The Short-Time Fourier Transform

Xavier Serra
Music Technology Group
Universitat Pompeu Fabra, Barcelona
http://mtg.upf.edu

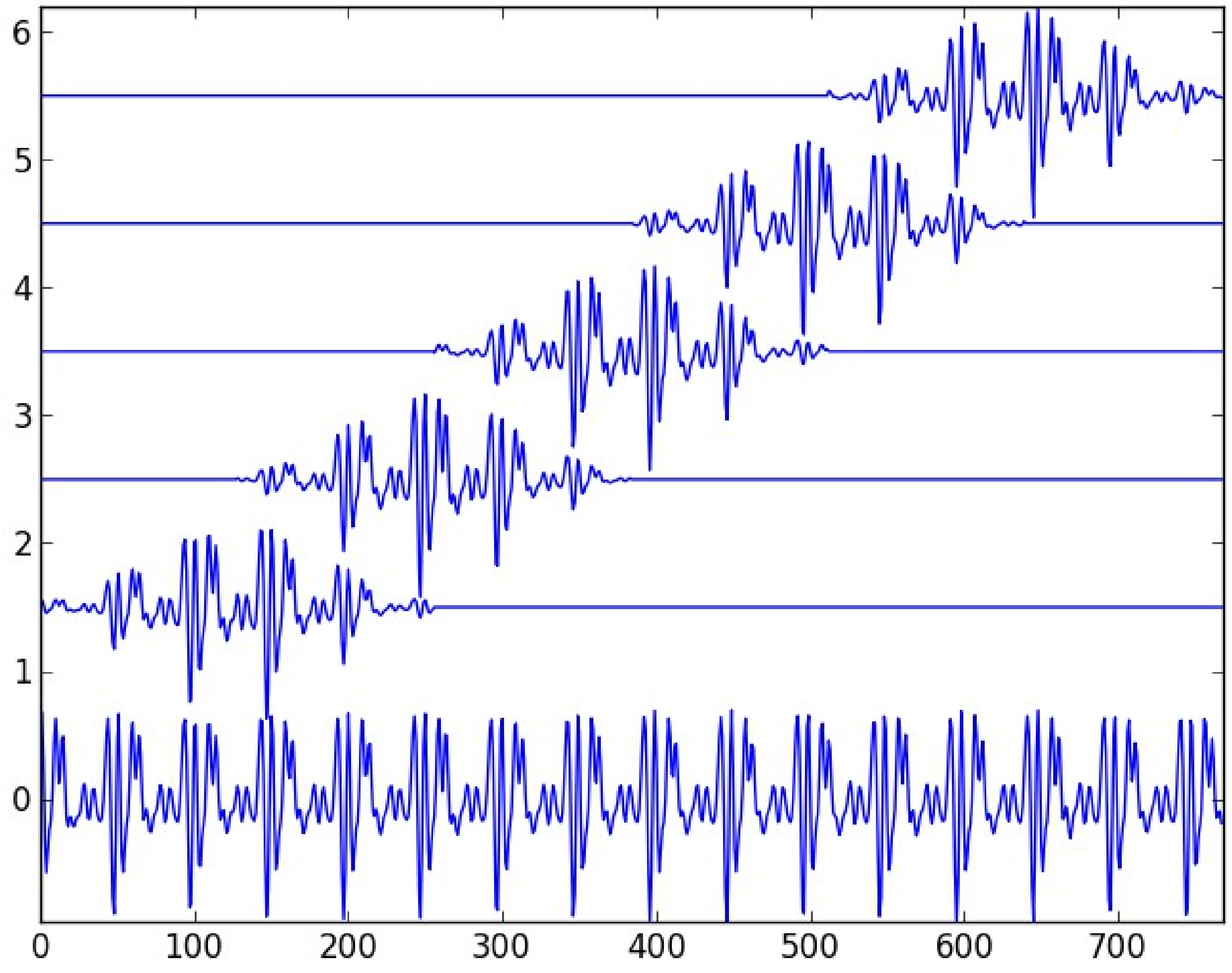
Index

Short-time Fourier Transform

$$X_l[k] = \sum_{n=0}^{N-1} w[n]\, x[n+lH]\, e^{-j2\pi kn/N} \qquad l = 0, 1, \ldots,$$

w: real window
l: number of frame
H: time advance of window (hop-size)

