Signal Processing for Communication

Ch. Bollinger

<2020-07-20 Mo>

Contents

Part I.

Week 1 Module 1:

1. Basics of Digital Signal Processing

1.1. Introduction to digital signal processing

1.1.1. Signal

- Description of the evoultion of a physical phenomenon

phenomenon	signal
weather	temperature
sound	pressure
sound	magnetic deviation
light intensity	gray level on paper

1.1.2. Processing

- **Analysis:** Understanding the information carried by the signal
- **Synthesis:** Creating a signal to contain the given information

1.1.3. Digital

- Discrete Time
  - Splice up time into a series of descrete instance without loosing information
  - Harry Nyquist and Claude Shannon state with the Sampling Theorem that continous time representation and discrete time representation are equivalent.
  - The Sampling Theorem: Under appropriate "slowness" conditions for x(t) we have

$$x(t) = \sum_{n=-\infty}^{\infty} x[n]\ sinc(\frac{t - nT_s}{T_s}) \tag{1.1}$$

  - The conditiion under which the Sampling Theorem holds was given by Fourier and it's Fourier Analysis.
  - The fouriere transform will give us a quantitive measure how fast a signal moves
- Discrete Amplitude
  - Through discretisation of ampltitudes only a set of predefined values are possible.
  - The set of levels is countable i.e. we can always map the level of a sample to an integer. If our data is represented by integer it becomes complete abstract and general which has very importand consequences in the following three domains:
    - **Storage** special devices for recoding needed
    - **Processing** General purpose microprocessor is sufficient
    - **Transmission** Reproduction of the original signal and therefore eliminating nois is easy

1.1.4. From Analog to Digital Signal Processing

- Analog asks for $f_{(t)} = ?$
- Digital represents data as a sequence of numbers (scaled with a factor of 1000)

-12 -12 -12 -11 -11 -12 -12 -11 -11 -10

-10 -10 -9 -10 -10 -9 -9 -9 -9 -9

-8 -8 -7 -7 -8 -8 -8 -7 -7 -7

1.2. Discrete-Time Signals

1.2.1. Basic Definitions

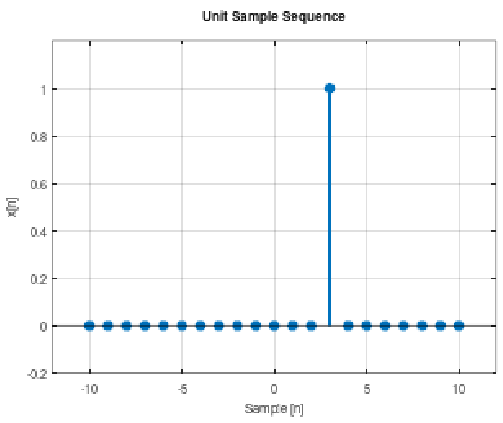
- Sequence: defined as complex-valued function
- Discrete-Time Signal: a sequnece of complex numbers
  - one dimension (for now)
  - notation: x[n]
  - two-sides sequencies: x: $\mathbb{Z} \rightarrow \mathbb{C}$
  - n is *a-dimensional* "time", sets an order on the sequence of samples
  - analysis: periodic measurement
  - synthesis: stream of generated samples, reproduce a physical phenomenon

1.2.2. Octave Algorithm for some basic Signals

Unit Impulse

```
function [x,n] = impseq(n0,n1,n2)
% Generates x(n) = delta(n-n0); n1 <= n0 <= n2
% ---------------------------------------------
% [x,n] = impseq(n0,n1,n2)
%
  n = [n1:n2]; x = [(n-n0) == 0];
end
```

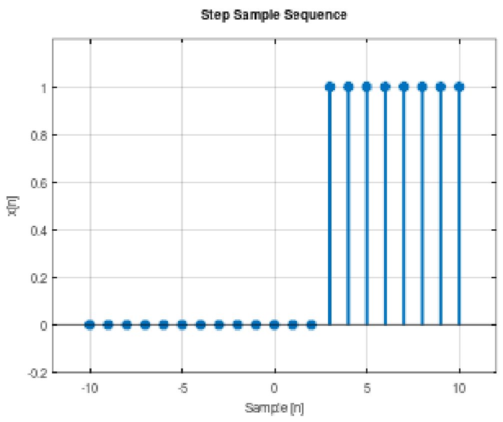
$$x[n] = \delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$



Unit Step

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = delta(n-n0); n1 <= n0 <= n2
% ----------------------------------------------
% [x,n] = stepseq(n0,n1,n2)
%
  n = [n1:n2]; x = [(n-n0) >= 0];
end
```

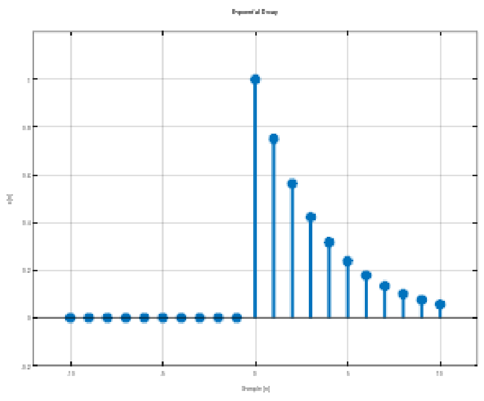
$$x[n] = u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$



Real-valued exponential Sequence

```
function [x,n] = expseq(n1,n2,a)
% Generates x(n) = a^n
% ----------------------------------------------
% [x,n] = expseq(n1,n2,A,omega,phi)
%
  n = [n1:n2];
  for (i = 1 : length(n))
    if (n(i) >= 0)
      x(i) = (a).^n(i);
    else
      x(i) = 0;
    end
  end
end
```

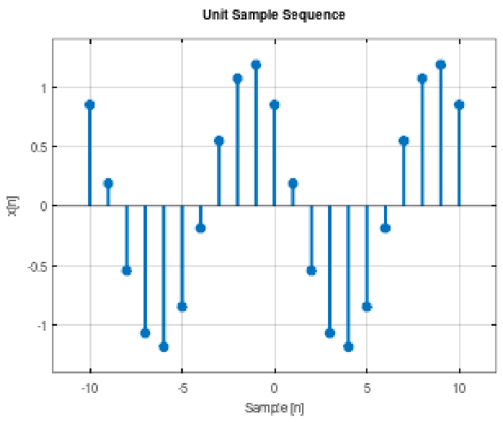
$$x[n] = a^n, \forall n \; a \in \mathbb{R}$$



Sinusoidal Sequence

```
function [x,n] = cosseq(n1,n2,A, omega, phi)
% Generates x(n) = A*cos(2*pi*omega*n + phi); n1 <= n2
% -----------------------------------------------
% [x,n] = cosseq(n1,n2,A,omega,phi)
%
  n = [n1:n2]; x = A*cos(2*pi*omega*n + phi);
end
```

$$x[n] = A\,cos(\omega_0 n + \Phi)$$



1.2.3. Classes of Discrete-Time signals

1.2.3.1. Finite-Length

- indicate notation: $x[n]$, $n = 0.1.2.....N-1$
- vector notation: $x = [x_0, x_1, ...x_{N-1}]^T$
- practical entities, good for numerical packages (e.g. numpy)

1.2.3.2. Infinte-Length

- sequence notation: $x[n]$, n $\in \mathbb{Z}$
- abstraction, good for theorems

1.2.3.3. Periodic

- N-periodic sequence: $\tilde{x}[n] = \tilde{x}[n + kN]$, n,k,N $\in \mathbb{Z}$
- same information as in finite-length of length N
- natural bridge between finite and infinite length

1.2.3.4. Finite-Support

Finite-support sequence

$$\overline{x}[n] = \begin{cases} x[n] & if 0 \leq n < N, n \in \mathbb{Z} \\ 0 & otherwise \end{cases} \tag{1.2}$$

- same information as in finite-length of length N
- another bridge between finite and infinite lengths

1.2.3.5. Elementary Operations

Scaling

$$y[n] = ax[n] \rightarrow \begin{cases} a > 0 & amplification \\ a < 0 & attenuation \end{cases} \tag{1.3}$$

Sum

$$y[n] = x[n] + z[n] \tag{1.4}$$

Product

$$y[n] = x[n] * z[n] \tag{1.5}$$

Shift

$$y[n] = x[n-k] \rightarrow \begin{cases} k > 0 & deleay \\ k < 0 & anticipate \end{cases} \tag{1.6}$$

Integration

$$y[n] = \sum_{k=-\infty}^{n} x[k] \tag{1.7}$$

Differentation

$$y[n] = x[n] - x[n-1] \tag{1.8}$$

Relation Operator and Signals

- The unit step can be optained by applying the integration operator to the discrete time pulse.
- The unit impulse can be optained by applying the differentation operator to the unit step.

1.2.4. Energy and Power

Energy Many sequencies have an infinity amount of energy e.g. the unit step u[n],

$$E_x = ||x||_2^2 = \sum_{k=-\infty}^{\infty} |x[n]|^2 \tag{1.9}$$

Power To describe the energetic properties of the sequencies we use the concept of power

$$P_x = ||x||_2^2 = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2 \tag{1.10}$$

Many signals have infi

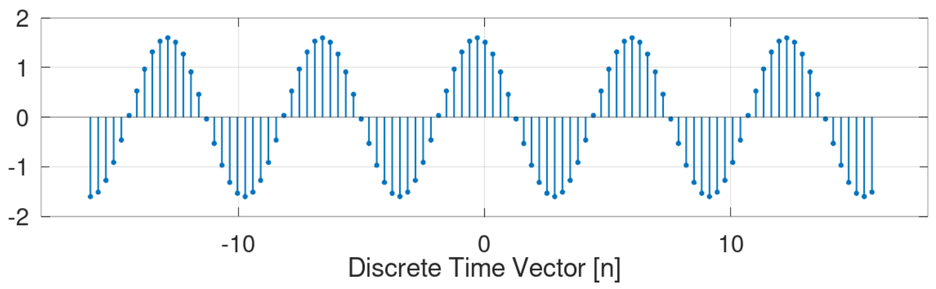
1.3. Basic signal processing

1.3.1. How a PC plays discrete-time sounds

1.3.1.1. The discrete-time sinusoid

$$x[n] = sin(\omega_0\, t + \Theta)$$

```
 N=33                          # Vector lenght
 n=-(N-1)/2:pi/10:(N-1)/2;  # Discrete Time Vector
 omega0 = pi/10;
 theta = pi/2

