Digital Signal Processing 2/ Advanced Digital Signal Processing
Lecture 7,
z-Transform, Filters

Gerald Schuller
TU-Ilmenau

The z-Transform

The z-Transform is a more general transform than the Fourier transform, and we will use it to obtain perfect reconstruction in filter banks and wavelets. Hence we will now look at the effects of sampling and some more tools in the z-domain.
Since we usually deal with causal systems in practice, we use the **1-sided z-Transform**, defined as

$$X(z)=\sum_{n=0}^{\infty} x(n)\,z^{-n}$$

Observe this simply takes our sequence $x(n)$ and **turns it into the polynomial** $X(z)$.
First observe that we get our usual frequency response (the Discrete Time Fourier Transform for a causal signal, starting at n=0) if we evaluate the z-tranform along the unit circle in the z-domain,

$$z=\mathrm{e}^{j\Omega}$$

This connects the z-Transform with the DTFT, except for the sample index n, which for the so-called one-side z-Tranform starts at n=0, and

for the DTFT starts at $n=-\infty$.

In general, we can write complex variable z with an angle and a magnitude,

$$z=r\cdot e^{j\Omega}$$

where we can interpret the Ω as the **normalized angular frequency**, and the r a damping factor for an exponentially decaying oscillation, if r<1 (or exponentially growing if r >1).

Observe: This damping factor is **not** in the DTFT. This means in the z-Transform we can have a converging sum of the transform even for unstable signals or system, by just choosing r large enough! This means the **Region of Convergence** (**ROC**) just becomes smaller. Remember, in the z-transform sum we have $z^{-1}=\frac{1}{r}\cdot e^{-j\Omega}$.

Properties of the z-Transfrom:

Shift property:

Take two causal sequences (causal means sample value 0 for negative indices): Sequence x(n), and x(n-1), which is the same sequence but delayed by one sample. Then their z-transforms are:

$$x(n)\rightarrow\sum_{n=0}^{\infty}x(n)\cdot z^{-n}=:X(z)$$

$$x(n-1) \rightarrow \sum_{n=0}^{\infty} x(n-1)\cdot z^{-n} = \sum_{n=1}^{\infty} x(n-1)\cdot z^{-n} =$$

Use the index substitution, $n' \leftarrow n-1$ or $n'+1 \leftarrow n$ to get rid of the " $n-1$ " in the transform:

$$= \sum_{n'=0}^{\infty} x(n')\cdot z^{-(n'+1)} = z^{-1}\cdot \sum_{n'=0}^{\infty} x(n')\cdot z^{-n'} = X(z)\cdot z^{-1}$$

This shows that a **delay by 1 sample** in the signal sequence (time domain) corresponds to the **multiplication with** z^{-1} in the z-domain:

$$x(n) \rightarrow X(z)$$

$$x(n-1) \rightarrow X(z)\cdot z^{-1}$$

Example:

Signal:

$x_0 = [1,2,3]$ => $X_0(z) = 1+2\cdot z^{-1}+3\cdot z^{-2}$

Signal, delayed by 1 sampling period:

$x_1 = [0,1,2,3]$ => $X_1(z) = 0+1\cdot z^{-1}+2\cdot z^{-2}+3\cdot z^{-3} =$

In the z-domain the delay shows up as multiplication with z^{-1} ,

$$= X_0(z)\cdot z^{-1}$$

Z-Transform Properties

Recommended reading:

Alan V. Oppenheim, Ronald W. Schafer: “Discrete Time Signal Processing”, Prentice Hall.

Related to the shift property is the z-transform of the shifted unit pulse. The unit pulse is defined as

$$\Delta(n)=\begin{cases}1, & if \quad n=0\\ 0, & else\end{cases}$$

so it is just a zero sequence with a 1 at time 0.
Its z-Transform is then:

$$\Delta(n)\rightarrow 1$$

The z-transform of the shifted unit pulse delayed by d samples is

$$\Delta(n-d)\rightarrow z^{-d}$$

Linearity: $a\cdot x(n)\rightarrow a\cdot X(z)$

$$x(n)+y(n)\rightarrow X(z)+Y(z)$$

Convolution:

$$x(n)*y(n)\rightarrow X(z)\cdot Y(z)$$

The z-transform turns a convolution into a multiplication.
Remember: the convolution is defined as:

$$x(n)*y(n)=\sum_{m=-\infty}^{\infty}x(m)\cdot y(n-m)$$

This is because the convolution of 2 sequences behave in the same way as the multiplication of 2 polynomials (the z-transforms) of these sequences. This is one of the main advantages of the z-Transform, since it turns convolution into a simpler multiplication (which in principle is invertible).

Example z-transform: Exponential decaying sequence:

$x(n)=p^n$, for n=0,1,..., meaning the sequence

$$1, p, p^2, p^3, \ldots$$

$$\rightarrow X(z)=\sum_{n=0}^{\infty} p^n \cdot z^{-n}$$

Remember from last time: we had a closed form solution for this type of **geometric sums**:

$$S=\sum_{k=0}^{N-1} c^k$$

its solution was:

$$S=\frac{c^N-1}{c-1}$$

But now we have an infinite sum, which means N goes towards infinity. But we have the expression c^N in the solution. If $|c|<1$, then this goes to zero $c^N \rightarrow 0$. Now we have $c=p\cdot z^{-1}$. Hence, if $|p\cdot z^{-1}|<1$ we get

$$\rightarrow X(z)=\frac{1}{1-p\cdot z^{-1}}=\frac{z}{z-p}$$

Observe that this fraction has a **pole** at position z=p, and a **zero** at position z=0. Hence if know the pole position, we know p, and if we know p we know the time sequence (except for a constant factor). So the location of the pole gives us very important information about the signal.

