BAK4_final

August 2, 2014

```
In [3]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        import random
        import math
```

set working directory:

```
In [4]: cd /Users/macuserin/Documents/Max/Projects/BAC3/data
```

```
/Users/macuserin/Documents/Max/Projects/BAC3/data
```

A function for parsing typical coll/dict Max/MSP data to an array:

```
In [5]: def parseFrame(jsonFile):
            import json
            json_data=open(jsonFile)
            data = json.load(json_data)
            json_data.close()
            array1 = []
            for i in range (len(data)):
                array1.append(data[str(i)][0])
            return array1
```

Here, the accumulation of the autocorrelation's spectrum is loaded and some constants are set:

```
In [6]: spectrum = parseFrame("Kuebel_Acc_autoCorrel.json")
        sr = 44100.
        frameSize = len(spectrum)
```

Helper function to convert a bin number to a frequency in Hz:

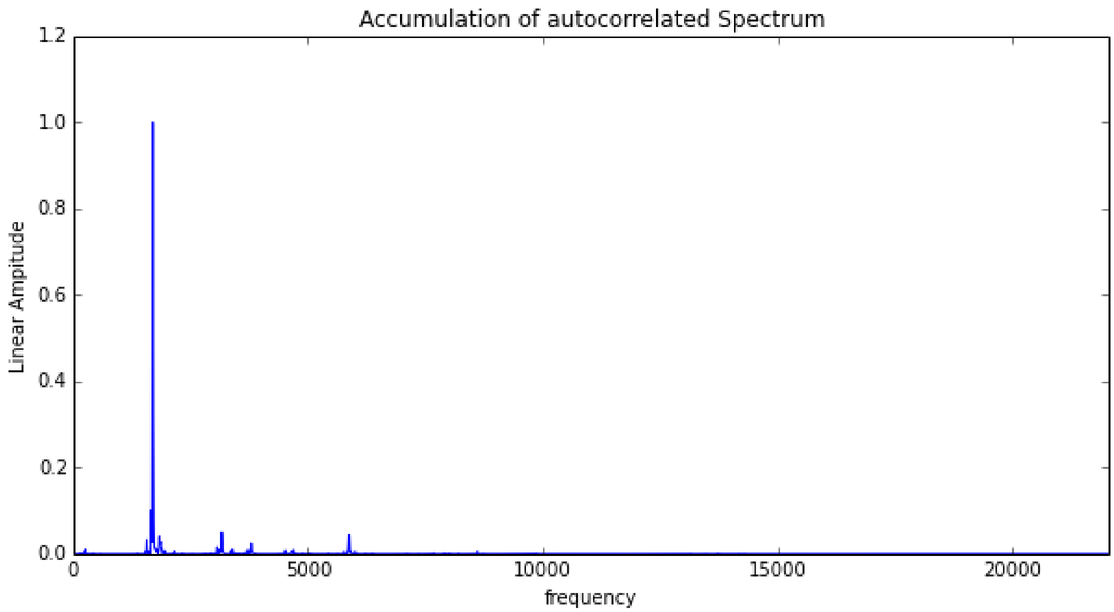
```
In [7]: def binToFreq(bins, sr, frameSize):
            freqs = []
            for i in range(len(bins)):
                f = (bins[i]/float(frameSize))*sr/2
                freqs.append(f)
            return freqs
```

Below, just the raw loaded data is plotted:

```
In [101]: fig = plt.figure(figsize=(10,5))
          ax = fig.add_subplot(111)
          frame = spectrum
          n = np.linspace(0, sr/2, frameSize)
          plt.plot(n, frame)
```

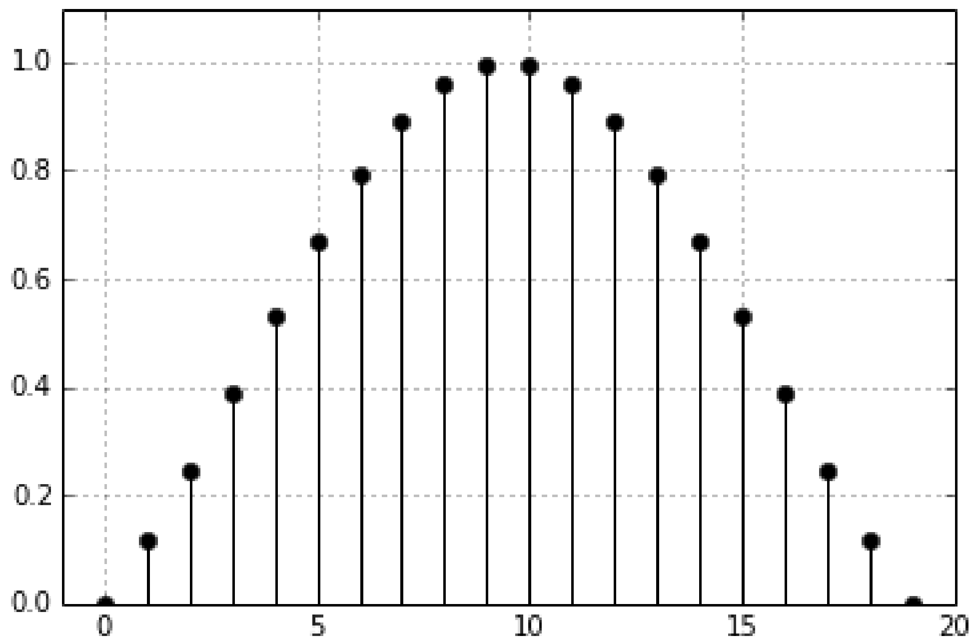
```
plt.title('Accumulation of autocorrelated Spectrum');
plt.axis([0, sr/2, 0, 1.2])
ax.set_xlabel('frequency')
ax.set_ylabel('Linear Ampitude')
```

