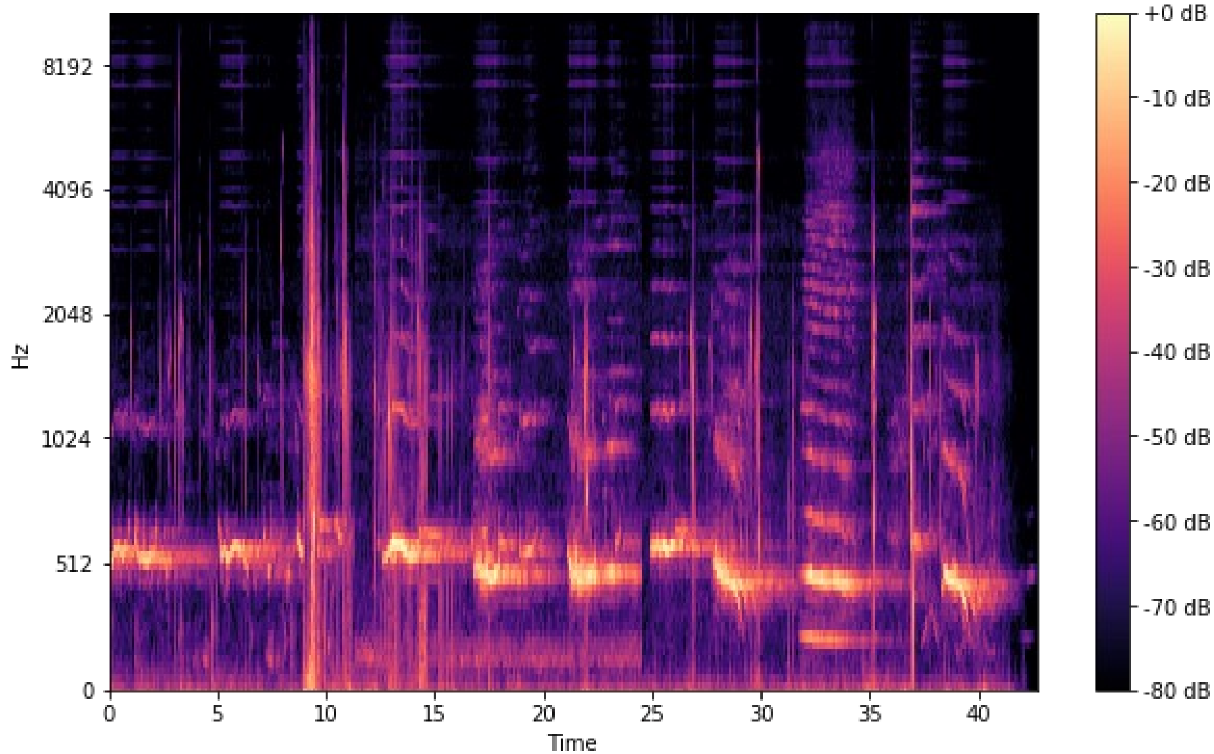
- Even **FFT takes long** to compute **for** a **long signal**
- In **real time** we want frequency information before the signal has ended
- Often, **very low frequencies** are **not of interest**
- Different **spectra over time** are of interest (**one FFT** gives **one spectrum**)

# Short Time Fourier Transform

- **Fourier transform** performed on **time windows**, i.e. short parts of the signal
- **Sequence** of window **spectra** describes the spectral development over time
- Result: **2-dimensional** structure
  **time vs. frequency**

# Spectrogram

- The resulting 2D image is called a **spectrogram**.



# Window Length

- The **length** of the **window** determines the
  - **Lowest** represented **frequency**
  - **Resolution** of the **spectrum**
- **Trade-off** between **time** and **frequency resolution** (Heisenberg's uncertainty principle)
  - **Long windows** offer more information in the spectra (better **frequency resolution**).
  - **Short windows** offer more information on the time-scale (better **time resolution**).

# Problems with Windowing

- Simplest window: **rectangular**

$$w_R(n) = \begin{cases} 1 & n = 0, \ldots, R-1 \\ 0 & elsewhere \end{cases}$$

- **Fourier Transform requires periodic** signals.
- **Signals** are **usually not periodic**, or not **at window length** period.