 f = 1.6*sin(omega0+n + theta); # The sinusoid

# Do not open the graphic window in org
 figure( 1, "visible", "off");

 stem(n,f, "filled", "linewidth", 2, "markersize", 6);
 axis([-(N-1+4)/2 (N-1+4)/2 -2 2])
 set(gca, "fontsize", 24);
 grid on ;
 xlabel("Discrete Time Vector [n]");
 print -dpng "-S1400,350" ./image/sin.png;
 # Org-Mode specific output
 ans = "./image/sin.png";
```



1.3.1.2. Digital vs physical frequency

- Discrete Time:
  - Periodicity: how many samples before the pattern repeats (M)
  - n: no physical dimension
- Physical World:
  - Periodicity: hoq many seconds before the pattern repeats
  - frequency measured in Hz
- Soundcard $\mathrm{T_s}$ System Clock
  - A sound card takes ever $\mathrm{T_s}$ an new sample from the discrete-time sequence.
  - periodicity of M samples → periodicity of $M\ T_s$ seconds
  - real world frequency

$$f = \frac{1}{M\ T_s} Hz \tag{1.11}$$

- Example
  - usually we choose $\mathrm{F_s}$ the number of samples per seconds
  - $\mathrm{T_s} = 1/\mathrm{F_s}$

$$\begin{aligned} F_s &= 48000 \text{e.g. a typical value} \\ T_s &= 20.8\mu\ s \\ f &= 440 Hz \text{ , with M} = 110 \end{aligned}$$

1.3.2. The Karplus Strong Algorithm

1.3.2.1. The Moving Average

- simple average (2 point average)

$$m = \frac{a+b}{2} \tag{1.12}$$

- moving average: take a ”local” average

$$y[n] = \frac{x[n] + x[n-1]}{2} \tag{1.13}$$

- Average a sinusoid

$$\begin{aligned} x[n] &= cos(\omega\ n) \\ y[n] &= \frac{cos(\omega\ n) - cos(\omega\ (n-1))}{2} \\ y[n] &= cos(\omega\ n + \theta) \end{aligned}$$

Linear Transformation

Applying a linear transformation to a sinusoidal input results in a sinusoidal output of the same frequency with a phase shift.

1.3.2.2. Reversing the loop

$$y[n] = x[n] + \alpha\ y[n-1] \rightarrow \text{ The Karplus Strong Algorithm} \tag{1.14}$$

- **Zero Initial Conditions:**
  - set a start time (usually $n_0 = 0$)
  - assume input and output are zero for all time before N_0

1.4. Digital Frequency

Digital Frequency

$$\begin{aligned} \sin\Big(n\big(\omega + 2k\pi\big)\Big) &= \sin\big(n\omega + \phi\big), \text{ k in } \mathbb{Z} \\ &= e^{i(\phi + n*2\pi\omega)} \end{aligned} \tag{1.15}$$

Complex Exponential

$$\omega = \frac{M}{N} \times 2 \times \pi \tag{1.16}$$

1.5. The Reproduction Formula

Reproduction Formula

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \tag{1.17}$$

Any signal can be expressed as a linear combination of wighted and shifted pulses.

Part II.

Week 2 Module 2:

2. Vector Spaces

Vector Space

Vector spaces build among others a common framework to work with the four classes of signals:

- Finite Length Signal
- Infinte Length Signal
- Periodic Signal
- Finite Support Signal

Finite length and periodic signal, i.e. the "practical signal processing" live in the $\mathbb{C}^N$ Space. To represent infinite length signals we need something more. We require sequneces to be square-summabe $\sum_{n=-\infty}^{\infty} |x[n]|^2$

$\mathbb{R}^2$, $\mathbb{R}^3$	Euclidean space, geomtery
$\mathbb{R}^n$, $\mathbb{C}^n$	Linear algebra
$\ell_2(\mathbb{Z})$	Square-Summable infinite sequences
$L_2([a,b])$	Square-integrable functions over an interval

2.1. Operationl Definitions

Inner Product

Measure of similarity between vectors

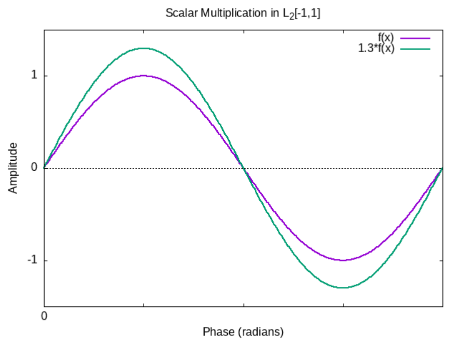
Inner Product	$\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{n=0}^{N-1} x_n \, y_n$ A vector space with an inner product is called an **inner product space**
Inner Product in $\mathbb{R}^2$	$\langle \mathbf{x}, \mathbf{y} \rangle = x_0 y_0 + x_1 y_1 = \mathbf{x} + \mathbf{y} cos(\alpha)$
Inner Product in $\mathbb{L}_{[-1,1]}$	$\langle \mathbf{x}, \mathbf{y} \rangle = \int_{-1}^{1} x(t) y(t) dt$
Norm of a Vector	$\mathbf{v} := \langle \mathbf{v}, \mathbf{v} \rangle = \vert\vert\mathbf{v}\vert\vert^2$ self inner product
Orthogonal	$\langle \mathbf{p}, \mathbf{q} \rangle = 0$ maximal different vectors inner product $= 0$
Distance	$d(x,y) = \mathbf{x} - \mathbf{y}_2$

2.2. Some Examples

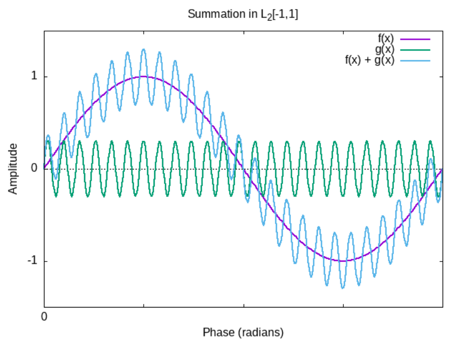
Not all vector spaces have got a graphical representation. The following table shows the graphical representation of vector spaces

graphical representation	no graphical respresentation
$\mathbb{R}^2$	$\mathbb{C}^N$ for N>1
$\mathbb{R}^3$	$\mathbb{R}^N$ for N>3
$\mathbb{L}_{[-1,1]}$	

Scalar Multiplication in $\mathbb{L}_2[-1,1]$

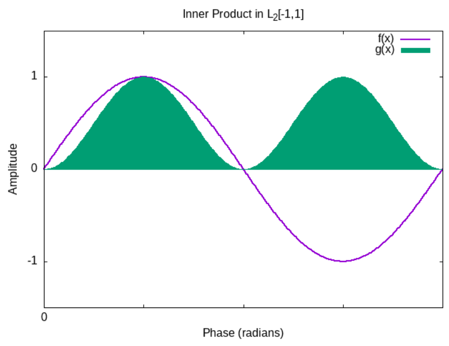


Summation of two Vectors in $\mathbb{L}_2[-1,1]$



Inner Product in $\mathbb{L}_2[-1,1]$ - The Norm:
with x = sin(π\,t)

$$\langle \mathbf{x}, \mathbf{x} \rangle = ||\mathbf{x}||^2$$
$$= \int_{-1}^{1} sin^2(\pi)dt = 1$$



2.3. Hilbert Space

A hilbert space is an **inner product space** which fulfills completeness.

2.4. Signal Spaces

Finite length signal live in $\mathbb{C}^N$

- all operations well defined and intuitive

- space of N-periodic signals sometimes indicated by $\tilde{\mathbb{C}}^N$

2.5. TODO Vecotor Bases

2.6. TODO Subspace Approximations

2.6.1. Least-Square Approximation

Consider a orthonormal basis for subspace S, called S of K

$$s-(k)_{k=0,1..,k-1} \text{ orthonormal basis for S}$$

orthonormal projection is defined as follows:

$$\hat{x}=\sum_{k=0}^{k-1}\langle s^{(k)},x\rangle s^{(k)}$$

- orthogonal projection has minimum-norm error:

$$arg\ min||x-y||=\hat{x}$$

- :

$$\langle x-\hat{x},\hat{x}\rangle=0$$

Part III.

Week 3 Module 3:

3. Part 1 - Basics of Fourier Analysis

3.1. Introduction to Fourier Analysis

3.1.1. The Frequency Domain

3.1.1.1. Sustainable dynamic systems exhibit oscillatory behavior

- A train has got an engine which makes the wheels turn in circular motion
- Waves, ebb and flow can be modeled as sinusoidal fashion
- Musical instruments generates sound by vibrating at a certain fundamental frequency
- Intuitivly: things that don't move in circles can't last
  - bombs
  - rockets
  - human beeings

3.1.1.2. Descriptin of the oscillations in the plane

Period P

Frequency $f = \frac{1}{P}$

Ordinate $\sin(ft)$

Abscissa $\cos(ft)$

3.1.1.3. Example Sinusoidal Detectors in our Body:

cochlea In the inner ear that detects air pressure sinusoids at frequenies from 20 to 20kHz

retina In the eye to detect electromagnetic sinusoids with frequency 430THz to 790THz. This is the frequency of lights in the visible spectrum

Humans anlayze complex signals (audio, images) in terms of their sinusoidal components

Frequency Domain semms to be as good a the time domain

3.1.1.4. Fundamental Questions: Can we decompse any signal into sinusoidal elements?

- Yes, Fourier showed us how to do it exactely
- Analysis
  - From time domain to frequency domain
  - Find the contribution of different frequencies
  - Discover "hidden" signal properties
- Synthesis
  - From frequency domain to time domain
  - Create signals with known frequency content
  - Fit signals to specific frequency regions

3.1.2. TODO The DFT as a change of basis

3.2. The Discrete Fourier Transform (DFT)

3.2.1. DFT definition

3.2.1.1. The Fourier Basis for $\mathbb{C}^N$ in "Signal" Notation

$$w_k[n] = e^{j\frac{2\pi}{N}nk} \text{with } n, k = 0, 1, ..., N-1 \tag{3.1}$$

3.2.1.2. The Fourier Basis in Vector Notation

$$\{\mathbf{w}^{(k)}\}_{k=0,1...N-1} \text{with } w_n^{(k)} = e^{j\frac{2\pi}{N}nk},\ n = 0, 1, ...N-1 \tag{3.2}$$

N N Dimension of vector space

k Index for different vectors and goes from 0..N-1

n Index of element in each vector goes from 0...N-1

3.2.1.3. Basis Expansion Vector Notation

- Analysis Formula

$$X_k = \langle \mathbf{w}^{(k)}, \mathbf{x} \rangle \text{ k = 0,...N-1} \tag{3.3}$$

X_k Coefficent for the new basis. Inner Product of $\mathbf{x}$ with each vector $\mathbf{w}^{(k)}$
$\mathbf{x}$ An arbitrary vector of $\mathbb{C}^N$
$\mathbf{w}^{(k)}$ New basis

- Synthesis Formula

$$\mathbf{x} = \frac{1}{N}\sum_{k=0}^{N-1} X_k \mathbf{w}^{(k)} \text{ k = 0,...N-1} \tag{3.4}$$

3.2.1.4. TODO Change of basis in matrix form

3.2.1.5. Basis Expansion Signal Notation

- Consider explicitely the operations involved in the transformation
- This notion is particulary useful if you want to consider the algorithmic nature of the transform
- Analysis Formula N-point signal in the frequency domain

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk},\ k = 0, 1, ..N-1$$

$X[k]$ Signal vector in the frequency domain

$x[n]$ Signal vector in the (discrete) time domain

Reminder This is the inner Product in explicite form

- Synthesis Formula N-point signal in the time domain

$$\boxed{x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]e^{j\frac{2\pi}{N}nk},\ k = 0, 1, ..N-1}$$

$X[k]$ Signal vector in the frequency domain

$\frac{1}{N}$ Normalisation coeficent

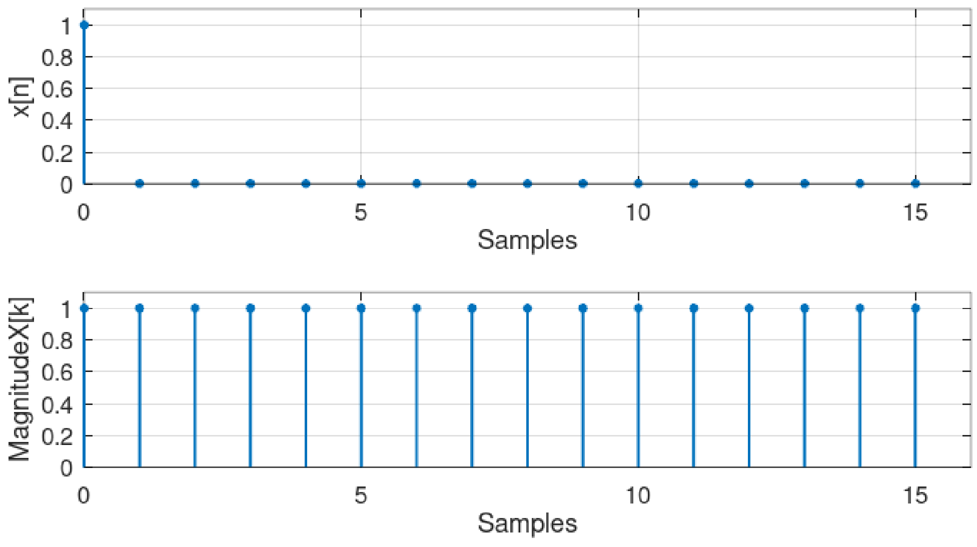
Reminder This is the inner Product in explicite fashion

3.2.2. Examples

3.2.2.1. DFT of the impulse function

$$x[n] = \delta[n]$$

$$X[k] = \sum_{n=0}^{N-1} \delta[n]e^{-j\frac{2\pi}{N}nk} = 1$$

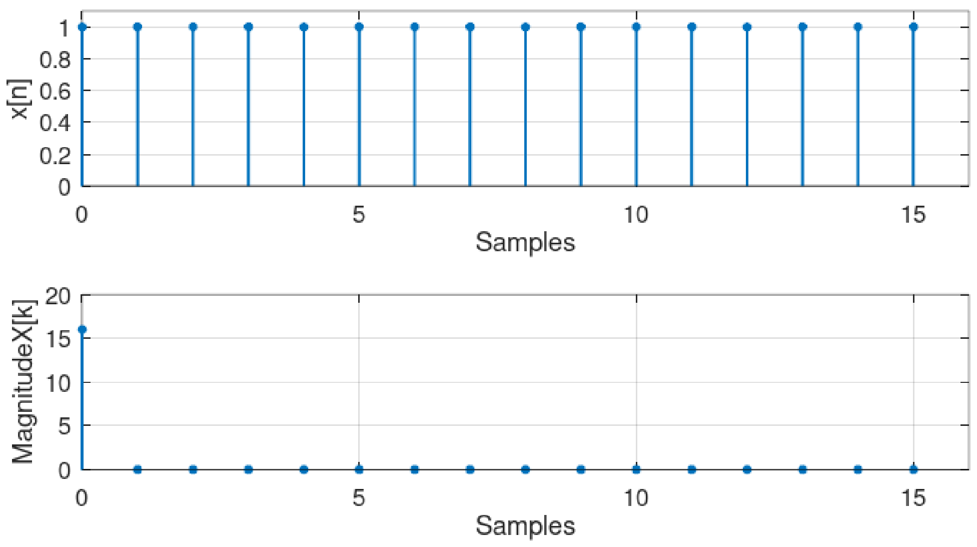


- The delata contains all frequencies over the range of all possible frequencies

3.2.2.2. DFT of the unit step

$$x[n] = 1$$

$$X[k] = \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}nk} = N\delta[k]$$



3.2.2.3. DFT Cosine Calculation Problem 1

$$x[n] = 3\cos(2\pi/16 \times n),\ x[n] = \mathbb{C}^{64}$$

1. Determine dimension and fundamental frequency of the signal
   - Dimension of space N = 64
   - Fundamental frequency $\omega = \dfrac{2\pi}{N} = \dfrac{2\pi}{64}$

     All frequencies in the fourier basis will be a multiple of the fundamental frequency ω. With this in mind we can start by expressing our sinuoid as a muiltiple of the fundamental frequenncy in space $\mathbb{C}^{64}$.

2. Express the signal as a multiple of the fundamental frequency in space.

$$\begin{aligned}
X[n] &= 3\cos(\frac{2\pi}{16}n) \\
&= 3\cos(\frac{2\pi}{64}4n) \\
&= \frac{3}{2}\left[e^{j\frac{2\pi}{64}4n} + e^{-j\frac{2\pi}{64}4n}\right], \text{ with Euler: } cos(\omega) = \frac{e^{j\omega}+e^{-j\omega}}{2} \\
&= \frac{3}{2}\left[e^{j\frac{2\pi}{64}4n} + e^{j\frac{2\pi}{64}60n}\right], \text{ with: } j\frac{2\pi}{64}60n = -j\frac{2\pi}{64}4n + j2\pi n \\
&= \frac{3}{2}\langle w_4[n] + w_{60}[n]\rangle
\end{aligned}$$

   - $w_4[n]$ Basis vector number 4
   - $w_{60}[n]$ Basis vector number 60

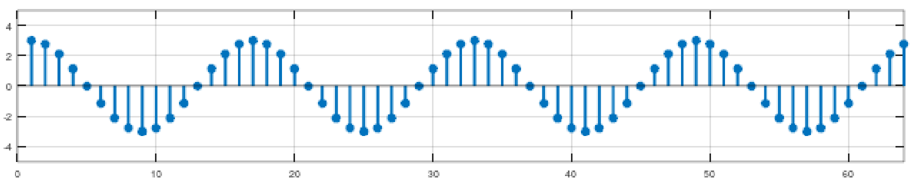
     Now we don't like this minus. So what we're going to do is exploit the fact that we can always add an integer multiple of 2pi to the exponent of the complex exponential. And the point will not change on the complex plane.

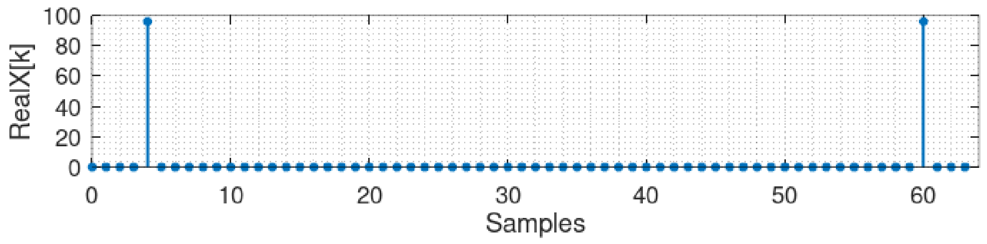
   - **The original signal is now expressed as the sum of two fourier basis vectors**

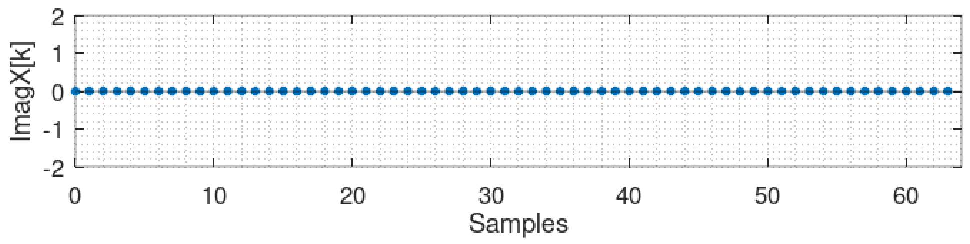
3. Calculate the DFT with the analysis forumla

$$X[k] = \langle w_k[n], x[n] \rangle, \text{ with: } k = 0, 1, ..N-1$$
$$= \begin{cases} 96 & \text{for } k = 4, 60 \\ 0 & \text{otherwise} \end{cases}$$

- $w_k[n]$ Canonical basis vector number k

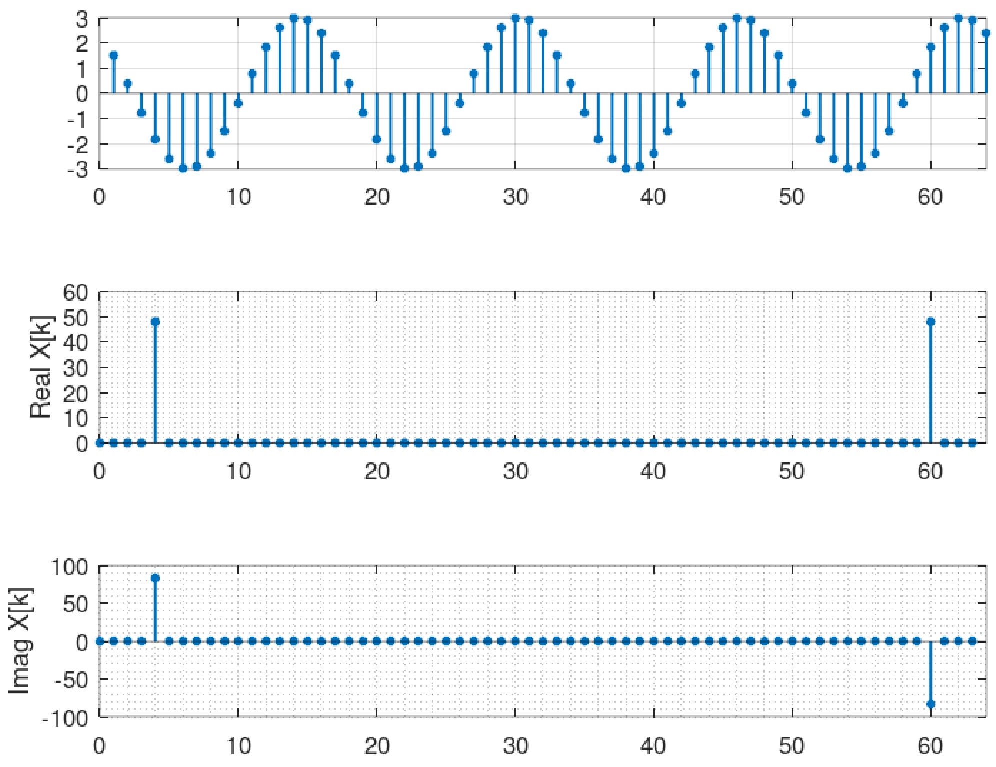


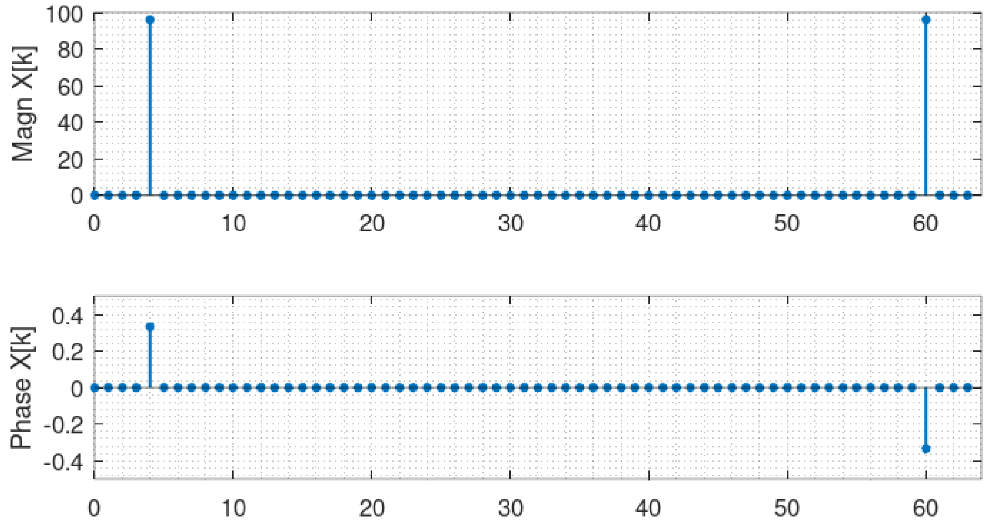




3.2.2.4. DFT Cosine Calculation Problem 2

$$x[n] = 3\, cos(2\, pi/16\, n + pi/3),\; x[n] \in \mathbb{C}^{64}$$
$$X[k] = \begin{cases} 96e^{j\frac{\pi}{3}} & \text{for } k = 4 \\ 96e^{-j\frac{\pi}{3}} & \text{for } k = 96 \\ 0 & \text{otherwise} \end{cases}$$



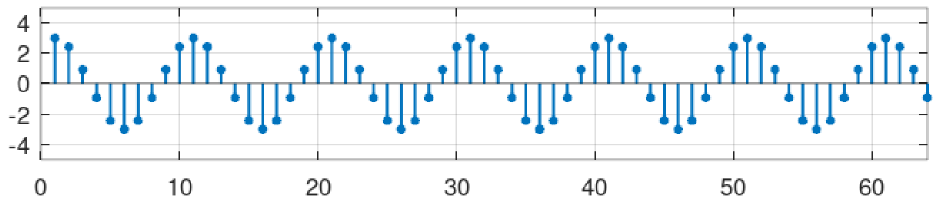


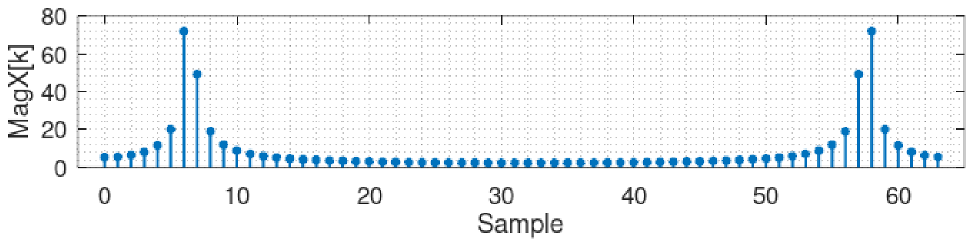
The calcution of the phase just does not work out of the box with octave.

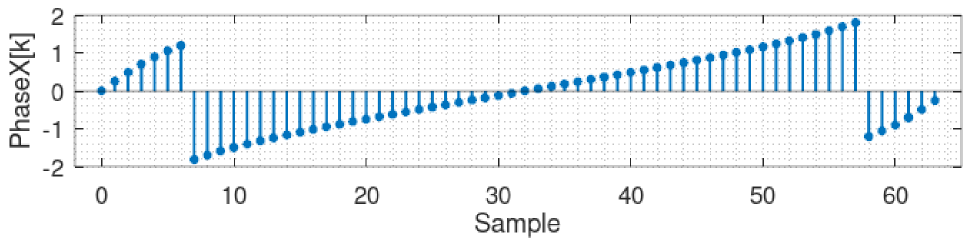
3.2.2.5. DFT Cosine Calculation Problem 3

$$x[n] = 3\,cos(2\,pi/10\;n),\; x[n] \in \mathbb{C}^{64}$$

$$X[k] = \begin{cases} 96e^{j\frac{\pi}{3}} & \text{for } k = 4 \\ 96e^{-j\frac{\pi}{3}} & \text{for } k = 96 \\ 0 & \text{otherwise} \end{cases}$$







3.2.3. Properties of the DFT

Linearity $DFT \alpha x[n] + \beta y[n] = DFT \alpha x[n] + DFT \beta y[n]$

3.2.4. Interpreting a DFT Plot

- Frequency coeffcience $< \pi[0...N/2]$ are interpreted as counter clock wise rotation in the plane
- Frequency coeffcience $> \pi[N/2...N-1]$ are interpreted as clock wise rotation in the plane
- The fastest frequency of the signal in the vector space is at N/2

Energy of a Signal

The square magnitude of the k-th DFT coefficent is proportional to the signal's energy at frequency $\omega = (\frac{2\pi}{N})k$

- Energy concentrated on single frequency (counterclockwise and clockwise combine to give real signal)

$$x1[n] = 3\ cos(2\ pi/16\ n),\ x[n] \in \mathbb{C}^{64}$$
$$x1[n] = u[n] - u[n-4]$$

- For real signals the DFT is symmetric in magnitude
  - $|X[k]| = |X[N-k]|$, for $k = 1, 2, ...[N/2]$
  - For real signals, magnitude plots need only $[N/2] + 1$ points

3.3. The DFT in Practice

3.3.1. TODO DFT Analysis

3.3.1.1. TODO Mystery Signal revisted

3.3.1.2. TODO Solar Spots

3.3.1.3. TODO Daily Temeperature (2920 days)

- The recorded signal
- average value (0-th DFT coefficient: 12.3°
- DFT main peak for $k = 8$, value 6.4°C
- 8 cycles over 29920 days
- $period = \frac{2920}{8} = 365 days$
- temperature exursion: 12.3° +/- 12.8°C

The fastest positive frequency of a singnal is at $fracN2$ samples. Since a full revolution of 2 π requires N samples, the discrete frequency corresponding with $\frac{N}{2}$) is π.

3.3.1.4. Labeling Frequency Band Axis

- If "clock" of a System is T_s
  - fastest (positive) frequency is $\omega = \pi$
  - sinosoid at $\omega = \pi$ needs two samples to do a full revolution
  - time between samples: $T_s = \frac{1}{F_s}$ seconds
  - real world period for fastest sinosoid: $2T_s$ seconds
  - reald world frequency for fastest sinosoid: $F_s/2$ Hz
- The discrete frequency x of a sinusoid compenent at peak k can be determined as follows:

$$\frac{x}{k} = \frac{N}{2\pi}, \text{ with k=0...N-1} \tag{3.5}$$

- The real world frequency of a siusoid compenent at peak k can be determined as follows:

$$\begin{aligned}\frac{x}{k} &= \frac{2\pi}{N}, \text{ with k=0...N-1}\\ \frac{f_s}{2} &= \rightarrow \pi,\ f_s \text{ sampling frequency}\\ \frac{x}{k} &= \frac{f_s}{N}\\ x &= \frac{kf_s}{N}\end{aligned}$$

3.3.1.5. TODO Example: train whistle

3.3.1.6. Example 1

A DFT analysis of a signal with length $N = 4000$ samples at a frequency $fs = 44.1kHz$ shows a peak at $k = 500$. What is the corresponding frequency in Hz of this digital frequency in Hz.

- Solution

$$\frac{x}{k} = \frac{2\ \pi}{N}$$
$$x \rightarrow \frac{2\ \pi\ k}{N}$$
$$\frac{f_s}{2} \rightarrow \pi$$
$$x = \frac{k}{N} f_s \qquad = 55125.5$$

3.3.1.7. Example 2

Calculation of the corresponding frequency vector for a signal for which its spectrum is analysed with the fourier transform

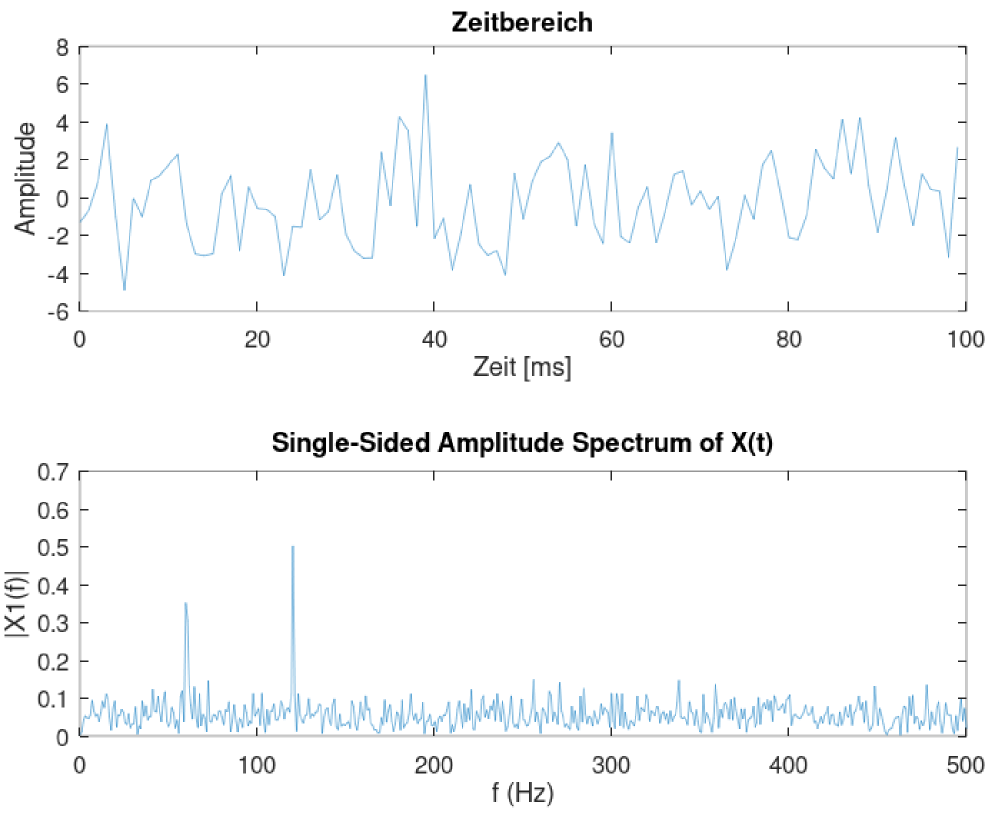
- Sampling Period: $T_s = 1/1000s$
- Sampling Frequency: $f_s = 1/T = 1000Hz$
- Vector Length $N = 2^10 = 1024$

$$\frac{X}{k} = \sum_{n=1}^{N} x[n]e^{-j2\pi(k-1)(\frac{n-1}{N})}$$
$$f(k) = \frac{k-1}{NT}, \text{ corresponding Frequency in Hz}$$

- StackOverflow

