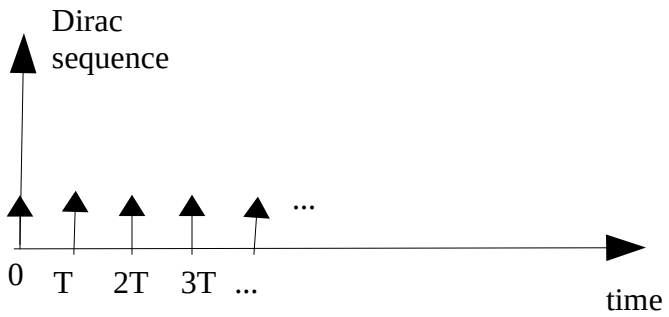
# Digital Signal Processing 2/ Advanced Digital Signal Processing
# Lecture 6,
# Sampling

Gerald Schuller, TU Ilmenau

## Sampling, Downsampling, Upsampling
## Sampling the analog signal, normalized frequency

To see what happens when we sample a signal, lets start with the analog signal s(t). Sampling it means to sample the signal at sample intervals T (example: 1/8000 s in the ISDN example), or the sampling frequency $f_s$ (8kHz in the ISDN example). Mathematically, sampling can be formulated as multiplying the analog signal with a Dirac impulse at the sampling time instances nT, where n is the number of our sample (n=0,1,... for causal systems).




The sequence or train of Dirac impulses can be written as:

$$\Delta_T(t) := \sum_{n=0}^{\infty} \delta(t - nT)$$

Remember that the integral over a Dirac impulse is 1:

$$\int_{t=-\infty}^{\infty} \delta(t)\,dt = 1$$

(see also: http://en.wikipedia.org/wiki/Dirac_delta_function)

The integral over the product of a function with a Dirac impulse is the value of the function at the position of the impulse:

$$\int_{t=-\infty}^{\infty} s(t)\,\delta(t - nT)\,dt = s(nT) \quad (1)$$

This is the mathematical formulation of sampling at time-point nT. We can now use this description to compute the resulting **spectrum** with the **Fourier transform**.
First, if the look at the Fourier transform of the analog signal or system, we get

$$S^c(\omega) = \int_{t=-\infty}^{\infty} s(t) \cdot e^{-j\omega t}\,dt$$

where the superscript c denote the continuous version, with $\omega = 2\pi f$ the angular frequency. If we now compute the Fourier Transform for the sampled signal $s(t) \cdot \Delta_T(t)$ , with the replacement

$$s(t) \leftarrow s(t) \cdot \Delta_T(t)$$

we obtain the following sum (since only at the sampling time instances the integral is non-zero, as we see in eq. (1)),

$$S^d(\omega) = \sum_{n=-\infty}^{\infty} s(nT) \cdot e^{-j\omega nT}$$

with the superscript d now denoting the discrete time version. Now we can see that the frequency variable only appears as $\omega nT$ , and T is the inverse of the sampling frequency. Hence we get

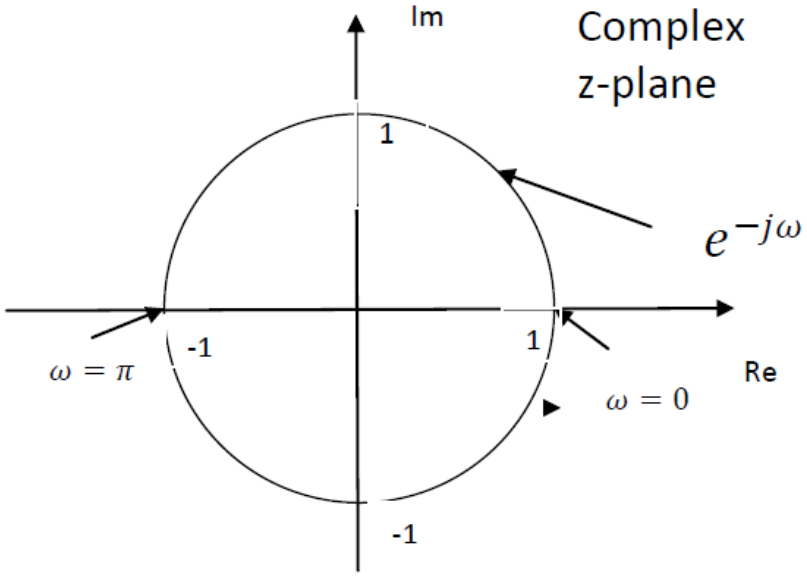
$$\omega T = \omega / f_s =: \Omega$$

This is now our **normalized frequency**, it is without a physical unit, since the unit Hertz in $\omega$ and $f_s$ cancel. In the normalized frequency $2\pi$ represents the sampling frequency and $\pi$ is the so called **Nyquist frequency** (the upper limit of our usable frequency range, defined as **half** the sampling frequency). Observe that we use the capital $\Omega$ to signify that this is the normalized frequency. The capitalized version is commonly used to distinguish it from the continuous, non-normalized, version, if both are used. Otherwise, also the small $\omega$ is used in the literature, for the normalized frequency, if that is all we need, if there is no danger of confusion, as we did before.
To convert a normalized frequency back to the non-normalized one, we simply multiply it with the sampling frequency: $\omega = \Omega \cdot f_s$ .
To indicate that we are now in the discrete time domain, we rename our signal to x(n)=s(nT).
Its spectrum or frequency response is then

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\Omega n}$$

We call this the "**Discrete Time Fourier Transform**" (**DTFT**), because we apply it to discrete time signals x(n). This can be distinguished from the Discrete Fourier Transform, because the above time signal has infinite length.

Because n is integer here (no longer real valued like t), we get a $2\pi$ **periodicity** for $X(\Omega)$ . This is the first important property for discrete time signals.




Here we can see that in general we obtain a $2\pi$ periodicity in the frequency domain, because of the $2\pi$ periodicity of the exponential function $e^{j\omega}$ .

Also observe that for **real valued** signals, the spectrum of the negative frequencies is the conjugate complex of the positive frequencies,

$$X(-\Omega)=X^*(\Omega)$$

where $^*$ denotes the conjugate complex operation, because in our DTFT, $e^{-j(-\Omega)n}=(e^{-j\Omega n})^*$ .

(we also have to use the fact that the conjugate complex of a real valued signal does not change it, to draw the conjugation out of the Fourier sum).

This also means that for real valued signals we only need to look at the frequency range between 0 and $\pi$ , since the negative frequencies are conjugate symmetric (and because

of the 2pi periodicity the range between 0 and pi is sufficient to look at). This is also again what the Nyquist Theorem tells us, that the frequencies between 0 and $\pi$ (the Nyquist frequency) are sufficient to completely describe a (real valued) signal.

## Sampling a discrete time signal

So what happens if we further downsample an already discrete signal x(n), to reduce its sampling rate?
**Downsampling by N** means we only keep every Nth sample and discard every sample in between. Observe that this results in a **normalized frequency which is a factor of N higher**.
This downsampling process can also be seen as first multiplying the signal with a sequence of **unit pulses** (a 1 at each sample position), zeros in between, and later dropping the zeros. This multiplication with the unit pulse train can now be used to mathematically analyse this downsampling, looking at the resulting spectra, first still including the zeros.
The frequency response now becomes

$$X^d(\Omega) = \sum_{n=mN} x(n) e^{-j\Omega n}$$

$$= \sum_{m=-\infty}^{\infty} x(mN) e^{-j\Omega mN}$$

for all integers m.

Again we can write down-sampling as a multiplication of the signal with a sampling

function. In continuous time it was the sequence of Dirac impulses, here it is a **sequence of unit pulses** at positions of multiples of N,

$$\Delta_N(n)=\begin{pmatrix}1,\ if\ \ n=mN\\ 0,\ else\end{pmatrix}$$

Then the sampled signal, with the zeros still in it, becomes

$$x^d(n)=x(n)\Delta_N(n)$$

This signal is now an intermediate signal, which gets the zeros removed before transmission or storage, to reduce the needed data rate. The decoder upsamples it by re-inserting the zeros to obtain the original sampling rate.

Observe that this is then also the signal that we obtain after this upsampling in the decoder. Hence this signal looks interesting to us, because it appears in the encoder and also in the decoder.

What does its **spectrum** or **frequency response** look like?

