ASSIGNMENT 10

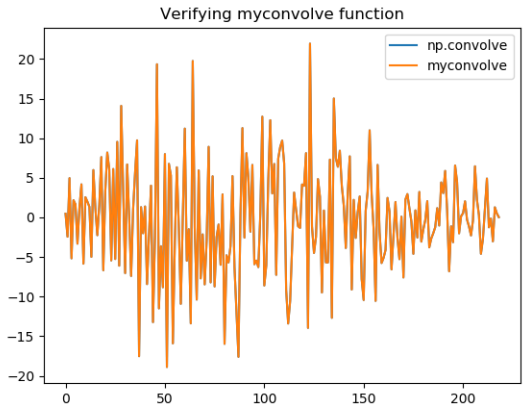
P HarshaVardhan - EE17B061

April 17, 2019

1 My algorithm for convolution

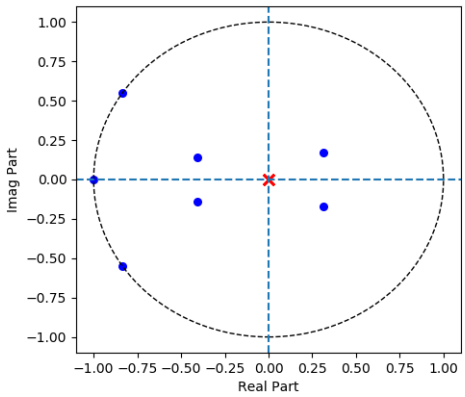
```
def myconvolve(a,b):
    a,b = a.copy(),b.copy()# copying is a good practice because
    # any changes to the array inside the function will be reflected in the global scope
    lenA,lenB = len(a),len(b)
    lenX = lenA+lenB-1
    x = np.zeros(lenX)
    a = np.hstack([a,np.zeros(lenB-1)])
    b = np.hstack([np.zeros(lenA-1),b[::-1]])
    for i in range(lenX):
        x[i] = a[:i+1].dot(b[-(i+1):].T)
    return x
```

verifying the algorithm by convolving two random signals.



2 Low pass filter

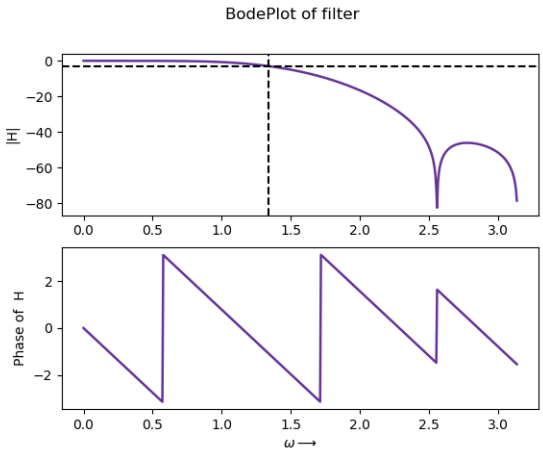
2.1 Position of poles and zeros



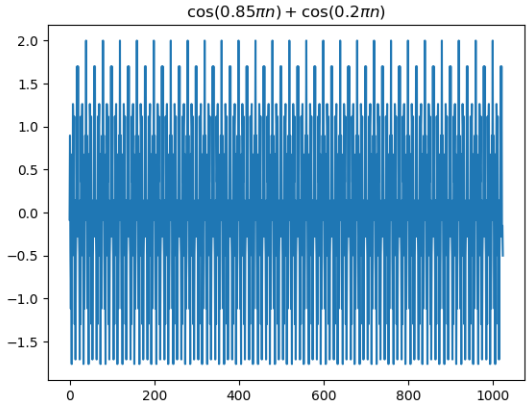
It has 12 trivial poles.The zeros inside the unit circle have a multiplicity of 1. There are four zeros at -1 and two zeros at each of the points(which are on either side of the real axis) on the unit circle except -1.From this we can say that this is a linear phase filter.This can also be infered from the phase plot of the filter which is piece wise linear.

2.2 Bodeplot of the Digital lowpass filter

From the Magnitude plot of the DTFT we can see that it is a lowpass filter with 3dB frequency somewhere around 1.3



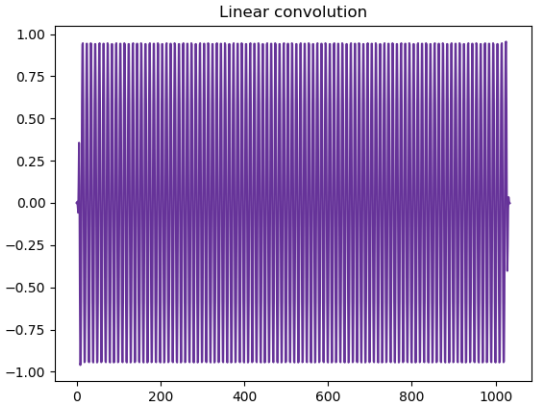
3 Output of the filter for $\cos(0.85\pi n) + \cos(0.2\pi n)$



Output of the filter can be calculated using the following methods.