Out[101]: <matplotlib.text.Text at 0x109f0cf90>



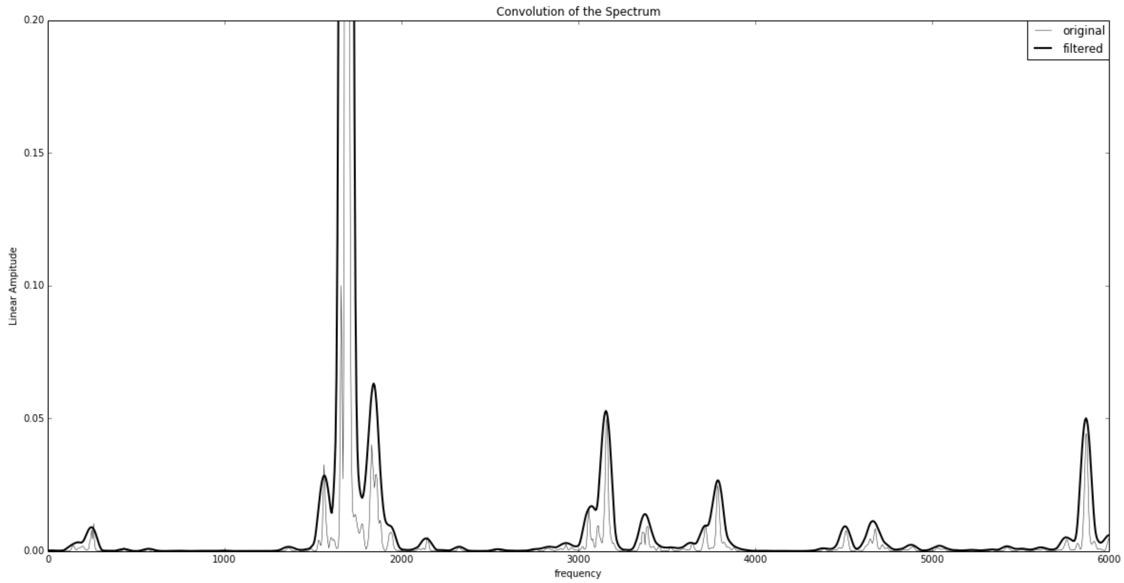
In [9]:
```
#A function for generating a Convolution Kernel
def sincKernel(length):
    x = np.linspace(-1,1,length)
    kernel = []
    for i in x:
        kernel.append(np.sinc(i))
    return kernel
```

In [110]:
```
#Generating and plotting the Kernel
#--create Convolution Kernel
convolutionLength = 20
kernel = sincKernel(convolutionLength)
n = range(convolutionLength)

plt.stem(n, kernel, linefmt='k', markerfmt='ko', basefmt='k--')
plt.grid(True)
a=plt.axis([-1, convolutionLength,0,1.1])
```



In [111]:
```
fig = plt.figure()
plt.axis([0, 6000, 0, 0.2])
ax = fig.add_subplot(111)

#-------IndexSIgnal
x2 = np.linspace(0, sr/2, frameSize)

#--plot OriginalFrame
plt.plot(x2, frame, color='0.1', label= "original", linewidth=0.5)

# -----Convolution
conv = np.convolve(frame,kernel)
conv = conv[:4096]
# ------post-amplify
conv[:] = [x*0.25 for x in conv]

# a negative delay applied to the result of the convolution as a compensation.
def shift(array, n):
    for i in range(len(array)):
        index = (i+n)
        if index < 0:
            array[i] = 0
        elif index >= len(array):
            array[i] = 0
        else:
            array[i]= array[i+n]
    return array
filtered = shift(conv,convolutionLength/2)
plt.plot(x2, filtered, 'k', label= "filtered", linewidth=2.)
```

```
plt.title('Convolution of the Spectrum');
ax.set_xlabel('frequency')
ax.set_ylabel('Linear Ampitude')
a=plt.legend(bbox_to_anchor=(1, 1), loc=1, borderaxespad=0.)
```