Keep in mind that this solution is only valid for all p which fullfill $|p\cdot z^{-1}|<1$. We see that this is true for $|z|>|p|$. This is also called the "**Region of Convergence" (ROC)**. The ROC is connected to the resulting stability of the system or

signal. The region of convergence is outside the pole locations. If the region of convergence includes the unit circle (which describes constant, bounded oscillations), we have a stable system. This means: if the **poles are inside the unit circle**, we have a **stable system.**
The sum of x(n) **converges** (we get the sum if we set $z=1$) if **abs(p)<1**. In this case we also say that the signal or system is **stable** (meaning for a bounded input we obtain a bounded output, so-called "BIBO stability"). In this case we see that the resulting pole of our z-transform is **inside the unit circle**. If abs(p)>1, we have an exponential growth of the impulse response, which is basically an "exploding" signal or system (meaning the output grows towards infinity), hence it is **unstable**.
In general we say that a system or a signal is **stable**, if the **poles** of its z-transform are **inside the unit circle** in the z-domain, or **unstable** if **at least one pole is outside the unit circle** (it will exponentially grow).

These are basic properties, which can be used to derive z-transforms of more complicated expressions, and they can also be used to obtain an inverse z-transform, by inspection.
For instance if we see a fraction with a **pole** in the z-Transform, we know that the underlying time sequence has an **exponential decay or oscillation** in it.
Observe that we can obtain a real valued decayed oscillation if we have 2 poles, each the conjugate complex of the other, or one with $+\Omega$ and one with $-\Omega$. In this

way, we cancel the imaginary part.

One of the main differences of the z-transform compared to the Discrete Time Fourier Transform (DTFT): With the z-transform we can see if a signal or system is stable by looking at the position of the poles in the z-domain. This is not possible for the DTFT, since there we don't know the positions of the poles.

Now take a look at our down sampled signal from last time:

$$x^d(n) = x(n) \cdot \Delta_N(n) = x(n) \cdot \frac{1}{N} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N} \cdot k \cdot n}$$

Now we can z-transform it

$$\sum_{n=0}^{\infty} x^d(n) \cdot z^{-n} = \sum_{n=0}^{\infty} x(n) \cdot \frac{1}{N} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N} \cdot k \cdot n} \cdot z^{-n}$$

Hence the effect of **multiplying our signal with the delta impulse train** in the z-domain is

$$X^d(z) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\mathrm{e}^{-j\frac{2\pi}{N} \cdot k} \cdot z\right)$$

Observe that here the aliasing components appear by multiplying z with $e^{-j\frac{2\pi}{N} \cdot k}$, which in effect is a shift of the frequency.

Remember from last time, the effect of the **removal or re-insertion of the zeros** (changing the sampling rate) from or into the signal $x^d(n)$ at the higher sampling rate and $y(m)$ at the lower sampling rate. In the z-domain is

$$Y(z) = X^d(z^{1/N})$$

Filters

Filters are linear, time-invariant systems. This means they fulfil the following properties:

If $F(x(n))$ is our filter function of input signal $x(n)$, then we have

Linearity: for 2 signals $x_1(x)$ and $x_2(x)$,

$$F(x_1(n)+x_2(n))=F(x_1(n))+F(x_2(n))$$

With a factor a:

$$F(a\cdot x(n))=a\cdot F(x(n))$$

which means we can "**draw out**" **sums and factors out of our function**.

Time-Invariance: if

$$y(n)=F(x(n))$$

then we have, for a delay of n_0:

$$y(n+n_0)=F(x(n+n_0))$$

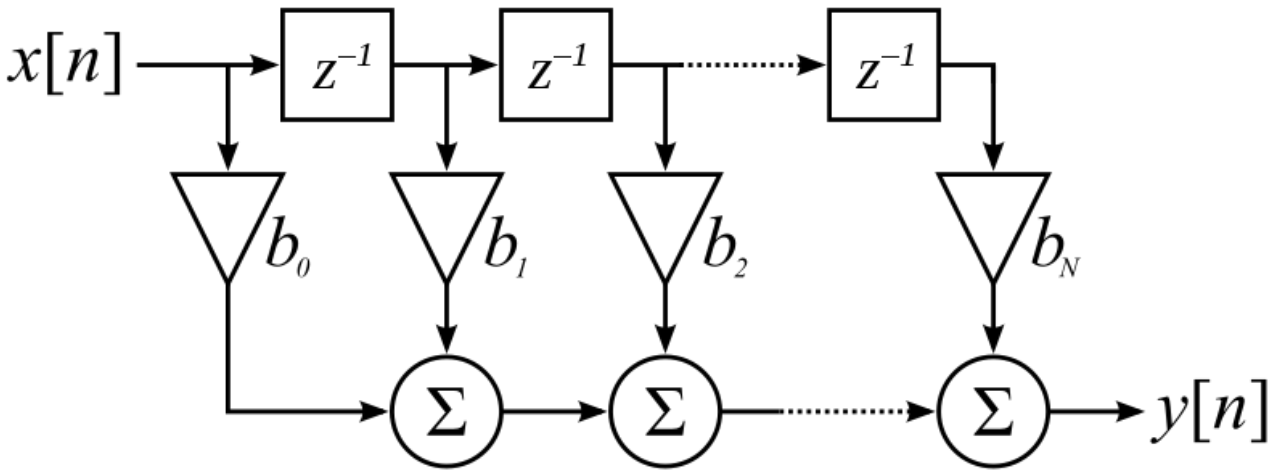
which means our function **stays the same no matter when** we apply it.