We saw that the downsampled signal $x^d(n)$ can be written as the **multiplication** of the original signal with the unit pulse train $\Delta_N(n)$. In the spectrum or frequency domain this becomes a **convolution with their Fourier transforms**. The Fourier transform of the unit pulse train is a dirac impulse train. But

that is not so easy to derive, so we just apply a simple trick, to make it mathematically more easy and get converging sums. We have a **guess** what the Fourier transform of the unit pulse train is: we get Dirac impulses at frequencies of $\frac{2\pi}{N}k$ (fundamental frequency $2\pi/N$ and its harmonics). The fundamental (angular) frequency is $2\pi/N$ , because our original sampling frequency was $2\pi$ , and now we reduced it by a factor N.
With a constant factor that becomes clear later, we get our guess to

$$\delta_{2\pi/N}(\Omega)=\sum_{k=0}^{N-1}\frac{2\pi}{N}\delta\left(\Omega-\frac{2\pi}{N}\cdot k\right)$$

Where $\delta_{2\pi/N}(\Omega)$ is (guessed to be) the Fourier transform of the unit impulse train (the Dirac impulse train). We just need to verify it. Its **inverse Discrete Time Fourier Transform** is

$$\Delta_N(n)=\frac{1}{2\pi}\int_0^{2\pi}\delta_{2\pi/N}(\Omega)e^{jn\Omega}\,d\Omega$$

$$= \frac{1}{2\pi} \int_0^{2\pi} \sum_{k=0}^{N-1} \frac{2\pi}{N} \delta\left(\Omega - \frac{2\pi}{N} \cdot k\right) e^{jn\Omega} d\Omega$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N} \cdot k \cdot n}$$

Now we have to prove that this is indeed our unit pulse train. To do that, we can look at the sum. It is a geometric sum (a sum over a constant with the summation index in the exponent),

$S = \sum_{k=0}^{N-1} c^k$, $S \cdot c = \sum_{k=1}^{N} c^k$, $Sc - S = c^N - 1$, hence we get

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

Here we get $c = e^{j\frac{2\pi}{N} n}$, and the sum can hence be computed in a closed form,

$$\sum_{k=0}^{N-1} e^{j\frac{2\pi}{N} \cdot n \cdot k} = \frac{e^{j\frac{2\pi}{N} \cdot n \cdot N} - 1}{e^{j\frac{2\pi}{N} \cdot n} - 1}$$

In order to get our unit pulse train, this sum must become N for n=mN. We can check this by simply plugging this into the sum. The right part of the sum is undefined in this

case (0/0), but we get an easy result by looking at the left hand side: $e^{j\frac{2\pi}{N}\cdot mN\cdot k}=1$. Hence the sum becomes N, as desired! For $n\neq mN$ we need the sum to be zero. Here we can now use the right hand side of our equation. The denominator is unequal zero, and the numerator becomes zero, and hence the sum is indeed zero, as needed! This proves that our **assumption was right**, and we have

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

Now we can use this expression for the unit pulse train in the time domain and compute the Fourier transform,

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

Taking its Discrete Time Fourier transform now results in

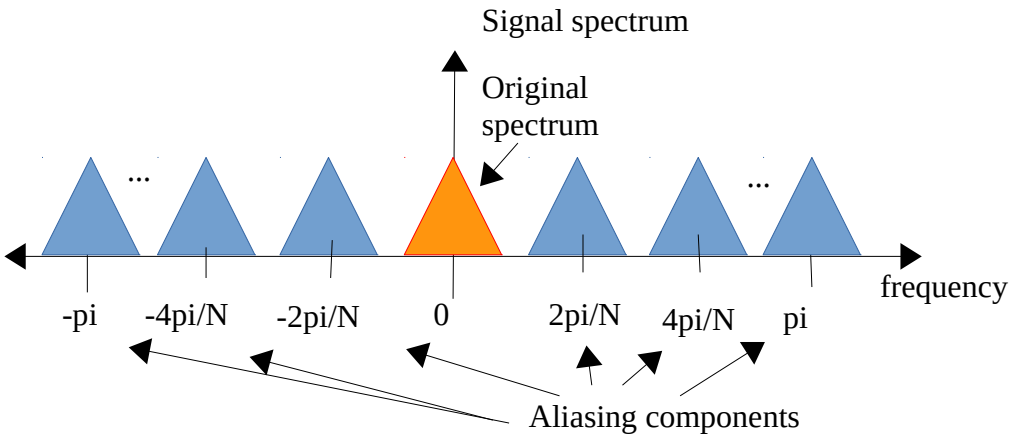
$$X^d(\Omega)=\sum_{n=-\infty}^{\infty} x(n)\cdot\frac{1}{N}\sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}\cdot k\cdot n}\cdot e^{-j\Omega n}=$$

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

$$= \frac{1}{N} \sum_{k=0}^{N-1} X\left(-\frac{2\pi}{N} k + \Omega\right)$$ ←

This shows, that sampling, still including the zeros, leads (in the frequency domain) to **multiple shifted versions** of the signal spectrum, the so-called **aliasing components**,

$$X^d(\Omega) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\frac{-2\pi}{N} \cdot k + \Omega\right) \quad \text{(eq.1)}$$

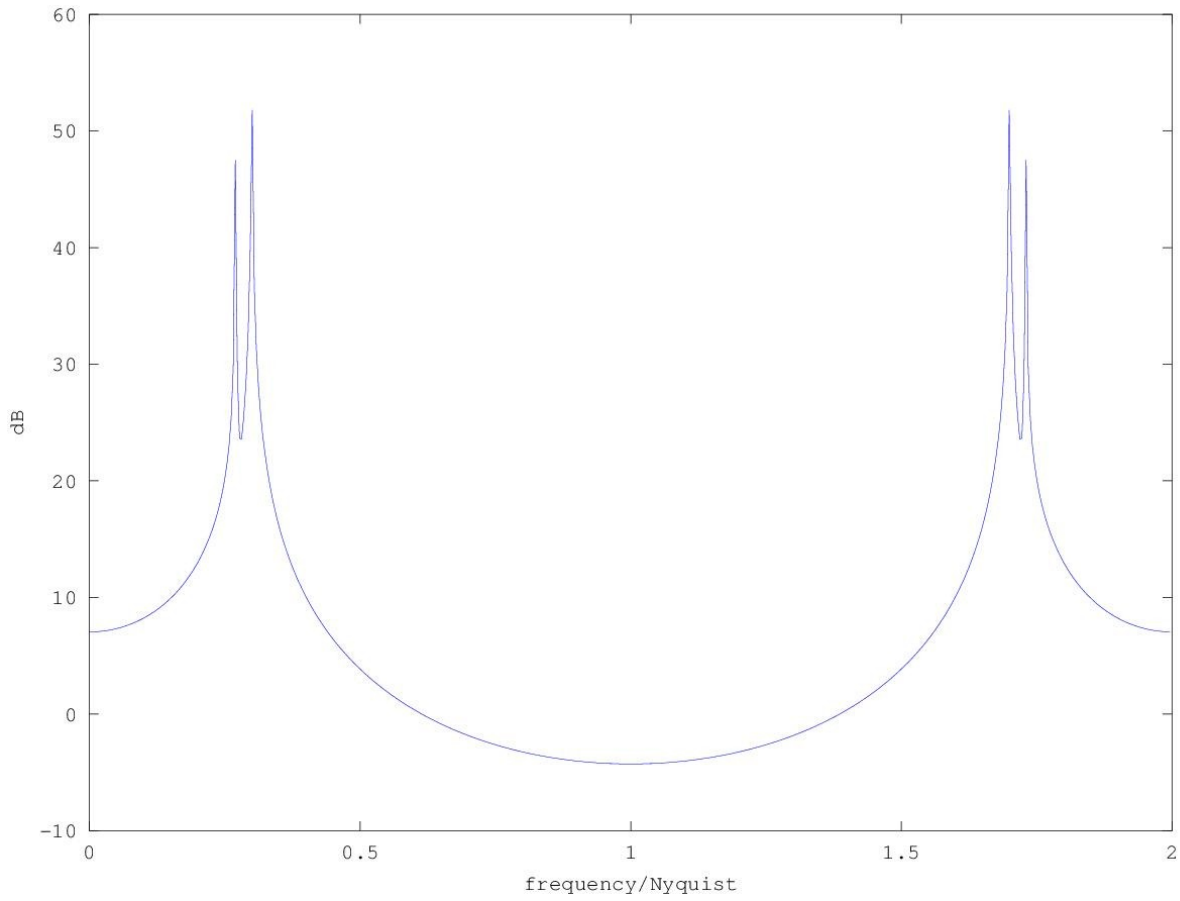



**Observe:** The aliasing components **periodify** the spectrum now according to the **new sampling rate**, which is 1/N of the old sampling rate, hence the new period is 1/N the old period (of $2\pi$ ).

