# 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

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

First observe that we get our usual frequency response if we evaluate the z-tranform along the unit circle in the z-domain,

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

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 z with an angle and a magnitude,

$$z=r\cdot e^{j\Omega}$$
where we can interpret the $\Omega$ as the normalized angular frequency, and the $r$ a damping factor for an exponentially decaying oscillation (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:

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

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

## Z-Transform Properties

z-Transform definition:

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

The z-transform turns a sequence into a polynomial in z.

Example: $x(n)=[2,4,3,1]$

$$X(z)=2+4\cdot z^{-1}+3\cdot z^{-2}+z^{-3}$$

Shift property:

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

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

**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 is

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

Shifted by d samples.
The “unit step” function is defined as:

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

**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-transform) of these sequences. This is one of the main advantages of the z-Transform, since it turns convolution into a simpler multiplication (which is in principle 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.

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 include the unit circle, 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 we obtain a bounded output for a bounded input, 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, which is basically an “exploding” signal or system (meaning the output grows towards infinity), hence **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** in it.

One of the main differences compared to the 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 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

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

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.

The effect is the **removal or re-insertion of the zeros** 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

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

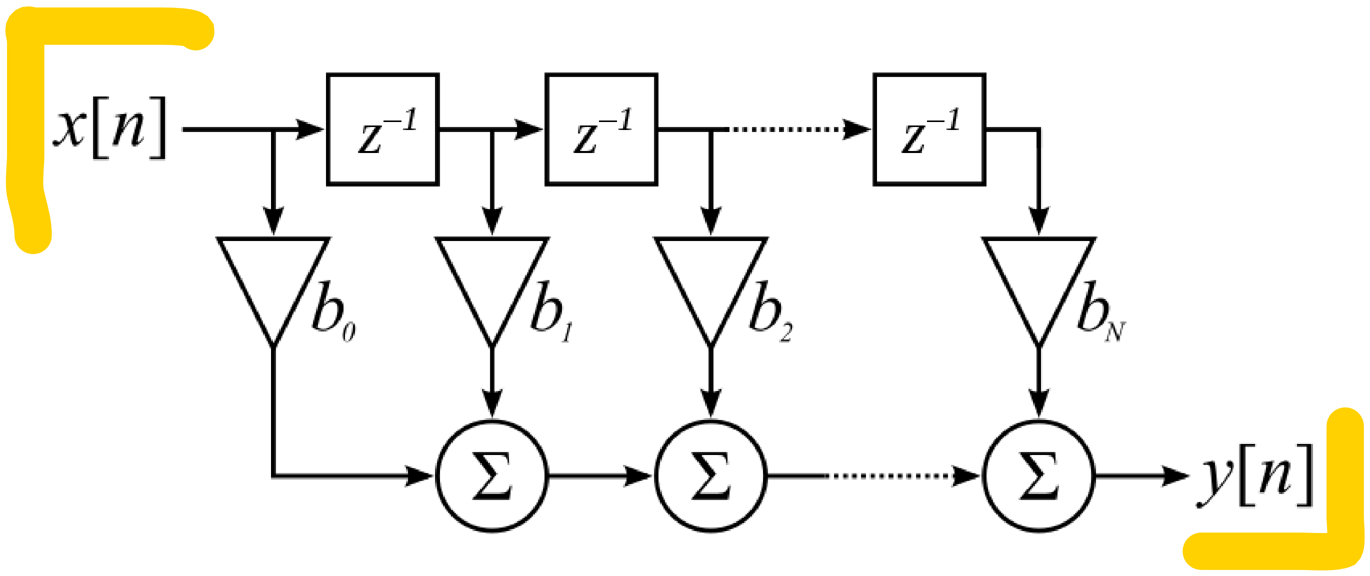
## Filters

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 other programming languages. A typical block diagram of an FIR filter is as follows



(From: http://en.wikipedia.org/wiki/Finite_impulse_response)

The z-transform of this difference equation is:

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

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 $\omega$. 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 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:

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

(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.
The z-transform of this difference equation 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** is defining the coefficients *a* with **opposite signs** as we and Oppenheim/Schafer are defining. See for instance “help filter”.
To obtain the transfer function, we first move the Y(z) to one side:

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

Hence the resulting transfer function is

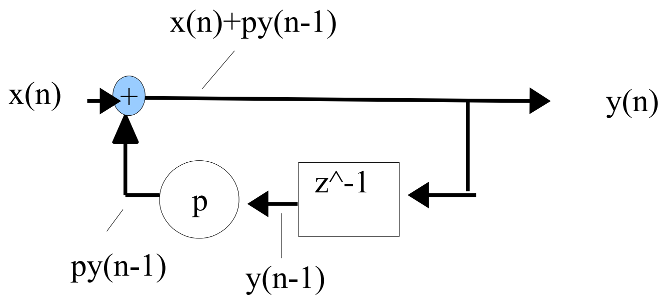
$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{m=0}^{L} b(m)\cdot z^{-m}}{1-\sum_{r=1}^{R} a(r)\cdot z^{-r}}$$

Here we can see that we obtain a polynomial in the denominator of our transfer function. The zeros of this denominator polynomial become the poles of the transfer function. If these poles are **inside the unit circle**, we have a **stable filter**!
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

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

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



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.