A simple Finite Impulse Response (**FIR**) filter has a difference equation like the following, with x(n) the input of our filter, and y(n) its output:

$$y(n)=\sum_{m=0}^{L} b(m)x(n-m)$$

Observe that this is the **convolution** of the signal *x(n)* with *b(n)*. Here, the *b(m)* are the coefficients of the filter, or its **impulse response**. These are commonly also referred to as "**taps**", because this system can be viewed as "tapping" a

delay line, as seen in the picture below.
This difference equation is also how usually filters are implemented in Matlab or Python other programming languages. A typical block diagram of an FIR filter is as follows



(From: http://en.wikipedia.org/wiki/Finite_impulse_response)
Observe that here the blocks with z^{-1} are implemented with a delay by 1 sampling interval, not a multiplication with z^{-1}, as we would in the z-domain!
After the first delay block z^{-1} we have x[n-1], after the second delay block we have x[n-2], and so on. Each delay block "memorizes" the value from the left for one sample clock cycle, and releases it to the right at the next sample clock cycle.
Hence they delay samples by 1 sample, or one sample clock cycle.

The z-transform of our difference equation of our convolution

$$y(n)=\sum_{m=0}^{L} b(m)\,x(n-m)$$

is (using the linearty and the delay property of the z-Transform):

$$Y(z)=\sum_{m=0}^{L} b(m)\cdot z^{-m}\cdot X(z)=X(z)\cdot\sum_{m=0}^{L} b(m)\cdot z^{-m}$$

Now we can compute the **transfer function**, defined as the output divided by the input,

$$H(z):=\frac{Y(z)}{X(z)}=\sum_{m=0}^{L} b(m)\cdot z^{-m}$$

Observe that this is the z-transform of the coefficients b(m)! This is the **z-transform of the impulse response** of the FIR filter!

Now we can obtain the **frequency response** (so that we can see which frequencies are attenuated and which are not) from our transfer function of the filter by just replacing z by $e^{j\Omega}$:

$$H(e^{j\Omega})=\sum_{m=0}^{L} b(m)\cdot e^{-j\Omega\cdot m}$$

Since $e^{j\Omega}$ is a complex number, our frequency response is also complex. Hence H is a complex number for each frequency Ω . Usually it is plotted as a **magnitude** plot and a **phase** plot over frequency. Its magnitude tells us the attenuation at each frequency, and the phase its phase shift for each frequency. Using those 2 plots or properties, we can also design our filters with desired properties (for instance a stop-band at given frequencies). The Matlab and

Python function to generate a magnitude and phase plot of a transfer function or signal is “freqz”, which we already saw.

IIR Filters

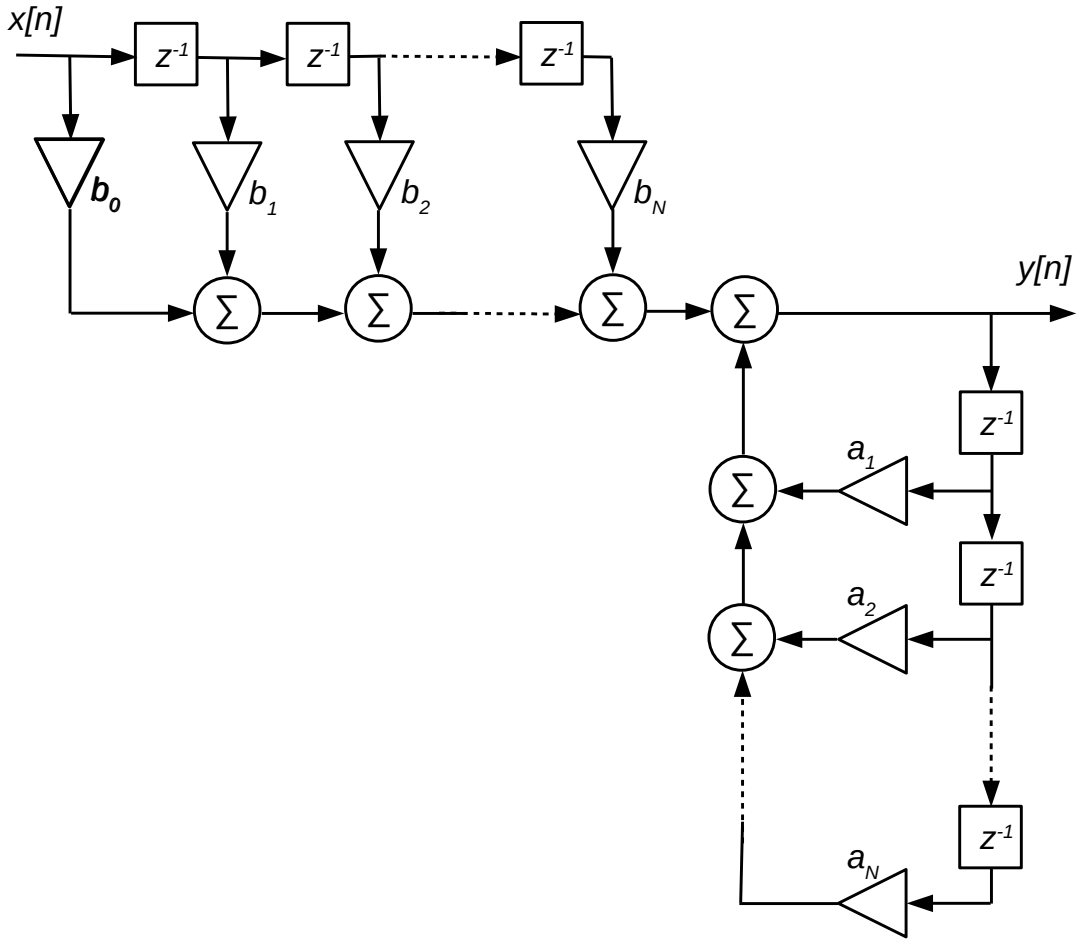
(Infinite Impulse Response Filters).
Their difference equation is:

$$y(n)=\sum_{m=0}^{L} b(m)\cdot x(n-m)+\sum_{r=1}^{R} a(r)\cdot y(n-r) \quad (1)$$

