Harmonic Modeling

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

Index

- Harmonic Model
- Sinusoids-Partials-Harmonics
- F0 detection
- Harmonic tracking
- Implementation

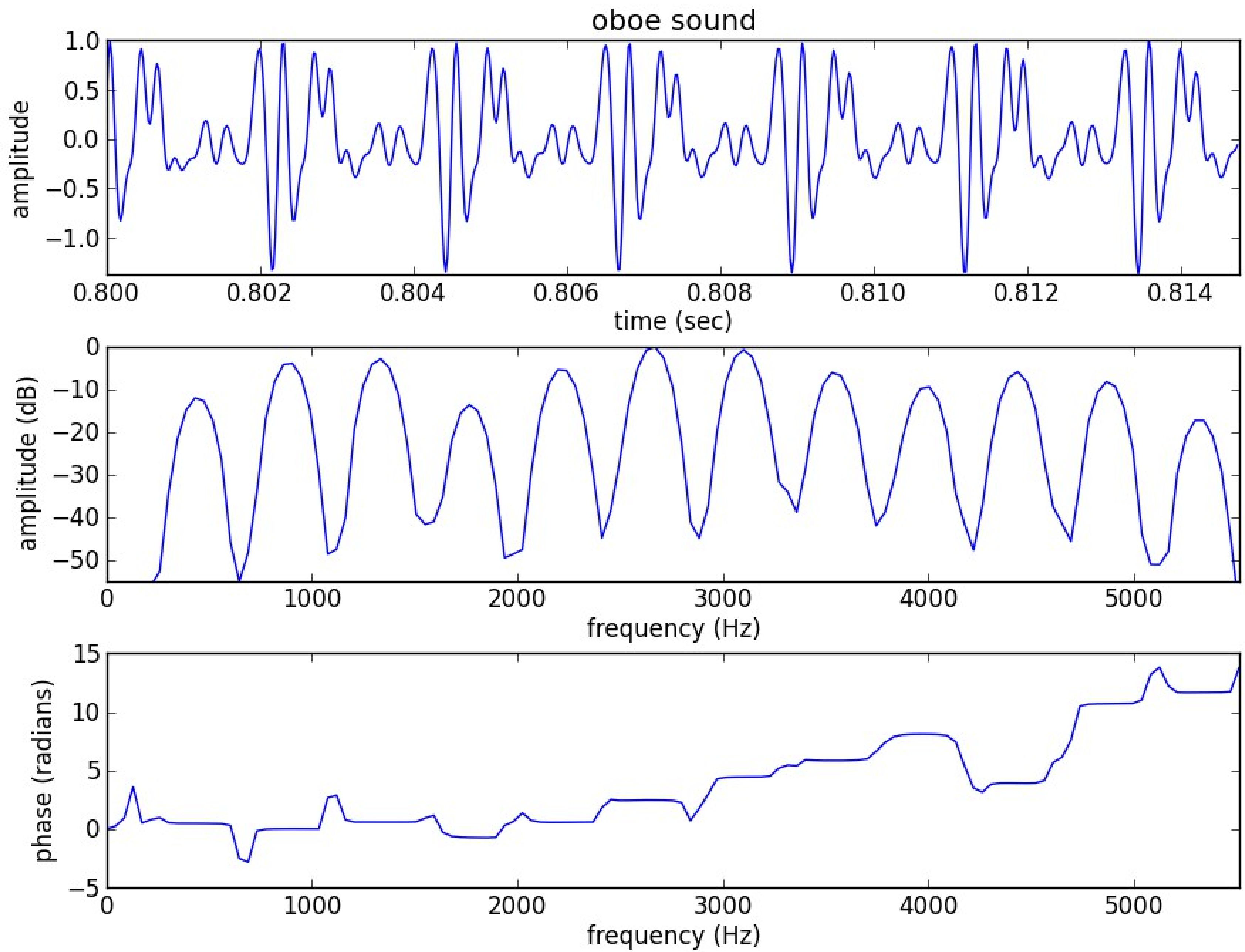
Harmonic model

$$yh[n]=\sum_{k=1}^{K} A_k[n]\cos\left(2\pi k f_0[n]n\right)$$

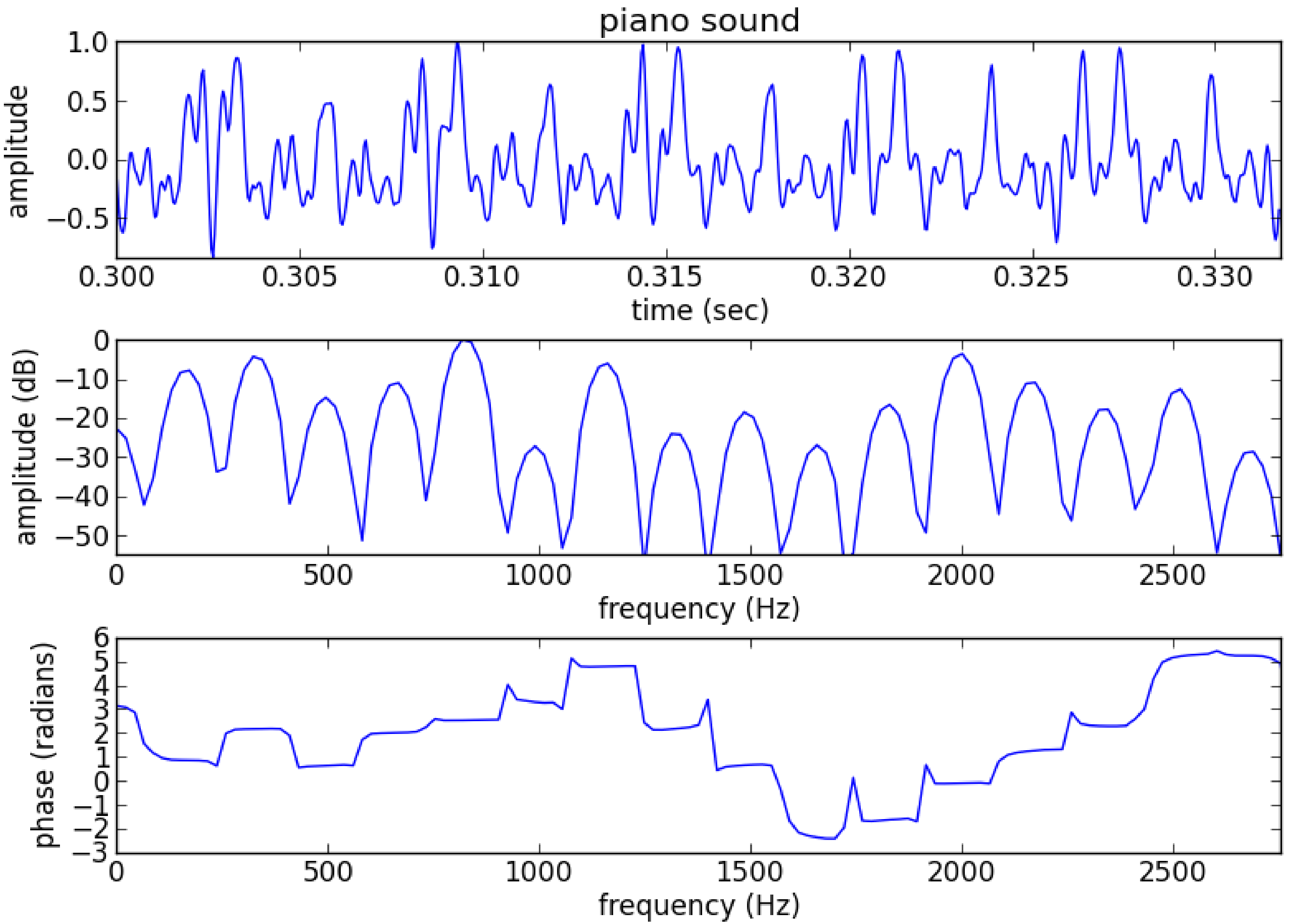
K: number of harmonic components
$A_k[n]$: instantaneous amplitude
$f_0[n]$: fundamental frequency

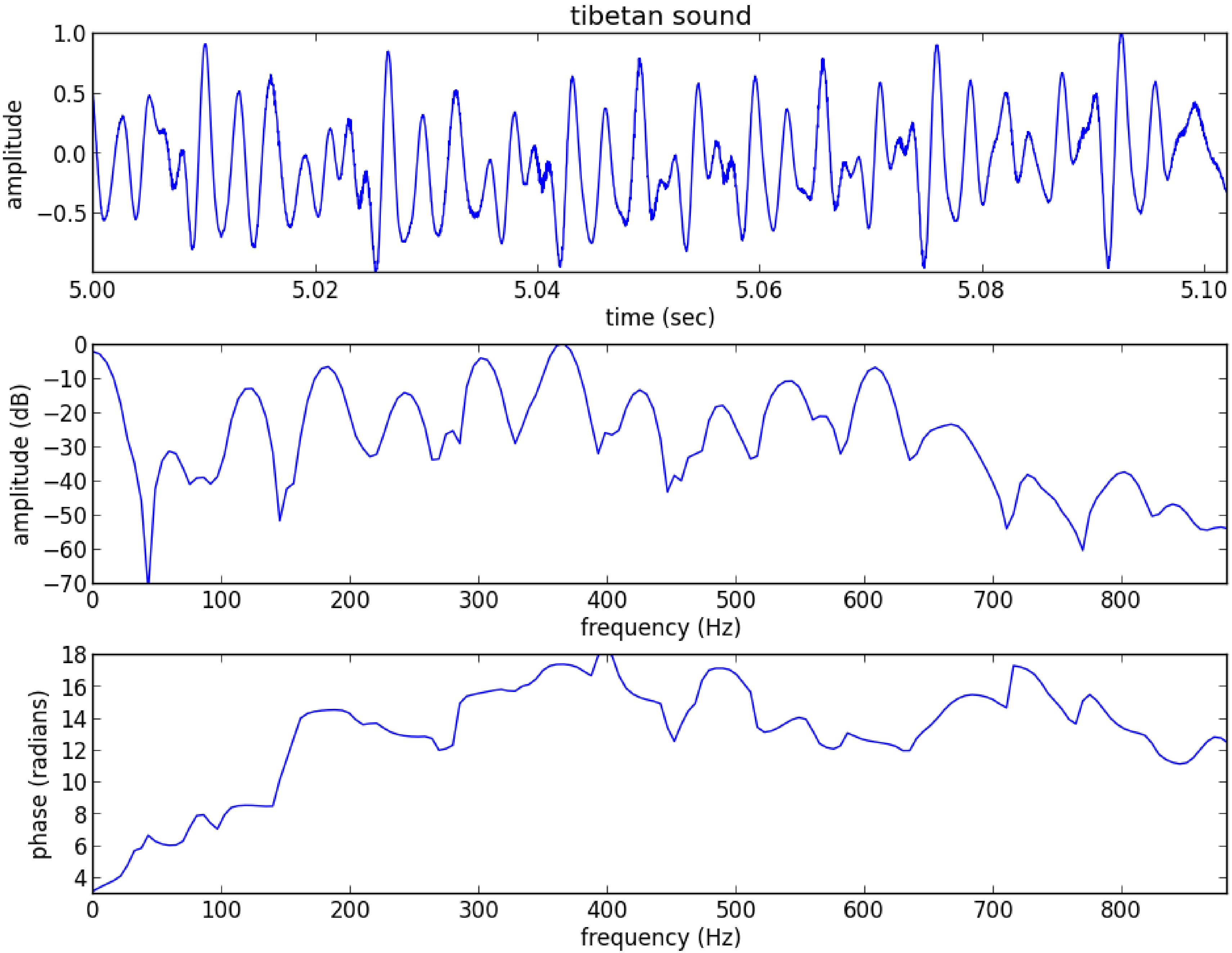
Sinusoids-Partials-Harmonics

- Any time-varying spectrum can be modeled as the sum of time-varying sinusoids.
- Most of the spectral bins in a spectrum do not correspond to actual partials of the analyzed sound.
- A sound partial is the result of a main mode of vibration of the generating system.
- Partials can be modeled as slowly time-varying sinusoids.
- A partial in the frequency domain can be identified by its spectral shape (magnitude and phase), its relation to other partials, and its time evolution.
- When the partials of a sound are related harmonically we call them harmonics.