**Observe:** The spectral components don't overlap if their bandwidths is below $2\pi/N$ for complex signals, or for a real low pass signal, it should be below $\pi/N$ ! If we want

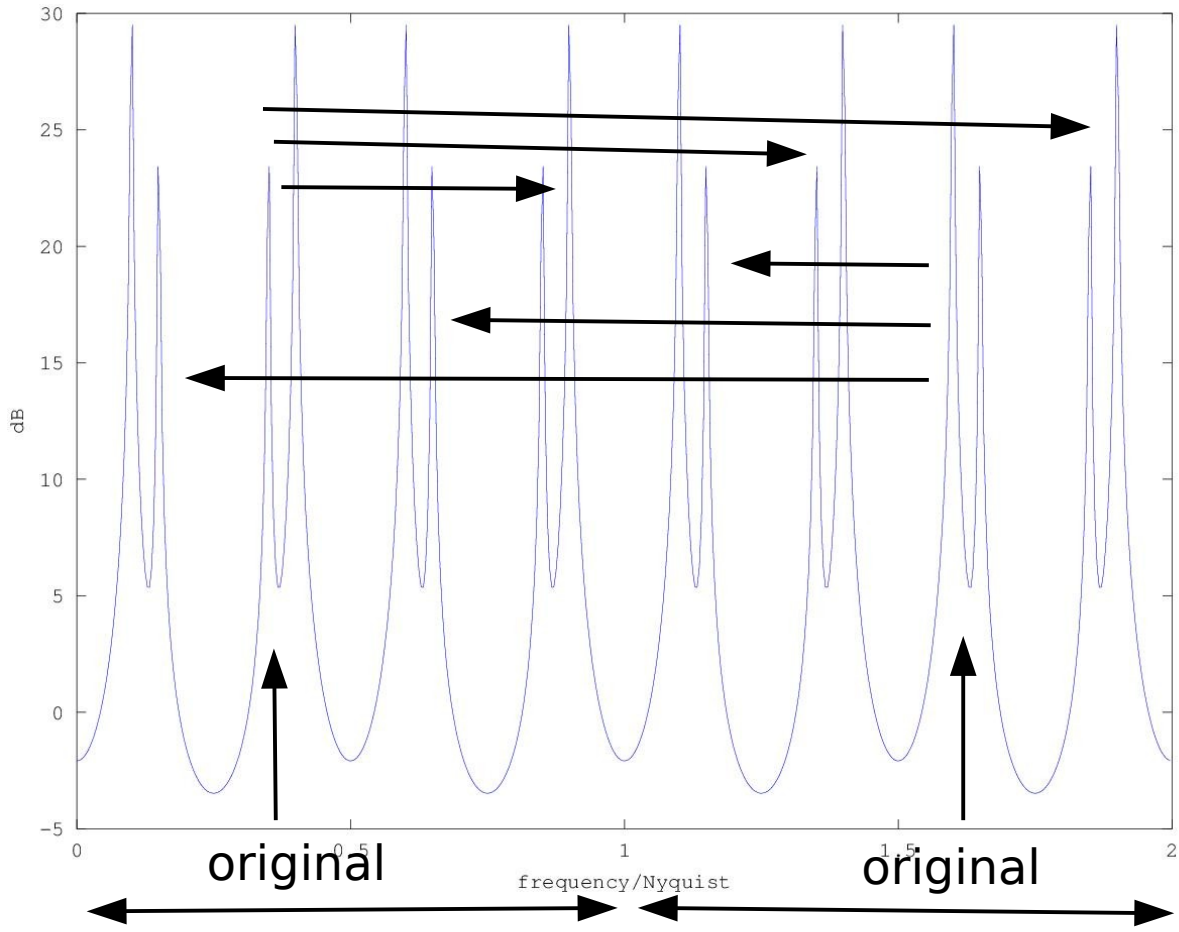
to reconstruct the original signal, hence we need to make sure the aliasing components don't overlap by suitable filtering at the high sampling rate, to prepare for the down-sampling. Observe that the term „Aliasing“ in the literature is sometime only used for overlapping alias components, and sometimes more broadly, like we do here, to mean any additional shifted frequency component.

Next is an **example**, also including the negative frequencies that now show up above normalized frequency 1 (**1** being the Nyquist frequency here), and showing 2 sine signals at different strength at normalized frequencies 0.4 and 0.35. This can also be seen as a narrow band signal, resulting e.g. from a passband filter.




After sampling by a factor of N=4, still including the zeros, we get the following spectrum:

# Spectral Copies




The picture shows that the spectrum still contains the **original spectrum**, **plus** the spectral **copies** at frequency shifts of $k \cdot 2 \cdot \pi / N$ from the originals.

**Observe**: Since we have a real valued signal (the sinusoids), the spectrum of negative and positive frequencies are **symmetric** around frequency zero. This then leads to the **mirrored appearance** between the neighbouring spectral images or aliasing components.

BTW, Nyquist tells us to sample in such a way, that the shifted spectra of our signal do not overlap. Otherwise, if they overlap, we cannot separate those parts of the spectrum anymore, and we loose information, which we cannot reconstruct.

**In conclusion**: Sampling a signal by a factor of N, with keeping the zeros between the sample points, leads to N-1 aliasing components or spectral copies.

**Example:**
Make a sine wave which at 44100 Hz sampling rate has a frequency of 400 Hz at 1 second duration. Hence we need 44100 samples, and 400 periods of our sinusoid in this second. Hence we can write our signal in Python as:

```
ipython --pylab
fs = 44100
f = 400.0
s=sin(2*pi*f*arange(0,1,1.0/fs))
```

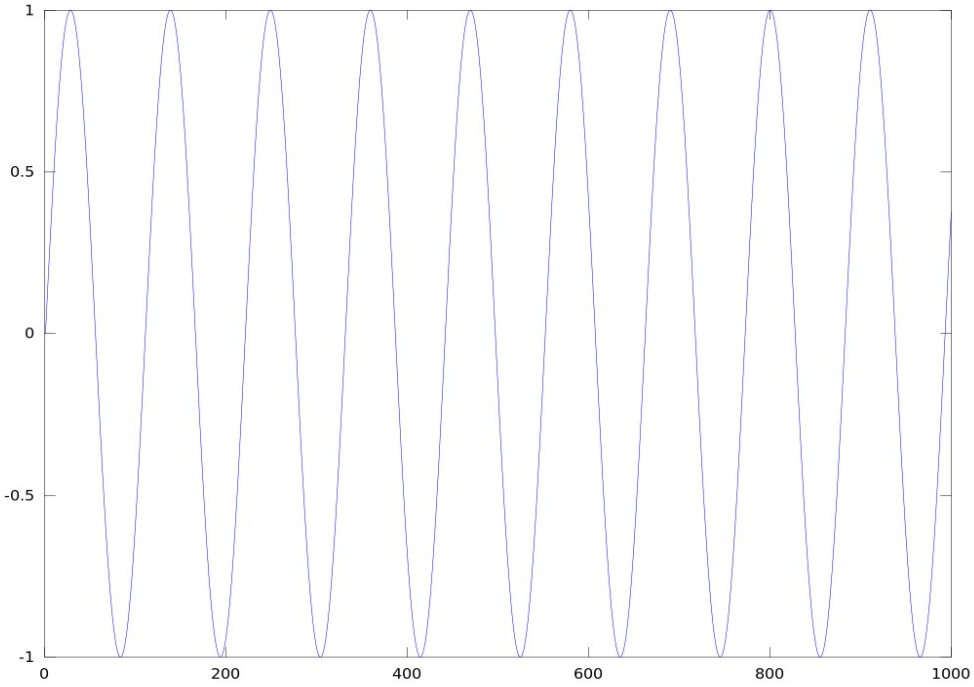
To listen to it, we use our sound library „sound.py“, which you can find on our Moodle Webpage:

```
from sound import sound
```

```
sound((2**15)*s,fs)
```