```
clear all;
close all;
N  = 1024;    # vector length
Fs = 1000;    # Sample Frequency Fs = 1000Hz
Ts = 1/Fs;    # Sampling Period  Ts = 0.001s
f1 = 60;      # 50Hz
f2 = 120;     # 120Hz

n  = 0:Ts:(N-1)*Ts;                        # time vector
x  = sin(2*pi*f1*n) + sin(2*pi*f2*n);      # a sinusoid signal
xr = x + 2*randn(size(n));                 # a noisy signal

X  =fft(xr);                               # FFT
X2 = 1/N*abs(X);                           # FFT magnitude full buffer length
F2 = Fs*(0:(N-1))/N;                       # Frequency vector full buffer length

X1 = X2(1:N/2+1)/2;                        # FFT magniture half buffer lenght
X1(2:end-1) = 2*X1(2:end-1);               # Arranged values
F1 = Fs*(0:(N/2))/N;                       # Frequency vector half buffer length

figure( 1, "visible", "off" )

subplot(2,1,1)
plot(Fs*n(1:100),xr(1:100));
title('Zeitbereich')
ylabel('Amplitude');
xlabel('Zeit [ms]')
set(gca, "fontsize", 24);
```

```
subplot(2,1,2)
plot(F1,X1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|X1(f)|')
set(gca, "fontsize", 24);


## subplot(2,1,3);
## plot(F2,X2);
## title('Two-Sided Amplitude Spectrum of X(t)')
## ylabel('|X2(f)|')
## xlabel('Frequenz [Hertz]')
## set(gca, "fontsize", 24);

                                                    # Org-Mode specific setting
print -dpng "-S800,600" ./image/eth-example.png;
ans = "./image/eth-example.png";
```



3.3.2. TODO DFT Example Analysis of Musical Instruments

- The fundamental note is the first peak in the spectrum
- The relative size of the harmonics gives the timber or the charachter of an instrument

3.3.3. TODO DFT Synthesis

3.3.4. TODO DFT Example - Tide Prediction in Venice

3.3.5. TODO DFT Example - MP3 Compression

- MP3 compression approx. factor 20 or more
- Compression introduces nois from approximation error
- Noise Shaping : Error shaped as the song in the Fourier domain.
- Perceptual Compression includes the human hearing system properties intto compression algorithm

3.3.6. TODO Signal of the Day: The first man-made signal from outer space

$$f = \frac{\omega f_s}{2\pi}$$

- A multiplication in time domain corresponds to a convolution in frequency domain

3.4. The Short-Time Fourier Transform STFT

- STFT is a clever way of using DFT
- Spectrogram, is a graphical way to represent the STFT data

3.4.1. The short-time Fourier transform

- DTMF Dual-Tone Multi Frequency dialing
- Time representation obfuscates frequency
- Frequency representation obfuscates time

$$x[m;k] = \sum_{n=0}^{L-1} x[m+n] e^{-j\frac{2\pi}{L}nk}$$

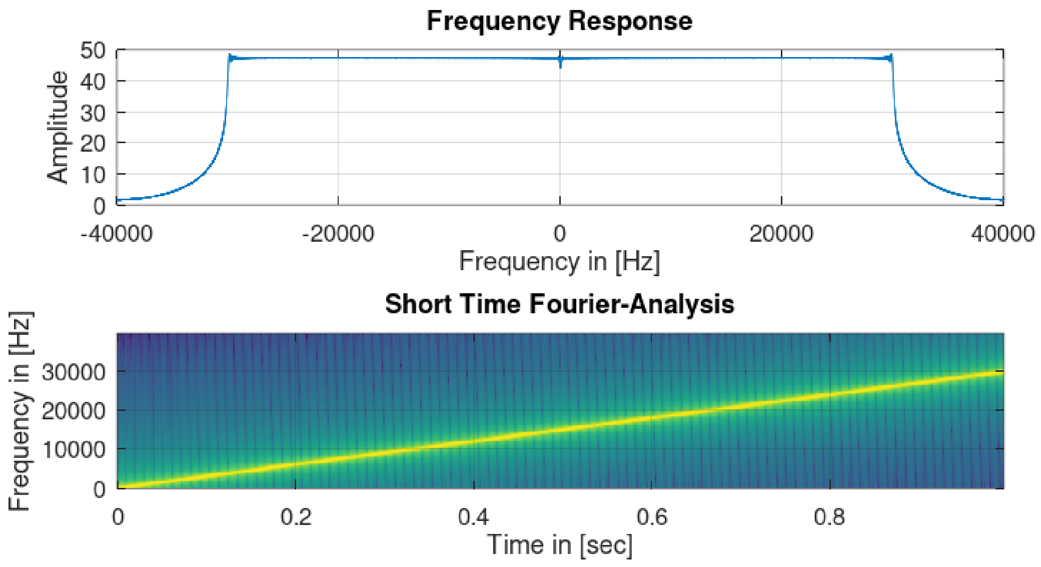
  - **m** Starting point of the localiced DFT
  - **k** Is the DFT index

3.4.2. TODO The spectrogram

- color-code the magnitued: dark is small, white is large
- use $10log_{10}(|X[m,k]|$ to see better (powr in dBs)
- plot spectral slices one after another

3.4.3. TODO Time-frequency tiling

3.4.4. STFT Example



Part IV.

Week 4 Module 3:

4. Part 2 - Advanced Fourier Analysise

4.1. Discrete Fourier Series DFS

4.1.1. TODO Discrete Fourier series

Discrete Fourier Series

DFS = DFT with periodicity explicit $\tilde{X}[k] = DFS\{x[n]\}$

- The DFS maps an N-Periodic signal onto an N-Periodic sequence of Fourier coeffients
- The inverse DFS maps $\text{n}_{\text{periodic}}$ sequence of Fourier coeffiencts a set onto an N-periodic signal
- DFS is an extension of the DFT for periodic sequencies
- A circular time-shift is an natural extension of a shift fo finite length signals.

4.1.1.1. Finite-length time shifts revisted

- The DFS helps us understand how to define time shifts for finite-lenght signals.

test

For an N-periodic sequence $\tilde{x}[n]$

$$\begin{aligned}
&\tilde{x}[n-M] \text{ is well-defined for all } M \in \mathbb{N}\\
&DFS\{\tilde{x}[n-M]\} = \boxed{e^{-j\frac{2\pi}{N}Mk}}\tilde{X}[k] \text{ delay factor}\\
&IDFS\left\{\boxed{e^{-j\frac{2\pi}{N}Mk}}\tilde{X}[k]\right\} = \tilde{x}[n-M] \text{ delay factor}
\end{aligned}$$

For an N-length signal $x[n]$

$$\begin{aligned}
&\tilde{x}[n-M] \text{ not well-defined for all } M \in \mathbb{N}\\
&build\ \tilde{x}[n] = x[n\ mod\ N] \Rightarrow\ \tilde{X}[k] = X[k]\\
&IDFT\left\{\boxed{e^{-j\frac{2\pi}{N}Mk}}X[k]\right\} = IDFS\left\{\boxed{e^{-j\frac{2\pi}{N}Mk}}\tilde{X}[k]\right\} = \tilde{x}[n-M] = x[(n-M)\ mod\ N]
\end{aligned}$$

Periodicity

Shifts for finite-length signals are "naturally" circular

4.1.2. TODO Karplus-Strong revisted and DFS

4.1.2.1. Analysis Formula for a N-Periodic Signal in the frequency domain

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n] e^{-j\frac{2\pi}{N}nk},\ k \in \mathbb{Z} \tag{4.1}$$

$X[k]$ Signal vector in the frequency domain

$x[n]$ Signal vector in the (discrete) time domain

Reminder This is the inner Product in explicite form

4.1.2.2. Synthesis Formula for a N-Periodic Signal in the time domain

$$\tilde{x}[n] = \frac{1}{N} \sum_{n=0}^{N-1} \tilde{X}[k] e^{j\frac{2\pi}{N}nk},\ k \in \mathbb{Z} \tag{4.2}$$

$x[n]$ Signal vector in the (discrete) time domain

$X[k]$ Signal vector in the frequency domain

$\frac{1}{N}$ Normalisation coeficent

Reminder This is the inner Product in explicite fashion

4.2. The Discret-Time Fourier Transform (DTFT)

4.2.1. Overview Fourier Transform

- N-Point finite-length siganls: DFT
- N-Point periodic signals: DFS
- Infinite length (non periodic) signals: DTFT

4.2.2. Karplus Strong revisted and the DTFT

4.2.2.1. Plotting the DTFT

- Frequencies go from $-\pi$ to π
- Positive frequencies are on the right hand side of the x-axis
- Negative frequencies are on the left hand side of the x-axis
- Low frequencies are centered around 0
- High frequnecies will be on the extreme of the bound

4.3. Existence and properties of the DTFT

4.3.1. Formal Definition of the DTFT

- $x[n] \in \ell_2(\mathbb{Z})$, the space of square summable infinity sequneces
- define the function of $\omega \in \mathbb{R}$

$$F(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}, \text{ with } \omega = \frac{2\pi}{N} \text{ and } N \to \infty$$

- inversion (when $F(\omega)$ exists):

$$x[n] = \frac{1}{2\,\pi} \int_{-\pi}^{\pi} F(e^{j\omega}); e^{j\ \omega\ n} \mathrm{d}\omega, \text{ with } n \in \mathbb{Z}$$

4.3.2. Properties of the DTFT

linearity	$DTFT\{\alpha x[n] + \beta y[n]\} = \alpha X(e^{j\omega}) + \beta Y(e^{j\omega})$
timeshift	$DTFT\{x[n-M]\} = e^{-j\omega M}\ X(e^{j\omega})$
modulation	$DTFT\{e^{-j\omega_0 M}\ x[n]\} = X(e^{j\ (\omega-\omega_0)})$
time reversal	$DTFT\{x[-n]\} = X(e^{-jw})$
conjugation	$DTFT\{x^*[n]\} = X^* X(e^{-j\ \omega})$

4.3.3. Some particular cases

- if $x[n]$ is symmetric, the DTFT is symmetric: $x[n] = x[-n] \iff X(e^{j\omega}) = X(e^{-j\omega})$
- if $x[n]$ is real, the DTFT is Hemitian-symmetric: $x[n] = x^*[n] \iff X(e^{j\omega}) = X^*(e^{-j\omega})$
- if $x[n]$ is real, the magnitude of th eDTFT is symmetric $x[n] \in \mathbb{R} \implies |X(e^{j\omega})| = |X(e^{-j\omega})|$
- if $x[n]$ is real and symmetric, $X(e^{j\omega})$ is also real and symmetric

4.3.4. TODO The DTFT as a change of basis

4.4. TODO Sinusoidal Modulation

4.4.1. TODO Sinusoidal modulation

4.4.2. TODO Tuning a guitar

4.4.3. TODO Signal of the day: Tristan Chord

4.5. TODO Notes and Supplementary Material

4.5.1. TODO Relation Ship between transforms

4.5.2. TODO The fast fourier transform

Part V.

Week 5 Module 4:

5. Part 1 Introduction to Filtering

5.1. Linear Time-Invariant Systems

LTI System

Linearity and Time Invariance taken together: A Linear Time Invariant System is completely characterized by its response to the input in particular by its the Impulse Response .

5.1.1. Linearity

Linearity is expressed by the equivalence

$$\mathfrak{H}\big\{\alpha\, x_1[n] + \beta\, x_2[n]\big\} = \alpha\, \mathfrak{H}\big\{x_1[n]\big\} + \beta\, \mathfrak{H}\big\{x_2[n]\big\} \tag{5.1}$$

- Fuzz-Box, example for a none linear device

5.1.1.1. TODO Add calculation examples

5.1.2. Time invariance

- The system behaves the same way independently of when a it's switched on

$$y[n] = \mathfrak{H}\big\{x[n]\big\} \Leftrightarrow \mathfrak{H}\big\{x[n-n_o]\big\} = y[n-n_o] \tag{5.2}$$

- Wah-Pedal, example of a time variant device

5.1.2.1. TODO Add calculation examples

5.1.3. Convolution

The impulse response is the output of a filter when the input is the delta function.

$$h[n] = \mathfrak{H}\big\{\delta[n]\big\} \tag{5.3}$$

Impulse Response

Impulse response fully characterize the LTI system!

We can always write

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \tag{5.4}$$

by linearity and time invariance

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] && (5.5)\\ &= x[n] * h[n] && (5.6) \end{aligned}$$

Performing the convolution algorithmically

$$x[n] \ast h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Ingredients
- a sequence x[m]
- a second sequence h[m]

The Recipe
- time-reverse h[m]
- at each step n (from $-\infty\, to\, \infty$):
  - center the time-reversed h[m] in n (i.e. by shift -n)
  - compute the inner product

Furthermore, the convolution can be defined in terms of the inner product between two sequencies.

$$\begin{aligned}(x \ast y)[n] &= \langle x^{*}[k], y[n-k]\rangle \\ &= \sum_{n=-\infty}^{\infty} x[k]y[n-k]\end{aligned}$$

5.2. Filtering in the Time Domain

For the convolution of two sequencies to exist, the convolution sum must be finite i.e. the both sequencies must be absolutely summable

5.2.1. The convolution operator

Linearity $x[n] \ast (\alpha \cdot y[n] + \beta \cdot w[n]) = \ast\alpha \cdot x[n] \ast y[n] + \beta \cdot x[n] \ast w[n])$
$w[n] = x[n] \ast y[n] \iff x[n] \ast y[n-k] = w[n-k]$

Commutative $x[n] \ast y[n] = y[n] \ast x[n]$

Associative $(x[n] \ast y[n]) \ast w[n] = x[n] \ast (y[n] \ast w[n])$

5.2.2. Convolution and inner Product

$$\boxed{x[n] \ast h[n] = \langle\, h^{*}[n-k], x[k]\rangle}$$

Filtering measures the time-localized similarity between the input sequence and a prototype sequence - the time reversed impulse response.
In general the convolution operator for a signal is defined with repsect to the inner product of its underlaying Hilbert space:

Square Summable Sequence $\ell_2(\mathbb{Z})$ $x[n] \ast h[n] = \langle\, h^{*}[n-k], x[k]\rangle$

N-Periodic Sequence $\tilde{x}[n] \ast \tilde{y}[n]) \sum_{k=0}^{N-1} \tilde{x}[n-k]\tilde{y}[k]$

Square Integrable Function $L_2([-\pi,\pi])$ $X(e^{j\omega}) \ast Y(e^{\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\sigma}) \cdot Y(e^{j(\omega-\sigma)})$

5.2.3. Properties of the Impulse Response

Causality A system is called causal if its output does not depend on futre values of the input. In practice a causual system is the only type of ”real-time” syste we can actually implement.

Stability A system is called bounded-input bounded-output stabel (BIBO stable) if its output is bounded for all bounded input sequencies. **FIR** Filter are always stable, since only in the convolution sum only a finite number of terms are involved.

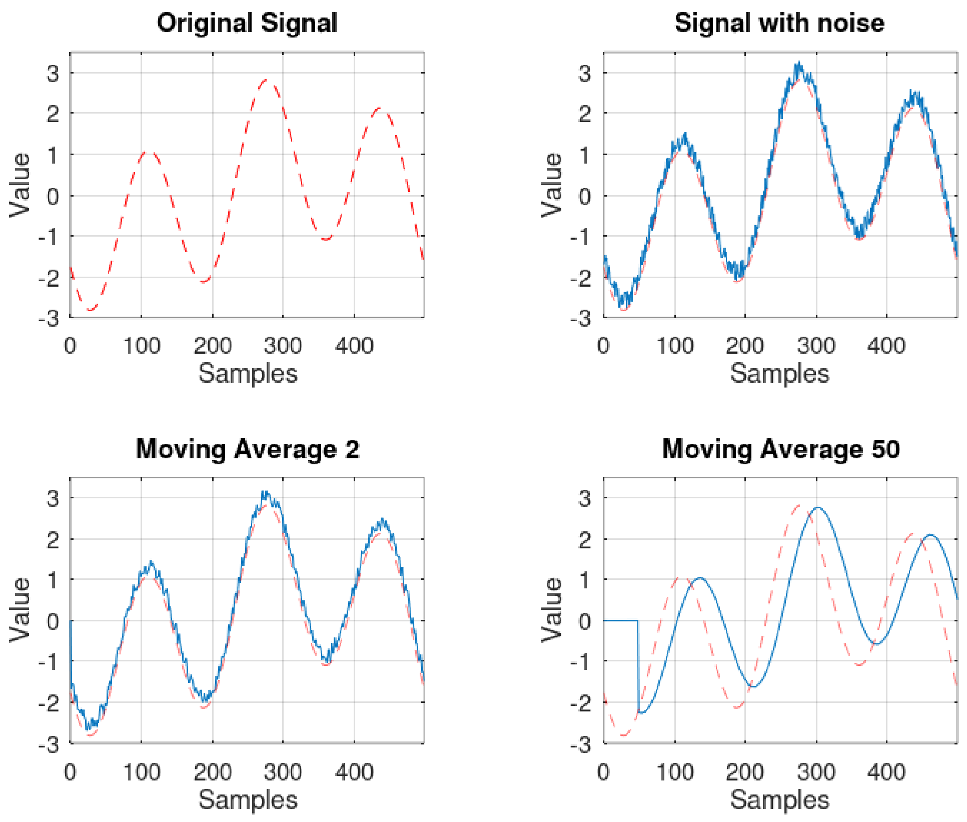
5.2.4. Filtering by Example

5.2.4.1. FIR Filter: Moving Average

Typicale filtering scenario: denoising

- idea: replace each sample by the local average. Average are useually good to eliminate random variation from which you don't know mutch about it.
- for instance: $y[n] = (x[n] + x[n-1])/2$
- more generally:

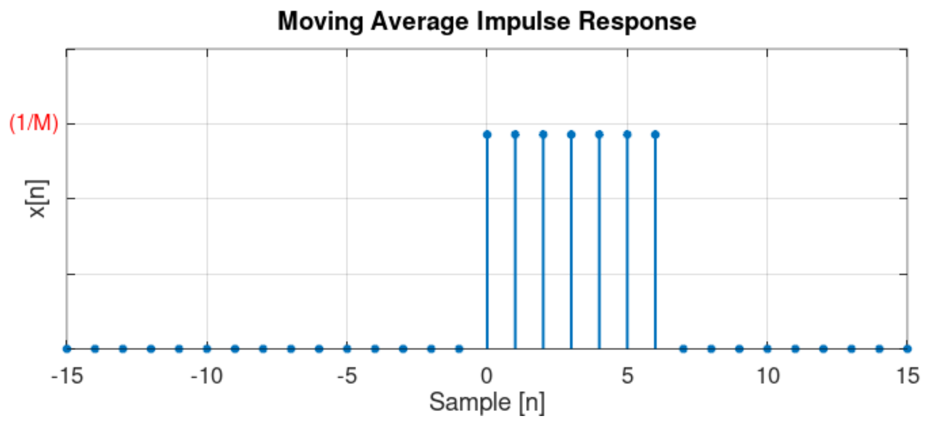
$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$



- Impulse Response

$$h[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n-k] \begin{cases} \frac{1}{M} & \text{for } 0 \leq n < M \\ 0 & \text{otherwise} \end{cases}$$