oboe sound
amplitude
time (sec)
0.800
0.802
0.804
0.806
0.808
0.810
0.812
0.814
1.0
0.5
0.0
−0.5
−1.0
amplitude (dB)
frequency (Hz)
0
−10
−20
−30
−40
−50
1000
2000
3000
4000
5000
phase (radians)
frequency (Hz)
15
10
5
0
−5


piano sound
amplitude
time (sec)
amplitude (dB)
frequency (Hz)
phase (radians)
frequency (Hz)

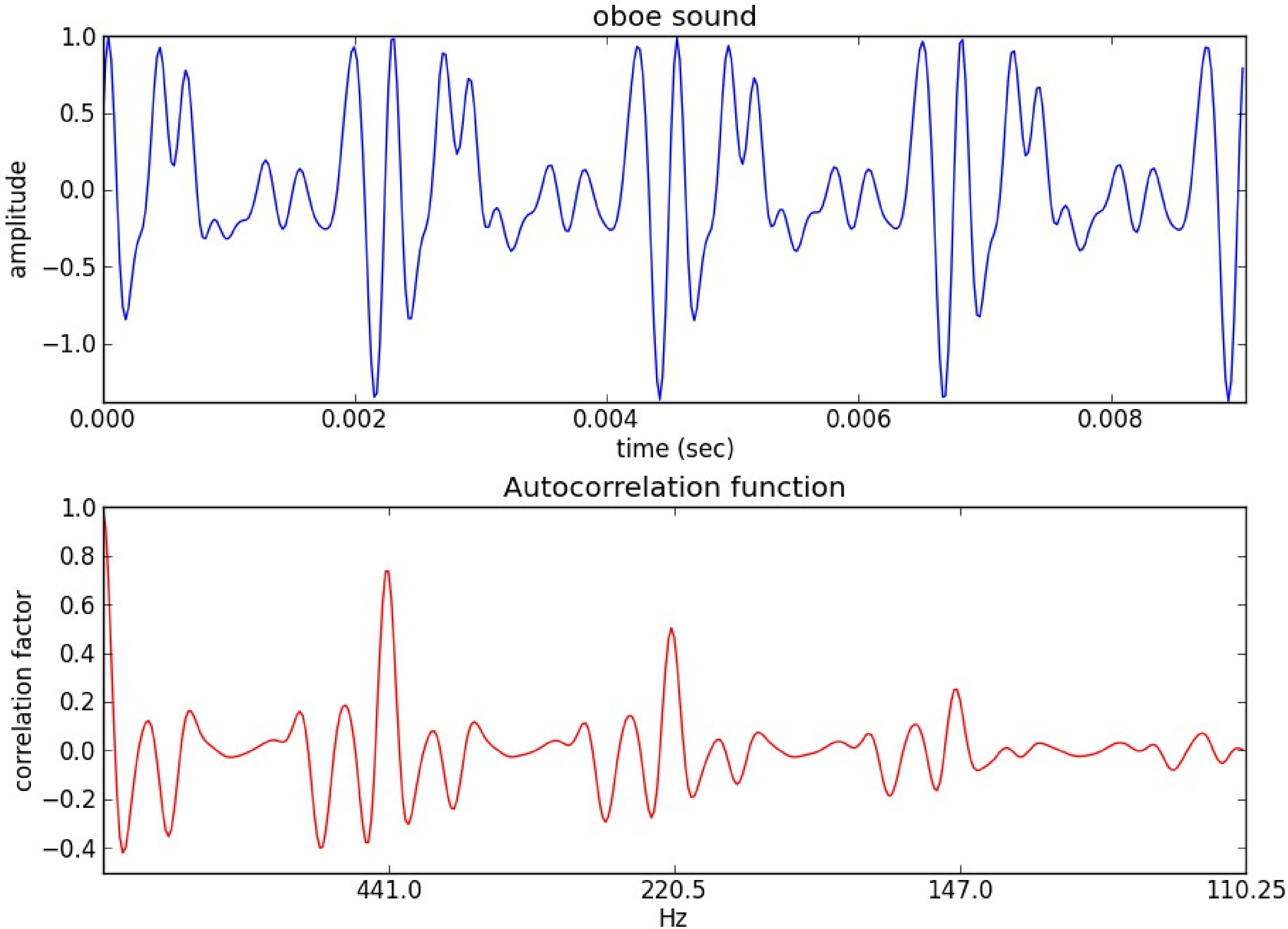


tibetan sound
amplitude
time (sec)
amplitude (dB)
frequency (Hz)
phase (radians)
frequency (Hz)