$$w[n]\,x[n+lH] \qquad l=0,1,\ldots,$$



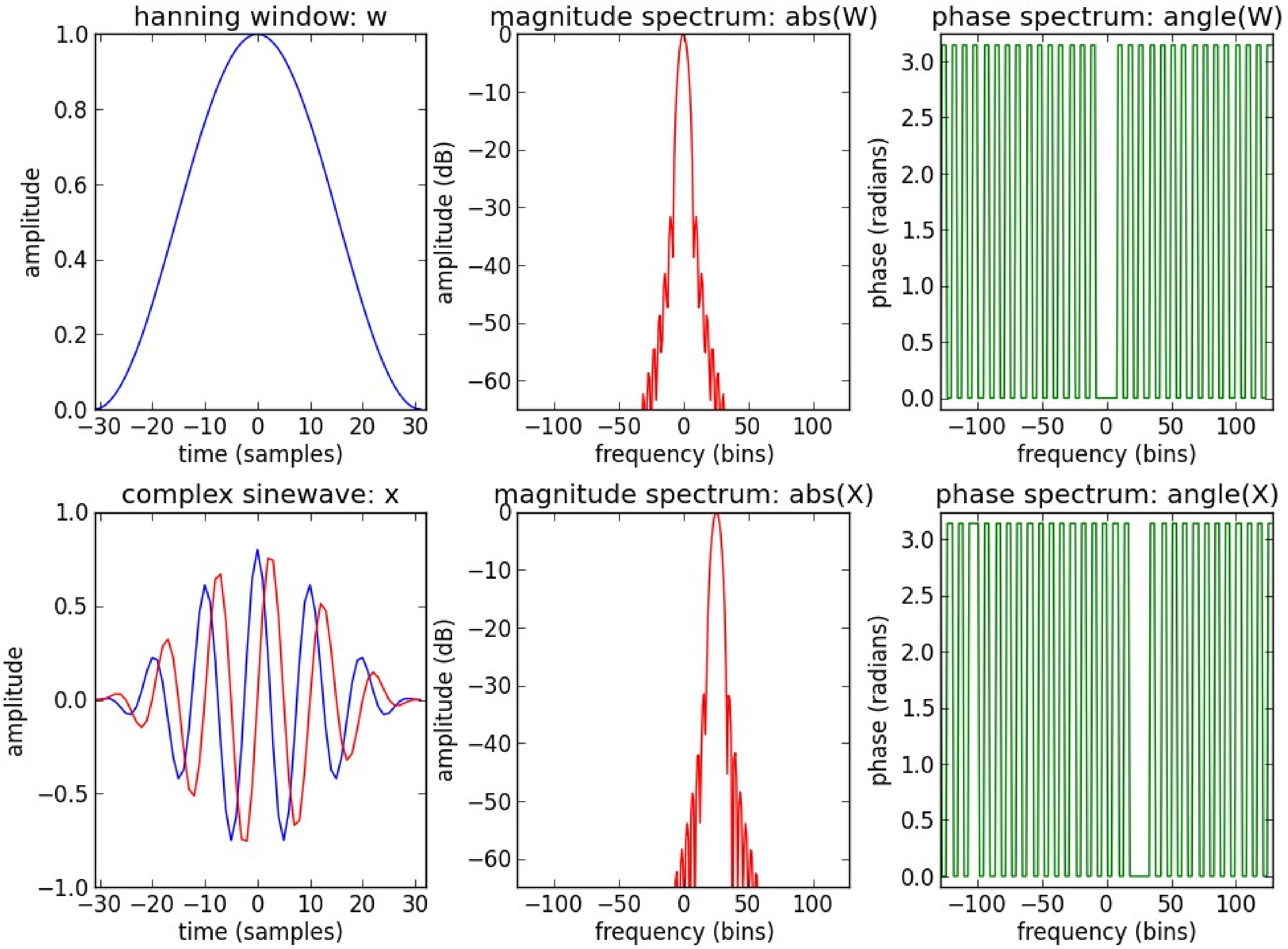
Transform of a windowed sinewave

$$x[n] = A e^{-j 2\pi k_0 n / N}$$

$$X[k] = A \sum_{n=0}^{N-1} w[n] e^{-j 2\pi k_0 n / N} e^{-j 2\pi k n / N}$$

$$= A \sum_{n=0}^{N-1} w[n] e^{-j 2\pi (k+k_0) n / N}$$

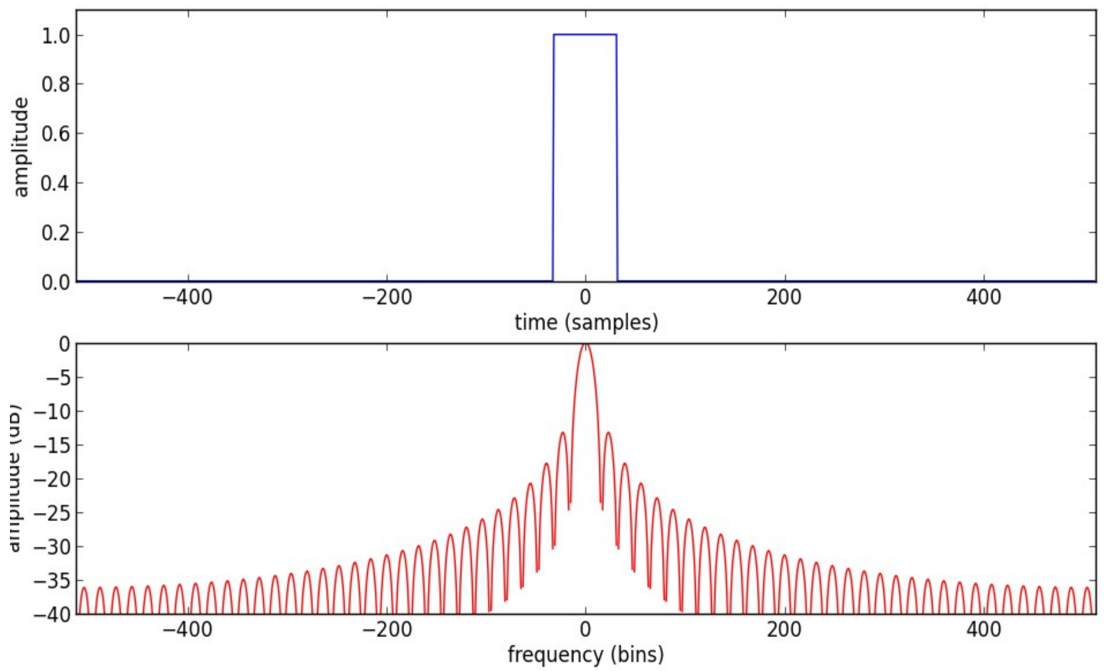
$$= A W[k + k_0]$$


hanning window: w
magnitude spectrum: abs(W)
phase spectrum: angle(W)
complex sinewave: x
magnitude spectrum: abs(X)
phase spectrum: angle(X)
amplitude
amplitude (dB)
phase (radians)
time (samples)
frequency (bins)

```
N = 256
M = 63
f0 = 1000
fs = 10000
A0 = .8
hN = N/2
hM = (M+1)/2
x = A0 * np.exp(1j*2*np.pi*f0/fs*np.arange(-hM+1,hM))

w = np.hanning(M)
plt.subplot(2,3,1)
plt.plot(np.arange(-hM+1, hM), w, 'b')

fftbuffer[:hM] = w[hM-1:]
fftbuffer[N-hM+1:] = w[:hM-1]
X = fft(fftbuffer)
X1[:hN] = X[hN:]
X1[N-hN:] = X[:hN]
mX = 20*np.log10(abs(X1))
plt.subplot(2,3,2)
plt.plot(np.arange(-hN, hN), mX-max(mX), 'r')

pX = np.angle(X1)
plt.subplot(2,3,3)
plt.plot(np.arange(-hN, hN), pX, 'g')

plt.subplot(2,3,4)
xwreal = np.real(x)*w
xwimag = np.imag(x)*w
plt.plot(np.arange(-hM+1, hM), xwreal, 'b')
plt.plot(np.arange(-hM+1, hM), xwimag, 'r')

fftbuffer[0:hM] = np.vectorize(complex)(xwreal[hM-1:],xwimag[hM-1:])
fftbuffer[N-hM+1:] = np.vectorize(complex)(xwreal[:hM-1], xwimag[:hM-1])
X = fft(fftbuffer)
X2[:hN] = X[hN:]
X2[N-hN:] = X[:hN]
mX2 = 20*np.log10(abs(X2))
plt.subplot(2,3,5)
plt.plot(np.arange(-hN, hN), mX2-max(mX2), 'r')

pX = np.angle(X2)
plt.subplot(2,3,6)
plt.plot(np.arange(-hN,hN), pX, 'g')
```

Window type

All standard windows are real and symmetric and have a frequency spectrum with a sinc-like shape.



rectangular window

magnitude spectrum

The choice is mainly determined by two of the spectrum’s characteristics:

1. Width of main lobe.
2. Highest side-lobe level.

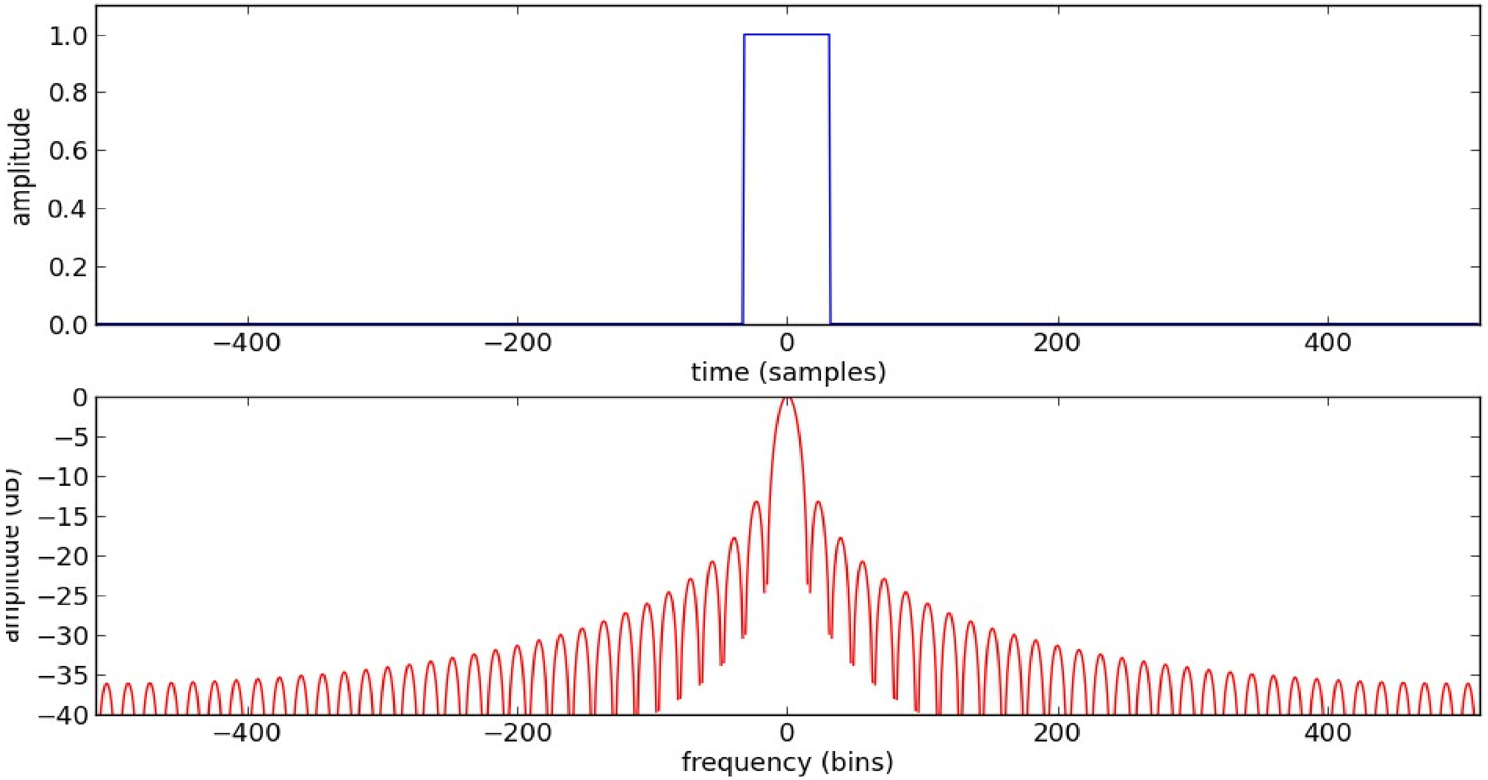
Window functions in Scipy

barthann (M[, sym])	Return a modified Bartlett-Hann window.
bartlett (M[, sym])	Return a Bartlett window.
blackman (M[, sym])	Return a Blackman window.
blackmanharris (M[, sym])	Return a minimum 4-term Blackman-Harris window.
bohman (M[, sym])	Return a Bohman window.
boxcar (M[, sym])	Return a boxcar or rectangular window.
chebwin (M, at[, sym])	Return a Dolph-Chebyshev window.
flattop (M[, sym])	Return a flat top window.
gaussian (M, std[, sym])	Return a Gaussian window.
general-gaussian (M, p, sig[, sym])	Return a window with a generalized Gaussian shape.
hamming (M[, sym])	Return a Hamming window.
hann (M[, sym])	Return a Hann window.
kaiser (M, beta[, sym])	Return a Kaiser window.
nuttall (M[, sym])	Return a minimum 4-term Blackman-Harris window according to Nuttall.
parzen (M[, sym])	Return a Parzen window.
slepian (M, width[, sym])	Return a digital Slepian window.
triang (M[, sym])	Return a triangular window.

Rectangular window

$$w(n) = 1, \qquad n = -M/2, \ldots, 0, \ldots, M/2$$
$$\quad = 0, \qquad \text{elsewhere}$$

$$W(\omega) = \frac{\sin(\omega M/2)}{\sin(\omega/2)}$$



main-lobe width: 2 bins
side-lobe level: -13.3 dB

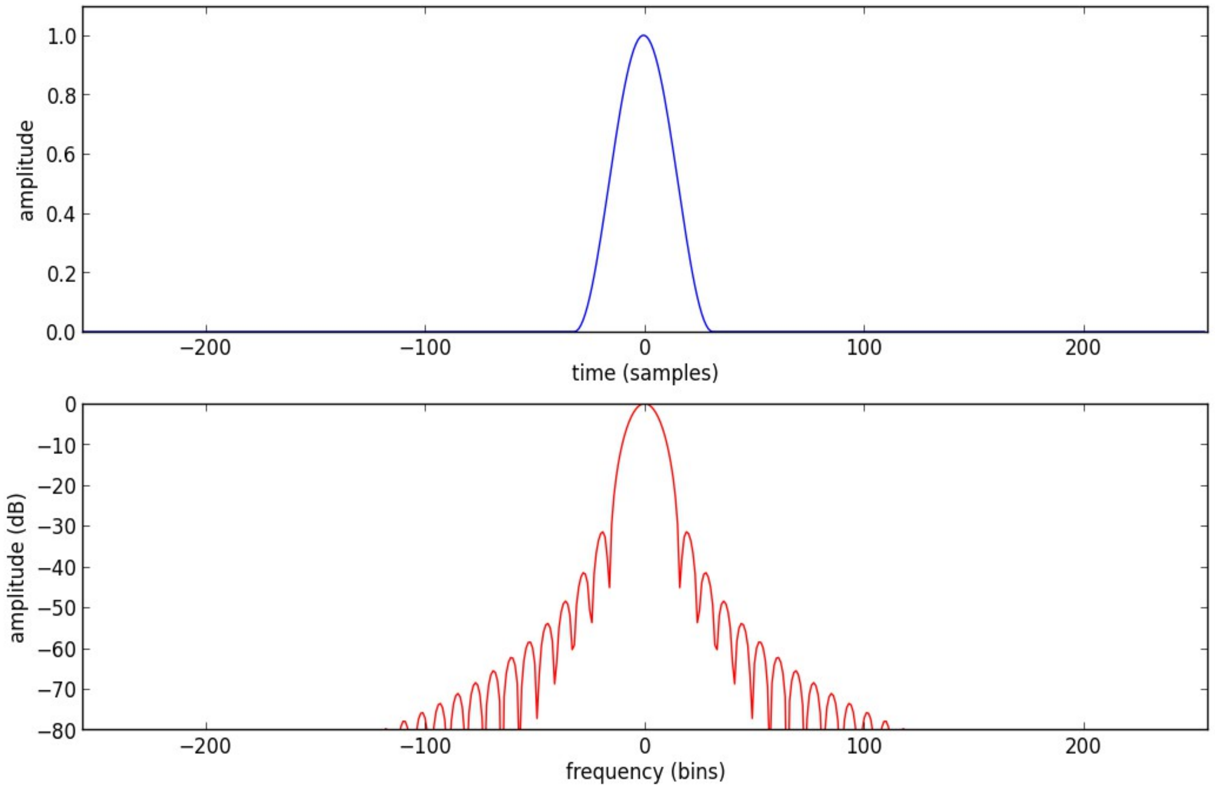
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft

M = 64
N = 1024
hN = N/2
hM = M/2
fftbuffer = np.zeros(N)

fftbuffer[hN-hM:hN+hM]=np.ones(M)
plt.subplot(2,1,1)
plt.plot(np.arange(-hN, hN), fftbuffer, 'b')

X = fft(fftbuffer)
mX = 20*np.log10(abs(X))
mX1[:hN] = mX[hN:]
mX1[N-hN:] = mX[:hN]
plt.subplot(2,1,2)
plt.plot(np.arange(-hN, hN), mX1-max(mX), 'r')
```

Hanning window

$$w(n) = .5 + .5\cos(2n\pi/M),$$
$$n = -M/2,...,0,...,M/2$$

$$W(\omega) = .5D(\omega) +$$
$$.25\left[D(\omega - 2\pi/N) + D(\omega + 2\pi/N)\right]$$

where $D(\omega) = \dfrac{\sin(\omega M/2)}{\sin(\omega/2)}$

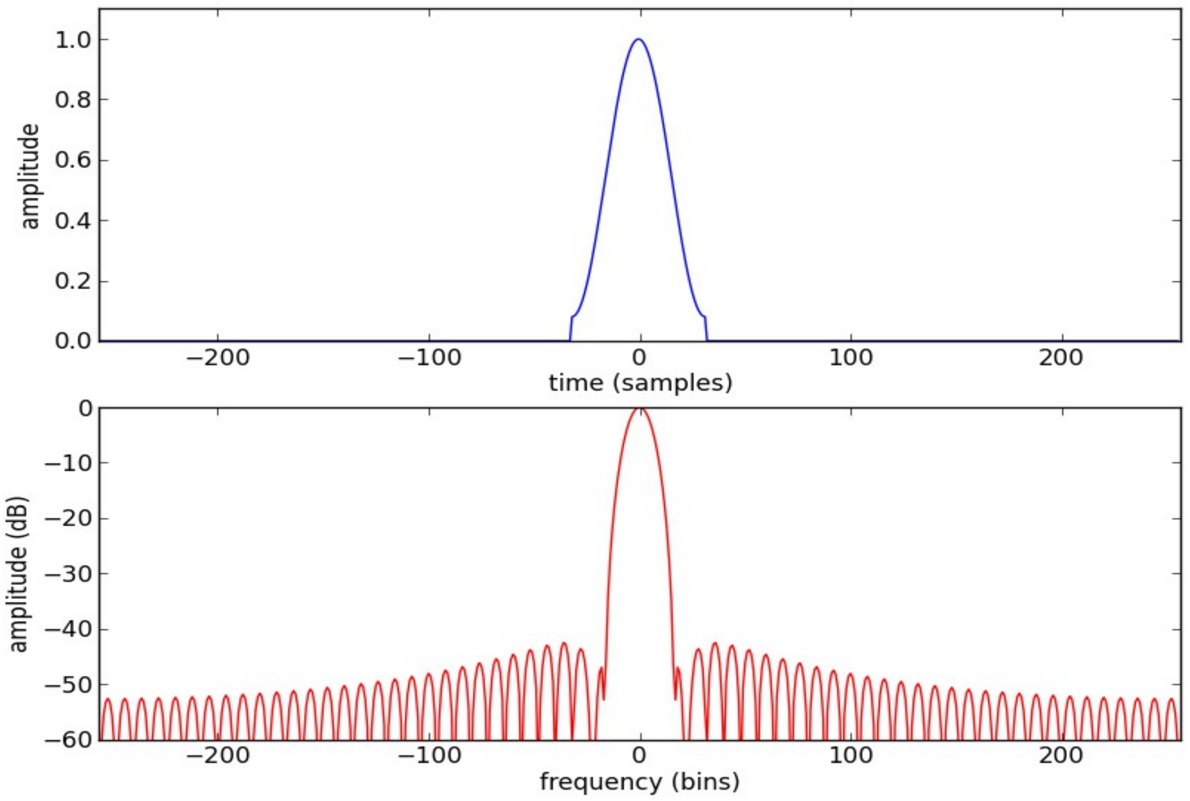


main-lobe width: 4 bins
side-lobe level: -31.5 dB

Hamming window

$$w(n) = .54 + .46\cos(2n\pi/M),$$
$$n = -M/2,\ldots,0,\ldots,M/2$$

$$W(\omega) = .5D(\omega) + .25\left[D(\omega - 2\pi/N) + D(\omega + 2\pi/N)\right]$$



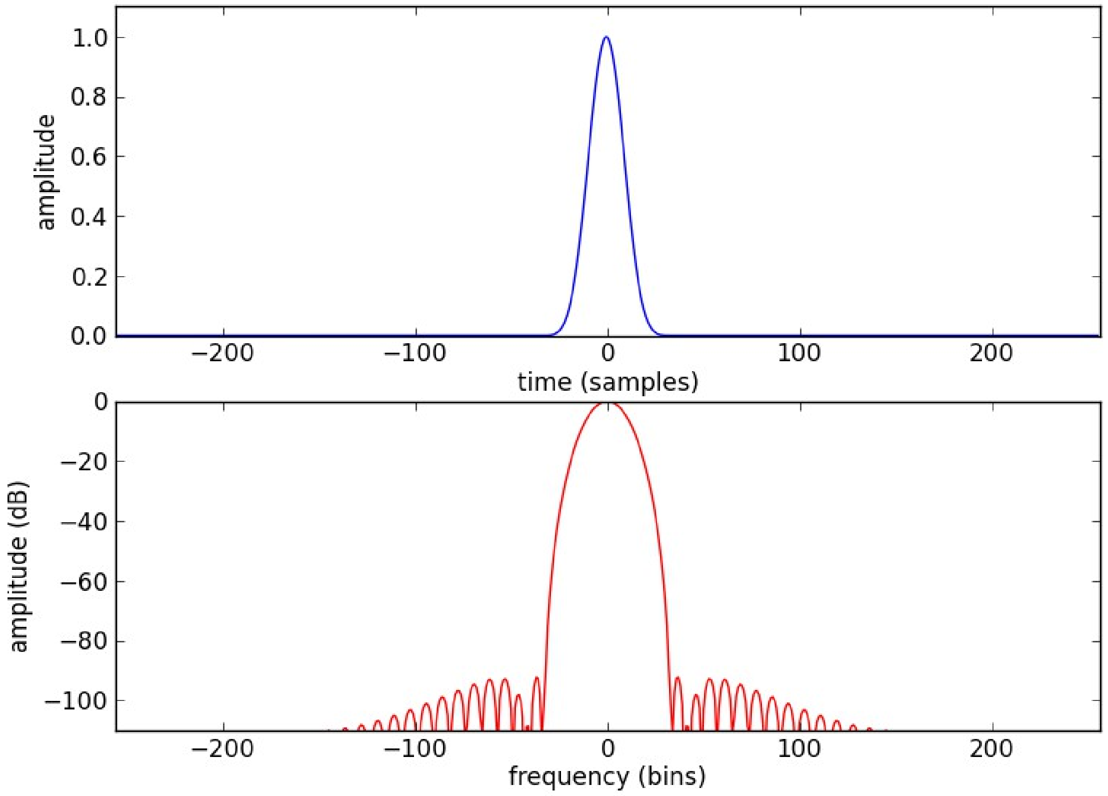
main-lobe width: 4 bins
side-lobe level: -42.7 dB

Blackman-Harris window

minimum 4-term Blackman-Harris window

$$w(n)=\frac{1}{M}\sum_{l=0}^{3}\alpha_l\cos(2nl\pi/M),\quad n=-M/2,\ldots 0,\ldots M/2$$

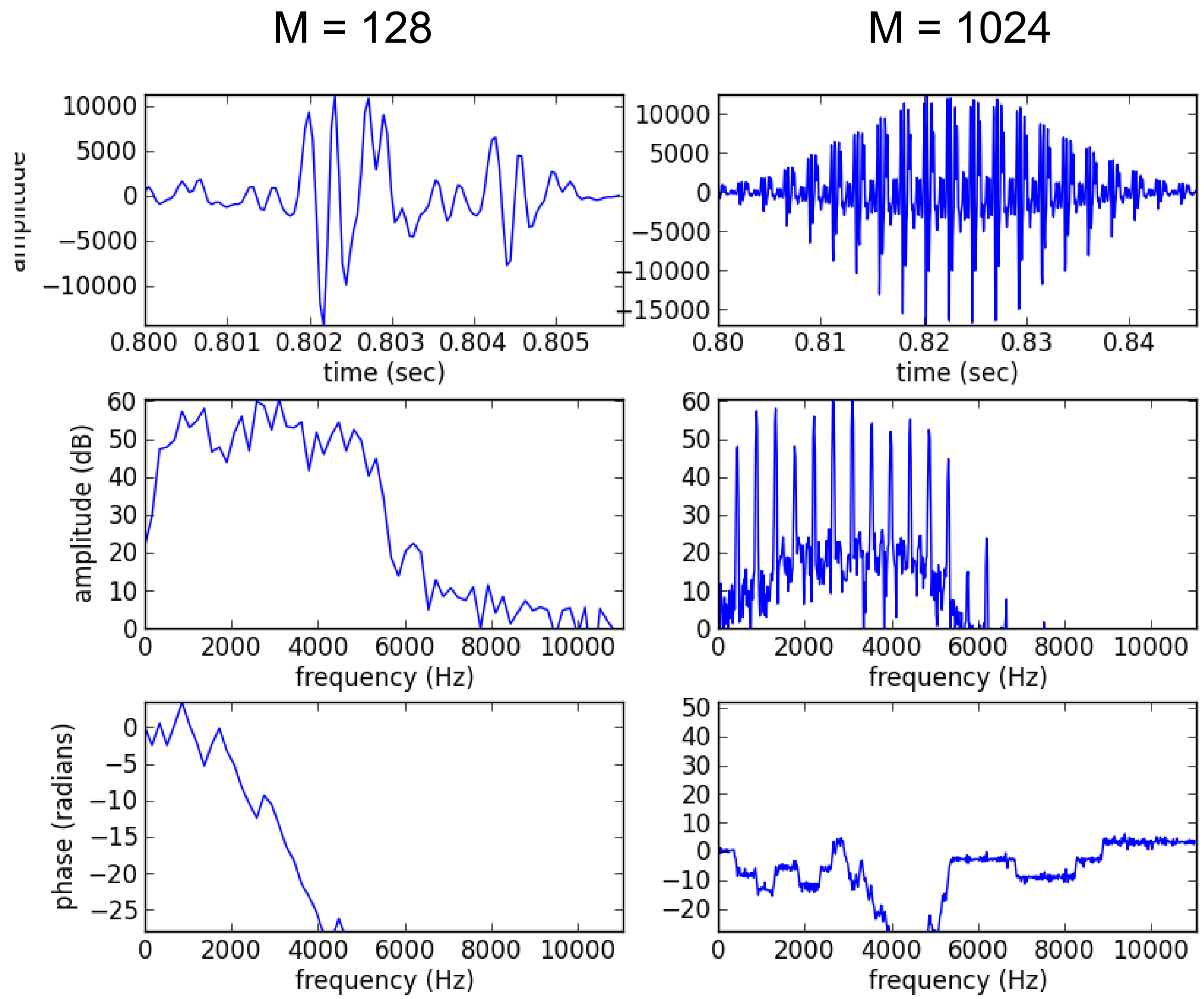