```
function [x,n] = ma_impresp(M,n1,n2)
% Generates x(n) = delta(n); 0 <= M
% ----------------------------------------------
% [x,n] = stepseq(n0,n1,n2)
%
  n = [n1:n2]; x = [ (n >= 0) & !((n-M) >= 0) ]./M;
end
```



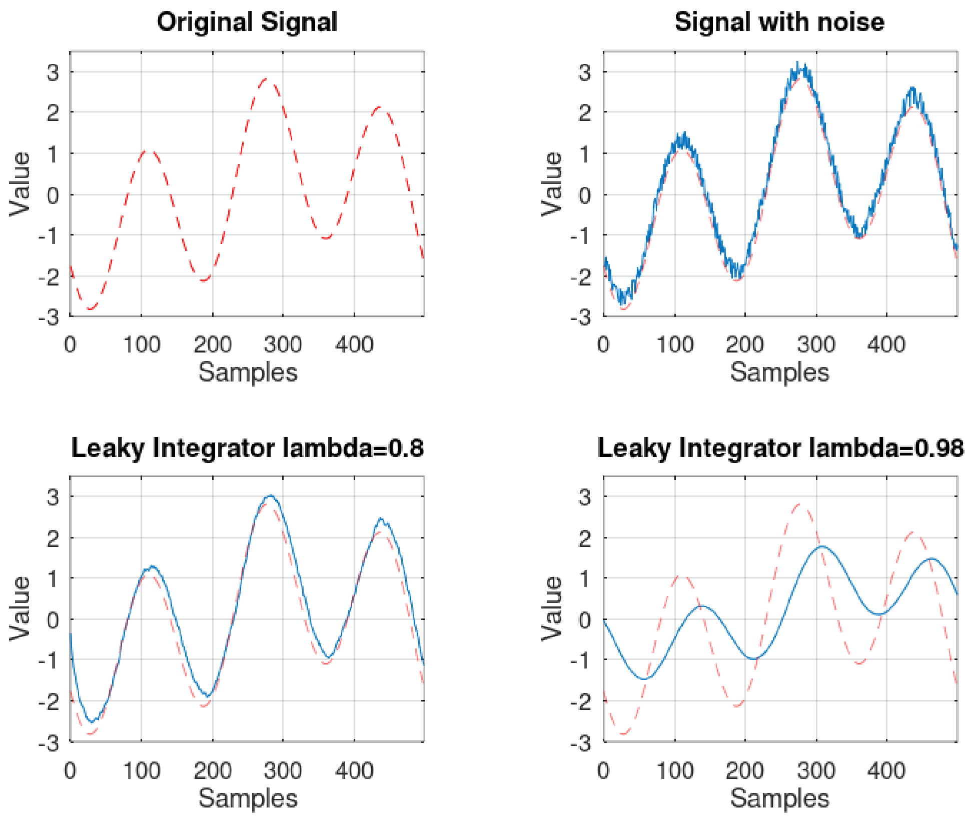
- MA Analysis
  - soomthin effect is proportional to M
  - number of operations and storage also proportional to M
- From the MA to first-order recursion

$$y_{M[n]} = \sum_{k=0}^{M-1} x[n-k] = x[n] X \sum_{k=1}^{M-1} x[n-k]$$
$$M_{y_{M[n]}} = x[n] + (M-1)y_{M-1}[n-1]$$
$$y_M[n] = \frac{M-1}{M} y_{M-1}[n-1] + \frac{1}{M} x[n]$$
$$y_M[n] = \lambda y_{M-1}[n-1] + (1-\lambda)x[n],\ \lambda = \frac{M-1}{M}$$

5.2.4.2. IIR Filter: The Leaky Integrator

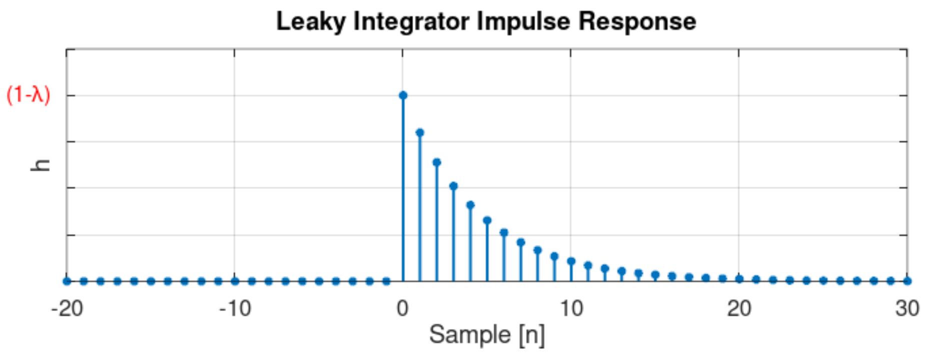
- when M is large, $y_{M-1}[n] \approx y_M[n]$and $(\lambda \approx 1)$
- the filter becomes: $y[n] = \lambda y[n-1] + (1-\lambda)x[n]$
- the filter is now recursive, since it uses its previous output value

```
function y = lky_impresp(a,b,lambda,x)
% Generates x(n) = a^n
% -----------------------------------------------
% [x,n] = lky_impresp(a,b, lambda, x)
% y[n] -lambda y[n-1] = (1-lambda) x[n]
% a = [1, -lambda];
% b = [(1-lambda)];