(See also: Oppenheim, Schafer: “Discrete Time Signal Processing”, Chapter 6 in Ed. 1989)

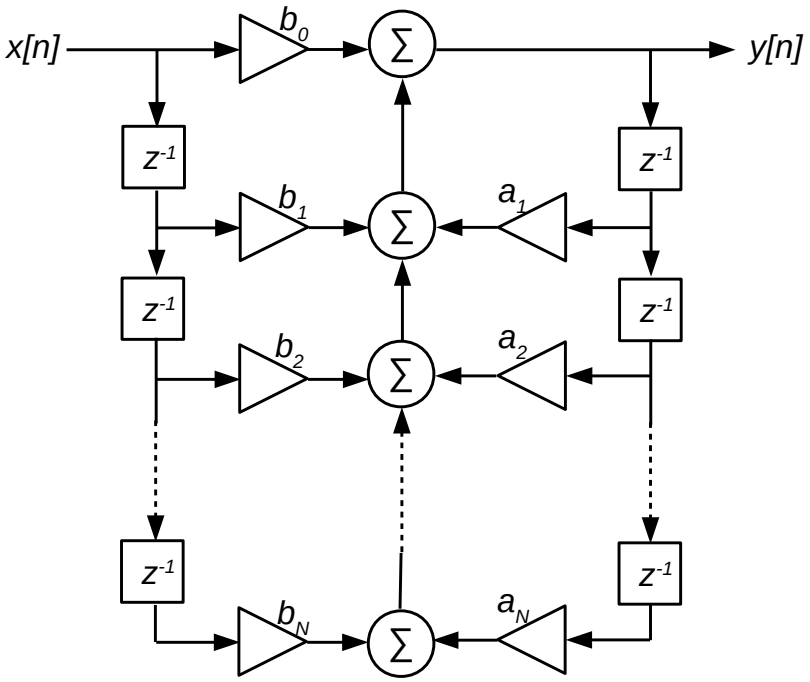
Here we have **2 convolutions**. Observe the feedback from the output y back to the input in this sum. Also observe that the feedback part starts with a delay of r=1. This is because we want to avoid so-called delayless loops. We cannot use the value y(n) before we computed it! Again, this difference equation is the usual implementation using Matlab or Octave or Python.

The following image shows the corresponding block diagram of our filter,



Again, here the boxes with z^{-1} symbolize a delay of 1 sampling period, and the triangles symbolize a multiplication with the factor written next to it.

We can simplify this structure by combining the summations,



(Figure 1)

The z-transform of its difference equation (1) is

$$Y(z)=\sum_{m=0}^{L} b(m)\cdot X(z)\cdot z^{-m}+\sum_{r=1} a(r)\cdot Y(z)\cdot z^{-r}$$

Observe: **Matlab and Octave** and **Python scipy.signal.lfilter** are defining the coefficients *a* with **opposite signs** as we and Oppenheim/Schafer are defining. See for instance "help filter" or "help(scipy.signal.lfilter)" in Python.

To obtain its transfer function, we first move the Y(z) to one side:

$$Y(z)\left(1-\sum_{r=1}^{R} a(r)\cdot z^{-r}\right)=X(z)\cdot\sum_{m=0}^{L} b(m)\cdot z^{-m}$$

Hence the resulting transfer function is

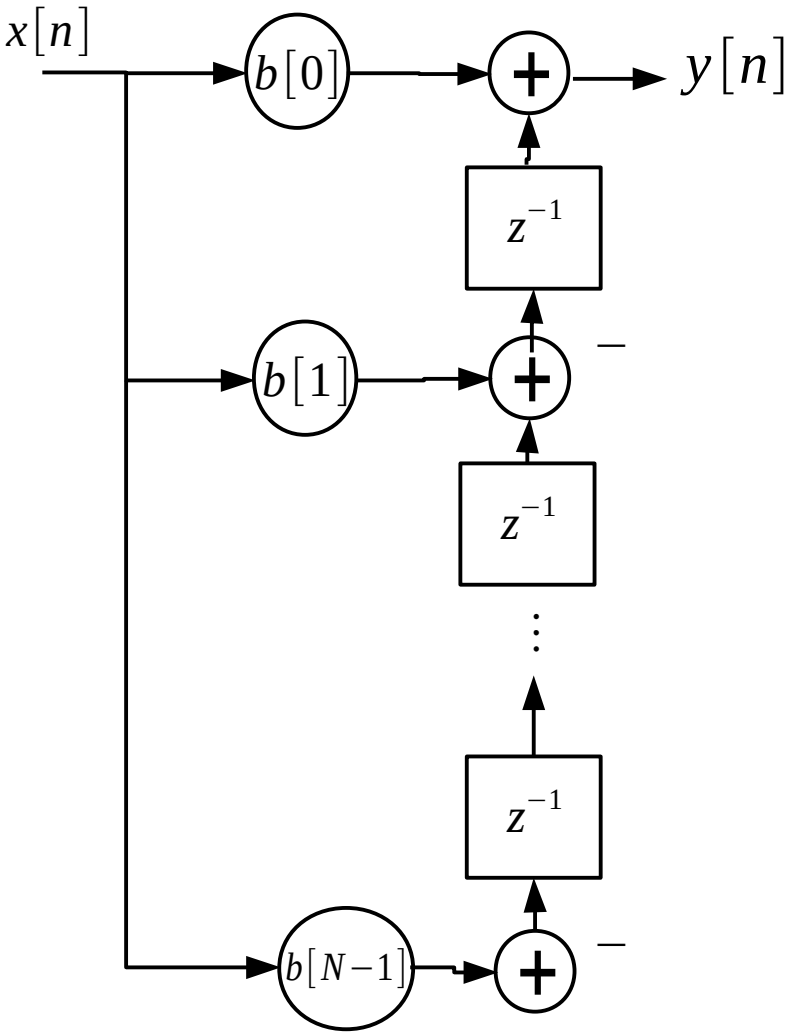
$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{m=0}^{L} b(m)\cdot z^{-m}}{1-\sum_{r=1}^{R} a(r)\cdot z^{-r}}$$

Observe: With the help of the z-Transform we were able to find a closed form solution for the transfer function, even though we have a feedback loop in our system! This is a **big advantage** for the z-Transform.