F0 detection in time domain

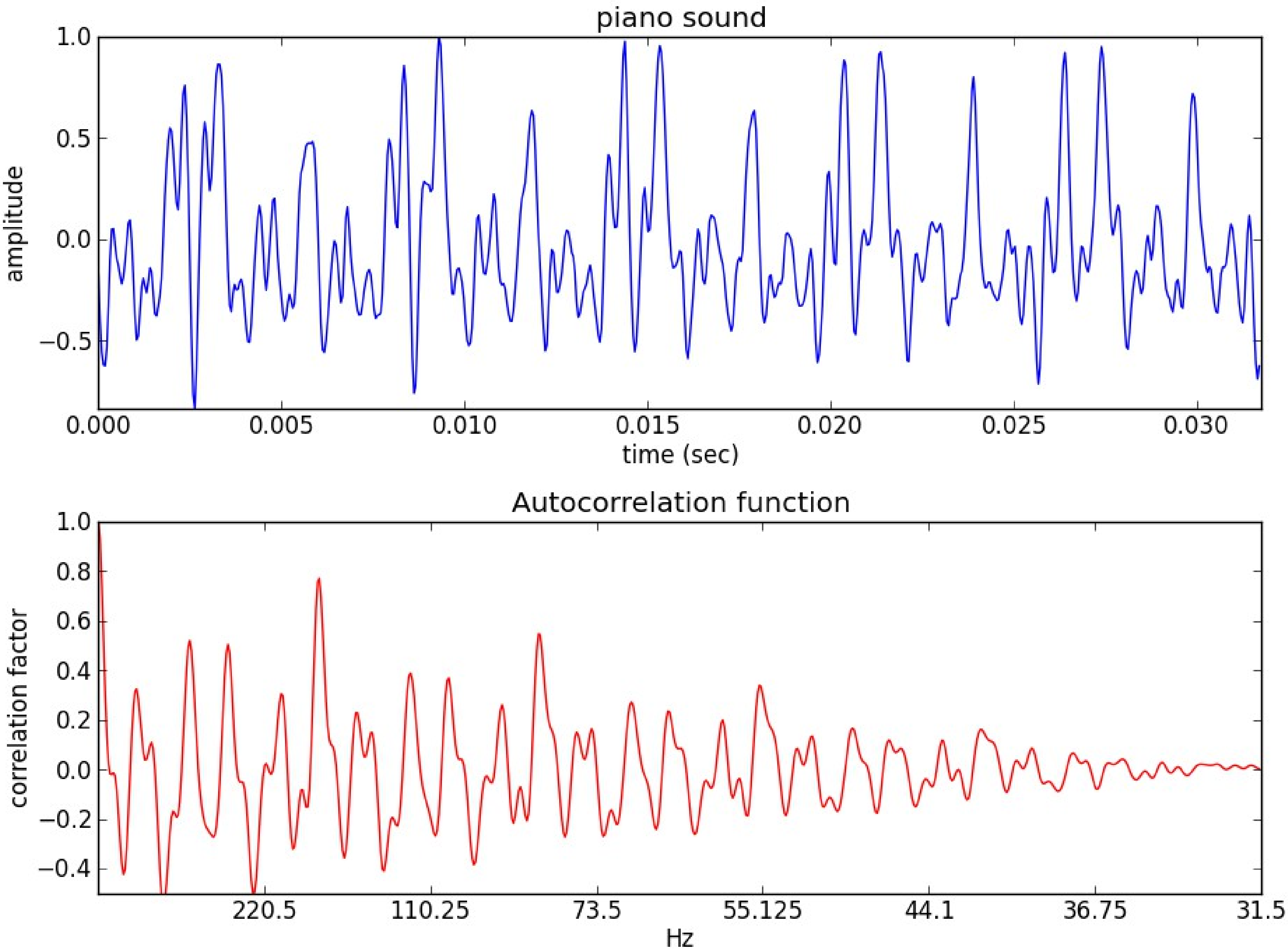
- Autocorrelation

$$Z_{xx}[k] = \sum_{n=0}^{n=N-1} x[n]\,x[n+k] \qquad k = -N+1, \ldots, N-1$$



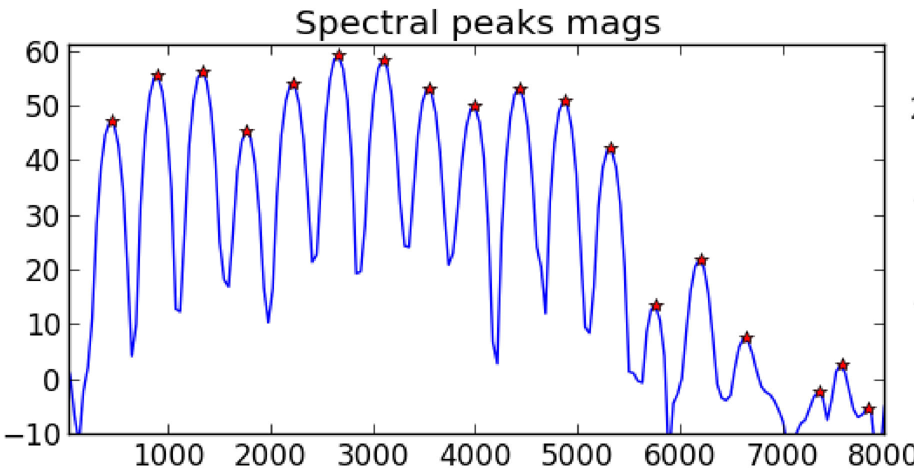
oboe sound
amplitude
1.0
0.5
0.0
−0.5
−1.0
0.000
0.002
0.004
0.006
0.008
time (sec)
Autocorrelation function
correlation factor
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
441.0
220.5
147.0
110.25
Hz

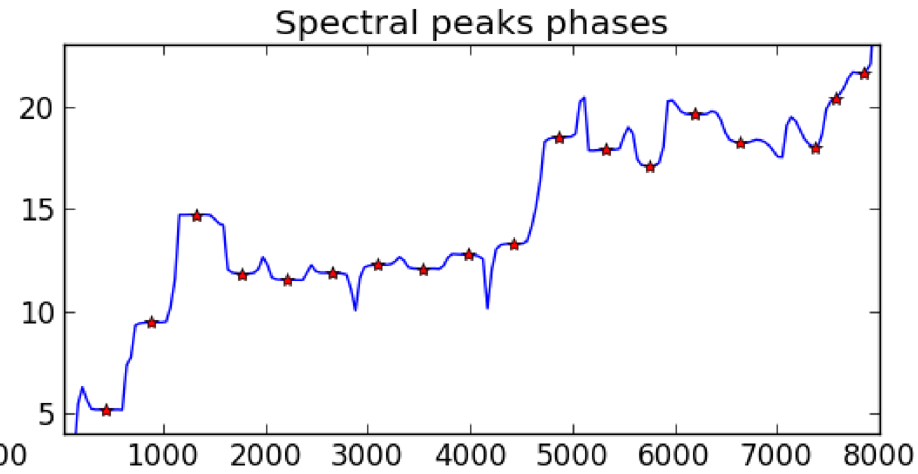
```
(fs, x) = read('oboe.wav')
M = 400
str = .8*fs
xp = x[str:str+M]/float(max(x[str:str+M]))
z = np.correlate(xp,xp,'full')

plt.subplot(211)
plt.plot(np.arange(M)/float(fs), xp)

plt.subplot(212)
plt.plot(z[M-1:]/max(z),'r')
```



piano sound
amplitude
1.0
0.5
0.0
−0.5
0.000
0.005
0.010
0.015
0.020
0.025
0.030
time (sec)
Autocorrelation function
correlation factor
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
220.5
110.25
73.5
55.125
44.1
36.75
31.5
Hz