  b = [1-lambda];
  a = [1, -lambda];
  y = filter(b,a,x);
end
```



- Impulse Response For the impulse we just need to plug the delta function

$$\begin{aligned} h[n] &= (\lambda y[n-1] + (1-\lambda))\delta[n] \\ &= (1-\lambda)\lambda^n u[n] \end{aligned}$$



The peak at n=0 is $1-\lambda$.

- The leaky integrator why the name
  - Discrete Time integrator is a boundless accumulator

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{n} x[k] \\ &= y[n-1] + x[n] \Rightarrow \text{ almost leaky integrator} \end{aligned}$$

To prevent "explosing" we scale the accumulator with λ:

$\lambda y[n-1]$	keep only a fraction λ of the accumulated value so far and forget ("leak") a fraction $\lambda - 1$

$(1-\lambda)x[n]$ add only a fraction $1-\lambda$ of the current value to the accumulator.

So we get the leaky integrator from the accumulator

$$y[n] = \lambda \cdot y[n-1] + (1-\lambda) \cdot x[n] \Rightarrow \text{ almost leaky integrator}$$

5.3. Classification of Filters

FIR Finite Impulse Response Filter

- Impulse response has finite support
- only a finite number of samples are involved in the computation of each output
- Example: Moving Average Filter

IIR Infinite Impulse Response Filter

- Impulse response has inifinte support
- a potentially infinite number of samples are involved in the computation of each output sample
- surprisingly, in many cases the computation can still be performed in a finite amount of steps
- Example: The Leaky Integrator

Casual

- impulse response is zero for n < 0
- only past samples are involved in the computation of each output sample
- causul filters can work ”on line” since they only need the past

Noncasual

- impulse response in nonzero for some (or all) n < 0
- can still be implemented in a offline fashing (e.g. image processing)

5.4. Filter Stability

FIR Filter

FIR filters are always stable

because their impuls response only contains a finite number of non-zero values, and therefore the sum of their absolute values will always be finite.

5.5. Frequency Response

5.5.1. References

- Signal and System for Dummies: Frequency Response

5.5.2. Eigensequence

If a complex exponential is applied to a LTI filter its response is the DTFT of the impulse response of the LTI filter times the complex exponential.

$$
\begin{aligned}
x[n] &= e^{j\omega_0 n} \\
y[n] &= \mathfrak{H}\{x[n]\} \\
y[n] &= x[n] * h[n] \\
y[n] &= e^{j\omega_0 n} * h[n] \\
y[n] &= H(e^{j\omega_0}) e^{j\omega_0 n}
\end{aligned}
$$

- DTFT of impulse response determinse the frequency characteristic of a filter
- Complex exponential are eignesequences of LTI systems, i.e. linear filters cannot change the frequency of a sinusoid.

5.5.3. Magnitude and phase

$$
\begin{aligned}
&\text{if } H(j^{j\omega_0}) = Ae^{j\theta}\text{, then} \\
&\mathfrak{H}\{e^{j\omega_0 n}\} = Ae^{j(\omega_0 n + \theta)}
\end{aligned}
$$

amplitude	**A**	**phase shift**	θ
amplification	>1	delay	< 0
attenuation	$0 \leq A < 1$	advancment	> 0

5.5.4. The convolution theorem

The convolution theorem summerizes this result in

$$\boxed{DTFT\{x[n] * h[n]\} = X(e^{j\omega})H(e^{j\omega})}$$

5.5.5. Frequency response

The DTFT of the impulse response is called the frequency response

$$\boxed{H(e^{j\omega}) = DTFT\{h[n]\}}$$

magnitude	$\mid H(e^{j\omega} \mid$	**phase**
amplification	> 1	overall shape and
attenuation	< 1	phase changes

5.5.6. Example of Frequency Response: Moving Average Filter

The difference equation from M-point averager is

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

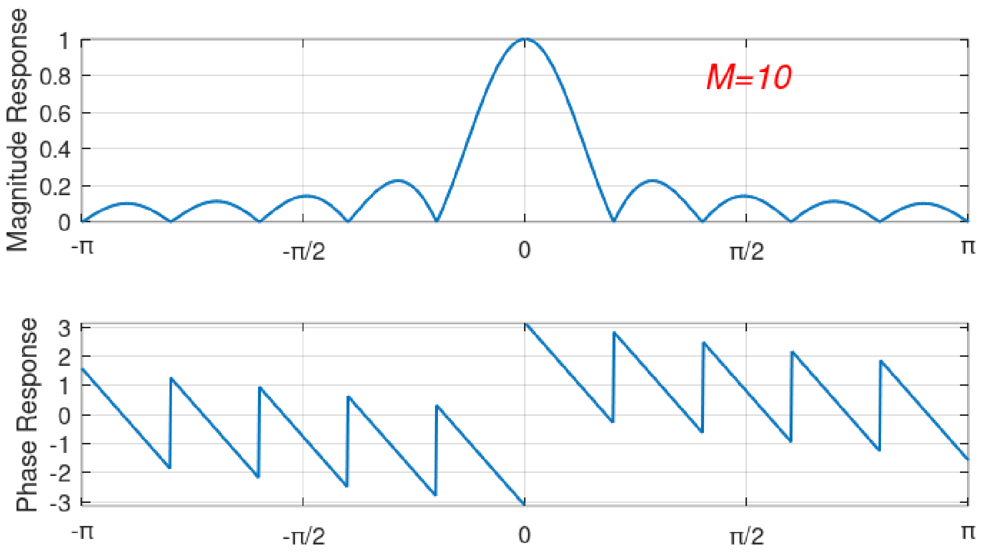
The Frequency response of the moving average filter

$$H(e^{j\omega}) = \frac{1}{M}\sum_{k=0}^{M-1} e^{-j\omega k} = \frac{1}{M}\sum_{k=0}^{M-1} \left(e^{-j\omega}\right)^k$$
$$= \frac{1}{M}\frac{(1-e^{-j\omega M})}{(1-e^{j\omega})}$$

- The frequency response is composed of a linear term $e^{-j\omega\frac{M-1}{2}}$ and $\pm\pi$ due to the sign changes of $\frac{sin(\frac{\omega}{2}M)}{sin(\frac{\omega}{2}M)}$

The Magnetute response of the moving average filter

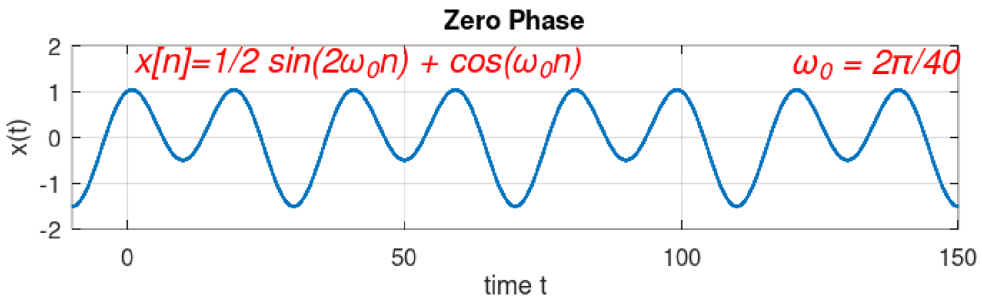
$$H(e^{j\omega}) = \frac{1}{M}\left|\frac{sin(\frac{\omega}{2}M)}{sin(\frac{\omega}{2})}\right|$$



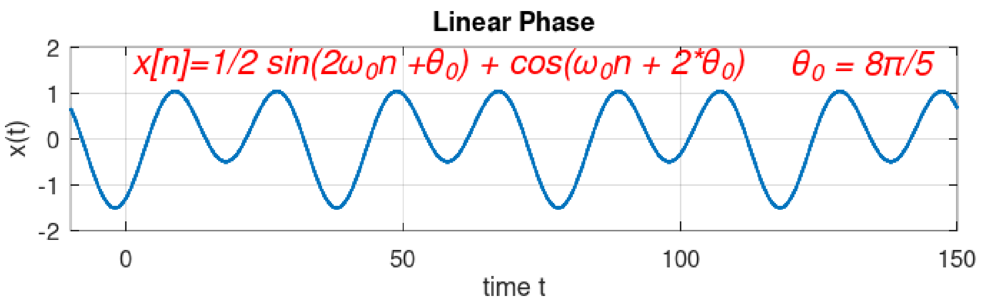
5.5.7. Phase and signal shape

To understand the effects of the phase on a signal is to distinguihs three different cases

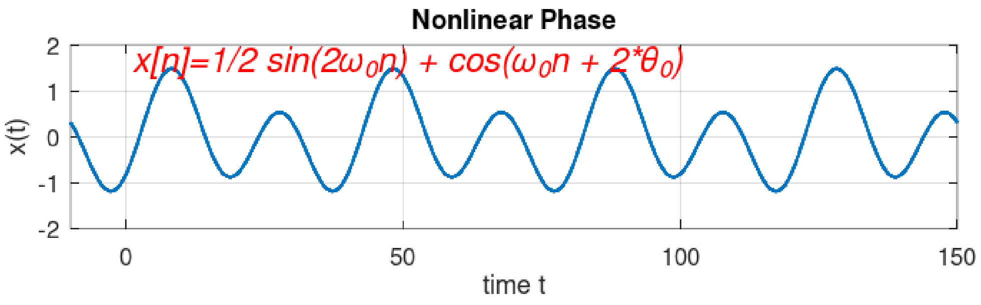
- zero phase: the spectrum is real: $\angle H(e^{jw}) = 0$



- linear phase: the phase is proportional to the frequency via a real factor, d: $\angle H(e^{jw}) = d\omega$ the phase is proportional to the frequency of the sinusoid. The net effect is a shift of the signal if the phase component is porportional to the frequency.



- non linear phase: which covers all the other properties now the shape of the signal in the time domain changes.



Spectrum

The spectrum of all three signals x[n] remains exactecly the same in magnitude.

5.5.8. Linear Phase

$$\begin{aligned} y[n] &= x[n-d] \\ Y(e^{j\omega}) &= e^{-j\omega d}\, X(e^{j\omega}) \\ H(e^{j\omega}) &= e^{-j\omega d} \Rightarrow linearphaseterm \end{aligned}$$

5.5.9. Moving Average is linear Phase

$$\begin{aligned} H(e^{j\omega}) &= A(e^{j\omega})e^{-j\omega d} \\ &\Rightarrow A(e^{j\omega})\text{: pure real term} \\ &\Rightarrow e^{-j\omega d}\text{: pure phase term} \\ &= \frac{1}{M}\frac{sin(\frac{\omega}{2}M)}{sin(\frac{\omega}{2}M)}e^{-j\omega\frac{M-1}{2}} \Rightarrow \frac{M-1}{2} = d \end{aligned}$$

5.5.10. Example of Frequency Response: Leaky Integrator

The Frequency response of the leaky integrator

$$H(e^{j\omega}) = \frac{1-\lambda}{1-\lambda e^{j\omega}}$$

Finding the magnitude and phaser requires a little algebra
From Complex Algebra

$$\frac{1}{a+jb} = \frac{1-jb}{a^2+b^2}$$

So that if $x = \dfrac{1}{a + jb}$

$$|x|^2 = \frac{1}{a^2 + b^2}$$
$$\angle x = tan^{-1}\left[-\frac{a}{b}\right]$$

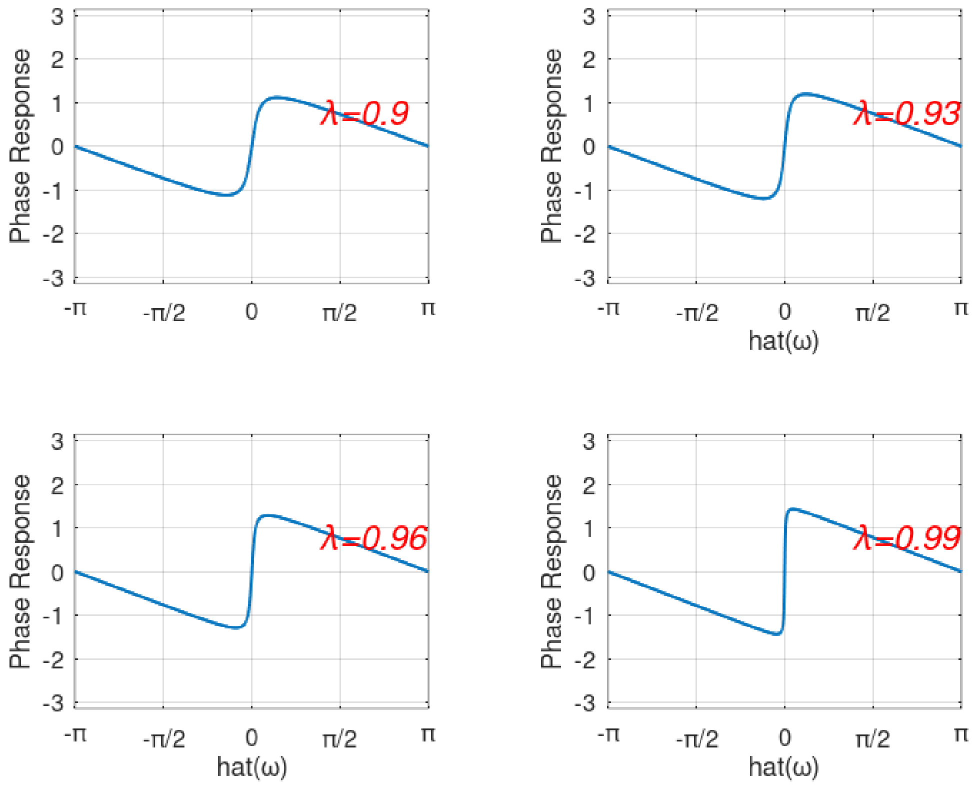
$$H(e^{j\omega}) = \frac{1-\lambda}{(1-\lambda cos\omega) - jsin\omega}$$

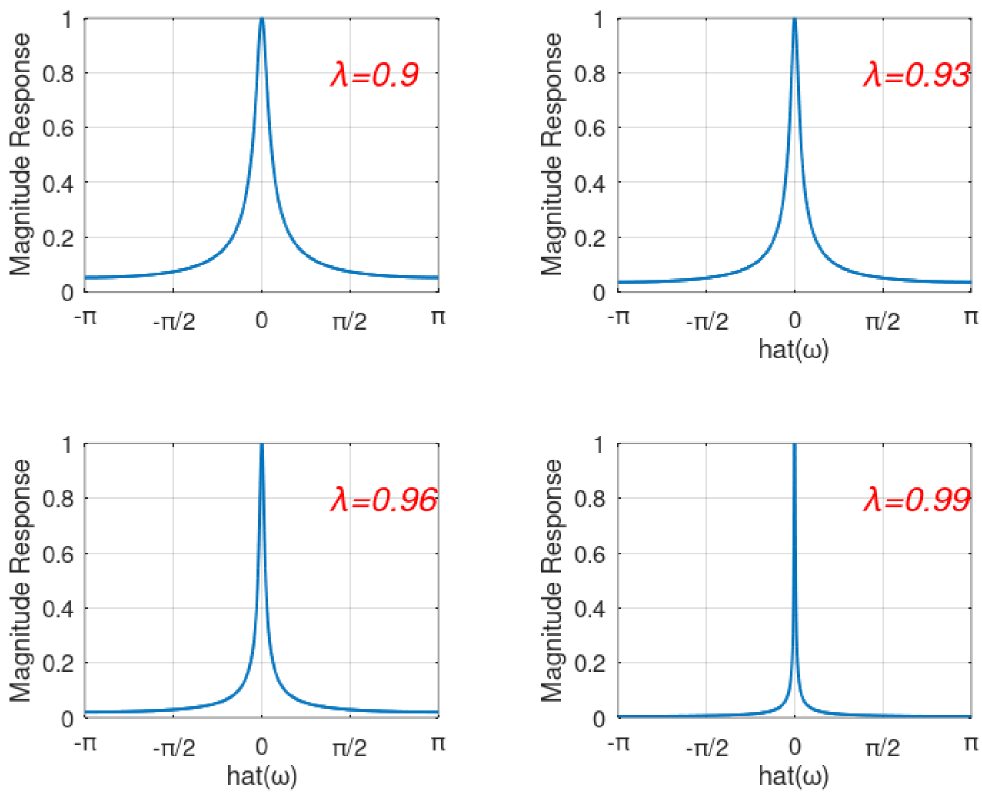
so that:

$$|H(e^{j\omega})|^2 = \frac{(1-\lambda)^2}{1 - 2\lambda cos\omega + \lambda^2}$$

$$\angle H(e^{j\omega}) = tan^{-1}\left[\frac{\lambda sin\omega}{1-\lambda cos\omega}\right]$$

The phase is nonlinear in this case





5.5.11. TODO Example of Frequency Response: Karplus Strong Algorithm

$$y[n] = \alpha y[n-M] + x[n]$$

The Karplus-Strong algorithm is initialized with a finite support signal x of support M. And then we use a feedback loop with a delay of M taps. To qproduce multiple copies of the original finite support signal, scaled by an exponentially decaying factor alpha.

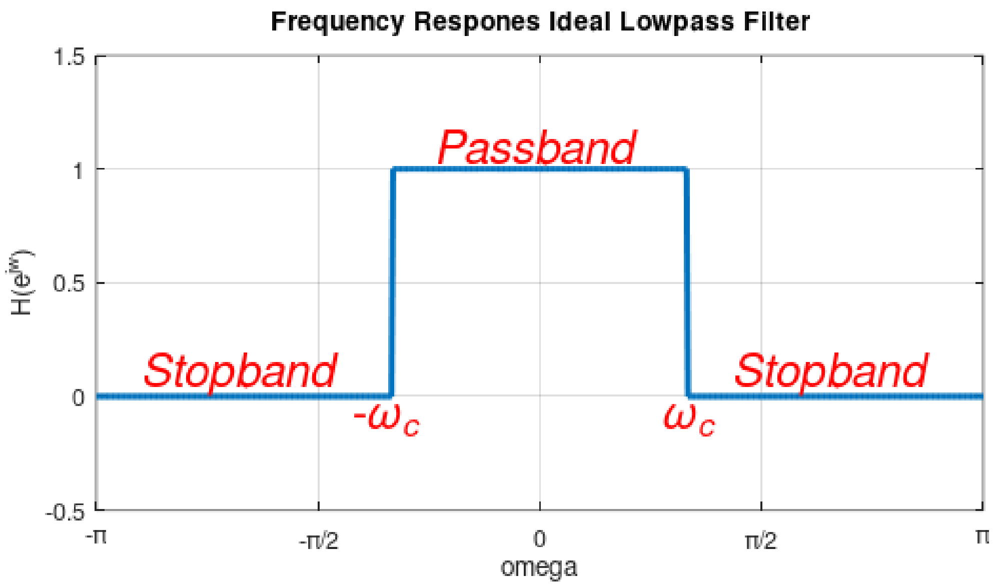
5.5.11.1. With Sawtooth Wave

$$\tilde{X}(j\omega)W(j\omega) = e^{-j\omega}\left(\frac{M+1}{M-1}\right)\frac{1-e^{-j(M-1)\omega}}{(1-e^{j\omega})^2} - \frac{1-e^{-j(M+1)\omega}}{(1-e^{j\omega})^2}$$

$$X(j\omega)W(j\omega) = \frac{1}{1-\alpha e^{-j\omega M}}$$

5.6. Ideal Filters

5.6.1. The ideal lowpass filter frequency response



5.6.2. Ideal lowpass filter impulse response

- Lets low frequencies go through
- Attenuates i.e. kills high frequencies

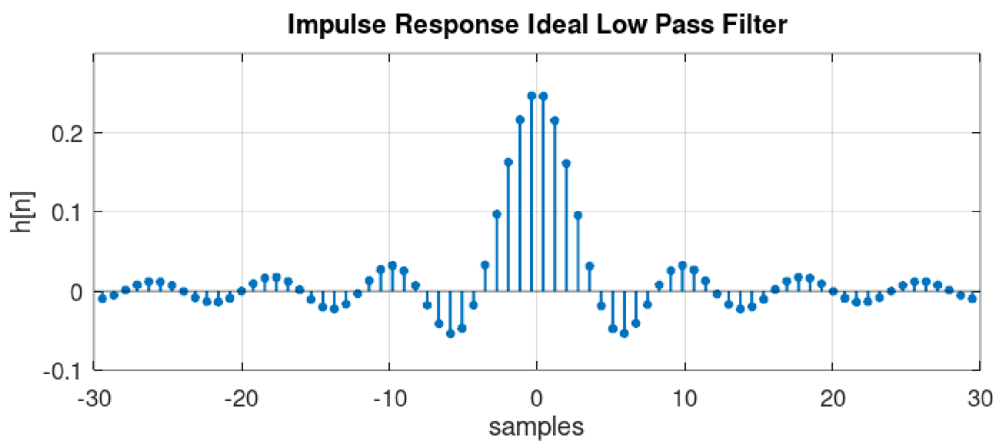
Cut off Frequency	ω_c - the frequency response transitions from 1 to zero
Passband	$\omega_b = 2\omega_c$

$$H(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega| \leq \omega_c \\ 0 & \text{otherwise} \end{cases}$$

- perfectly flat passband
- infinite attenuation in stopband
- zero-phase (no delay)

Calculation of the impulse response from the frequency response of an ideal low pass filter. Impulse Respones

$$\begin{aligned} h[n] &= IDFT\{H(e^{j\omega})\} \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} H(e^{j\omega})e^{j\omega\ n}d\omega \\ &= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} e^{j\omega\ n}d\omega \\ &= \frac{1}{\pi\ n}\frac{e^{j\omega_c\ n} - e^{-j\omega_c\ n}}{2j} \\ &= \frac{sin(\omega_c\ n)}{\pi\ n} \end{aligned}$$



- from Mathworks
- The impulse response has infinite support to the right and to the left
- Independant of how the convolution is computed, it will always take an inifintie number of operations.
- The impulse response decays slowly in time $\left(\frac{1}{n}\right)$, we need a lot of samples for a good approximation.

5.6.2.1. Impulse Response: From normlized Algorithm to Octave Implementation

$$\begin{aligned}\frac{sin(\omega_c\, n)}{\pi\, n} &= \frac{\omega_c}{\pi} . \cdot sinc(n\frac{\omega_c}{\pi});\\ &= \frac{\frac{\pi}{c}}{\pi} . \cdot sinc(n\frac{\frac{\pi}{c}}{\pi});\\ &= \frac{1}{c} . \cdot sinc(n\frac{1}{c});\\ &= \frac{1}{c} . \cdot sinc(\frac{n}{c});\end{aligned}$$

5.6.2.2. The sinc-rect pair:

$$rect(x) = \begin{cases} 1 & |x| \le \frac{1}{2} \\ 0 & |x| > \frac{1}{2} \end{cases}$$

$$sinc(x) = \begin{cases} \frac{sin(\pi\, x)}{\pi\, x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

- rect is the indicator function from $-\frac{1}{2}$ to $\frac{1}{2}$

5.6.2.3. Canonical form of the ideal low pass filter

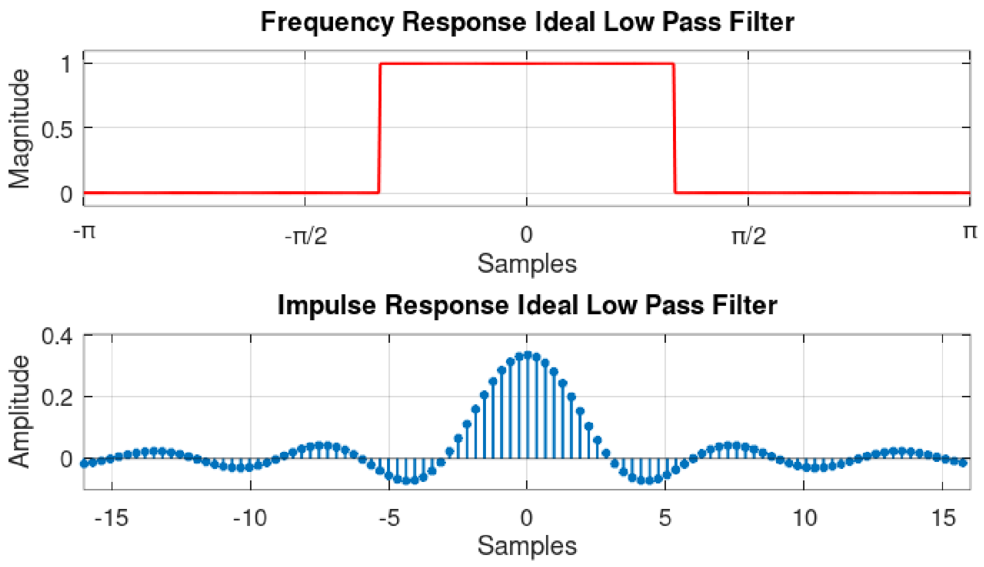
The sinct-rect pair can be written in canonical form as follow: \$~\$

$$\boxed{H(e^{j\,\omega}) = rect\left(\frac{\omega}{2\,\omega_c}\right)} \quad \underleftrightarrow{DTFT} \quad \boxed{\frac{\omega_c}{\pi}\, sinc\left(\frac{\omega_c}{\pi}\, n\right) = h[n]}$$

- The Impulse response is normalized by $\frac{\omega_c}{\pi}$

5.6.3. Example

Calculation of the impulse- and frequency response for an ideal low pass filter with $\omega_c \frac{\pi}{3}$

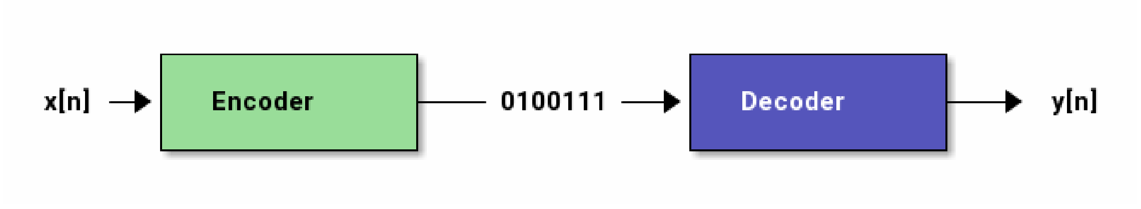


5.6.4. TODO Ideal filters derived from the ideal low pass filter

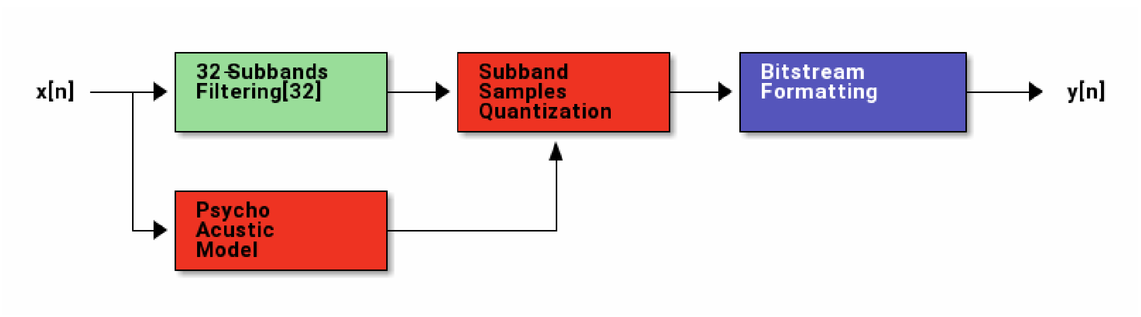
5.6.5. TODO Demodulation revisted

5.7. MP3 Encoder

- **Goal:** Reduce number of bits to represent original signal x[n]
- MP3: Motion Picture Expert G3roup



- **Lossy Compression:** $x[n] \neq y[n]$
- Put noise where not perceptible by human ear
- **Example:** Raw Storage Consumption DVD
  - Sample Rate: 48kHz
  - Bits per Sample: 16
  - Bit Rate: $\frac{48000\text{samples}}{second} \frac{16bits}{samples} = 768kbits/s$
  - Duration: 60s
  - Mono Raw Data Storage Usage: $60s \times 76.8kbits/s = 46Mbit = 5.8MByte$
  - Stereo Raw Data Storage Usage: $2 \times 5.8MBytes = 12MBytes$
  - MP3 Compressed Storage Usage: $1.5MBytes$



- Clever Quantiziation Scheme: Number of bits allocated to each subband is dependent on the perceptual importance of each sub-band with respect to overall quality of the audio wave-form
- Masking Effect of the human auditory system.

5.7.1. Psycho Acoustic Model, How it Works

- The psycho acoustic model is not part of the mp3 standard
- calculate the minimum number of bits that we need to quantize each of the 32 subband filter outputs, so that the perceptual distortion is as little as possible

step 1	Use FFT to estimate the energy of the signal in each subband
step 2	Distinguish beween tonal (sinusoid like) and non-tonal (nois-like) compnent
step 3	Determine indicidual masking effect of tonal and non-tonal component in each critical band
step 4	Determine the total masking effect by summing the individual contirbution
step 5	Map this total effect to the 32 subbands
step 6	Determine bit allොction by allocating priority bits to subbands with lowest singal-to-mask ratio

5.7.2. Subband Filter

$$h_i[n] = h[n]cos\left(\frac{pi}{64}(2i+1)(n-16)\right)$$

5.8. Programing Assignment 1

```python
import matplotlib
import numpy as np
matplotlib.use('Agg')
import matplotlib.pyplot as plt


def scaled_fft_db(x):
     """ ASSIGNMENT 1:
         Module 4 Part 1:
         Apply a hanning window to len(x[n]) = 512
     """

     N = len(x)                 # number of samples
     n = np.arange(N)           # time vector
     # a) Compute a 512-point Hann window and use it to weigh the input data.
     sine_sqr = np.sin((np.pi*n)/(N-1))**2     # sin(x)^2 = 1/2*(1 - cos(2x))
     c = np.sqrt(511/np.sum(sine_sqr))
```

```
    w = c/2 * (1 - np.cos((2 * np.pi * n)/(N - 1)))
    # b) Compute the DFT of the weighed input, take the magnitude in dBs and
    #    normalize so that the maximum value is 96dB.
    y = w * x
    Y = np.fft.fft(y) / N
    # c) Return the first 257 values of the normalized spectrum
    Y = Y[0: np.int(N/2+1)]
    # Take the magnitude of X
    Y_mag = np.abs(Y)
    nonzero_magY = np.where(Y_mag != 0)[0]