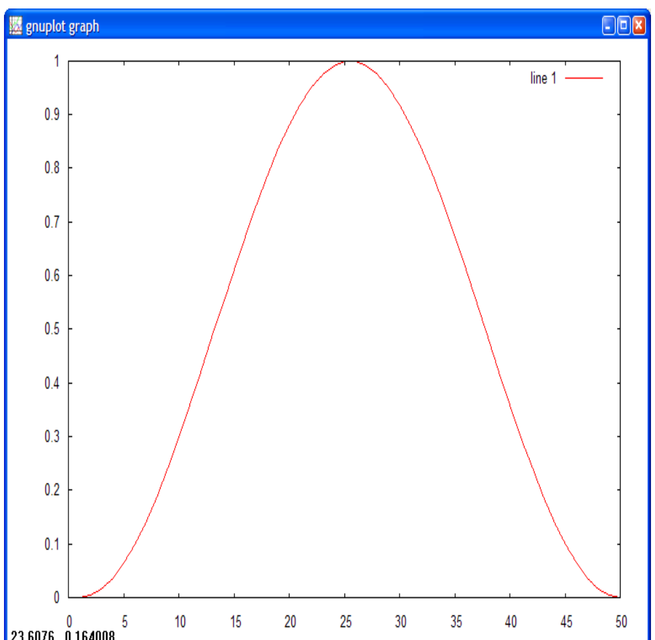
# Better Windows

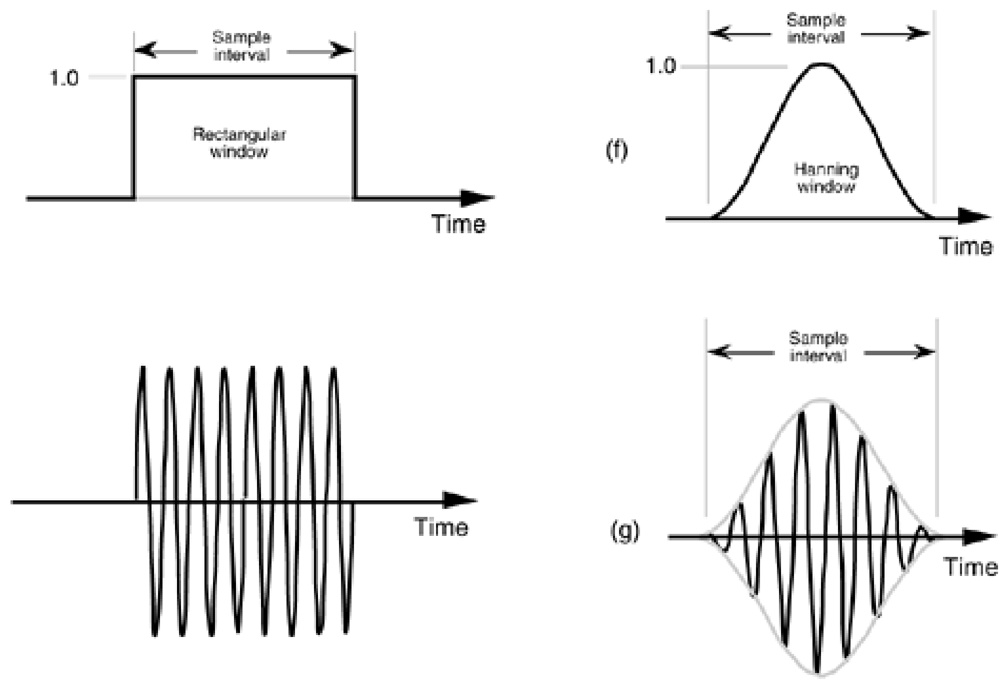
- A **window** function that **fades to 0 smoothly avoids problems** with non-periodic signals.
- Smooth window **reduces frequency resolution**.
- **Common choice**: the **Hann** window, defined as:

$$w[n] = \frac{1}{2} - \frac{1}{2}\cos(2\pi n/N)$$

# Hann Window

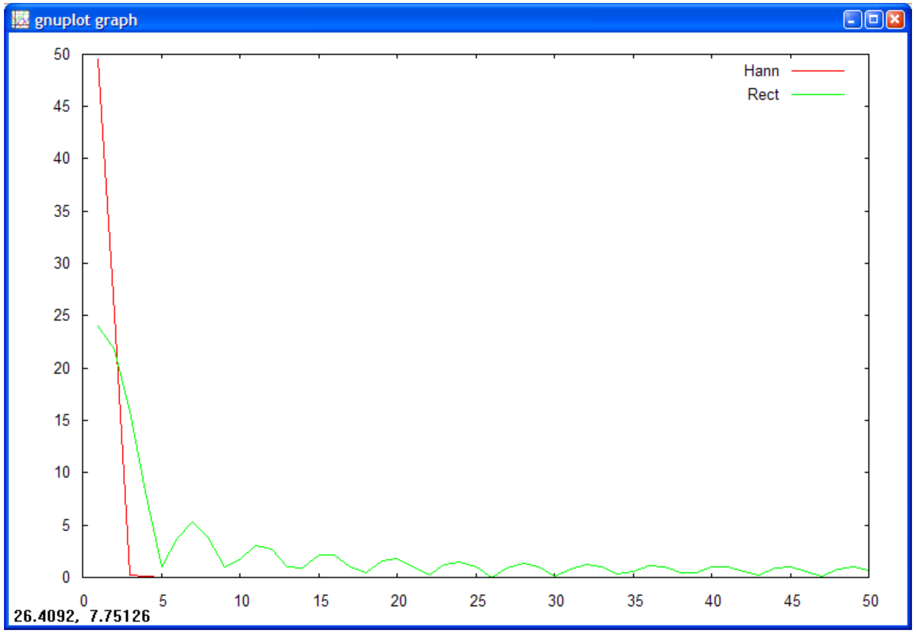
- The Hann window is available in Python (SciPy) as `scipy.signal.windows.hann(N)`



# Spectrum Leaking

- The FT of the window function shows how the **windowed spectrum 'leaks'** for different window functions (red: Hann, green: Rectangular).



# Python Functions for FFT

Get the **spectrum** (**complex** numbers)

`spec = `**`scipy.fft.rfft`**`(sig)` or **`scipy.fft.fft`**`(sig)`

Use **`np.abs`**`(spec)` for **magnitude** and **`np.angle`**`(spec)` for **phase**.

Apply **`np.real`**`()` and **`np.imag`**`()` if needed (rare).

Get the signal back with

`sig = `**`scipy.fft.irfft`**`(spec)` or **`scipy.fft.ifft`**`(sig)`

# Resynthesis from the Spectrogram

# Resynthesis

- From the STFT we can **return** to the **signal** by **iFFT** on **every window**.
- ... but we **can change** it before doing that :-)
- interesting applications:
  - **equalizing**: just amplify or damp
  - **denoising**: subtract the noise-floor
  - **watermarking**: imprint a specific pattern
  - **vocoding**: transferring spectrum peaks