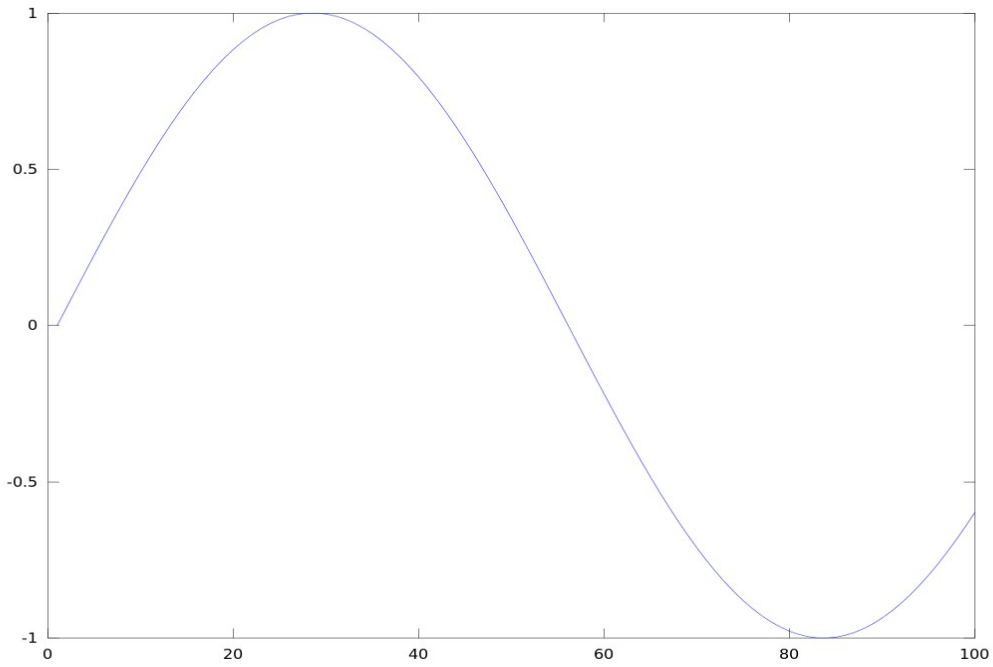
Now plot the first 1000 samples:

```
plot(s[0:1000])
```



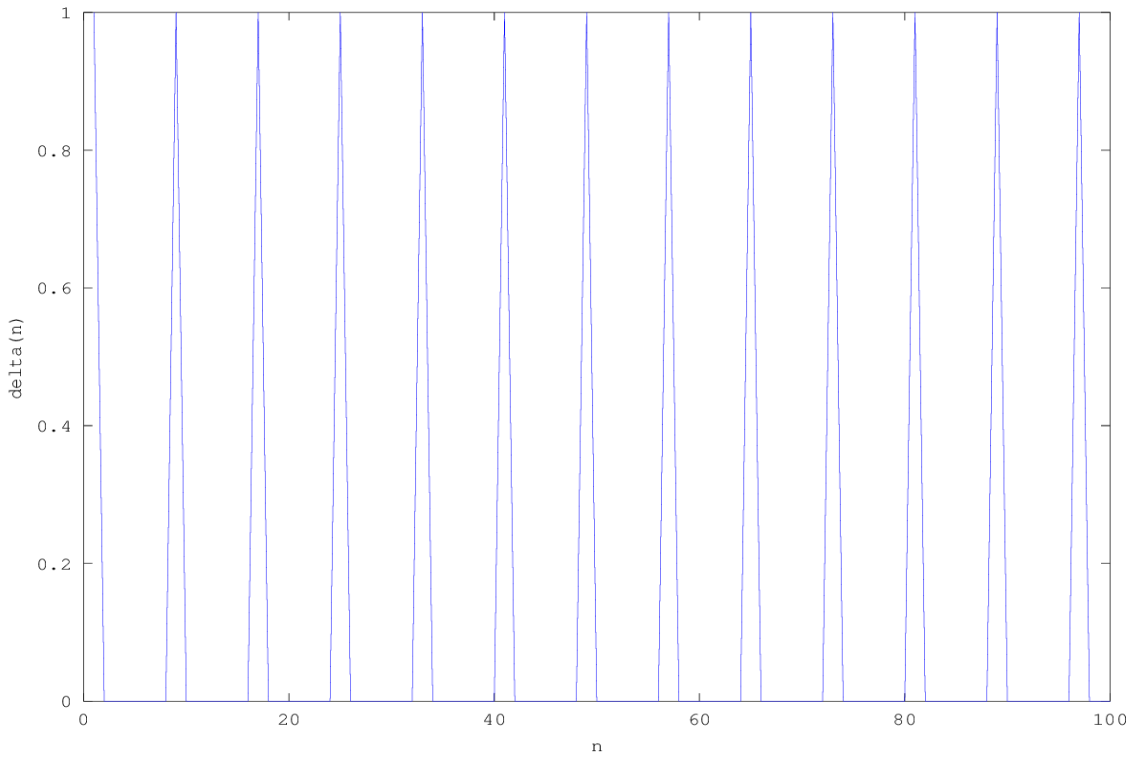

Next plot the first 100 samples:

```
plot(s[0:100])
```




Now we can multiply this sine tone signal with a unit pulse train, with N=8.
We generate the unit impulse train,

```
unit = zeros(44100)
unit[0::8] = 1
plt.plot(unit[0:100])
plt.xlabel('n')
plt.ylabel('unit(n)')
```




The triangle shape for the pulse comes from the plot function always drawing a line between neighbouring points.
Listen to it, with scaling to the value range for 16 bit/sample:

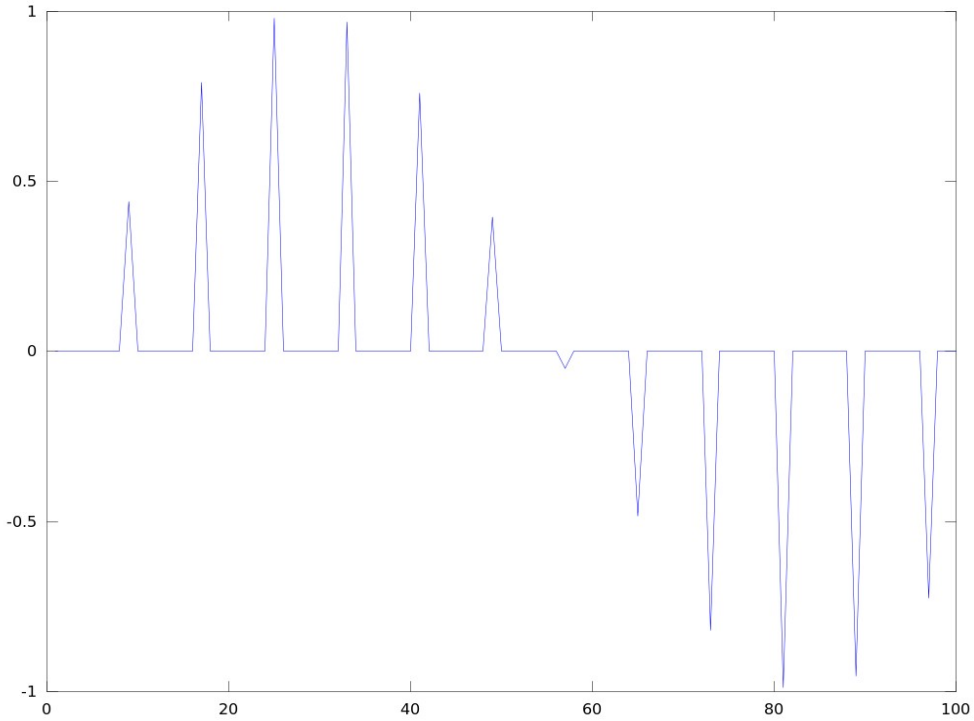
```
from sound import sound
sound(unit*2.0**15,44100)
```

The multiplication with the unit impulse train:

```
sdu=s*unit
```

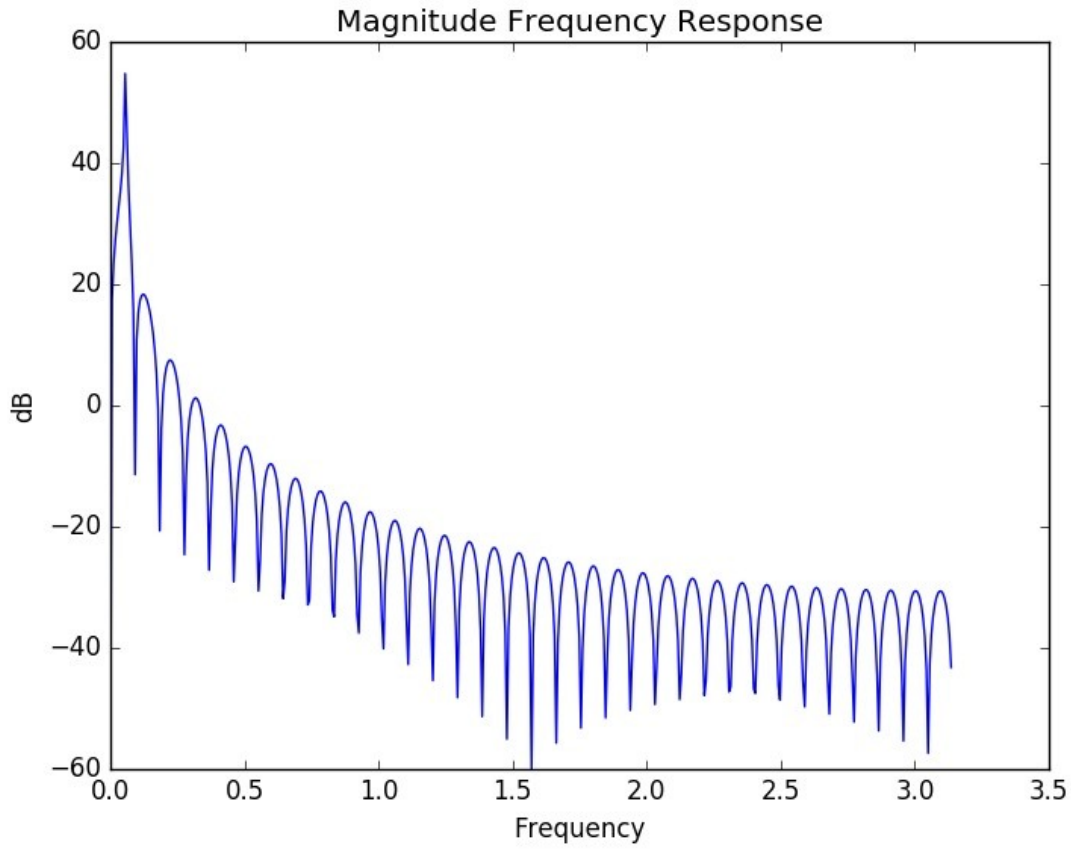
(This multiplication is also called „frequency mixing“).
Now plot the result, the first 100 samples:

```
plot(sdu[0:100])
```