    # Convert the magnitudes to dB
    Y_db = -100 * np.ones_like(Y_mag)    # Set the default dB to -100
    Y_db[nonzero_magY] = 20*np.log10(Y_mag[nonzero_magY])  # Compute the dB for nonzero magnitude ind

    # Rescale to amx of 96 dB
    max_db = np.amax(Y_db)
    Y_db = 96 - max_db + Y_db

    return Y_db

def test():
    N = 512
    n = np.arange(N)
    x = np.cos(2*np.pi*n/10)

    # Y = scaled_fft_db(x)
    Y = scaled_fft_db(x)

    fig=plt.figure(figsize=(6,3))
    plt.semilogy(abs(Y))
    plt.grid(True)

    fig.tight_layout()
    plt.savefig('image/python-matplot-fig-04.png')
    return 'image/python-matplot-fig-04.png' # return filename to org-mode

return test()
```

Part VI.

Week 6 Module 4 Part 2:

6. Introduction to Filtering

- First strategy of filter design: Imitation
  - uuImpulse truncation
  - Window Method
  - Frequency Sampling

  Trying to replicate the structure of either the impulse response or the frequency response of ideal filters.

6.1. Filter Design Part 1 (FIR Filter)

- An ideal filter is not realizable in practice because the impulse response is a two-sided infinite support sequence.

6.1.1. Reference

- The Scientist and Engineers Guide to DSP: Recurscive Filter

6.1.2. Impulse truncation

Impulse Truncation

1. Pick ω_c
2. Compute ideal impulse response h[n] (analytically)
3. truncate h[n] to a finite-support $\hat{h}[n]$
4. $\hat{h}[n]$ defines an FIR filter

FIR approximation of lenght M = 2N+1

$$\hat{h}[n] = \begin{cases} \frac{\omega_c}{\pi}\, sinc(\frac{\omega_c}{\pi} n) & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

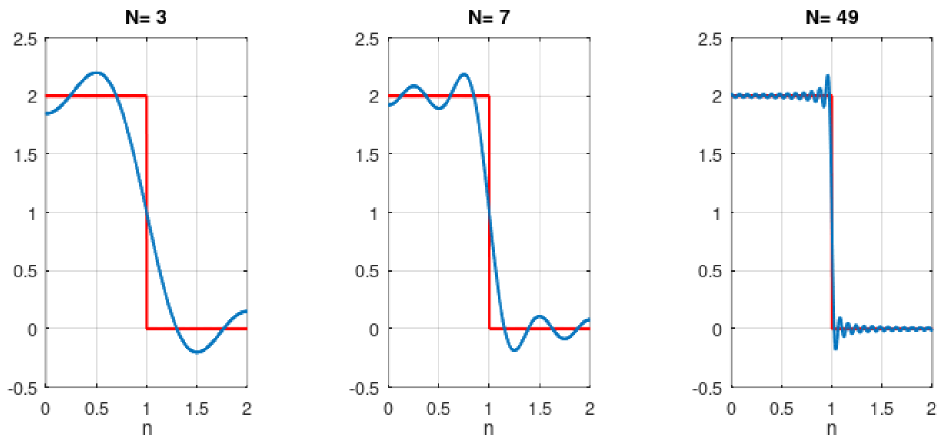
- **Why approximation by truncation could be a good idea** A justification of this method is the computation of the mean square error:

$$\begin{aligned} MSE &= \frac{1}{2\pi}\int_{-\pi}^{\pi} |H(e^{j\omega}) - \hat{H}(e^{j\omega})|^2 d\omega \\ &= ||H(e^{j\omega}) - \hat{H}(e^{j\omega})||^2 \\ &= ||h[n] - \hat{h}[n]||^2 \\ &= \sum_{n=-\infty}^{\infty} |h[n] - \hat{h}[n]|^2 \end{aligned}$$

The means square error MSE is minimized by symmetric impulse truncation around zero

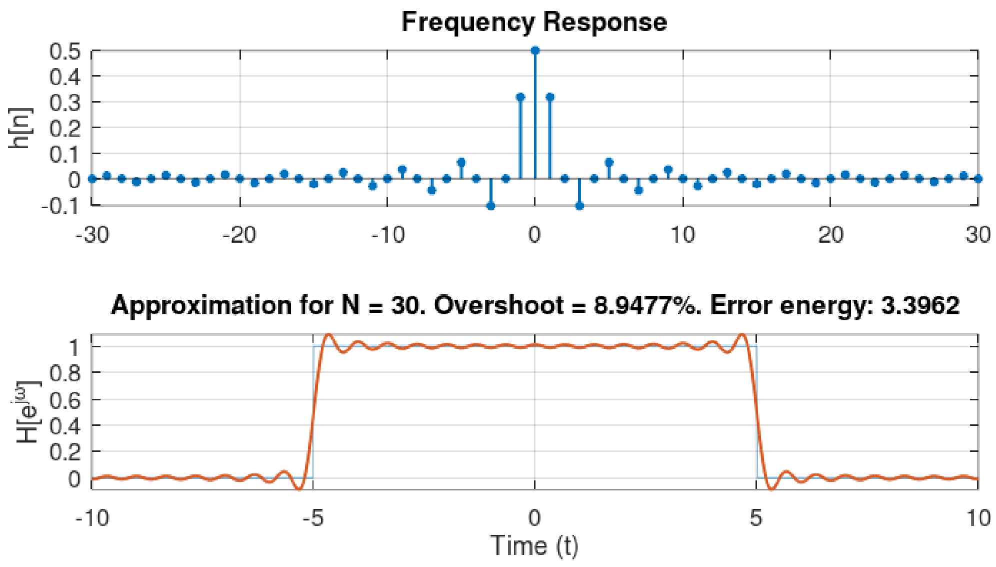
- **Why approximation by truncation is not such a good idea** The maximum error around the cutoff frequency is around 9% of the height of the jump regardless of N. This is known as the Gibbs Phenomenon.

6.1.2.1. The Gibbs Phenomenon



References:

- Matlab Answers



6.1.3. Window method

The impulse truncation can be interpreted as the product of the ideal filter response and a rectangular window of N points.
From the modulation theorem, the DTFT of the product f two signals is equivalent to the convolution of their DTFTs. Hence, the choice of window influences the quality of the approximation results.

Window Method

The window method is just a generalization of the impulse truncation method where we use a different window shape.

For example, by using a triangular window, we reduce the Gibbs error at the price of a longer transition.

6.1.3.1. The modulation theorem revisited.

We can consider the approximated filter as

$$\hat{h}[n] = h[n]\, w[n]$$

with the indicator function w[n]

$$w[n] = \begin{cases} 1 & |n| \leq N \\ 0 & \text{otherwise} \end{cases}$$

The question is how can we express the Fourier Transform $\hat{H}(e^{j\omega})$ =? of the filter as the product of two sequences? For that, we have to study the modulation theorem.

- Convolution Theorem states that the Fourier Transform of the convolution of two sequences is the product in the frequency domain of the Fourier Transforms.

$$\boxed{DTFT\{(x * y)[n]\} = X(e^{j\omega})\, Y(e^{j\omega})}$$

- Modulation Theorem The modulation theorem states that the Fourier Transform of the product of two sequences is the convolution in the frequency domain of the Fourier Transform

$$\boxed{DTFT\{(x[n]\ y)[n]\} = (X * Y)(e^{j\omega})}$$

- Convolution in the Frequency Domain

in $\mathbb{C}^{\infty}$ the space of infinite support signals, the convolution can be defined in terms of the inner product of the two sequences.

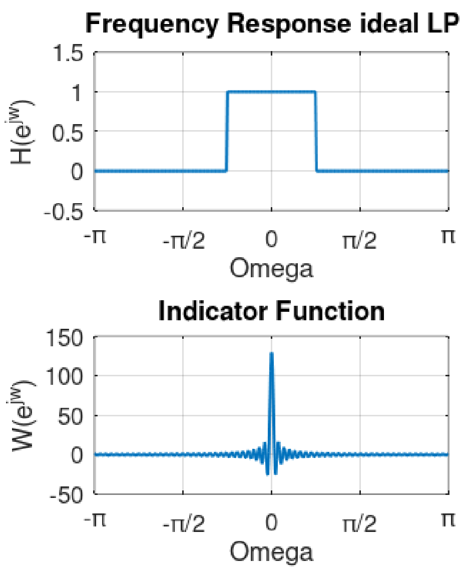
$$\begin{aligned}(x * y)[n] &= \big\langle x^*[k], y[n-k]\big\rangle \\ &= \sum_{n=-\infty}^{\infty} x[k]y[n-k]\end{aligned}$$

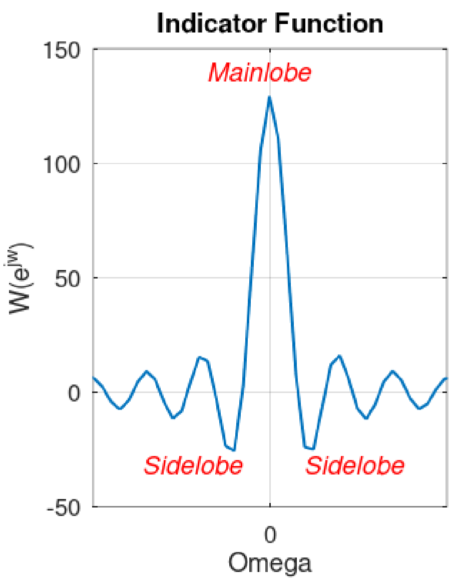
We can adapt the same strategie in $\mathbb{L}\Big(\big[-\pi, \pi\big]\Big)$, which is the space where the DTFT life's. So we find the convolution of two Fourier Transforms as the inner product of the first Fourier Transform conjugated and the second Fourier Transform frequence reversed and delayed by ω

$$\begin{aligned}(X * Y)(e^{j\,\omega}) &= \big\langle X^*(e^{j\,\sigma}), Y(e^{j\,\omega-\sigma})\big\rangle \\ &= \frac{1}{2\pi}\int_{-\pi}^{pi} X^*(e^{j\,\sigma})\, Y(e^{j\,\omega-\sigma})\, d\sigma\end{aligned}$$

If we apply the definition of the inner product for $L2([-\pi, \pi])$ we get that the convolution between two Fourier Transforms.

6.1.3.2. Mainlobe and Sidelobes





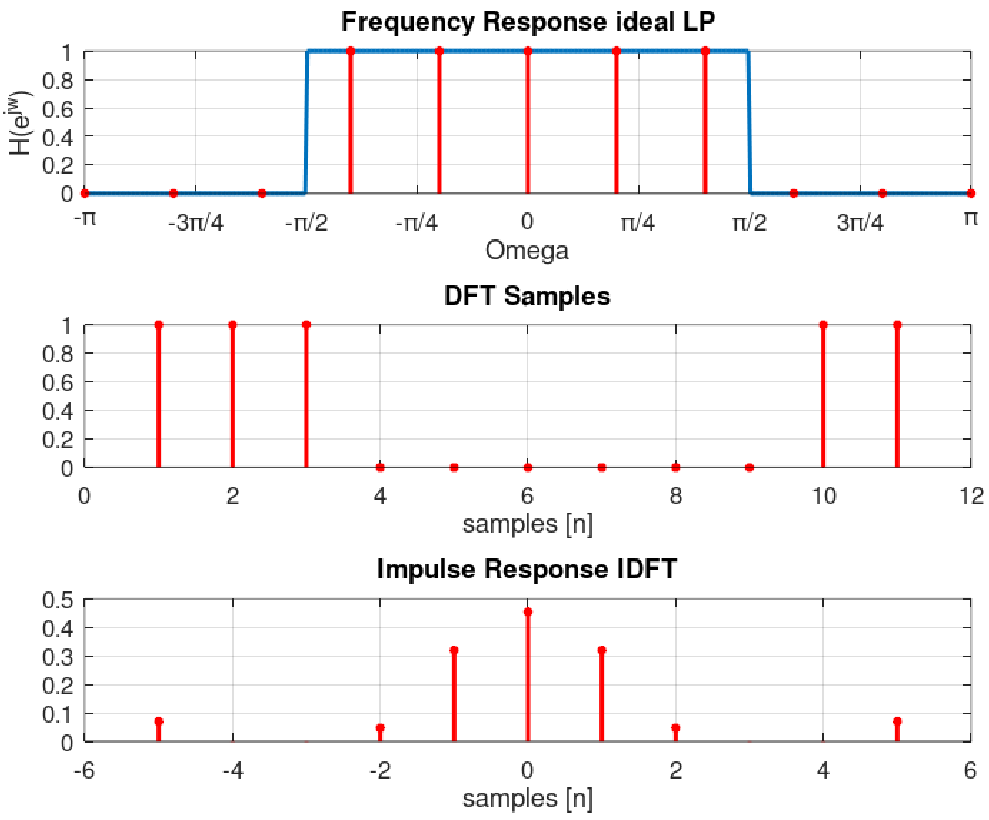
We want:

- narrow mainlobe $\Rightarrow$ to have sharp transition
- small sidelobe $\Rightarrow$ gibbs error is small
- short window $\Rightarrow$ FIR is efficient

6.1.4. Frequency sampling

Frequency Sampling

1. Draw desired frequency response $H(e^{j\omega})$
2. take M values at $\omega_k = \frac{2\pi}{M} \cdot k$
3. compute IDFT of values
4. use result as M-tap impulse response $\hat{h}[n]$



- **Why Frequency Sampling is not such a good idea:**
  - frequency response is DTFT of finite-support, whose DFT we know
  - frequency response is interpolation of frequency samples
  - interpolator is transform N-tap rectangular window (no escape from the indicator function)
  - again no control over main- and sidelobe

Summery Imitation

These methods to approximate ideal filters are certainly very useful when we want to derive a quick and dirty prototype, and we don't have time to use more sophisticated filter design methods

6.2. Signal of the Day: Camera Resolution and space exploration

6.2.1. Rosettta Mission: Spacecraft

- Reaching Comet 67P. 10 years to get momentum to get its orbit.
- Resolution of taken pictures:

Resolution	at Distance	Year	
1km/pixel	86'000km	28. June 2014	
	12'000km	14. July 2014	
100m/pixel	5'500km	20. July 2014	
5.3m/pixel	285km	3. August 2014	
11cm/pixel	6km	14. February 2015	most detailed pictures of a planet

Is it necessary to send a probe for 10years into space to get high resolution pictures?

6.2.2. Image Formation

$$\begin{aligned} i(x,y) &= s(x,y) * h(x,y), \text{ i: image that is formed,} \\ &= s(x,y) * t(x,y) * p(x,y) \end{aligned}$$

- i: image that is formed on the retina or camera
- s: light sources (source image)
- h: transfer function of the light
- t: medium through the light is traveling
- p: point spread function (PSF), lenses and focal distance

The major enemy to image quality of telescope on earth are the atmospheric disturbances.