# FFT Normalisation

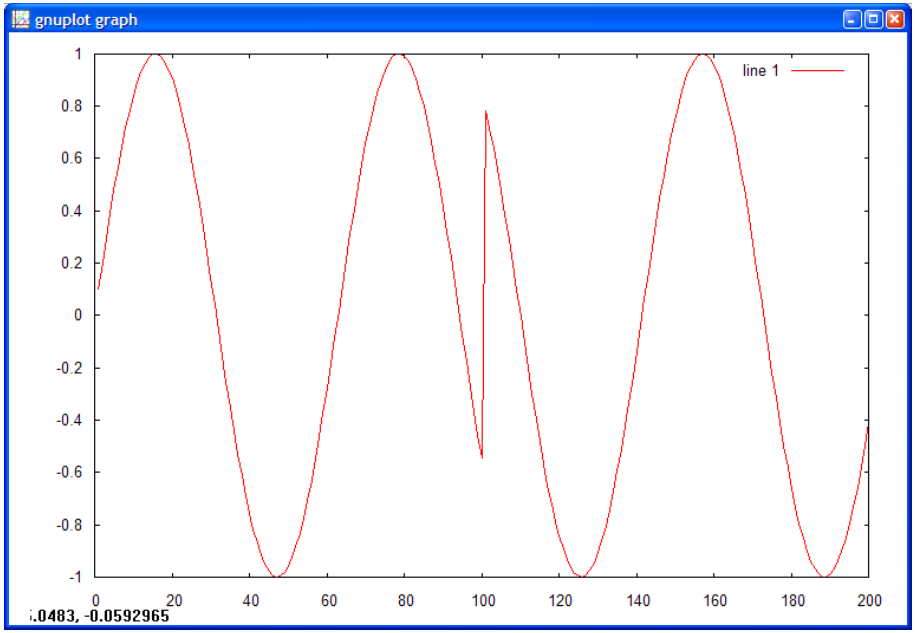
- The **signal** (*s*) has the same **energy** as the **spectrum** (S) divided by $\sqrt{N}$. $\sum s^2 = \sum \left( \frac{S}{\sqrt{N}} \right)^2$ (this is called ***Parseval's theorem***)
- Alternatively, divide the power spectrum (i.e. S.^2 ) by N to achieve normalisation: $\sum s^2 = \frac{1}{N} \sum S^2$
- The whole **FFT / iFFT** cycle produces a multiplication by N which is **normally cancelled only** in the **iFFT** for efficiency reasons.

# Reconstruction after Windowing

- **Windowing** means **y[n] = x[n]*w[n]**
- **Reverse** after STFT by **x'[n] = y[n]/w[n]**
- Problems:
  - **rounding errors** near the margins
  - **large values** near the margins **if** the **spectrum was processed**
- Idea: **let windows overlap**

# Getting the Joints Right

- Windows must **overlap** to **get good quality**.
- If the **signal** was **changed**, there will be **clicks:**

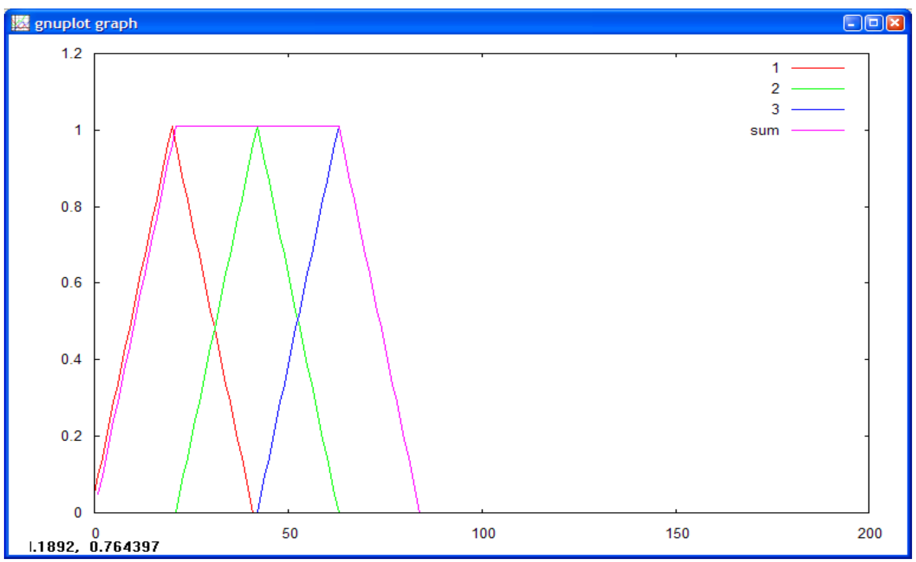


# Windowing and Crossfading

- **Crossfading**: blending from one signal to another.
- Idea: use **overlapping** windows, that fade in and out in the overlap region.
- Window should be designed so that values **add up to 1** in the **overlap range**.