F0 detection from spectral peaks





The fundamental frequency can be defined as the common divisor of the harmonic series that best explains the spectral peaks.

Steps:

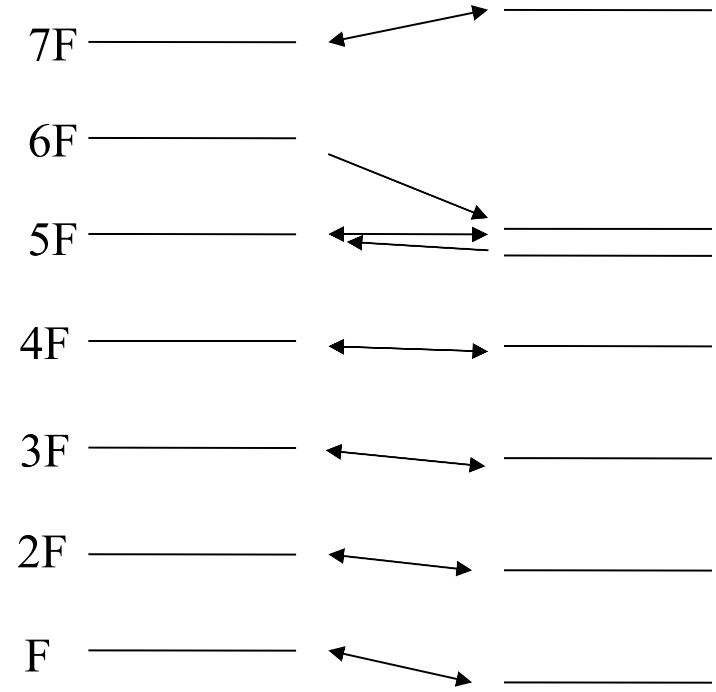
- Choose possible fundamental candidates.
- Measure the “goodness” of the resulting harmonic series compared with the spectral peaks.
- Get the best candidate.

Two-way mismatch
error calculation
(Maher and Beauchamp, 94)

$$Err_{p \to m} = \sum_{n=1}^{N} E_{\omega}(\Delta f_n, f_n, a_n, A_{\max})$$

$$= \sum_{n=1}^{N} \Delta f_n \cdot (f_n)^{-p} +$$

$$\left(\frac{a_n}{A_{\max}}\right) \times \left[q \Delta f_n \cdot (f_n)^{-p} - r \right]$$



Δf_n: difference between a predicted and its closest measured peak,

f_n, a_n: frequency and magnitude of predicted peaks,

$A_{\max}$: maximum peak magnitude.

$$Err_{m \to p} = \sum_{k=1}^{K} E_w(\Delta f_k, f_k, a_k, A_{\max})$$

$$= \sum_{k=1}^{K} \Delta f_k (f_k)^{-p} + \left(\frac{a_k}{A_{\max}}\right) \times \left[q \Delta f_k \cdot (f_k)^{-p} - r \right]$$

Δf_k: difference between a measured and its closest predicted peak,

f_k, a_k: frequency and magnitude of measured peaks,

$A_{\max}$: maximum peak magnitude.

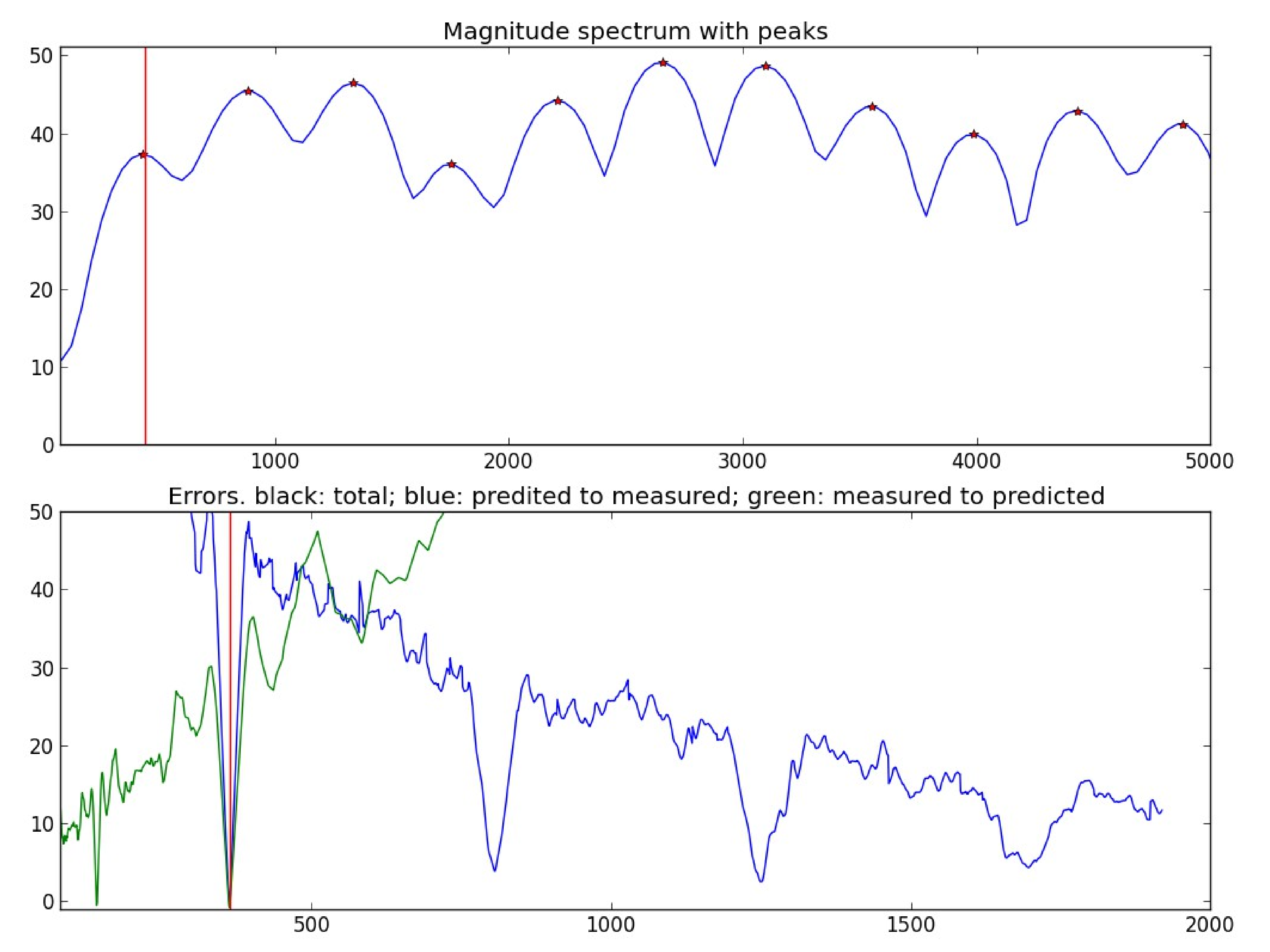
Total error:

$$Err_{total} = Err_{p \to m} / N + \rho Err_{m \to p} / K$$

Maher and Beauchamp propose: $p = 0.5, q = 1.4, r = 0.5, \rho = 0.33$

	Err_{pm}	Err_{mp}	Err
50 Hz	122.58	-3.0	7.49
100 Hz	32.0	-3.0	3.83
200 Hz	10.0	30.66	4.2

TWM calculation from a harmonic series that includes the frequencies: 200, 300, 500, 600, 700, 800.

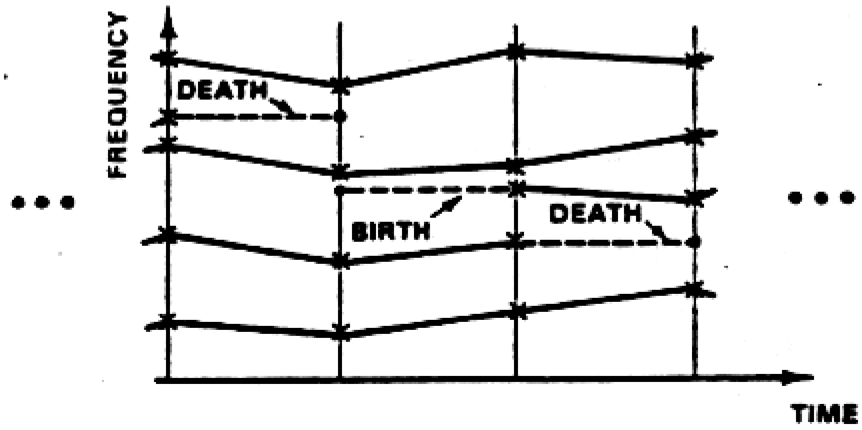


Magnitude spectrum with peaks
0
10
20
30
40
50
1000
2000
3000
4000
5000
Errors. black: total; blue: predited to measured; green: measured to predicted
0
10
20
30
40
50
500
1000
1500
2000

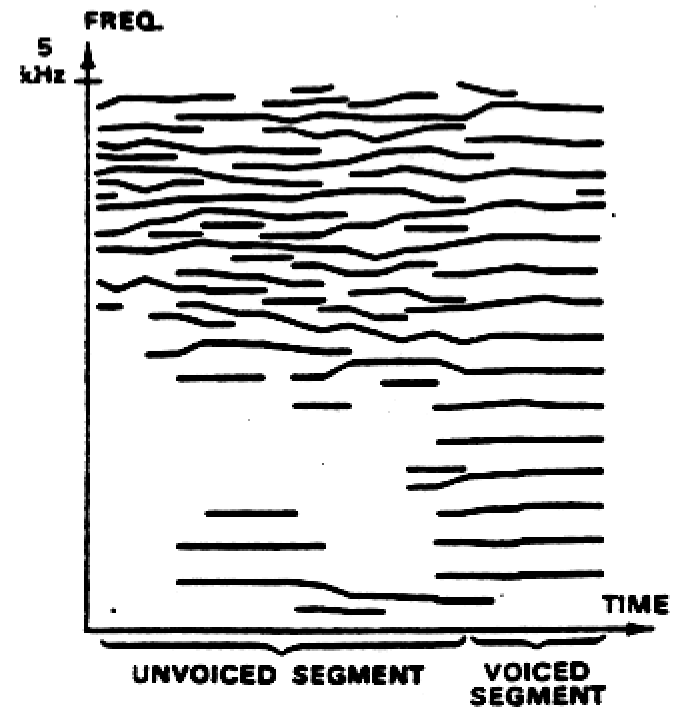
```
p = 0.5
q = 1.4
r = 0.5
rho = 0.33
Amax = max(pmag)

harmonic = np.matrix(f0cand)
MaxNPM = min(maxnpeaks, pfreq.size)
for i in range(0, MaxNPM) :
  difmatrixPM = harmonic.T * np.ones(pfreq.size)
  difmatrixPM = abs(difmatrixPM - np.ones((harmonic.size, 1))*pfreq)
  FreqDistance = np.amin(difmatrixPM, axis=1)
  peakloc = np.argmin(difmatrixPM, axis=1)
  Ponddif = np.array(FreqDistance) * (np.array(harmonic.T)**(-p))
  PeakMag = pmag[peakloc]
  MagFactor = 10**((PeakMag-Amax)/20)
  ErrorPM = ErrorPM + (Ponddif + MagFactor*(q*Ponddif-r)).T
  harmonic = harmonic+f0cand

MaxNMP = min(maxnpeaks, pfreq.size)
for i in range(0, f0cand.size) :
  nharm = np.round(pfreq[:MaxNMP]/f0cand[i])
  nharm = (nharm>=1)*nharm + (nharm<1)
  FreqDistance = abs(pfreq[:MaxNMP] - nharm*f0cand[i])
  Ponddif = FreqDistance * (pfreq[:MaxNMP]**(-p))
  PeakMag = pmag[:MaxNMP]
  MagFactor = 10**((PeakMag-Amax)/20)
  ErrorMP[i] = sum(MagFactor * (Ponddif + MagFactor*(q*Ponddif-r)))

Error = (ErrorPM/MaxNPM) + (rho*ErrorMP/MaxNMP)
f0index = np.argmin(Error)
f0 = f0cand[f0index]
```