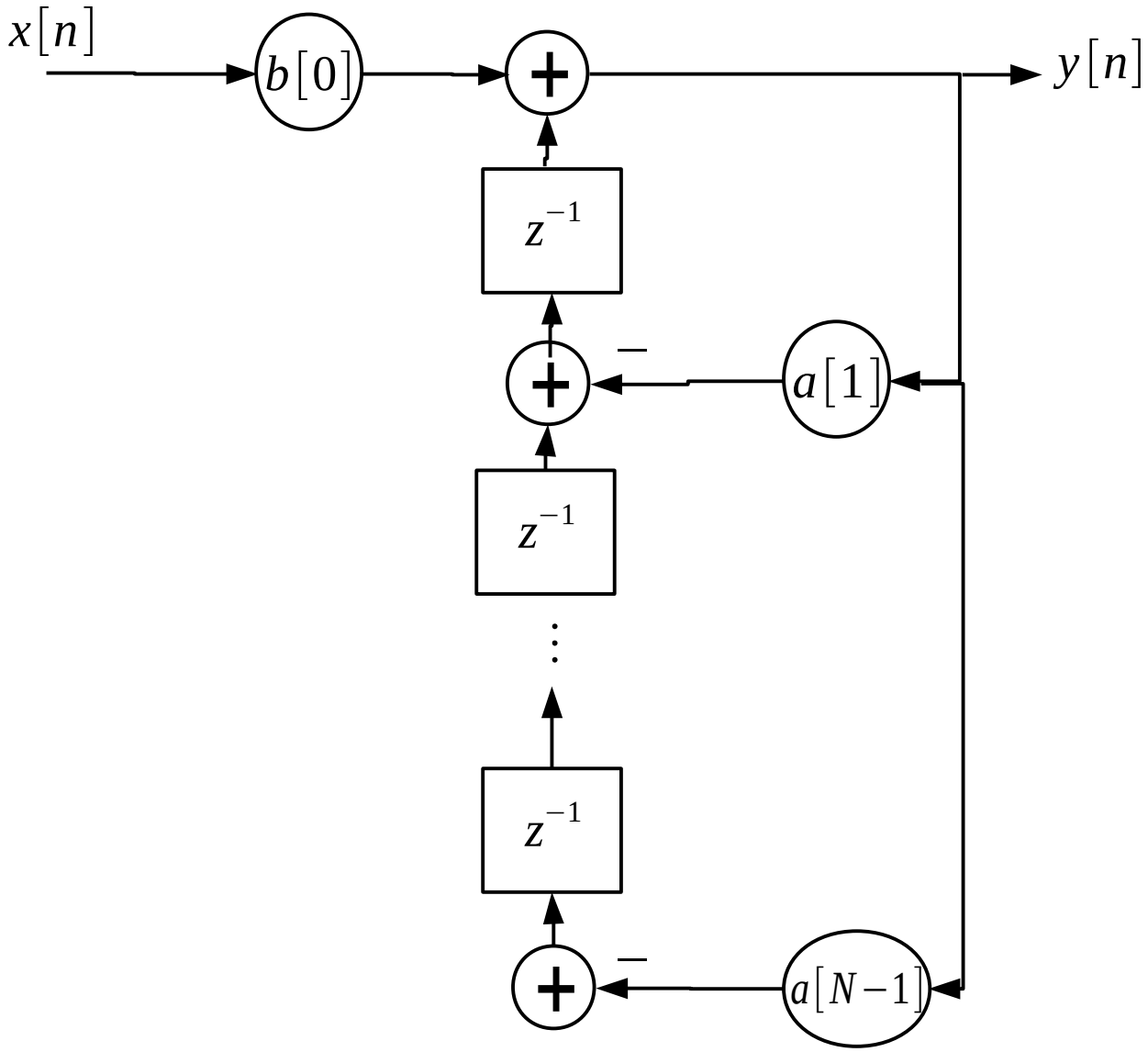
Here we can see that we obtain a polynomial in the **denominator** of our transfer function, and hence poles. And the filter can potentially become unstable! The zeros of this denominator polynomial become the poles of the transfer function. If these poles are all **inside the unit circle**, we have a **stable filter**! This also shows that we just need to design our coefficients a(n) such that the **poles are inside the unit circle for stabiltity**.

We can take Figure 1 and shift the delays z^{-1} after the multiplication, and obtain the following structure. Here, the circles symbolize a multiplication with the values inside the circle,

FIR:

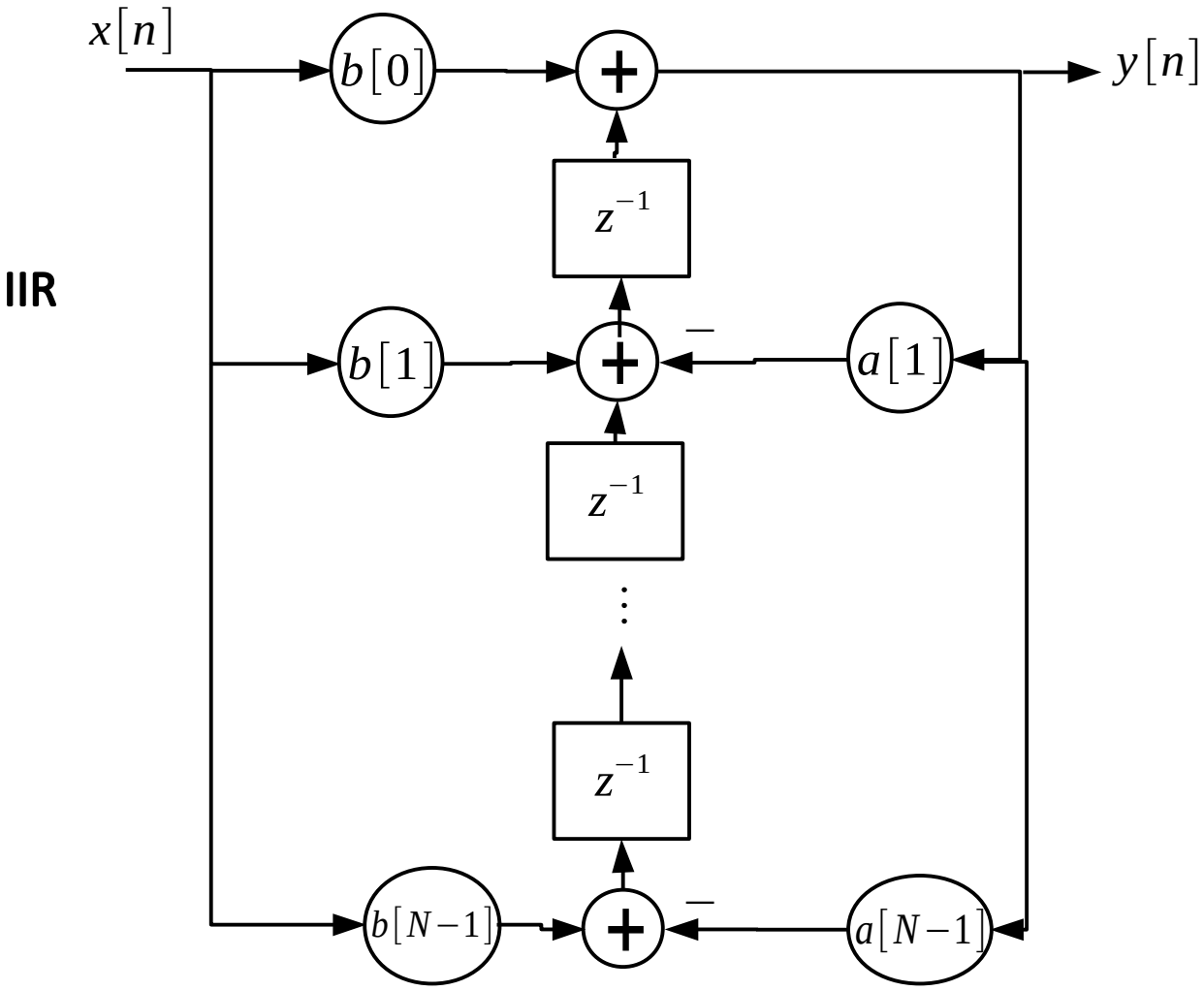


IIR Structure (feedback part):



Combined FIR-IIR Structure used in the Python “lfilter” Function:

The two previous FIR and IIR (feedback part) structures had the delays in the middle. Since a delay is a linear operator, we can shift it after a summation, and hence can combine the delay chain for the FIR and IIR part. This reduces the memory requirement for an implementation, and leads to the following structure,



Filter Example:

Going back to our simple example of an exponential decaying signal, this shows how to implement it. We just need a system with a pole at position *p*. In the above equation we obtain it by setting *b(0)=1* and *a(1)=p*. Hence we obtain a simple difference equation

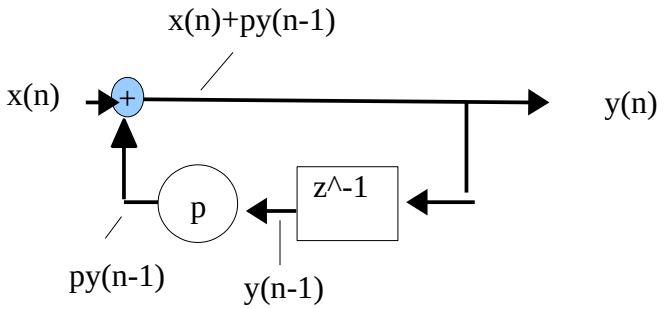
$$y(n)=1\cdot x(n)+ p\cdot y(n-1)$$

Here you can see: if x(n) is the unit pulse, the output is the exponential decaying sequence:

$$1, p, p^2, p^3, \ldots$$

which is an infinitely long impulse response. Hence the name IIR.

This can also be written in the form of a block diagram:



(also compare it with the Python implementaion block diagram above, for b[0]=1 and a[0]=1, a[1]=p. Python always wants to have a[0] and it should be 1! See also help(lfilter)).

In the z-domain this is

$$Y(z) = X(z) + p \cdot z^{-1} \cdot Y(z)$$

$$\rightarrow H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - p \cdot z^{-1}}$$

In this structure we can now see the feedback loop. Observe that this is the same as the z-Transform of our exponential series before. This means, when we transform this back to the time domain, we obtain an exponential function, which is the **filters impulse response**, which confirms what we just saw by using an impulse in the time domain for our filter. So the result of the inverse z-Transform of our transfer function is indeed,

$$1, p, p^2, p^3, \ldots .$$

Computing the Resulting Frequency Response

Example: The Matlab or Octave or Python function "*freqz*" (we have a Python freqz function also in Moodle) can be used to plot the magnitude and phase plot of the transfer function of this filter. Its input are directly the coefficients *a* and *b* of the transfer function H(z), in the form:

freqz(b,a),

where b and a are vectors containing the coefficients.