This is our signal still with the zeros in it. Now take a look at the magnitude spectrum (in dB) of the original signal s:

```
from scipy.signal import freqz
w,H=freqz(s)
plot(w, 20*log10(abs(H)+1e-3))
xlabel('Normalized Frequency')
ylabel('dB')
title('Magnitude Frequency Response')
```

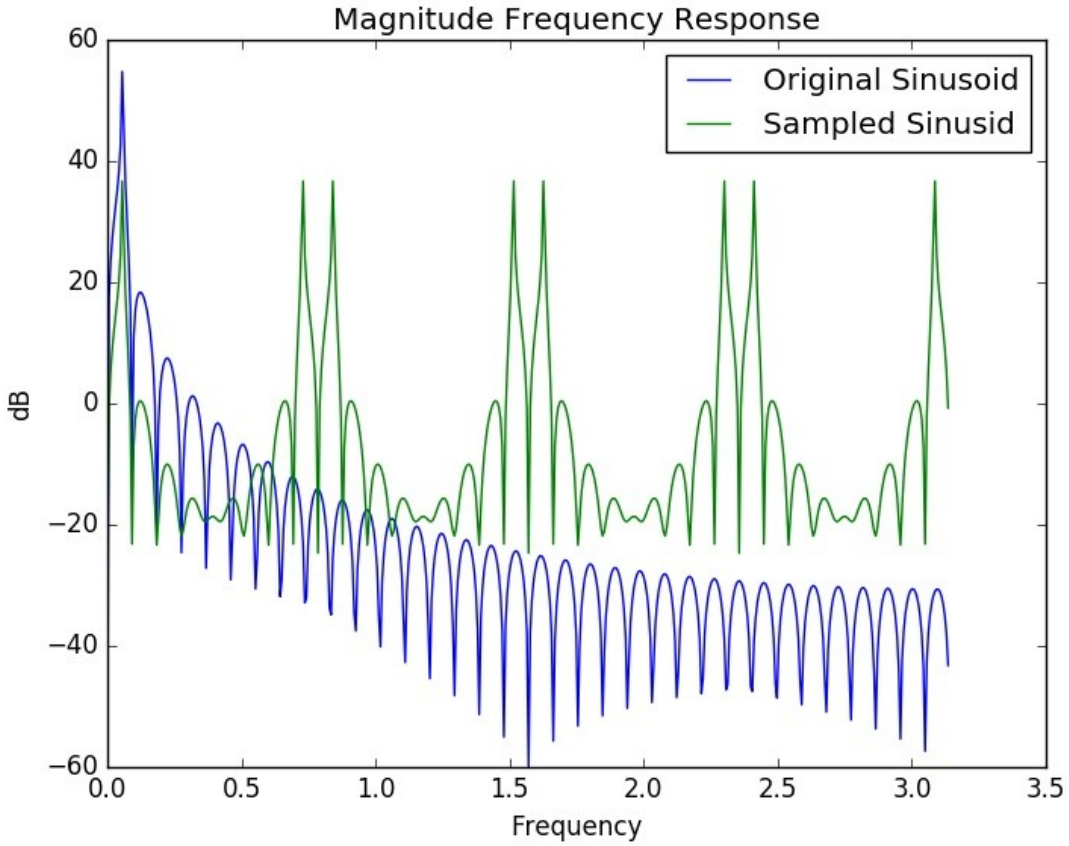



The plot shows the magnitude of the frequency spectrum of our signal. Observe that the frequency axis (horizontal) is a **normalized** frequency, normalized to the Nyquist frequency as $\pi$, in our case 22050 Hz. Hence our sinusoid should appear as a peak at normalized frequency 400.0/22050*pi=0.05699, which we indeed see.

Now we can compare this to our signal with the zeros, sdu:

```
w,H=freqz(sdu)
plot(w, 20*log10(abs(H)+1e-3))
```

```
legend(('Original Sinusoid','Sampled Sinusid'))
```




Here we can see the original line of our 400 Hz tone, and now also the 7 new aliasing components. Observe that always 2 aliasing components are close together. This is because the original 400 Hz tone also has a spectral peak at the negative frequencies, at -400 Hz, or at normalized frequency -0.05699.

Now also listen to the signal with the zeros:

```
sound(2**15*sdu,44100);
```

Here you can hear that it sounds quite different from the original, because of the strong aliasing components!
**Python** real time audio **example**: This example takes the microphone input and samples it, without removing the zeros, and plays it back the the speaker in real time. It constructs a unit pulse train, with a 1 at every N'th sample, using the modulus function „%“,

```
s=(np.arange(0,CHUNK)%N)==0
```

here we need that in Python, „True“ is a 1 and „False“ is a 0.
Start it with:

```
python pyrecplay_samplingblock.py
```

## Removing the zeros

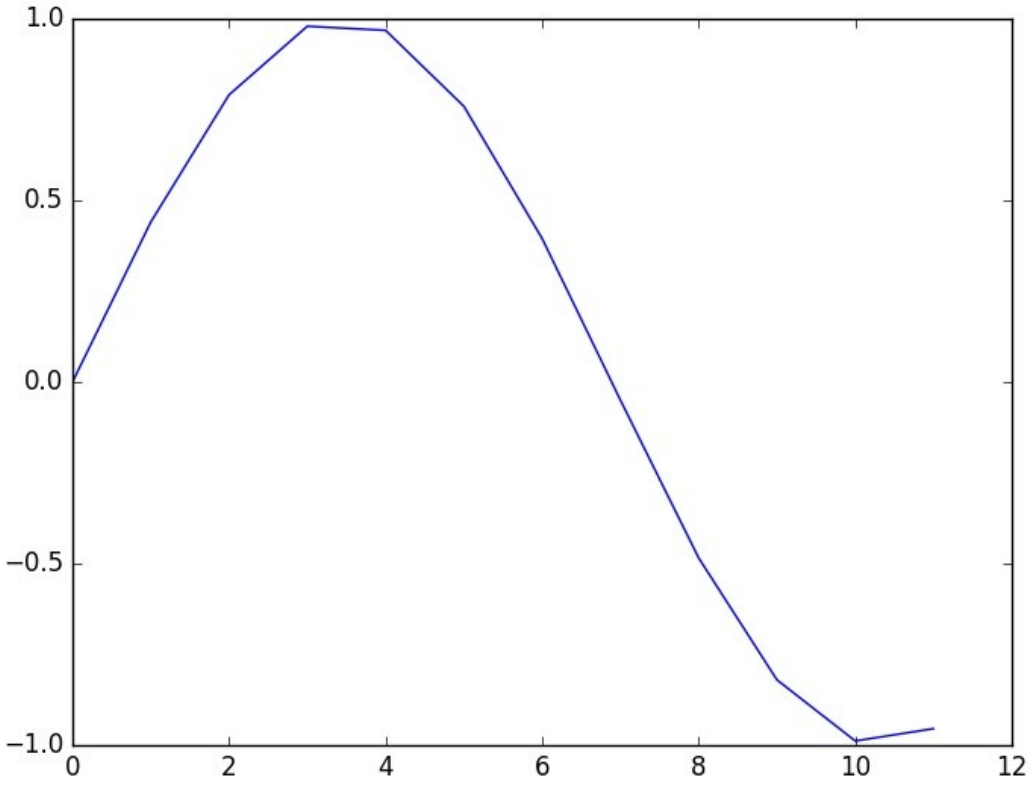
The final step of downsampling is now to omit the zeros between the samples, to obtain the lower sampling rate. Let's call the signal without the zeros

$$y(m),$$

where the time index m denotes the **lower sampling rate** (as opposed to n, which denotes the higher sampling rate).