where $\alpha_0=0.35875, \alpha_1=0.48829, \alpha_2=0.14128, \alpha_3=0.01168$



main lobe width $=9$
side-lobe level $=-92$dB

Window size

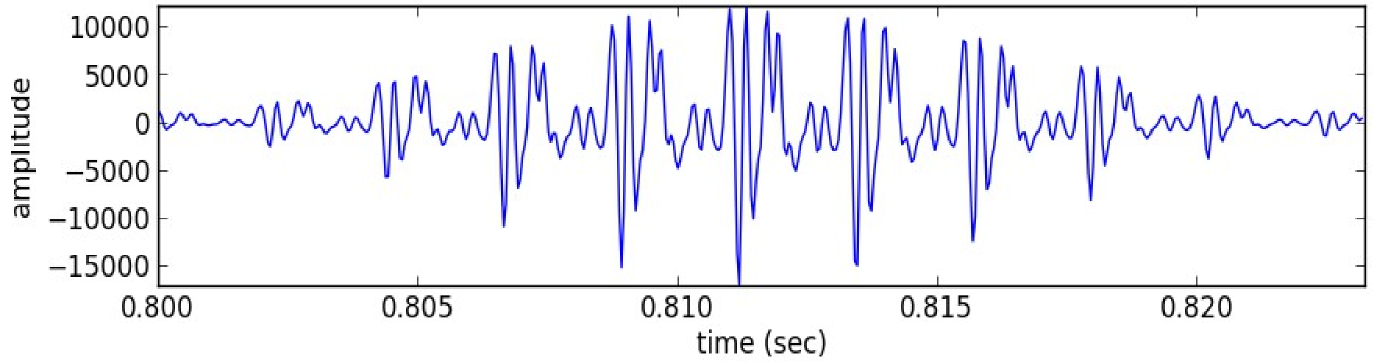
Affects the time versus frequency resolution



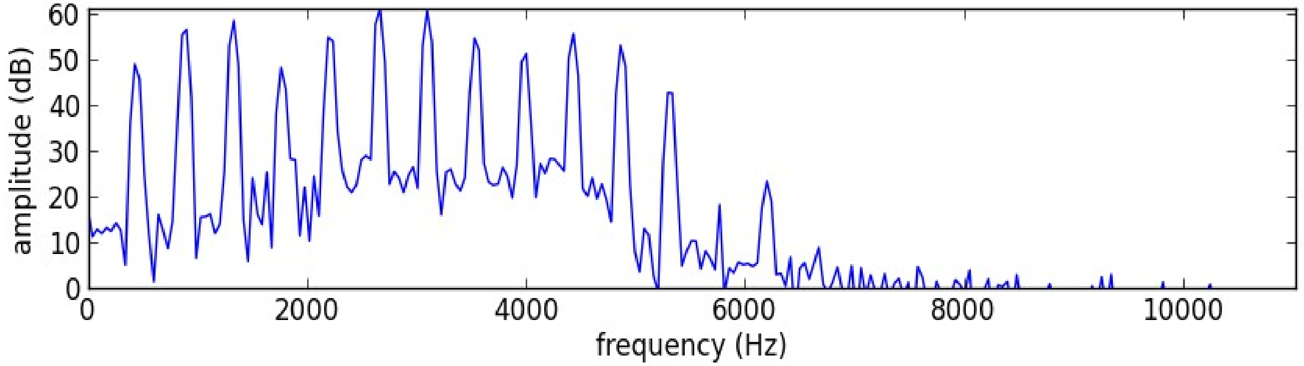
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.wavfile import read
from scipy.fftpack import fft

(fs, x) = read('oboe.wav')
N = 128
start = .8*fs
xw = x[start:start+N] * np.hamming(N)
plt.figure(1)
plt.subplot(321)
plt.plot(np.arange(start, (start+N), 1.0)/fs, xw)

X = fft(xw)
mX = 20 * np.log10(abs(X)/N)
pX = np.unwrap(np.angle(X))
plt.subplot(323)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), mX[0:N/2])

plt.subplot(325)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), pX[:N/2])

N = 1024
start = .8*fs
xw = x[start:start+N] * np.hamming(N)
plt.subplot(322)
plt.plot(np.arange(start, (start+N), 1.0)/fs, xw)

X = fft(xw)
mX = 20 * np.log10(abs(X)/N)
pX = np.unwrap(np.angle(X))
plt.subplot(324)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), mX[0:N/2])

plt.subplot(326)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), pX[:N/2])
```