**Example:** The Matlab or Octave function "*freqz*" 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.

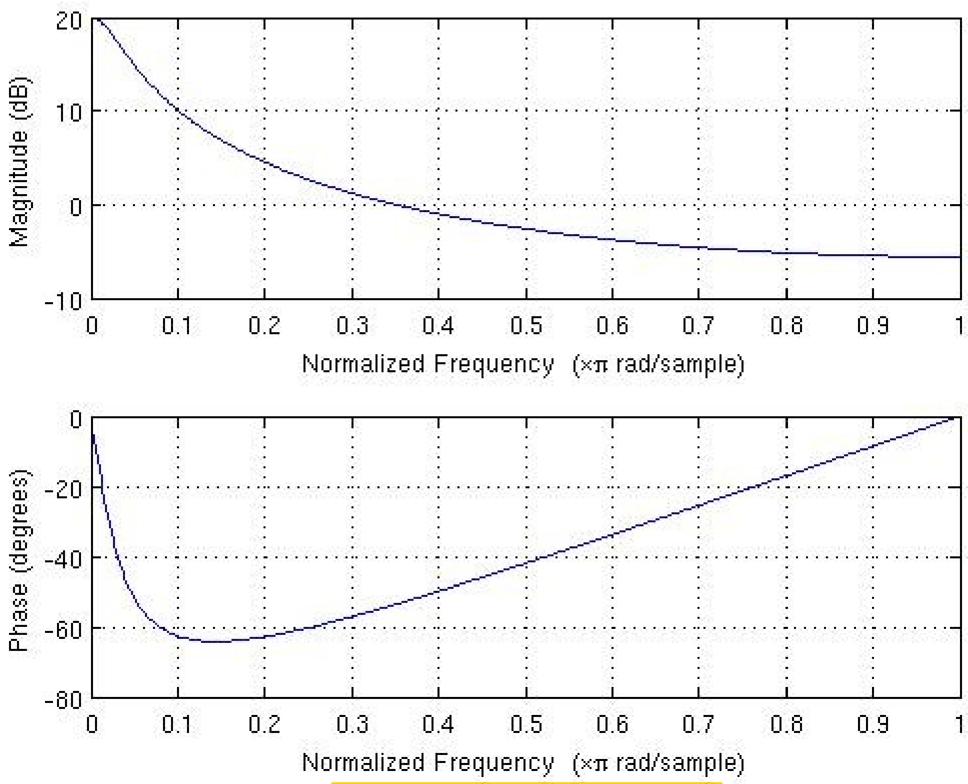
If we choose a(1)=*p=0.9* in our example, we obtain

*A=[1,-0.9]* and *B=1*.

Hence we type:

*freqz(1,[1,-0.9])*

and obtain the following plot:

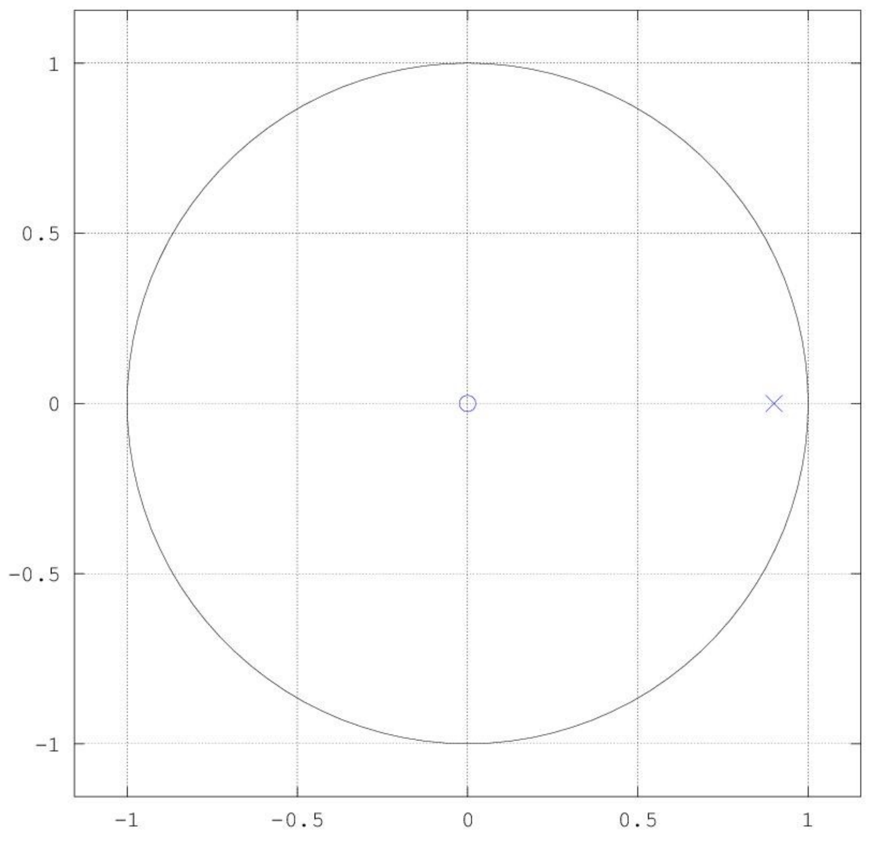


Observe that the horizontal axis is the normalized frequency (see last lecture), its right hand side is pi, which is the Nyquist frequency or half the sampling frequency. The frequency response we see here has a low pass characteristic.

We can use the command "zplane" to plot the location of the zeros and poles in the complex z-plane, with

*zplane(B,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. Here we have an angle of 0 degrees. In the example we can indeed observe a peak in the magnitude

response at normalized frequency $\Omega=0$ above.

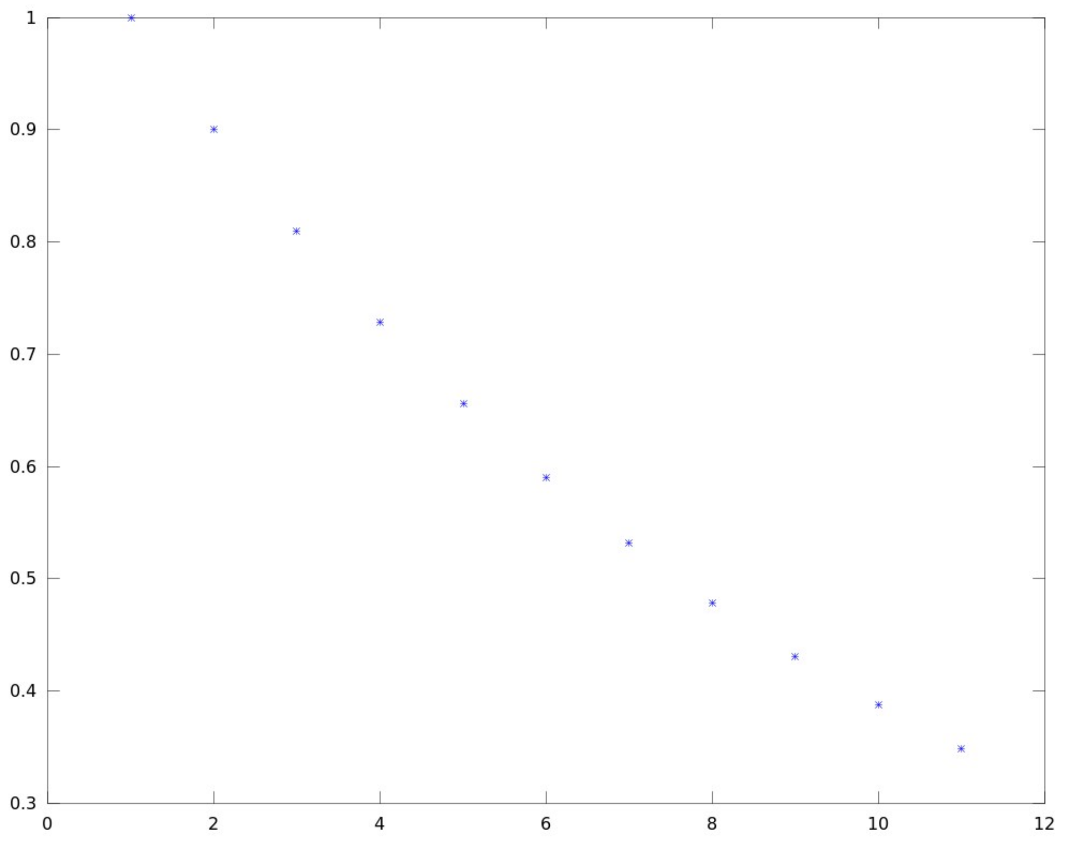
The **filtering** operation itself works similarly in Matlab or Octave, in the time domain. The function is "*filter*". Given an input signal in the vector x, and filter coefficients in vectors A and B, the filtered output y of our filter is simply:

*y=filter(B,A,x);*

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

**Octave/Matlab example:**
Use the function *filter* to obtain the impulse response of this IIR filter.
Start with a unit impulse as input x:

```
x=[1,zeros(1,10)];
%B and A are given as before:
A=[1,-0.9]; B=1;
%Now calculate the impulse response:
y=filter(B,A,x);
plot(y,'*')
```



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!