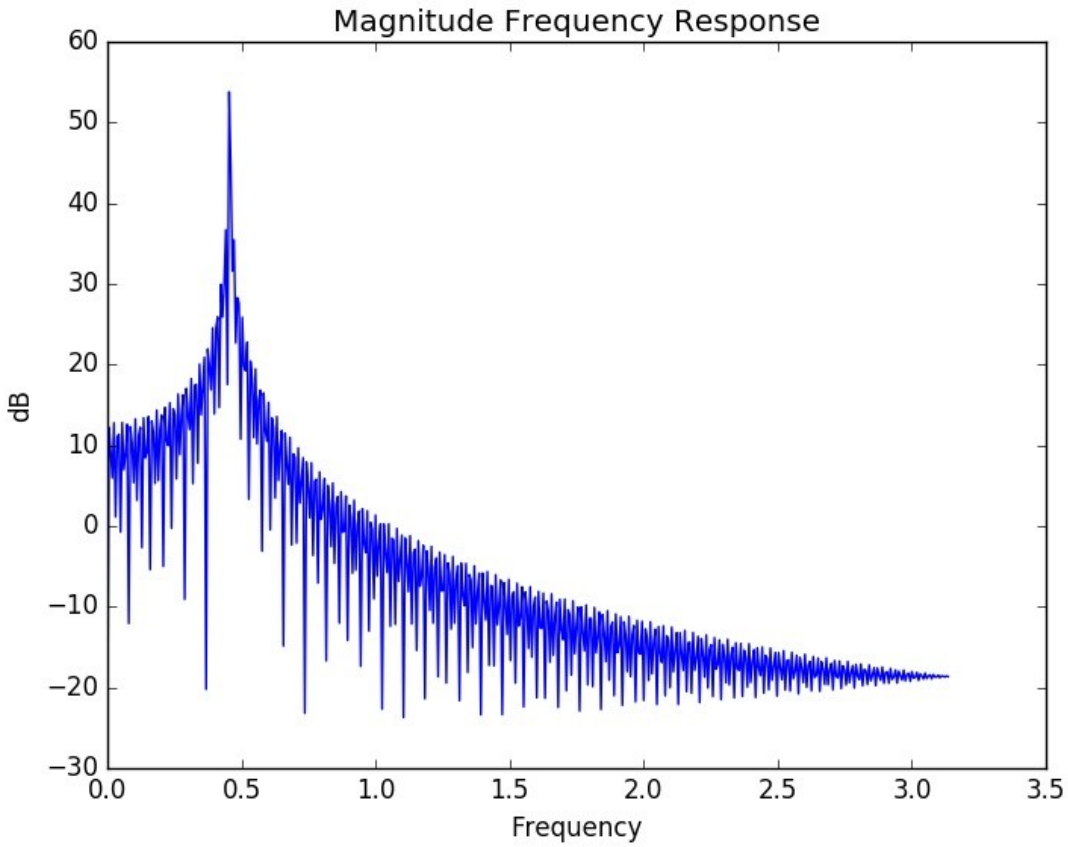
In our Python example this is:

```
sd = sdu[0:44100:8]
plot(sd[0:(100/8)])
```




We can now take a look at the spectrum with

```
w,H=freqz(sd)
plot(w, 20*log10(abs(H)+1e-3))
xlabel('Frequency')
ylabel('dB')
title('Magnitude Frequency Response')
```




Observe that the sine signal now appear at normalized frequency of 0.455, a **factor of 8 higher** than before, with the zeros in it, because we **reduced the sampling rate by 8**. This is because we now have a new Nyquist frequency of 22050/8 now, hence our normalized frequency becomes $400*3.14/22050\cdot 8\approx 0.455$ . This means removing the zeros scales or stretches our frequency axis.

Observe that here, for our plot, we only have $100/8 \approx 12$ samples left.
How are the frequency responses or spectra of $y(m)$ and $x^d(n)$ connected? We can simply take the Fourier transforms of them,

$$X^d(\Omega) = \sum_{n=-\infty}^{\infty} x^d(n) \cdot \mathrm{e}^{-j\Omega n}$$

still with the zeros in it. Hence most of the sum contains only zeros. Now we only need to let the sum run over the non-zeros entries (only every Nth entry), by replacing n by mN, and we get

$$X^d(\Omega) = \sum_{n=mN} x^d(n) \cdot \mathrm{e}^{-j\Omega n} \ ,$$

for all integer m, now without the zeros. Now we can make the connection to the Fourier transform of y(m), by making the index substitution m for n in the sum,

$$X^d(\Omega) = \sum_{m=-\infty}^{\infty} y(m) \cdot \mathrm{e}^{-j\Omega \cdot N m} = Y(\Omega \cdot N)$$

This is now our result. It shows that the downsampled version (with the removal of the zeros), has the same frequency response, but the normalized frequency variable $\Omega$ is scaled by the factor N. For instance, the normalized frequency $\pi/N$ before downsampling becomes $\pi$ after removing the zeros! It shows that a small part of the spectrum before downsampling

becomes the full usable spectrum after downsampling.
Observe that we don't loose any frequencies this way, because by looking at eq. (1) we see that we obtain multiple copies of the spectrum in increments of $2\pi/N$ , and hence the spectrum already has a periodicity of $2\pi/N$ . This means that the spectrum between $-\pi/N$ and $\pi/N$ for instance (we could take any period of length $2\pi/N$ ) contains a unique and full part of the spectrum, because the rest is just a periodic continuation.
This can be seen in following pictures,

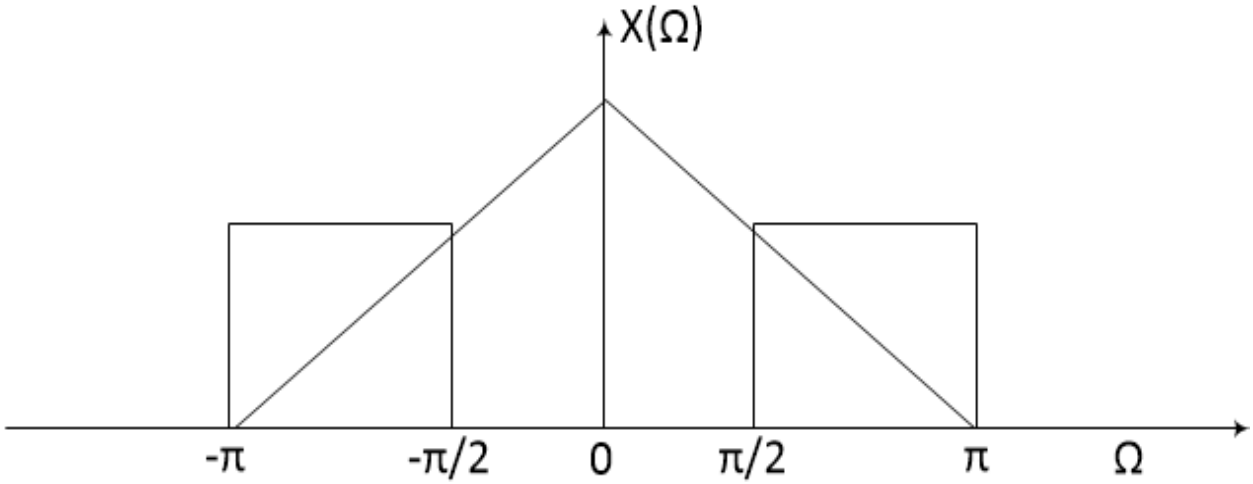



Figure 1: The magnitude spectrum of a signal. The 2 boxes symbolize the passband of an ideal bandpass, here a high pass.

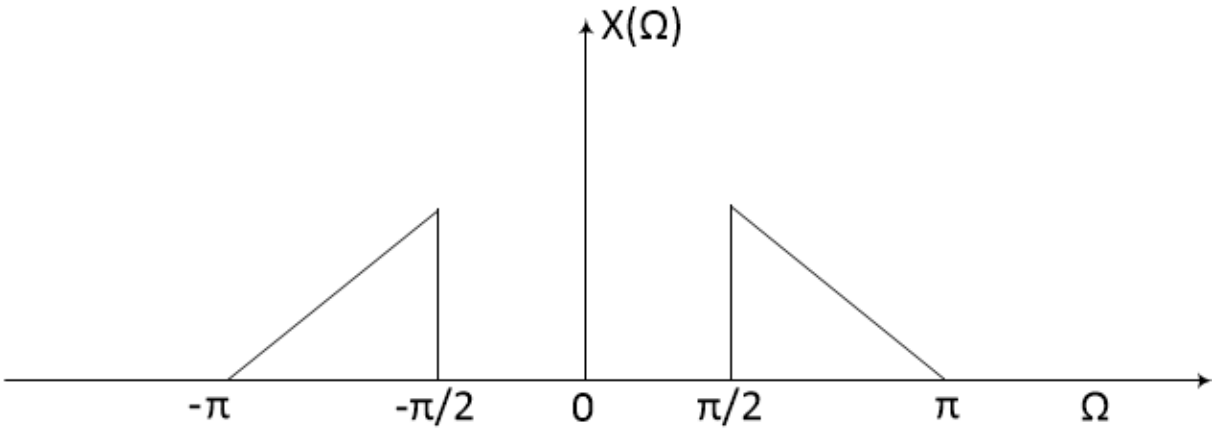



Figure: The signal spectrum after passing through the high pass.