FFT size

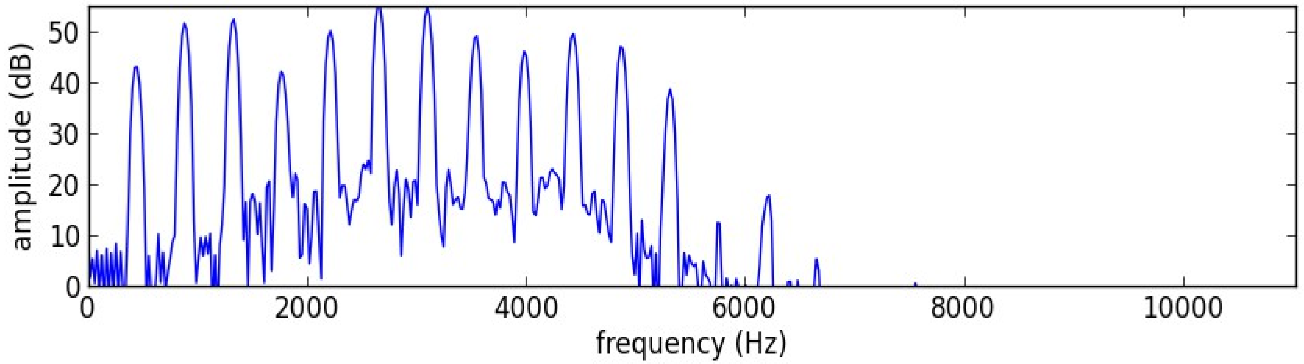
FFT size, N, should be equal or larger than the window size, M



$M = 512$



$M = 512$
$N = 512$



$M = 512$
$N = 1024$

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.wavfile import read
from scipy.fftpack import fft, ifft

(fs, x) = read('oboe.wav')
M = 512
N = 512
start = .8*fs
xw = x[start:start+M] * np.hamming(M)
plt.figure(1)
plt.subplot(311)
plt.plot(np.arange(start, (start+M), 1.0)/fs, xw)

X = fft(xw, N)
mX = 20 * np.log10(abs(X)/N)
pX = np.unwrap(np.angle(X))
plt.subplot(312)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), mX[0:N/2])