The pinhole camera A certain pixel density is required to distinguish light sources on the image plane. We might be tempted to say the maximum achievable resolution is only depend on the **resolution** of the sensor at the back of the camera. In reality the resolution is limited by pixel density resolution is limited by diffraction.

Diffraction (Beugung) The image of an original point light source will appear as a diffraction pattern. The diffraction pattern through a small circular aperture is called **Airy disk**.

Rayleigh's criterion Minimum angle θ between light point sources that guarantees resolution

$$\theta = 1.22\frac{\lambda}{D}$$

- λ : wave length of the light that hits the camera
- D : Diameter of the aperture

6.2.3. Seeing the Lunar Excursion Module (LEM)

- size of LEM $\approx$ 5m
- distance to the Moon $\approx$
- $Rightarrow\ \theta$ subtended by the LEM is $\approx 0.003 arcsec$
- Hubble's aperture: 2.4m
- visible spectrum $\lambda \approx 550nm$
- Rayleigh's criterion: $\theta \approx 0.1 arcsec$
  $\Rightarrow$ to see the LEM, Hubble should have an aperture of 80m!!!!

6.2.4. Rayleigh's criterion, Spatial Resolution

$$\delta x = 1.22\, f\, \frac{f}{D} = \theta \cdot f$$

If the pixel separation on the camera sensor is not less than δx our camera will be resolution limited rather than diffraction limited.

- f: foco length
- f/D: f-number

pixel density takes into account the size of the sensor.

6.2.5. What about mega pixels?

How many mega pixels one need on an commercial camera. This actually depends on the size of the sensor and on the optics:

- f-number of all trades: f/8
- spatial Rayleigh's criterion: $\delta x \approx 4\mu m$
- max pixel area $16 \cdot 10^{-5}$
  $\Rightarrow$ to opperate at the diffraction limit we need $62'500 pixels/mm^2$
  Highend camera usually have one of the following sensors:
- APS-C sensor (329mm^2): 20 MP $\Rightarrow$ the camera is operating at the defraction limit
- 35-mm sensor (864mm^2): 54 MP $\Rightarrow$ the camera is operating at the defraction limit

6.3. Realizable Filters

6.3.1. The Z-Transform

6.3.1.1. References

. Signals and Systems for Dummies: Z-Transform

6.3.1.2. Z-Transform

maps a discrete-time sequence x[n] onto a function of $\sum_{n=-\infty}^{\infty} x[n]\, z^{-n}$.

$$\boxed{x[n] = \sum_{n=-\infty}^{\infty} x[n]\, z^{-n} \qquad (6.1)}$$

The z-Transform is an extension of the DTFT to the whole complex plane and is equal to the DTFT for $z = e^{j\omega}$.

$$X(z)|_{z=e^{j\omega}} = DTFT\{x[n]\} \qquad (6.2)$$

Key properties of the z-Transform are:

- linearity: $\mathcal{Z}\{\alpha x[n] + \beta y[n]\} = \alpha X(z) + \beta Y(z)$
- time shift: $\mathcal{Z}\{x[n-N]\} = z^{-N} X(z)$

Applying the z-transform to CCDE's

$$\boxed{\begin{gathered}\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k] \\ Y(z) \sum_{k=0}^{N-1} a_k z^{-k} = X(z) \sum_{k=0}^{M-1} b_k z^{-k} \\ Y(z) = H(z) X(z)\end{gathered}}$$

- **M input values**
- **N output values**

6.3.1.3. Constant Difference Equation

A constant coefficent difference equation (CCDE) expresses the input-, output relationship of an LTI system as a linear combination of output samples equal to a linear combination of input samples

$$\boxed{\sum_{k=0}^{N-1} a_k y[n-k]} = \boxed{\sum_{k=0}^{M-1} b_k x[n-k]}$$

In the z-domain, a Constant Coefficent Difference Equation CCDE is represented as a ration $H(z)$ of two polynomials of z^{-1}.

$$\boxed{H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}} \qquad (6.3)}$$

6.3.1.4. Frequency Response

The frequency response of a filter is equal to this transfer function evaluated at $z =^{j\omega}$.

$$H(j\omega) = H(z)|_{Z=e^{j\omega}} = \frac{\sum\limits_{k=0}^{M-1} b_k z^{-k}}{\sum\limits_{k=0}^{N-1} a_k z^{-k}} \qquad (6.4)$$

6.3.2. Z-Transform of the leaky integrator

$$\begin{aligned} y[n] &= (1-\lambda)x[n] + \lambda y[n-1] \\ Y(z) &= (1-\lambda)X(z) + \lambda z^{-1}Y(z) \\ Y(z) - \lambda z^{-1}Y(z) &= (1-\lambda)X(z) \\ Y(z)\left(1-\lambda z^{-1}\right) &= (1-\lambda)X(z) \\ Y(z) &= H(z)X(z) \\ H(z) &= \frac{Y(z)}{X(z)} = \frac{1-\lambda}{1-\lambda z^{-1}} \\ H(e^{j\omega}) &= \frac{1-\lambda}{1-\lambda e^{-j\omega}} \end{aligned}$$

6.3.2.1. LTI Systems

An LTI system can be represented as the convolution $y[n] = x[n] * h[n]$. From the convolution property of the Z-transform, it follows that the z-transform of y[n] is:

$$Y(z) = H(z)\, X(z) \qquad (6.5)$$

6.3.3. Region of convergence

Conditions for convergences

- The zeros/poles are the roots of the numerator/denominator of the rational transfer function
- the region of convergence is only determined by the magnitude of the poles
- the z-transform of a causal LTI system extends outwards from the largest magnitude pole

BIBO-Stable

An LTI system is stable if its region of convergence includes the unit circle

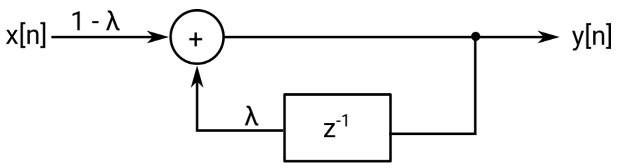
6.4. Filter Design Part 2

- many signal processing problems can be solved using simple filters
- we have seen simple lowpass filters already (Moving Average, Leaky Integrator)
- simplel (low order) transfer functions allow for intuitive design and tuning

6.4.1. Intuitive IIR Designs

6.4.1.1. Leaky Integrator

- Filter Structure



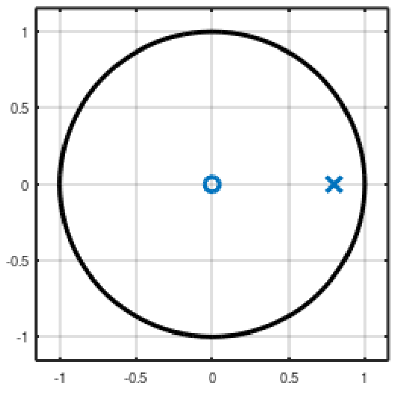
- Transfer Function

$$H(z) = \frac{1-\lambda}{1-\lambda z^{-1}}$$

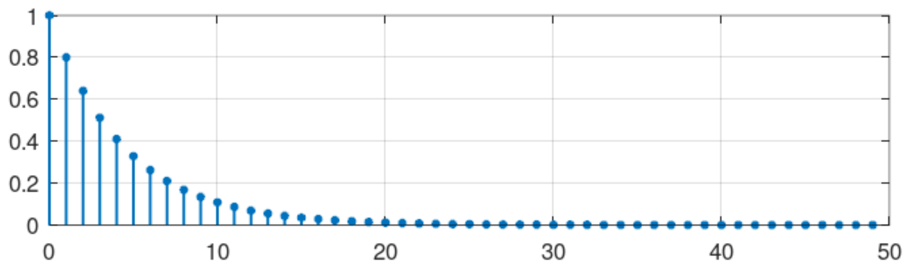
- CCDE

$$y[n] = (1-\lambda)\,x[n] + \lambda\,y[n-1]$$

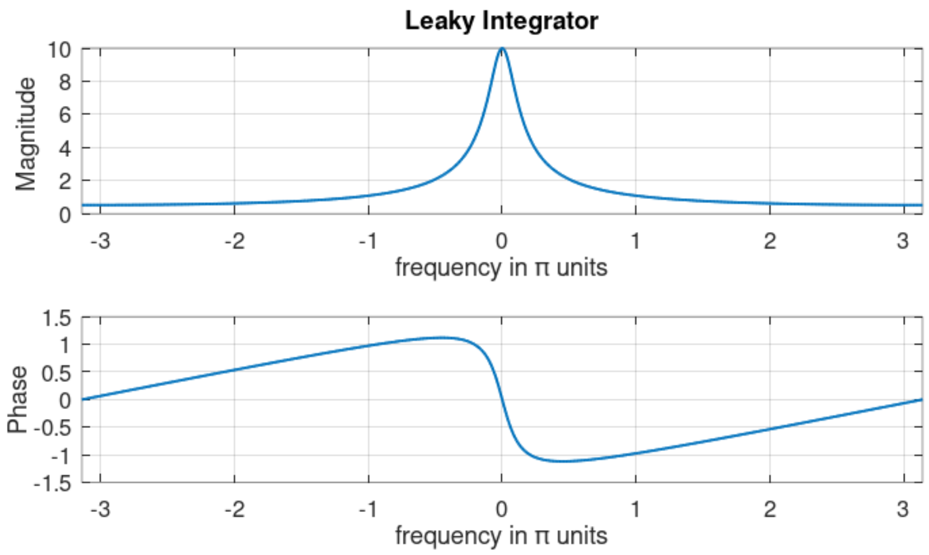
- Pole-Zero Plot



- Impulse response



- Frequency Response



6.4.1.2. Resonator

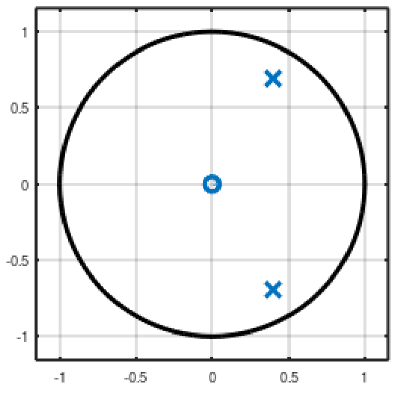
- a resonator is a narrow bandbass filter
- used to detect presence of a given frequency
- useful in communication systems and telephone (DTMF)
- Idea: shift passband of the Leaky Integrator
- Transfer Function

$$
\begin{aligned}
H(z) &= \frac{G_0}{(1-pz^{-1})(1-p^*z^{-1})} \\
p &= \lambda e^{j\omega_0} \\
H(z) &= \frac{G_0}{1-2\mathcal{R}pz^{-1}+|p|^2z^{-2}} \\
H(z) &= \frac{G_0}{1-2\lambda\omega_0 z^{-1}+|\lambda|^2z^{-2}}
\end{aligned}
$$

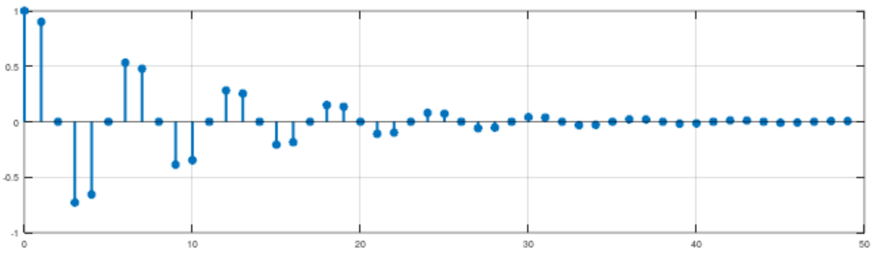
The coeffience to be used in the CCDE

$$
\begin{aligned}
a_1 &= 2\lambda cos\omega_0 \\
a_2 &= -|\lambda|^2
\end{aligned}
$$

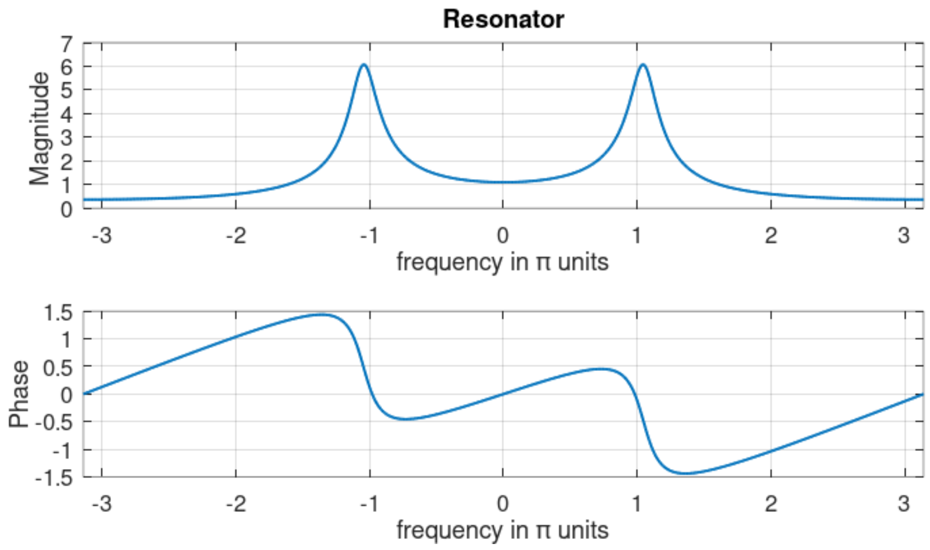
- Pole-Zero Plot
  - Move the pole of the leaky integrator radially around the circle of radius lambda to shift the passband at the frequency that we are interested in, i.e. ω_0. interested in selecting. Since we want a real filter, we also have to create a complex conjugate pole at an angle that is $-\omega_0$.



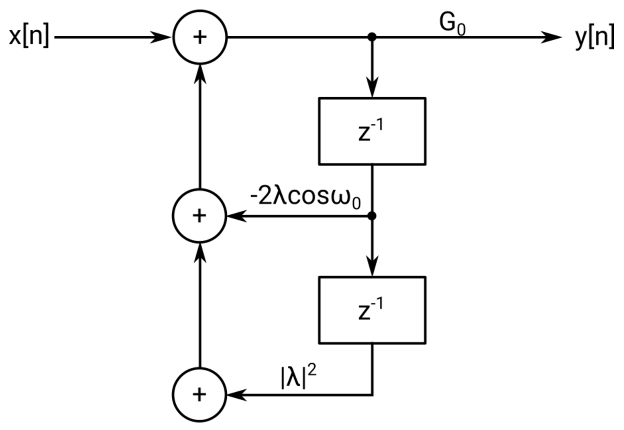
- Impulse response



- Frequency Response



- Filter Structure



6.4.1.3. DC Removal

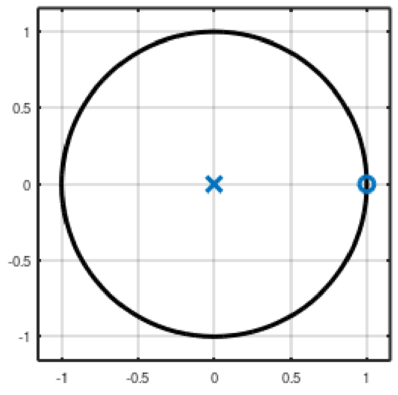
- a DC-balances signal has zero sum: $\lim_{N\to\infty} \sum_{n=-N}^{N} x[n] = 0$ i.e. there is no Direct Current component
- its DTFT value at zero is zero for an $\omega = 0$