Peak continuation

McAulay and Quatieri approach:



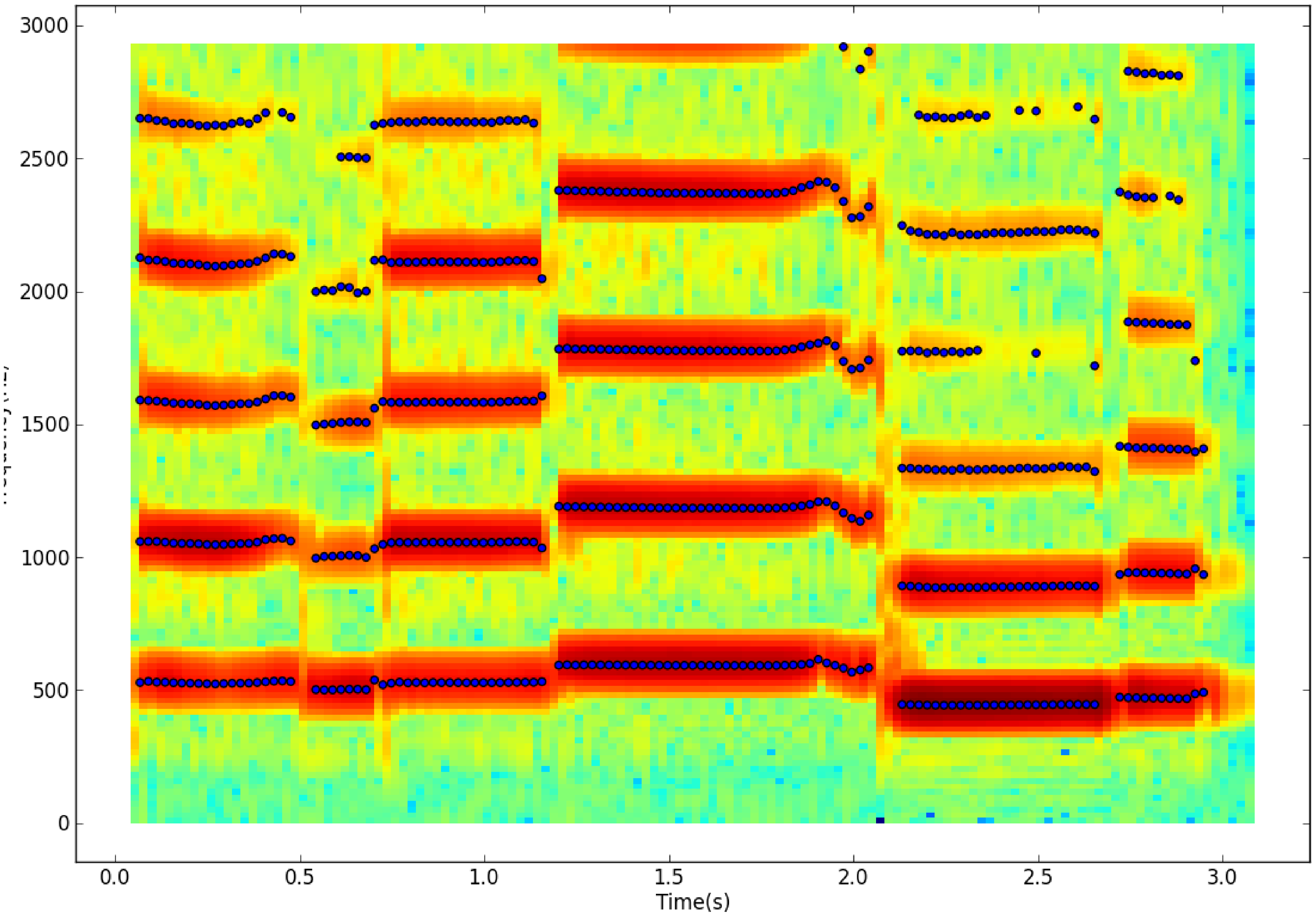
frequency tracker



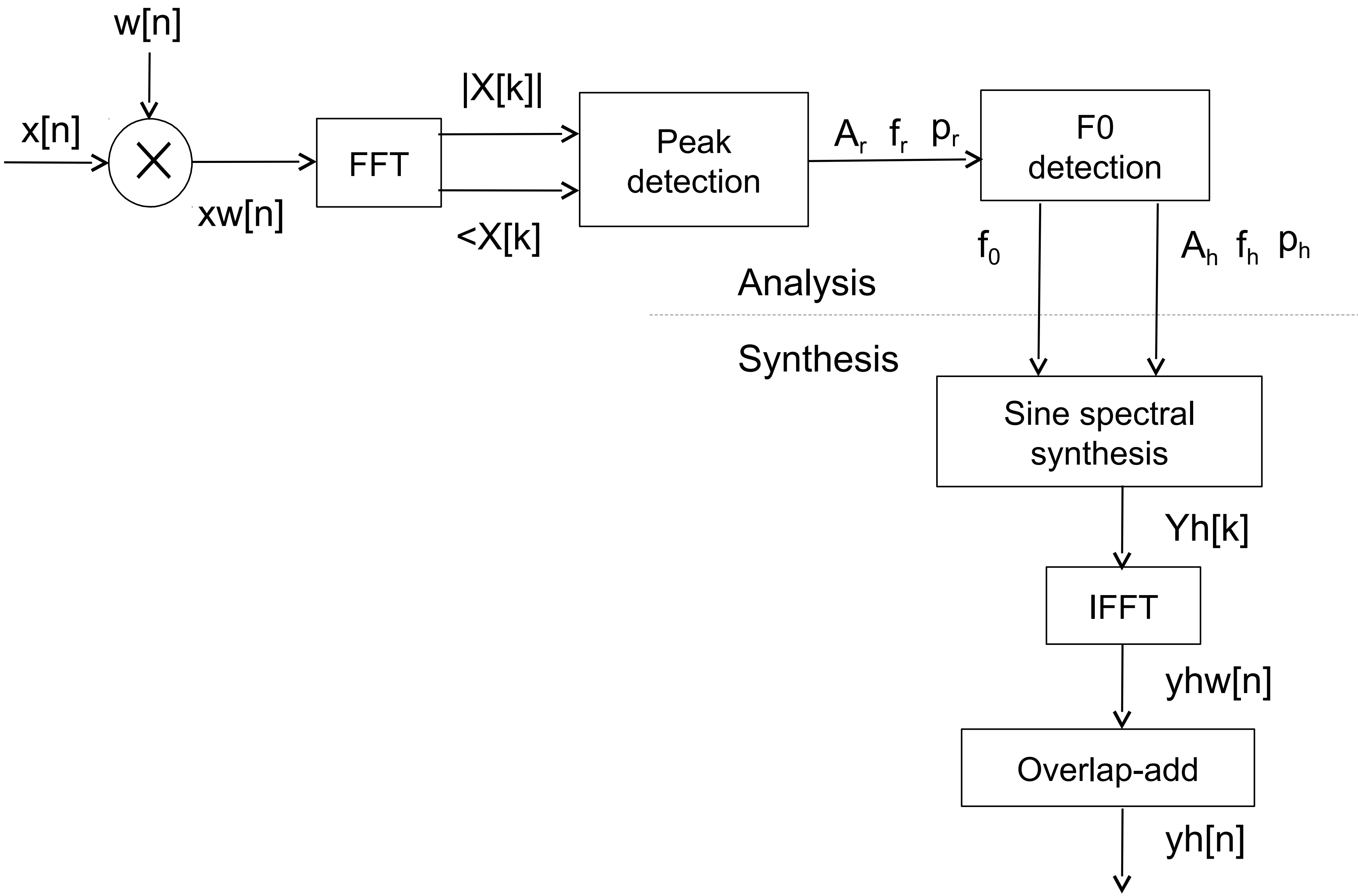
frequency tracks of speech

Harmonic tracking

```
f0 = fd.f0DetectionTwm(iploc, ipmag, N, fs, f0et, minf0, maxf0)
hf = (f0>0)*(f0*np.arange(1, nH+1))
hi = 0
npeaks = ploc.size
while f0>0 and hi<nH and hf[hi]<fs/2 :
   dev = min(abs(iploc/N*fs - hf[hi]))
   pei = np.argmin(abs(iploc/N*fs - hf[hi]))
   if (hi==0 or not any(hloc[:hi]==iploc[pei])) and dev<maxhd*hf[hi]:
      hloc[hi] = iploc[pei]
      hmag[hi] = ipmag[pei]
      hphase[hi] = ipphase[pei]
    hi += 1
hloc = (hloc!=0) * (hloc*Ns/N)
```


3000
2500
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Time(s)

Implementation: Harmonic model



```
def harmonicModel(x, fs, w, N, t, nH, minf0, maxf0, f0et, maxhd):
 Ns = 512
 w = w / sum(w)
 ow = triang(2*H)
 sw[hNs-H:hNs+H] = ow
 bh = blackmanharris(Ns) / sum(bh)
 sw[hNs-H:hNs+H] = sw[hNs-H:hNs+H] / bh[hNs-H:hNs+H]
 while pin<pend:
 #-----analysis-----
  xw = x[pin-hM1:pin+hM2] * w