M = 512
N = 1024
start = .8*fs
xw = x[start:start+M] * np.hamming(M)
X = fft(xw, N)
mX = 20 * np.log10(abs(X)/N)
plt.subplot(313)
plt.plot(np.arange(0, fs/2.0, float(fs)/N), mX[0:N/2])
```

Hop size

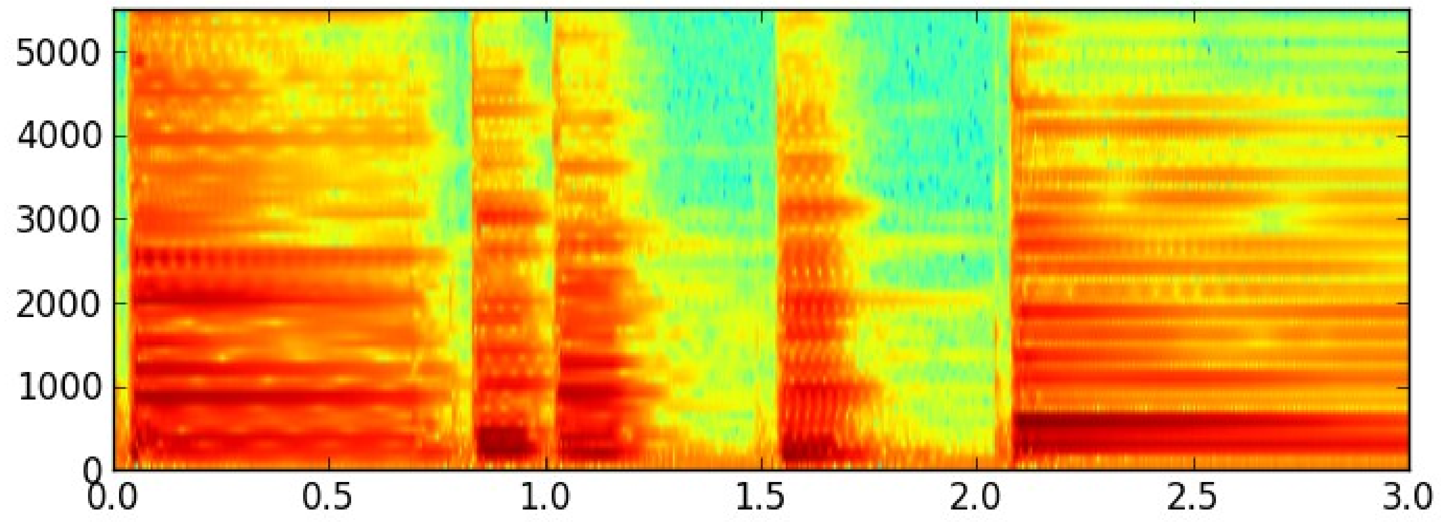
- Successive frames should overlap in time in such a way that all the data are weighted equally.

- For certain windows exists perfect overlap factors. Rectangular: M/j, Hanning and Hamming: (M/2)/j, where j = 0, 1, ...

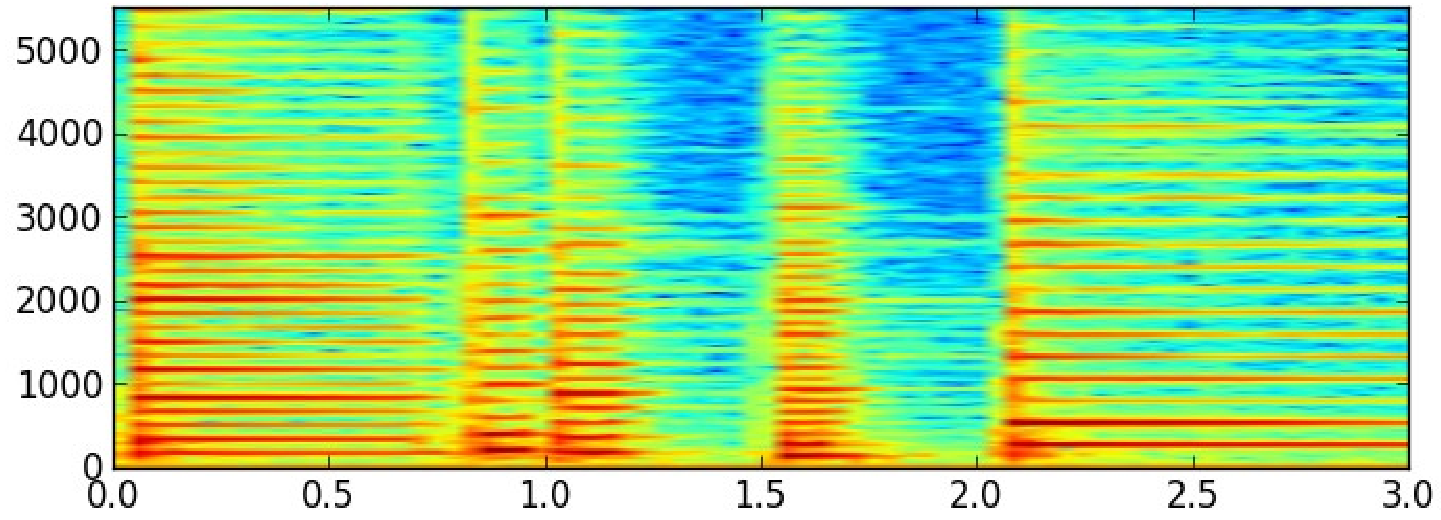
The overlap factor can be expressed by:

$$A_w(m) = \sum_{n=-\infty}^{\infty} w(m - nH) = c$$

Time-frequency compromise



M=226
N=256
H=128



M=1024
N=1024
H=128