# Crossfading Functions

- Condition: Function ranges $\boldsymbol{w}_{in}$ and $\boldsymbol{w}_{out}$ should **add up to 1** in the **overlap**.
- E.g. **triangle** waves or a **Hann** window:



# Convolution and Digital Filtering

# Convolution

- **Convolution** combines two signals, **similarly** to **cross-correlation**
  - it's the correlation with a reversed signal

$$conv(s1, s2)[t1] = \sum_{t=0}^{N2-1} s1[t1-t]\, s2[t]$$

  N2 is the length of s2, s1[i] = 0 assumed where i<0 or i>=N

- Often **written** as **s1 * s2**

# Properties

- Convolution is **commutative**:
  s1 * s2 = s2 * s1
- Convolution is also **associative:**
  (x * y) * z = x * (y * z)
- The **length** of s1 * s2 is **N1 + N2 -1**

# Convolution Example

```
    s1 = [1,0,2,3,0,1]  s2 = [2,0,1]
    1,0,2,3,0,1
1 0 2                2   (0·1 + 0·0 + 1·2)
  1 0 2                0   (0·1 + 1·0 + 0·2)
    1 0 2                5   (1·1 + 0·0 + 2·2)
      1 0 2                6   (0·1 + 2·0 + 3·2)
        1 0 2                2   (2·1 + 3·0 + 0·2)
          1 0 2                5   (3·1 + 0·0 + 1·2)
            1 0 2                0   (0·1 + 1·0 + 0·2)
              1 0 2                1   (1·1 + 0·0 + 0·2)
         s1 * s2 = [2,0,5,6,2,5,0,1]
```

# Convolution Theorem

- The most **important** property of the convolution is given by the **convolution theorem**:
  *A **convolution** in the **time domain***
  *is **equivalent** to*
  *a **multiplication** in the **frequency domain**:*

$$x * y \circ\!\!-\!\!\bullet X \cdot Y$$

$$meaning:\ FT(conv(x, y)) = FT(x) \cdot FT(y)$$

# Digital Filters

- Sound **spectra** are **changed** by **filters**
- **STFT** manipulation and resynthesis - a form of **filtering in the frequency domain**
- **Most filtering** happens in the **time domain** by **convolution**

# Linear Filters

- Linear filters **sum scaled and delayed copies** of the signal to itself (**convolution** with the **scaling factors**)
- **2 types**, depending on where they take the signal from
  - **Finite Impulse Response** (**FIR**) filters (use input signal)
  - **Infinite Impulse Response** (**IIR**) filters (use input & output signal)

# The Order of Filters

- An **FIR** filter ***f*** of **order *k*** has this **structure**
  $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$

- An **IIR** filter *g* of **order *k*** has this **recursive** structure
  $g(x[n]) = + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$
  $- a_1 g(x[n-1]) - a_2 g(x[n-2]) - \ldots - a_k g(x[n-k])$

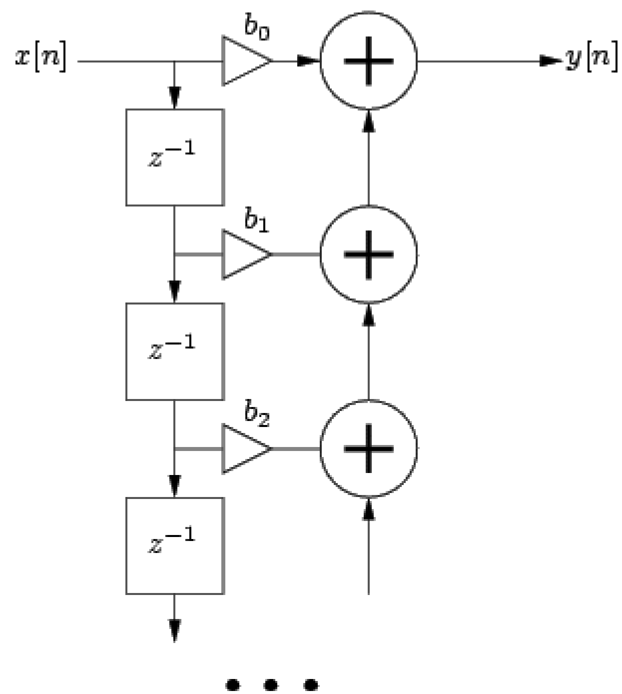
- or as a **difference equation**

$$y[n] = -\sum_{i=1}^{k} a_i y[n-i] + \sum_{i=0}^{k} b_i x[n-i]$$

- $\mathbf{a_n}$ and $\mathbf{b_n}$ are called **filter coefficients**

# An FIR Filter

- Am **FIR** filter *f* of order *k* has this **structure**
  $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$
  with **coefficients** $b = [b_0, b_1, b_2, \ldots, b_k]$
- **Graphically**:




**$z^{-1}$**: delay by 1 sample

# How does an FIR Filter work ?

- $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$ with **coefficients b = [$b_0, b_1, b_2, \ldots, b_k$]**
- **FIR** is just the **convolution x * b**

$$f(x[n]) = \sum_{i=0}^{k} x[n-i]\, b[i]$$

- a small **example**:

```
x = [2,3]   b = [1,2]
    2,3
2   1              2
    2  1              7
       2  1              6
f(x) = [2,7,6]
```

# Impulse Response

- Impulse Response is **system output** in response to a **unit impulse** [1,0,0,...].
- It **completely describes** the **behaviour** of a **linear filter** (or linear **system**) because
  - **every signal** can be described as a **sum of differently scaled unit impulses** (one per sample)
  - the **system's** signal **response** is then a **sum** of the differently **scaled impulse responses**
- Remember: a **linear system** satisfies the **superposition principle** f(ax[n]+by[y]) = a f(x[n]) + b f(y[n])

# The Impulse Response of an FIR Filter

- The **impulse response** of a **FIR** filter $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$ **is** $h = [b_0, b_1, b_2, \ldots, b_k]$, i.e. the **coefficients**

- An **example:**

```
    s1 = [1,0,0]  h = [2,0,1]
    1,0,0
1 0 2           2          f(x[0])
  1 0 2           0        f(x[1])
    1 0 2           1      f(x[2])
    f(s1) = [2,0,1]
```

# The Frequency Response of an FIR Filter

- The **frequency response** of an FIR filter describes its **effect on** different **frequency components** of a signal.
- From the **convolution theorem** we know, that the **convolution** in the **time** domain leads to a **multiplication** in the **frequency** domain: $x * h \circ\!\!-\!\!\bullet X \cdot H$
- **H** is called the (complex) **frequency response** of the filter
  - `np.abs(H)` gives the **amplitude response**
  - `np.angle(H)` gives the **phase response**

# FIR Examples

- **b = [1,0,0,0]** does **not change** the signal, its frequency response is [1,1,1,1]
- **b = [0,1,0,0] delays** by one sample, its frequency response is [1,-i,-1,i] (**magnitude stays**, but **phase changes**)
- **b = [1,1,1,1]** (**averaging filter**) lets only the low frequencies pass, its frequency response is [1,0,0,0]

- **Frequency resolution** depends on the **length** of the **filter**
- We can **calculate** filter **coefficients** for a given frequency response H by using `scipy.fft.irfft(H)`

# Python Functions for Filters

For simple FIR filtering you can use convolution

```
y = scipy.signal.convolve(b, x)
```

The filter function lets you add IIR and truncates the output

```
y = scipy.signal.lfilter(b, a, x)
```

You can get the frequency response like this:

`scipy.signal.freqz(b)` or `scipy.signal.freqz(b,a)`

And the circular convolution treats the signals as periodic

```
y = dsp_ap.operations.circ_convolve(b, x)
```

(for an exact match between freq and time domain filtering)

# Take-Home Messages

- **Short Term Fourier Transform** (STFT) for non-stationay (i.e. time varying) signals (spectrograms)
- Fourier Transform (FT) assumes **periodicity**,
  - artefacts (**spectral leakage**) for non-periodic signals
  - can be improved (but not avoided) with **Hann window**
- **Convolution** filtering (**FIR**) modifies frequency mix
- **Convolution theorem** → TD conv equiv to FT mult
- **Recurrent** filtering (**IIR**) is more **efficient**, **hard to control**
- **Impulse response** defines linear filter fully

# READING

http://www.dspguide.com/ Chapter 6 and 14.

# NEXT WEEK:
# Audio Programming
# More Filtering