  fftbuffer[:hM1] = xw[hM2:]
  fftbuffer[N-hM2:] = xw[:hM2]
  X = fft(fftbuffer)
  mX = 20 * np.log10( abs(X[:hN]) )
  ploc = PP.peakDetection(mX, hN, t)
  pX = np.unwrap( np.angle(X[:hN]) )
  iploc, ipmag, ipphase = PP.peakInterp(mX, pX, ploc)
  f0 = fd.f0DetectionTwm(iploc, ipmag, N, fs, f0et, minf0, maxf0)
  hf = (f0>0)*(f0*np.arange(1, nH+1))
  while f0>0 and hi<nH and hf[hi]<fs/2 :
    dev = min(abs(iploc/N*fs - hf[hi]))
    pei = np.argmin(abs(iploc/N*fs - hf[hi]))
    if (hi==0 or not any(hloc[:hi]==iploc[pei])) and dev<maxhd*hf[hi]:
      hloc[hi] = iploc[pei]
      hmag[hi] = ipmag[pei]
      hphase[hi] = ipphase[pei]
    hi += 1
  hloc = (hloc!=0) * (hloc*Ns/N)
  #-----synthesis-----
  Yh = GS.genSpecSines(hloc, hmag, hphase, Ns)
  fftbuffer = np.real( ifft(Yh) )
  yh[:hNs-1] = fftbuffer[hNs+1:]
  yh[hNs-1:] = fftbuffer[:hNs+1]
  y[pin-hNs:pin+hNs] += sw*yh
  pin += H
 return y
```

References

- http://en.wikipedia.org/wiki/Fundamental_frequency
- http://en.wikipedia.org/wiki/Pitch_detection_algorithm

Credits