In [96]:
```
def delta(data):
    delta = []
    for i in range(len(data)):
        if i != 0:
            d = data[i] - data[i - 1]
        else:
            d = 0
        delta.append(d)
    return delta


def findPeaks(data, iterarions,thresh):
        import numpy as np
        import copy
        tdata = copy.copy(data)
        length = len(data)
        peaks = []
        delt = delta(data)

        for i in range(iterarions):

                tmax = np.argmax(tdata)
                peaks.append(tmax)
                deriv = -1.
                #thresh = 0.001
                j = 1
                tdata[tmax]= 0
```

```
                while (deriv)<thresh and j < 100:
                        index = np.clip(tmax+j, 0, length-1)
                        deriv = delt[index+1]
                        tdata[index] = 0
                        j = j + 1

                j = 1
                deriv = 1.
                while deriv>(thresh*-1) and j < 100:
                        index = np.clip(tmax-j, 0, length)
                        deriv = delt[index]
                        tdata[index] = 0
                        j = j + 1
        return peaks, tdata

def markers(data, peaks, length):
        import numpy as np
        markers= np.zeros(length)
        markers = markers-1
        for i in range(len(peaks)):
                markers[peaks[i]]=data[peaks[i]]
        return markers
```

The parameters of the Peak Detection are set: The number of desired/estimated peaks and a Threshold Value, that regulates the algorithm's sensitivity to the values of the derivative of the input:

```
In []: numberOfPeaks = 15
       threshold = 0.001
```

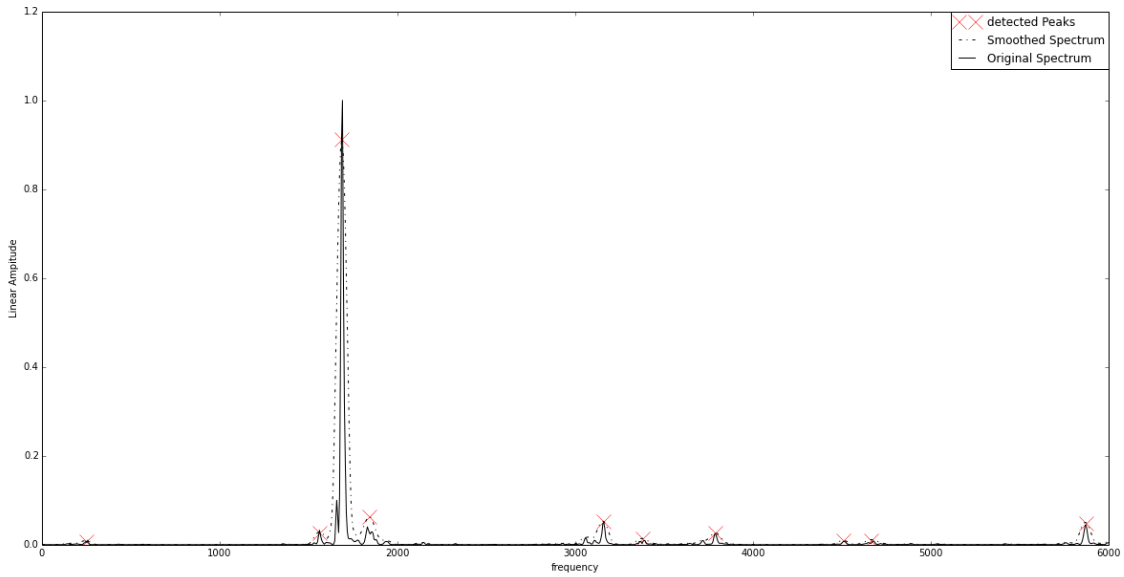
Finally, the peaks are dected:

```
In [117]: result = findPeaks(filtered,numberOfPeaks,threshold)
          peaks = result[0]
          tdata = result[1]
          marks = markers(filtered, peaks, frameSize)
          peakFreqs = binToFreq(peaks, sr,frameSize)
          peakFreqs=np.sort(peakFreqs)
          print "Detected Peak Frequencies:"+"\n"+str(peakFreqs)
```

```
Detected Peak Frequencies:
[   247.63183594   1561.15722656   1684.97314453   1841.08886719
   3154.61425781   3375.32958984   3784.46044922   4505.82275391
   4661.93847656   5867.79785156   7671.20361328   8602.51464844
   9846.05712891  13135.25390625  13727.41699219]
```

And plotted:

```
In [116]: fig = plt.figure()
          ax = fig.add_subplot(111)
          plt.axis([0, 6000, 0, 1.2])
          plt.plot(x2, marks, "rx",markersize=15, label="detected Peaks")
          plt.plot(x2, filtered, "k-.", label="Smoothed Spectrum")
          plt.plot(x2, frame, "k", label="Original Spectrum")
          ax.set_xlabel('frequency')
          ax.set_ylabel('Linear Ampitude')
          a=plt.legend(bbox_to_anchor=(1, 1), loc=1, borderaxespad=0.)
```


detected Peaks
Smoothed Spectrum
Original Spectrum
Linear Amplitude
frequency
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1000
2000
3000
4000
5000
6000