3.1 Direct linear convolution

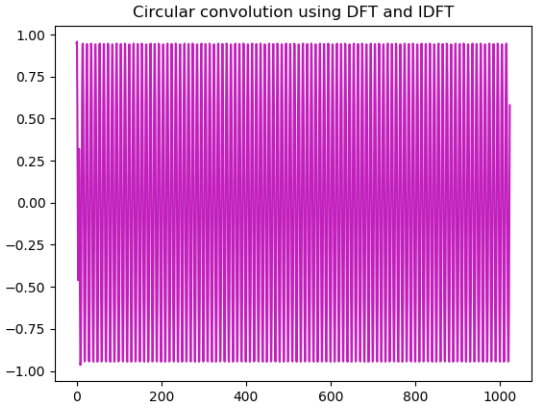
evaluating x(t)*h(t) using np.convolve



3.2 Circular convolution

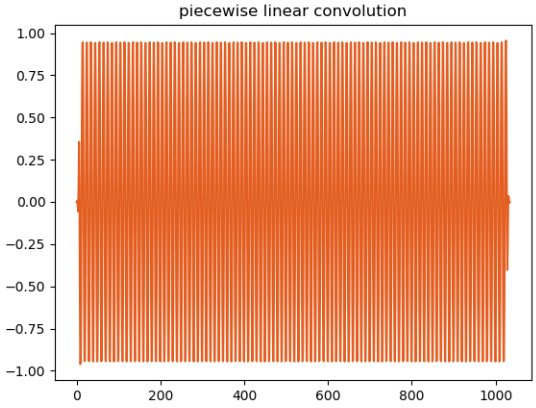
we can evaluate circular convolution faster than linear convolution(in time domain) by multiplication(of DFT of the signals) in frequency domain.

```
y = ifft(fft(xi)*fft(np.hstack([LPfilter,np.zeros(len(xi)-len(LPfilter))])))
```



3.3 linear convolution by splitting the array

This is how we would have convolved a real time signal with impulse response if we didn't have fft. Parts of the input signal are convolved with the impulse response and the result is not correct on its own the first p-1 terms are to be added to the last p-1 terms of the previous convolution and the lasts p-1 terms wait for the next convolution's result.Output is identical to linear convolution.




```
L,P=2**5,len(LPfilter)
N=L+P-1
y=np.zeros(len(xi)+P-1)
```

```
for i in range(int(len(xi)/L)):
    y[i*L:((i+1)*L)+P-1] += np.convolve(xi[i*L:(i+1)*L],LPfilter)
plot('aliased linear convolution',y)
```

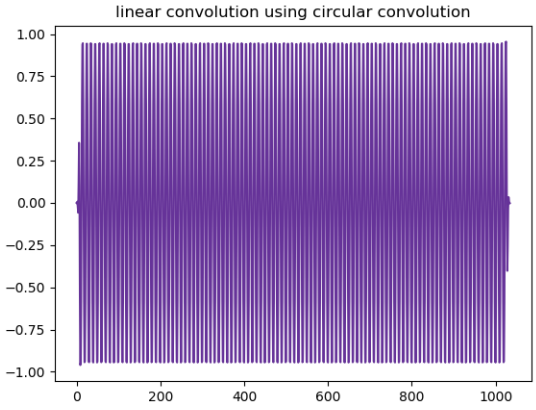
3.4 linear convolution using circular convolution

P = len(impulse response) L = length of each segment of the input

we take each segment add P-1 values of previous segment at its start do circular convolution in frequency domain and remove the first P-1 terms. since these are not identical to linear convolution.

we pad the input with **P-1 zeros** at the start,since first segment previous values will be zeros.

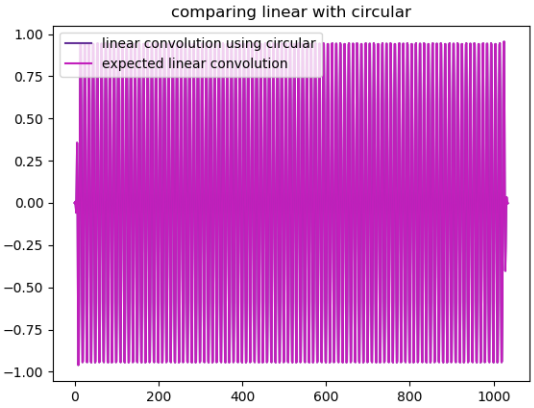
we also add **L zeros** at the end.These are to get last P-1 values of linear convolution.



even though we need to discard part of the output of each convolution this is still much faster than time domain implementation of linear convolution.

```
n = np.arange(1,1025)
L,P=2**4,len(LPfilter)# L is length of segment for circular convolution
xi = np.cos(0.85*np.pi*n) + np.cos(0.2*np.pi*n)#np.random.randn(1024)
plt.figure(6)
xi1 = np.hstack([np.zeros(P-1),xi,np.zeros(L)])
y2=np.zeros(len(xi)+L,dtype=np.complex)
for i in range(int(len(xi1)/L)):
    y2[i*L:(i+1)*L] += ifft(fft(xi1[i*L:(i+1)*L+P-1])*fft(np.hstack([LPfilter,np.zeros(L+P-1
plt.title('linear convolution using circular convolution')
```

```
plt.plot(y2[:-L+P-1].real,color='#663399')
```



4 zadoff chu sequence

since correlation is not commutative the order in which we correlate matters, however the outputs will just be mirror images of one another if we change the order of correlation.