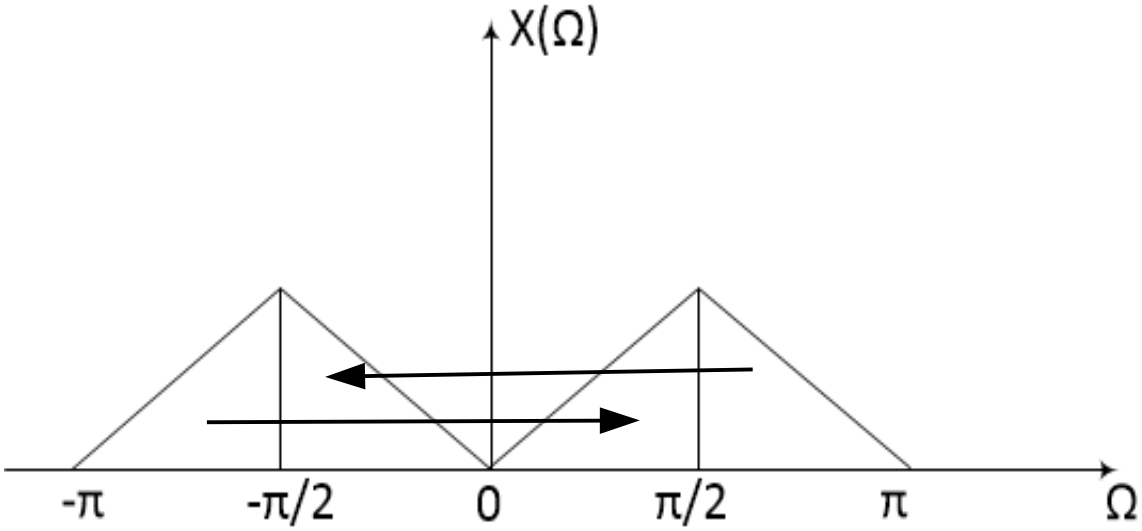



Figure: Signal spectrum after multiplication with the unit pulse train, for N=2, hence setting every second value to zero (the zeros still in the sequence). Observe the we shift and add the signal by multiples of $2\pi/2=\pi$ , and in effect we obtain „mirrored“ images of the high frequencies to the low frequencies (since we assume a real valued signal). Observe that the mirrored spectra

and the original spectrum don't overlap, which makes reconstruction easy.

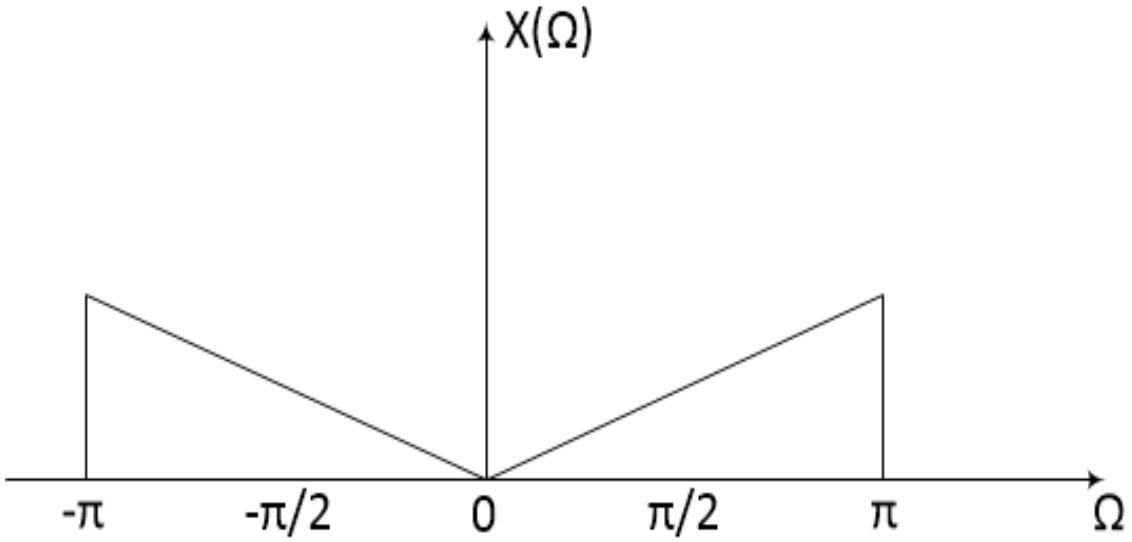



Figure: Signal spectrum after downsampling (removing the zeros) by N (2 in this example). Observe the stretching of the spectrum by a factor of 2.

## Upsampling

What is still missing in our system is the upsampling, as the opposite operation of downsampling, for the case where we would like to increase our sampling rate. One of the first (**wrong!**) approaches to upsampling that often comes to mind if we want to do upsampling by a factor of N, is to simply repeat every sample N-1 times. **But** this is equivalent to first inserting N-1 zeros after each sample, and then filter the resulting

sequence by a **low pass** filter with an impulse response of N ones. This is a very special case, and we would like to have a more general case. In our case, we actually a high pass filter, not a low pass. Hence we assume that we upsample by always first inserting N-1 **zeros** after each sample, and then have some interpolation filter for it (can also be a **high or band pas**s, as needed to reconstruct a signal). This is to „fish out“ the correct spectral copy of the original.
Again we take the signal at the lower sampling rate as

$$y(m)$$

with index m for the lower sampling rate, and the signal at the higher sampling rate, with the zeros in it, as

$$x^d(n)$$

with index n for the higher sampling rate. Here we can see that this is simply the reverse operation of the final step of removing the zeros for the downsampling. Hence we can take our result from downsampling and apply it here:

$$X^d(\Omega)=Y(\Omega\cdot N)$$

or

$$X^d(\Omega/N)=Y(\Omega)$$

We are now just coming from y(m), going to the now upsampled signal $x^d(n)$ .

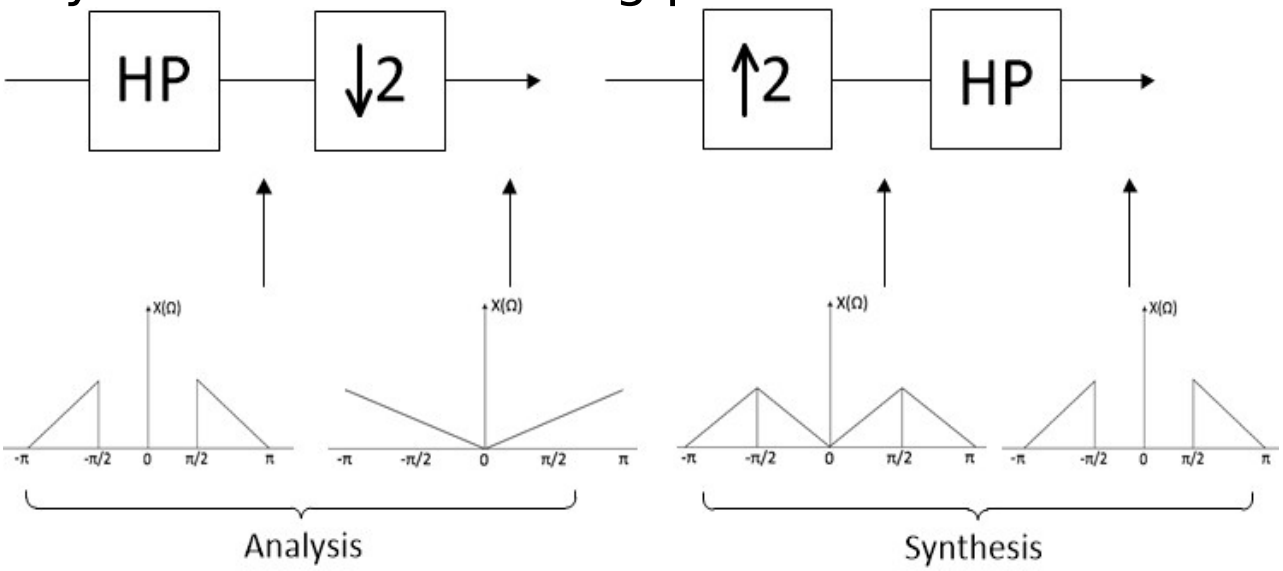
For instance if we had the frequency $\pi$ before upsampling, it becomes $\pi/2$ for the upsampled signal, if we have N=2. In this way we now get an „extended“ frequency range.

