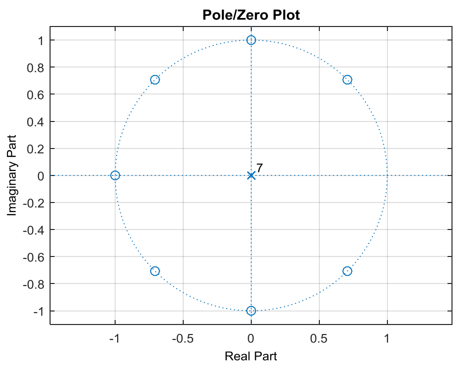
Multimedia Signal Processing 1st Module

03/03/2017

Ex.1 (Pt.14)

A LTI digital filter has the following zero/pole plot and it is composed of the cascade of 4 filters, each of them has a single real zero or a couple of conjugate zeros (the angular position of the zeros are $\pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3}{4}\pi, \pi$), furthermore it has a unitary gain for the continuous components ($0Hz$).



1. Define the z-transform of the filter.
2. Define its difference equation.
3. A signal $x(t) = 5 + \cos(100\pi t)$ is sampled at $400Hz$ and filtered with this filter. What will be the output signal $y(n)$?

Ex.2 (Pt.8)

You need to apply the previous filter in a real time application and you want to adopt the *overlap and add* approach using blocks of 128 samples of the input signal working in the frequency domain.
Describe the overlap and add approach in the **time domain** for **this** specific application.
Describe how we can got the results **working in the Frequency domain** (define the **W** matrix for this specific application) and the expedient required in order to prevent undesired effects of the circular convolution.

Ex.3 (Pt. 10 – MATLAB code)

Re-implement using MATLAB the steps of the first and second exercise:

1. Generate 1000 samples from the signal $x(t)$,
2. define the filter $h(n)$
3. get the output signal $y(n)$ in the time domain
4. define the **W** matrix for the DFT to be used for the *Overlap and Add* approach and apply in to the proper segment of the input signal and of the impulse response (avoiding drawbacks of the circular convolution).
5. Calculate the output signal for every block of the input signal and manage the buffer for the *Overlap and Add* approach.

Solutions

Ex.1

$$H(z)=A\left(1-\sqrt{2}z^{-1}+z^{-2}\right)\left(1+z^{-2}\right)\left(1+\sqrt{2}z^{-1}+z^{-2}\right)\left(1+z^{-1}\right)=$$
$$=A\left(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6}+z^{-7}\right)$$

To obtain a unitary gain for the continuous component($z=1$)-> $A=\frac{1}{8}$

$$y(n)=\frac{1}{8}\left(x(n)+x(n-1)+x(n-2)+x(n-3)+x(n-4)+x(n-5)+x(n-6)+x(n-7)\right)$$

the sampled signal $x(n)=5+\cos\left(2\pi\frac{50}{400}n\right)$ is filtered and the sinusoid at $\omega=\frac{\pi}{4}$ is completely removed due to the zeros in the filter at those normalized frequencies while the constant is preserved so: $y(n)=5$

Ex.2

See the description of the Overlap and Add on the course slides.

Since the length of the filter is 8 samples and each block of samples is made of 128 samples, the convolution will have a length of 128+8-1=135 samples:

The $W_{135}=e^{-j\frac{2\pi}{135}}$ and the $\mathbf{W}_{135}^{r,c}=W^{r\cdot c}$ where *r* and *c* indicate the row and column of the 135x135 **W** matrix.

Every block $b(n)$ of 128 samples of the input signal is processed according to the DFT procedure:

seven zeros are pasted to the end of the signal

The buffer length has to be of 7 samples that will be added to the first 7 samples of the result from the next block.

Both $b(n)$ and $h(n)$ have to be extended to 135 samples pasting zeros to their ends.

We get the DFT of the signal pre-multiplying $b(n)$ by the $\mathbf{W}_{135}^{r,c}$ matrix and the same for the impulse response $b(n)$, we then multiply sample by sample the two resulting vectors of 135 samples and calculate the inverse DFT pre-multiplying with the inverse of the $\mathbf{W}_{135}^{r,c}$ matrix.

From the output we preserve 128 samples and the remaining 7 are stored into the buffer.

Ex.3

```
n=0:999;
x=5+cos(100*pi/400*n);
h=[1 1 1 1 1 1 1 1]/8;
buffer=zeros(1,length(h)-1);
bufferLength=length(buffer);
samples=length(n);
blockSize=128;
filterLength=length(h);
convolutionLength=blockSize+filterLength-1;
W=exp(1j*2*pi/convolutionLength*(0:convolutionLength-1)'*(0:convolutionLength-1));
h=[h zeros(1,convolutionLength-filterLength)];
H=W*h';
```

```
y=[];
x=[x zeros(1,ceil(samples/blockSize)*blockSize-length(x))];
for iteration = 0:length(x)/blockSize-1;
    block=x(iteration*blockSize+1:(iteration+1)*blockSize);
    block=[block zeros(1,convolutionLength-blockSize)];
    Block=W*block';
    Output=Block.*H;
    output=(W\Output)';
    y=[y (output(1:bufferLength)+buffer) output(bufferLength+1:end-bufferLength)];
    buffer=output(end-bufferLength+1:end);
end
```