```
from pylab import *
from scipy.io.wavfile import read

(fs, x) = read('piano.wav')
figure(1)
subplot(2,1,1)
specgram(x, NFFT=256, Fs=fs, noverlap=128)
axis([0,size(x)/fs,0,fs/4])
subplot(2,1,2)
specgram(x, NFFT=1024, Fs=fs, noverlap=128)
axis([0,size(x)/fs,0,fs/4])
```

Inverse STFT

The overlap-add interpretation of the STFT yields a particular synthesis method.

$$s(n) = \sum_{l=0}^{L-1} Shift_{lH,n}\left[\frac{1}{K}\sum_{k=0}^{K-1} X_l(k) e^{j\omega_k m}\right]$$

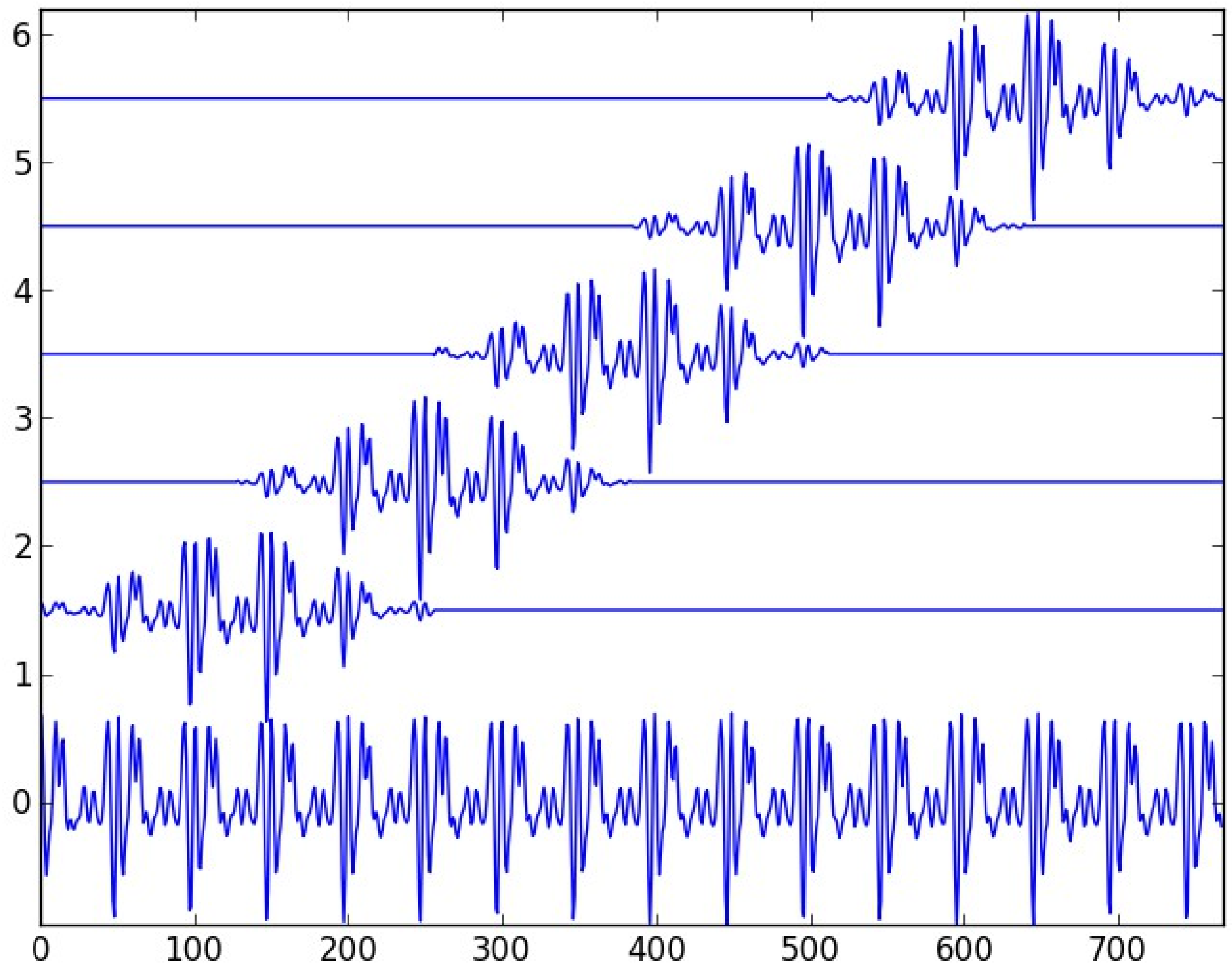
each synthesized frame is:

$$s_l(n) = x(n+lH)\,w(n)$$

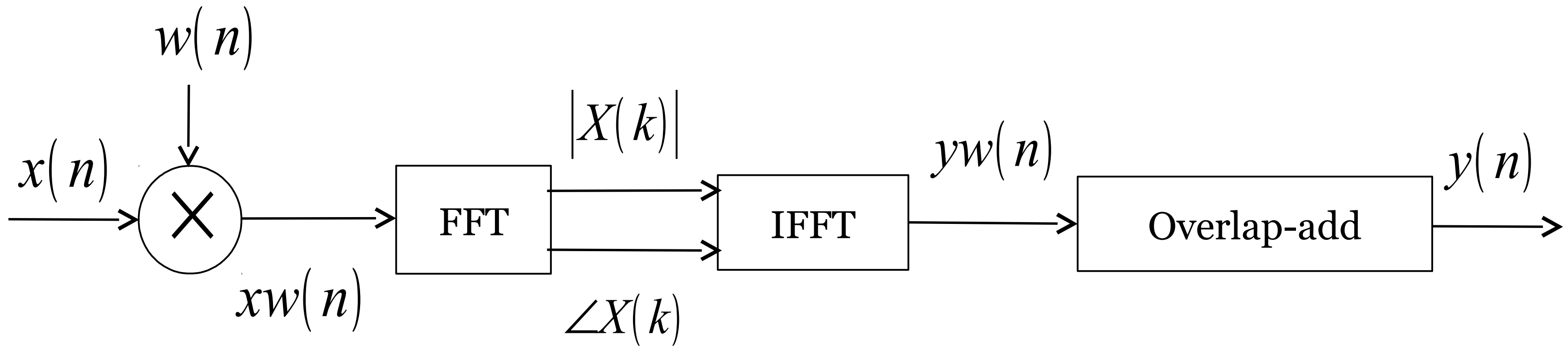
and the synthesized sound is:

$$s(n) = \sum_{l=0}^{L-1} s_l(n) = x(n)\sum_{l=0}^{L-1} w(n-lH)$$

$$w[n]x[n+lH] \qquad l=0,1,\ldots,$$



STFT implementation diagram



```
import numpy as np
from scipy.fftpack import fft, ifft

def stft(x, fs, w, N, H) :
  hN = N/2
  hM = (w.size+1)/2
  pin = hM
  pend = x.size-hM
  fftbuffer = np.zeros(N)
  yw = np.zeros(w.size)
  y = np.zeros(x.size)
  w = w / sum(w)
  while pin<pend:
  #-----analysis-----
    xw = x[pin-hM:pin+hM-1] * w
    fftbuffer = np.zeros(N)
    fftbuffer[:hM] = xw[hM-1:]
    fftbuffer[N-hM+1:] = xw[:hM-1]
    X = fft(fftbuffer)
    mX = 20 * np.log10( abs(X[:hN]) )
    pX = np.unwrap( np.angle(X[:hN]) )
  #-----synthesis-----
    Y = np.zeros(N, dtype = complex)
    Y[:hN] = 10**(mX/20) * np.exp(1j*pX)
    Y[hN+1:] = 10**(mX[:0:-1]/20) * np.exp(-1j*pX[:0:-1])
    fftbuffer = np.real( ifft(Y) )
    yw[:hM-1] = fftbuffer[N-hM+1:]
    yw[hM-1:] = fftbuffer[:hM]
    y[pin-hM:pin+hM-1] += H*yw
    pin += H
  return y
```

References

- https://ccrma.stanford.edu/~jos/sasp/
- https://en.wikipedia.org/wiki/STFT
- https://en.wikipedia.org/wiki/Window_function
- Full code of plots and accompanying labs available at:https://github.com/MTG/sms-tools

Credits