We can use `ipython -pylab`

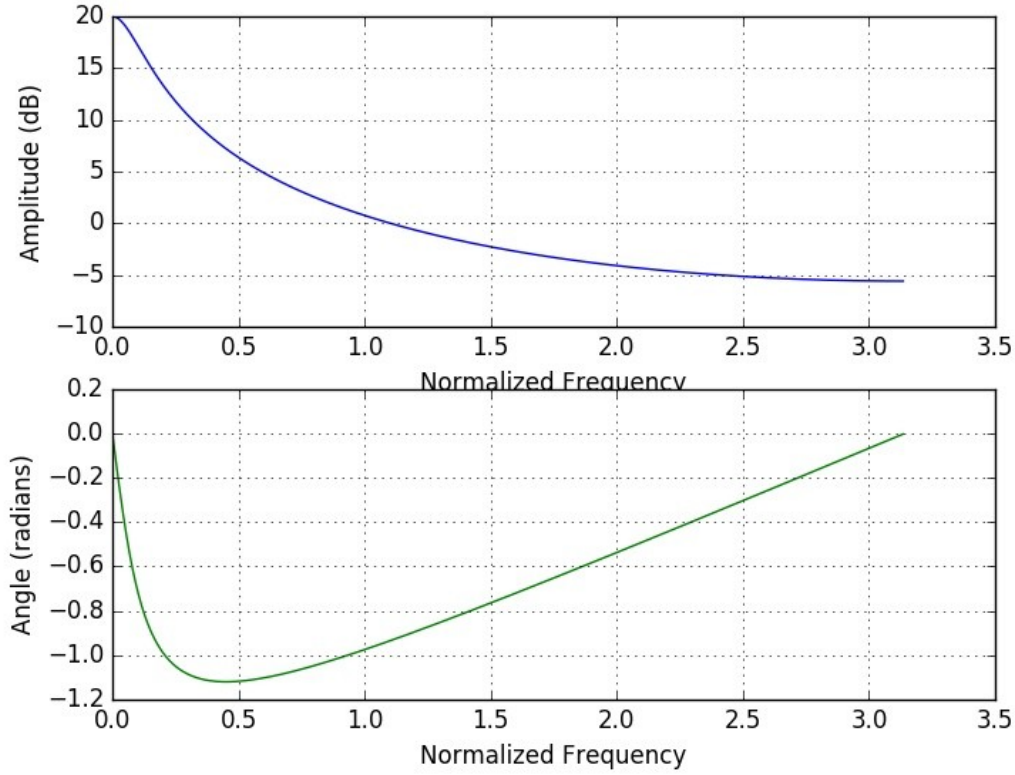
If we choose a(1)=*p=0.9* in our example, we obtain

a=[1,-0.9] and b=*[1]*.

Then we type:

from freqz import freqz
w,H=freqz(b,a)

and obtain the following plot:



Observe that the horizontal axis is the normalized frequency (see last lecture), its right hand side is pi, which is the Nyquist frequency or half the sampling frequency. The frequency response we see here has a low pass characteristic.

We can use the command “zplane” (also in Moodle) to plot the location of the zeros and poles in the complex z-plane. For that we first need to calculate the pole and zero positions with Pythons function `np.roots`.

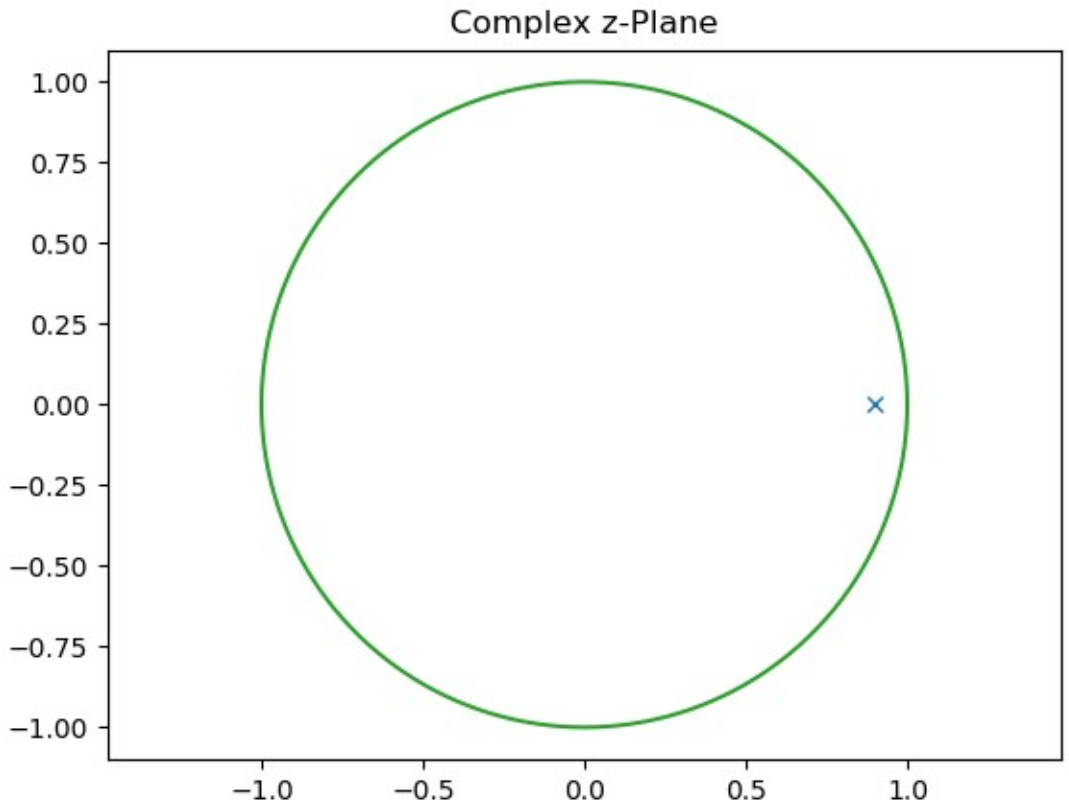
We have $H(z)=\frac{1}{1-p\cdot z^{-1}}$

and `roots` expects as argument the ordered coefficients of our polinomial in z^{-1} (which makes it compatible to `lfilter`). Hence our pole positions are at `np.roots(a)` and the zeros at `np.roots(b)`.