Since we now have again the signal including the zeros, $x^d(n)$ , we again have the periodic spectrum, as before, as we progress through the same steps backwards now. We can also see that the result of upsampling is periodic in frequency, because the signal was $2\pi$ periodic before upsampling anyway, and after upsampling the frequency scale replaces $2\pi \cdot N$ by $2\pi$ .

## Reconstruction

Observe that if the **aliasing components don't overlap**, we can **perfectly reconstruct** the signal by using a suitable filter. We can make sure that they don't overlap by **filtering** the signal at the **higher sampling rate**, before they can overlap (as we did in our high pass example). If they already overlap at the lower sampling rate, it would be too late to separate the different components, the signal would already be „destroyed“.

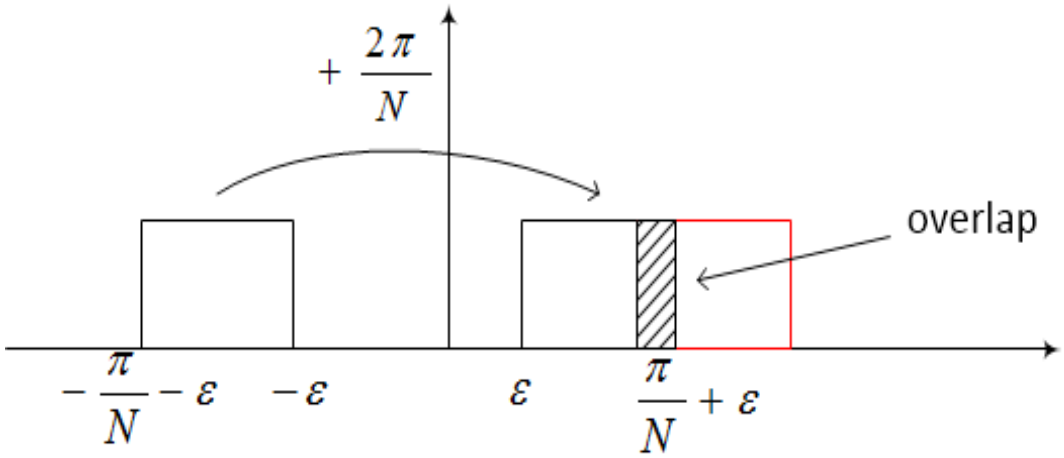
We can perfectly reconstruct the high pass signal in our example if we use ideal filters, using upsampling and ideal high pass filtering.
In this way we have for the analysis and synthesis the following picture




Observe that we violate the conventional Nyquist criterium, because our high pass passes the high frequencies. But then the sampling mirrors those frequencies to the lower range, such that we can apply the traditional Nyquist sampling theorem. This method is also known as bandpass Nyquist.

This is an important principle for filter banks and wavelets. It says that we can perfectly reconstruct a bandpass signal in a filter bank, if we sample with twice the rate as the **bandwidth** of our bandpass signal (assuming ideal filters, to avoid spectral overlap of aliasing components).
In genereal this is true for **complex** filters. For **rea**l valued filters observe that this simple assumption only works if we have bandpass filters which start at frequencies $\pi/N \cdot k$ (integer k) . Otherwise we could have overlap to the aliased components!
**Example:** We have a real valued bandpass filter which starts its passband at $\epsilon$ and ends at $\epsilon+\pi/N$ (since it is real values the the passband also appears at the same negative frequencies, $-\epsilon$ to $-\epsilon-\pi/N$ ). After multiplication with the unit pulse train, we get one (out of N) aliasing component by shifting the negative passband by $2\pi/N$ to the positive frequencies, which results in the range of $\pi/N-\epsilon$ to $2\pi/N-\epsilon$ . Hence we have an overlap from $\pi/N-\epsilon$ to $\pi/N+\epsilon$ and we could not perfectly reconstruct our signal!
Hence, sampling of twice the bandwidth for real valued signals and filters only works if the bandwidth are aligned with $\pi/N \cdot k$ .
What could we do otherwise to avoid

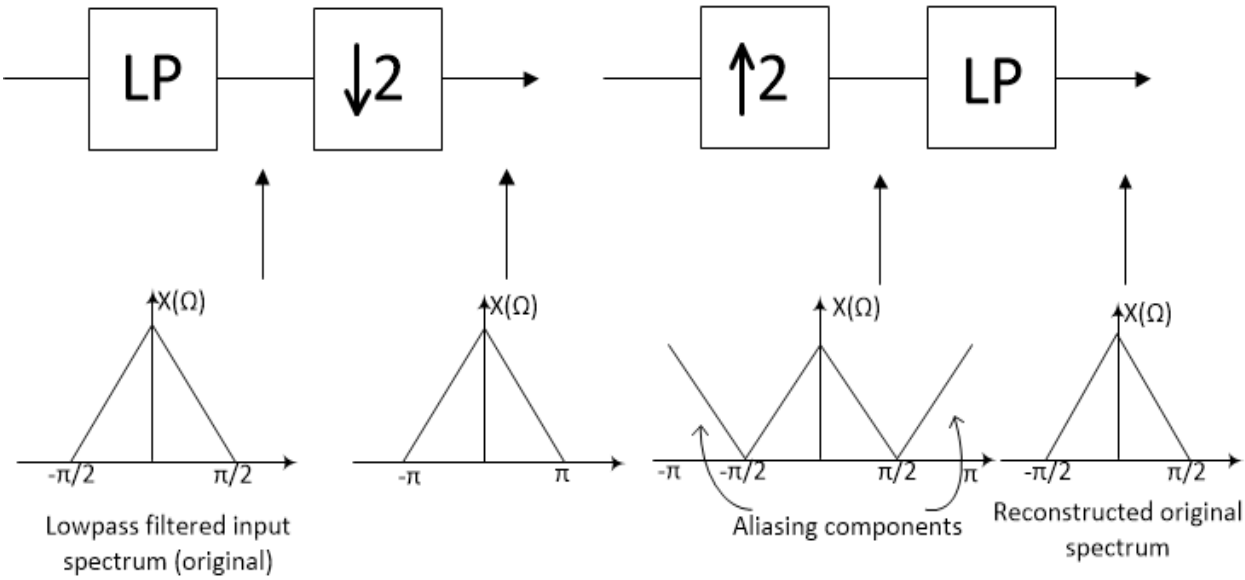
overlapping aliasing components? We could simply increase the sampling rate, for instance to twice the usual sampling rate (4 times the bandwidth), to have a "safety margin".
Another possibility would be to shift the bandpass signals in frequency, such that they are now aligned with the above mentioned grid.
Observe that this restriction is not needed for complex signals or filter banks. That is also why complex filter banks are used for instance in acoustic echo cancellation, because there the sampling rate can be increased by a certain fraction (less than a factor of 2) to reduce aliasing artifacts, and still have a lower complexity.




**Summary**: if our band boundaries are alligned with multiples of $\pi/N$ then we can donwsample by N, otherwise we are on the save side by using N/2 as downsampling

rate for real valued signals.  For complex, one-sided signals (for instance only positive frequencies) we can always downsample by N, regardles of the exact placement of the bandpass filter.

Compare with the standard Nyquist case: here we have a lowpass signal which we downsample and reconstruct:




## Python Example

This program can be **controlled by keyboard**, and features **sampling** by a factor of N=8 (with keeping the zeros), which can be turned on and off, and a **low pass filter** before and after sampling to

supress the alias components, which can be turned on and off.
It shows the magnitude of the resulting DFT spectrum after reconstruction (after the last low pass filter) in a so-called **waterfall spectogram:**
-The **magnitude** of the DFT is displayed as **color** (yellow is high, blue is low),
**-horizontal** axis is **frequency** (0 to Nyquist frequency)
**-vertical** axis is **time** (up is past).

Start it with

```
python pyrecspecwaterfallsampling.py
```

**Observe**: When we turn on **sampling**, we clearly **hear** the **aliasing artifacts**, and we also **see** them as spectral copies in the waterfall spectrogram.
When we turn on the **low pass filter**, most of the aliasing artifacts are **not audible** anymore, and we also **don't see** most of them anymore in the waterfall spectrogramm, but it also sounds **more muffled**.