We plot their positions with,

```
from zplane import zplane
zplane(np.roots(b),np.roots(a));
```

The resulting plot is:



Zeros are marked with an "**o**", and **poles** are marked with an "**x**". Here we see the pole at location z=0.9.

In general, the **closer a pole** is to the **unit circle**, the larger is the corresponding **peak** in the **magnitude** of the frequency response at a normalized **frequency** identical to the **angle of the pole** to the origin.
This can be seen by turning the z-transform into a DTFT by replacing z by $e^{j\Omega}$. Hence the frequency response is

obtained by “running” on this unit circle, Ω is our angle and also normalized frequency, and the closer we get to a pole, the higher the magnitude of the frequency response becomes. This is opposite for zeros, the closer we come to a zero, the smaller magnitude of the frequency response becomes.

Here our pole has an angle of 0 degrees. In the example we can indeed observe a peak in the magnitude response at normalized frequency $\Omega=0$ above.

The **filtering** operation itself works similarly in Matlab or Octave or Python, in the time domain. The function is “*filter*” or “*lfilter*”. Given an input signal in the vector x, and filter coefficients in vectors a and b, the filtered output y of our filter is simply:

```
y=lfilter(b,a,x);
```

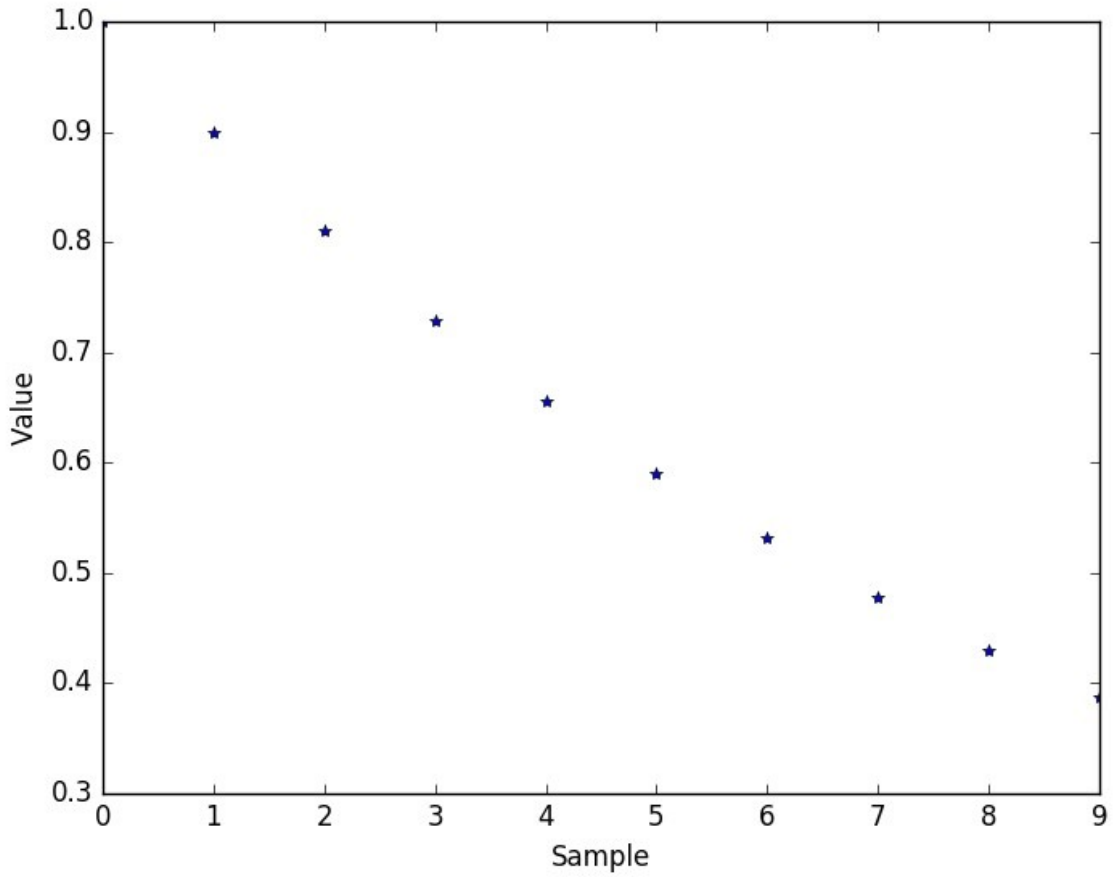
Often used orders for the zeros and poles L and R are a few to a few dozen coefficients.

Python example:

Use the function l*filter* to obtain the impulse response of this IIR filter.

```
ipython -pylab
import scipy.signal
#Start with a unit pulse as input x:
x=np.zeros(10)
x[0] = 1
```

```
#B and A are given as before:
A=[1,-0.9];
B=[1];
#Now calculate the impulse response:
y=scipy.signal.lfilter(B,A,x);
plot(y,'*')
xlabel('Sample')
ylabel('Value')
```



Here we can see the indeed exponetial decaying function (the sequence $ir(n) = p^n$ for p=0.9). In this way we can also test more complicated IIR filters. This exponential decaying impulse response again shows the stability of the

filter, which was to be expected because the pole of its transfer function in the z-domain is placed at z=0.9, and hence inside the unit-circle!