- we want to remove the DC bias from a non zero-centered signal
- we want to kill the frequency component at $\omega = 0$
- Transfer Function

$$H(z) = 1 - Z^{-1}$$

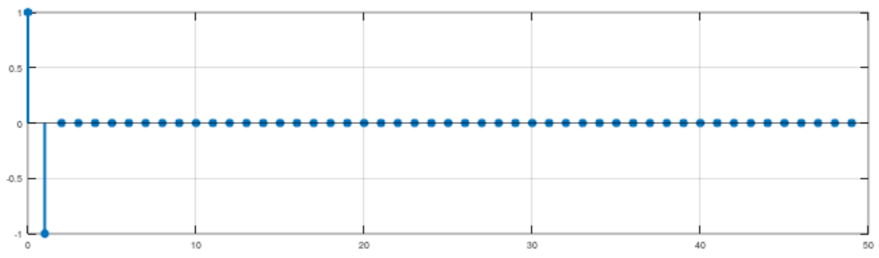
- CCD

$$y[n] = x[n] - x[n-1]$$

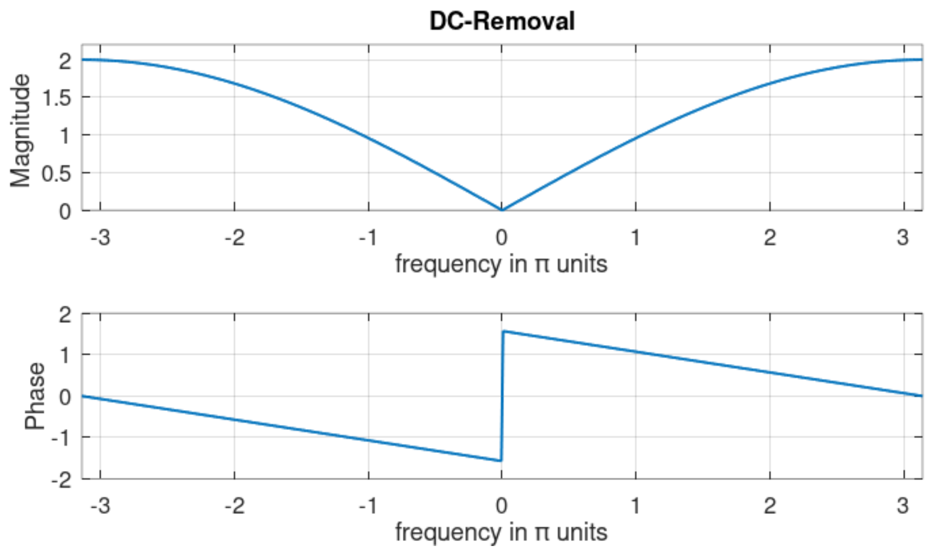
- Pole-Zero Plot
  - Simply place a zero at $ z = 1$



- Impulse response



- Frequency response



This is not an acceptable characteristic because it introduces a very big attenuation over almost the entety of the frequency support.

6.4.1.4. DC Removal Improved - DC-Notch Filter

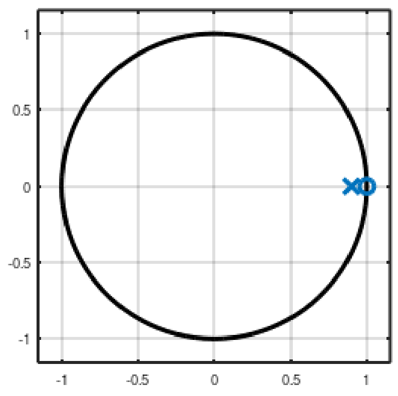
- Transfer Function

$$H(z) = \frac{1 - z^{-1}}{1 - \lambda z^{-1}}$$

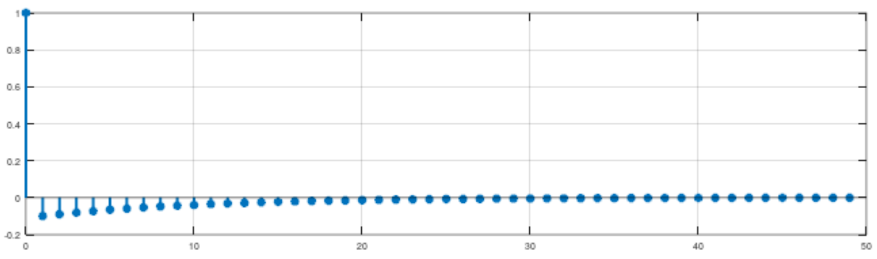
- CCDE

$$y[n] = \lambda y[n-1] + x[n] - x[n-1]$$

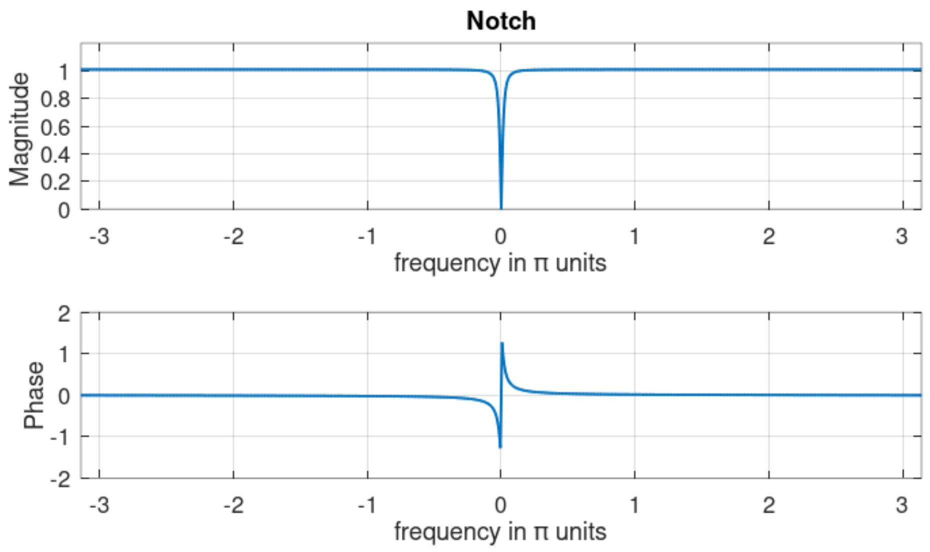
- Pole-Zero Plot
  - and if we remember the circus tent method, we know that we can push up the z-transform by putting a pole in the vicinity of the 0. So we try and do that and we combine therefore, the effect of a 0 and 1 with the effect of a pole close to one, and inside the unit circle, for obvious reasons of stability.



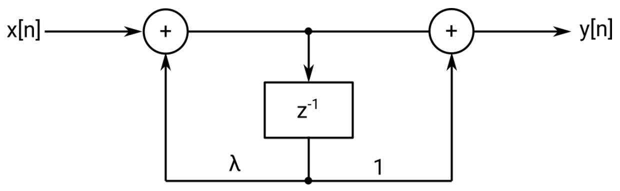
- Impulse response



- Frequency Response



- Filter Structure



6.4.1.5. Hum Removal

- The hum removal filter is to the dc notch what the resonator is to the leaky integrator
- similar to DC removal but want to remove a specific nonzero frequency
- very usful for musicaians amplifiers for electronic guitars pick up the hum from the electronic mains (50Hz in Europe and 60Hz in North America)
- we need to tune the hum removal according the country

- Transfer Function

$$
\begin{aligned}
H(z) &= \frac{(1-e^{j\omega_0}z^{-1})(1-e^{-j\omega_0}z^{-1})}{(1-\lambda e^{j\omega_0}z^{-1})(1-\lambda e^{-j\omega_0}z^{-1})} \\
p &= e^{j\omega_0} \\
q &= \lambda e^{j\omega_0} \\
&= \frac{(1-pz^{-1})(1-p*z^{-1})}{(1-qz^{-1})(1-q*z^{-1})} \\
H(z) &= \frac{1-2\mathcal{R}pz^{-1}+|p|^2z^{-2}}{1-2\mathcal{R}qz^{-1}+|q|^2z^{-2}} \\
&= \frac{1-2\omega_0 z^{-1}+z^{-2}}{1-2\lambda\omega_0 z^{-1}+|\lambda|^2z^{-2}}
\end{aligned}
$$

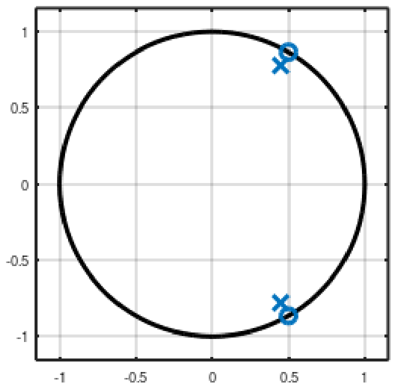
The coeffience to be used in the CCDE

$$
\begin{aligned}
a_1 &= -2\lambda cos\omega_0 \\
a_2 &= |\lambda|^2 \\
b_1 &= -2\omega_0 \\
b_2 &= 1
\end{aligned}
$$

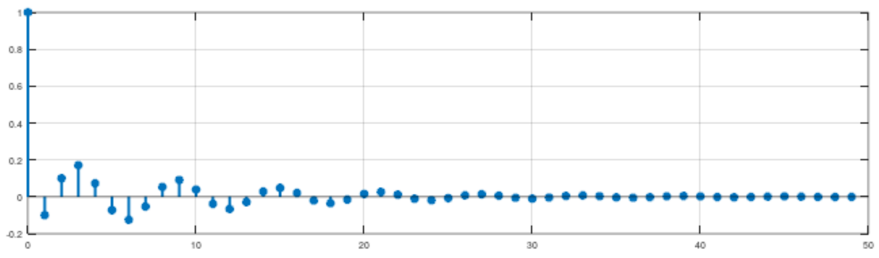
- CCDE

$$y[n] = 2\lambda\ cos\omega_0\ y[n-1] + |\lambda|^2\ y[n-2] + x[n] - 2\ cos\omega_0\ x[n-1] +\ x[n-2]$$

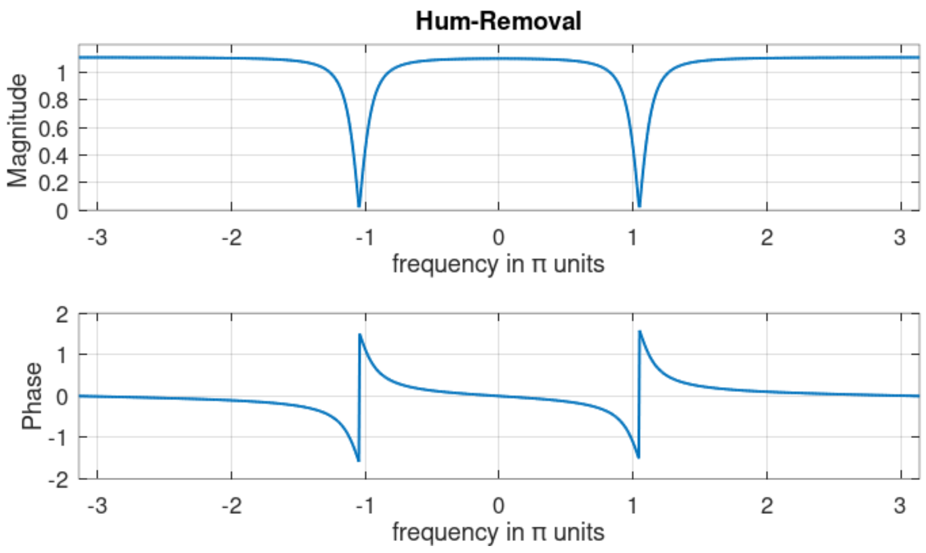
- Pole-Zero Plot
  - and if we remember the circus tent method, we know that we can push up the z-transform by putting a pole in the vicinity of the 0. So we try and do that and we combine therefore, the effect of a 0 and 1 with the effect of a pole close to one, and inside the unit circle, for obvious reasons of stability.



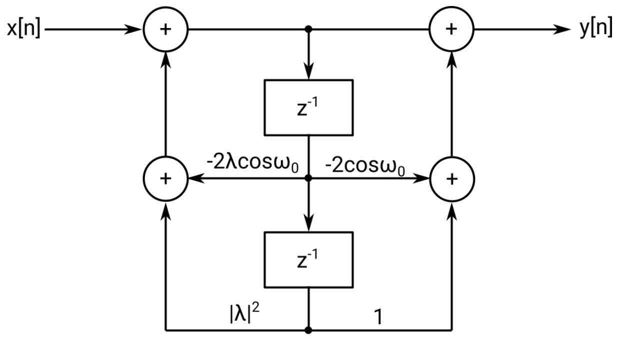
- Impulse response



- Frequency Response



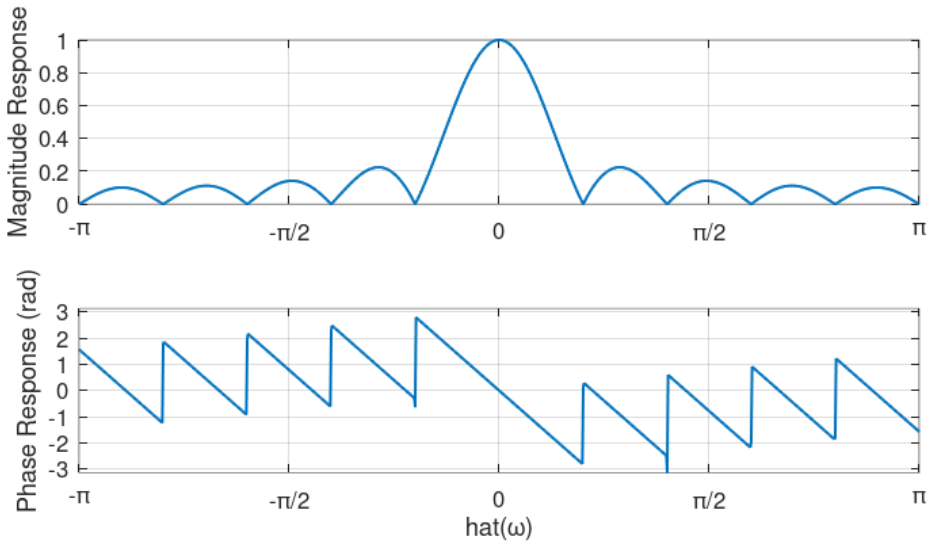
- Filter Structure



6.4.2. Matlab

Dirichlet The Dirichlet or periodic sync function can be used to analyze Moving Average Filters $D_M(j\omega) = diric(\omega, M) = \dfrac{sin(\frac{\omega}{2}M)}{sin(\frac{\omega}{2}M)}$

Freqz The frequency response can be plotted most easily using freqz() function.



6.5. Filter Design Part 3

6.5.1. Filter Specification

6.5.2. IIR Design

Filterdesign was established art long before digital processing appeared

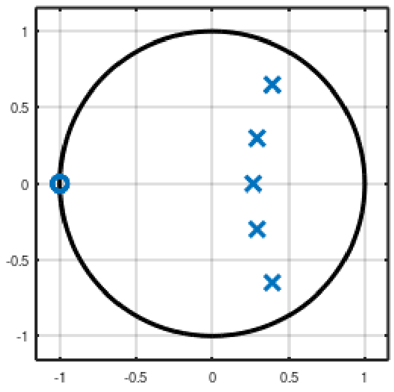
- AFD: Analog Filter Design
- lots of nice analog filters exist
- methods exist to ”translate” the analog design into a rational transfer function
  - **impulse invariance transformation**, preserves the shape of the impulse response
  - finite difference approximation, converts a differential equation into a ccde
  - step invariance, preserves the shape of the step response
  - matched-z transformation, matches the pole-zero representation
  - **bilinear transformation**, preserves the system function representation
- most numerical packages (Matlab, etc.) provide ready-made routines
- design involves specifying some parameters and testing that the specs are fulfilled

6.5.2.1. Butterworth lowpass

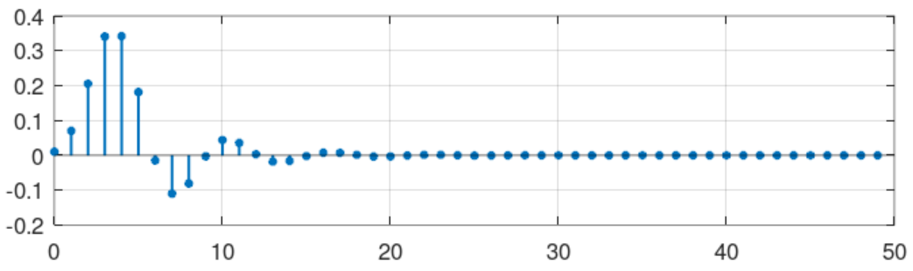
Table 6.1.: Butterworth lowpass

Magnitude response	**Design Parameters**	**Test values**
maximally flat	order N	width of transition band
monotonic over $[0, \pi]$	cutoff frequency	passband error

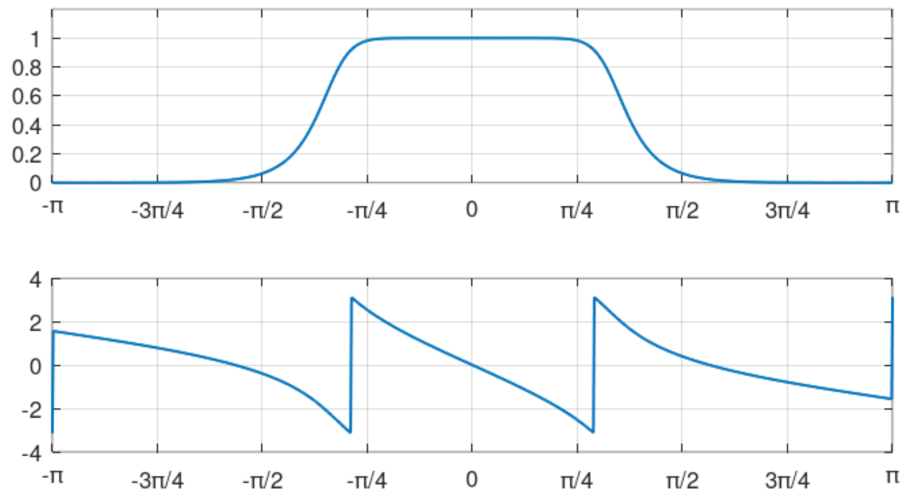
- Pole-Zero Plot



- Impulse Response



- Frequency Response

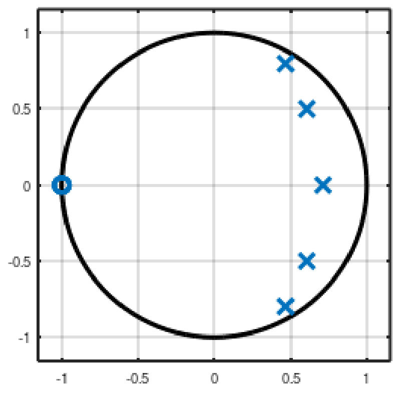


6.5.2.2. Chebyshev lowpass

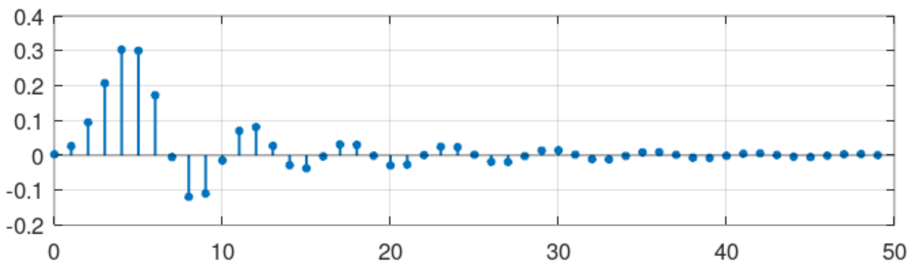
Table 6.2.: Chebyshev lowpass

Magnitude response	Design Parameters	Test values
equiripple in passband	order N	width of transition band
monotonic in stopband	passband max error	stopband error
	cutoff frequency	

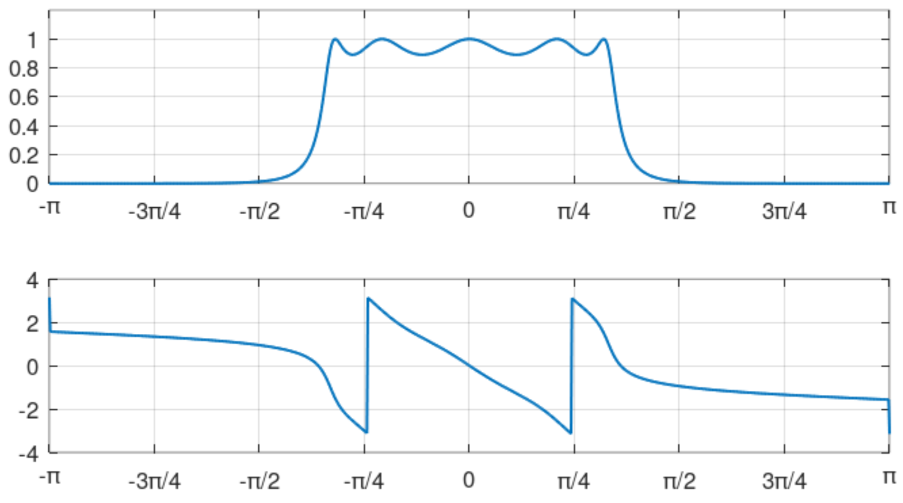
- Pole-Zero Plot



- Impulse Response



- Frequency Response

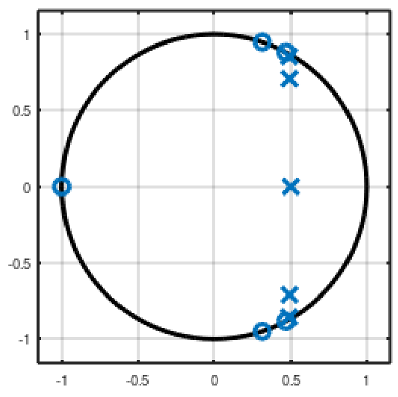


6.5.2.3. Elliptic Lowpass

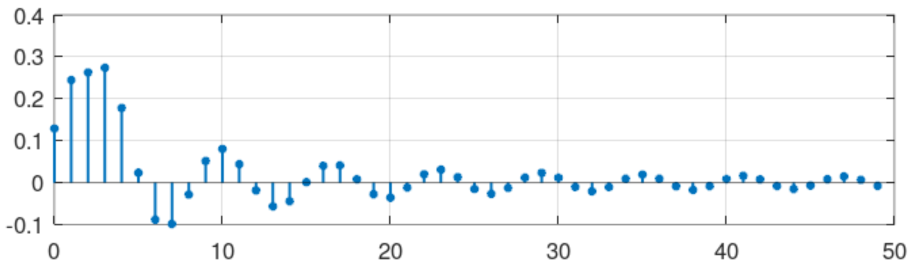
Table 6.3.: Elliptic Lowpass

Magnitude response	Design Parameters	Test values
equiripple in passband	order N	width of transition band
equiripple in stopband	cutoff frequnecy	
	passband max error	
	stopband min attenuation	

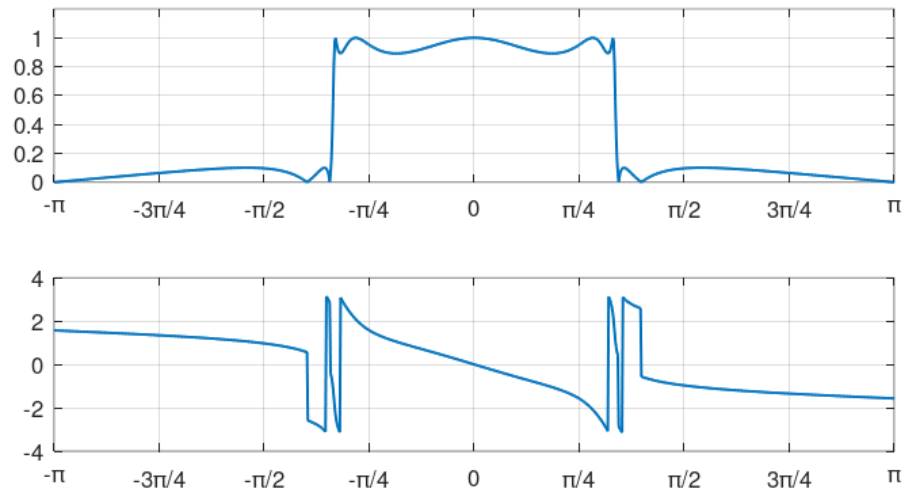
- Pole-Zero Plot



- Impulse Response



- Frequency Response



6.5.3. FIR Design

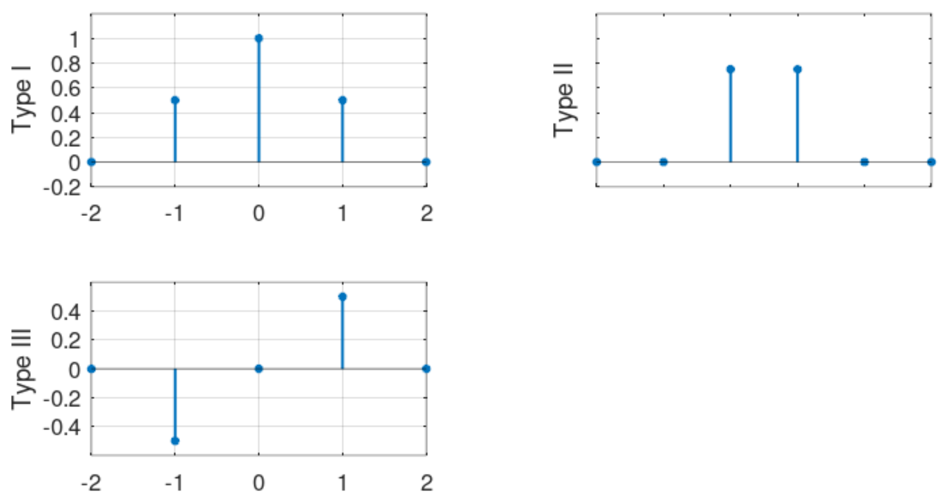
6.5.3.1. Optimal minmax design

FIR filters are `digital` signal processing ”exclusivity”. In the 70s Parks and McClellan developed an algorithm to design optimal FIR filters:

- linear phase
- equiripple error in passband and stopband

algorithm proceeds by **minimizing** the maximum error in passpand and stopband

- Linear Phase Linear phase derives from a symmetric or antisymmetric impulse respones



Type I-Filters Odd length impulse response, and are symmetric

Type II-Filters Even length impulse response, and are symmetric

Type III-Filters Odd length impulse response, and are antisymmetric

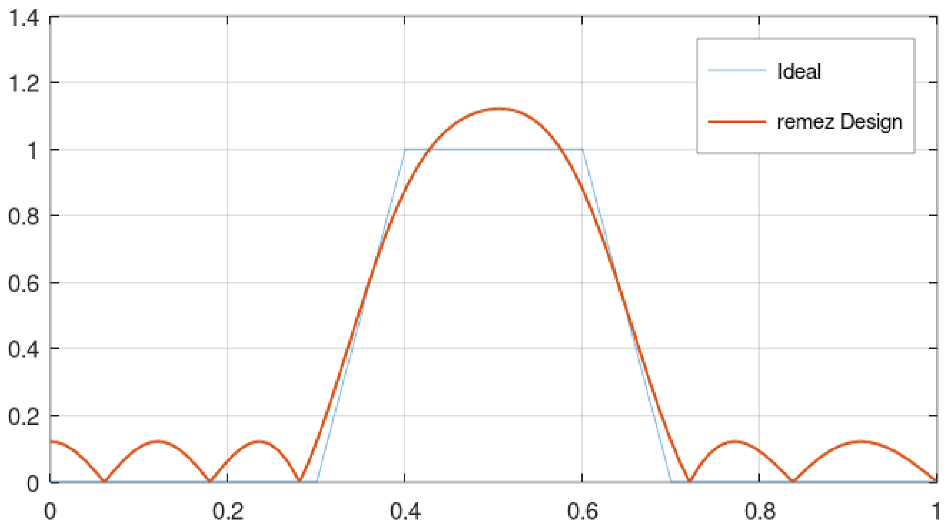
Type IV-Filters Even length impulse response, and are antisymmetric

Type-II and Type-IV Filters are symmetric and antisymmetric filters, respectively, both of which have an even number of taps. That means that the center symmetry of these filters fall in between samples. And so they both introduce a non integer linear phase factor, of one half sample.

6.5.4. The Park McMellon Design Algorithm

Table 6.4.: Park McMellon

Magnitude response	Design Parameters	Test values
equiripple in passband and stopband	order N	passband max error
	passband edge ω_p	stopband max error
	stopband edge ω_s	
	ratio of passband to stop-band error $\frac{\delta_p}{delta_s}$	



6.6. ONGOING Notes and Supplementary Materials

6.6.1. The Fractional Delay Filter (FDF)

$$\begin{tikzcd} x[n] \arrow[r] & \boxed{z^{-d}} \arrow[r] & y[n] \end{tikzcd}$$

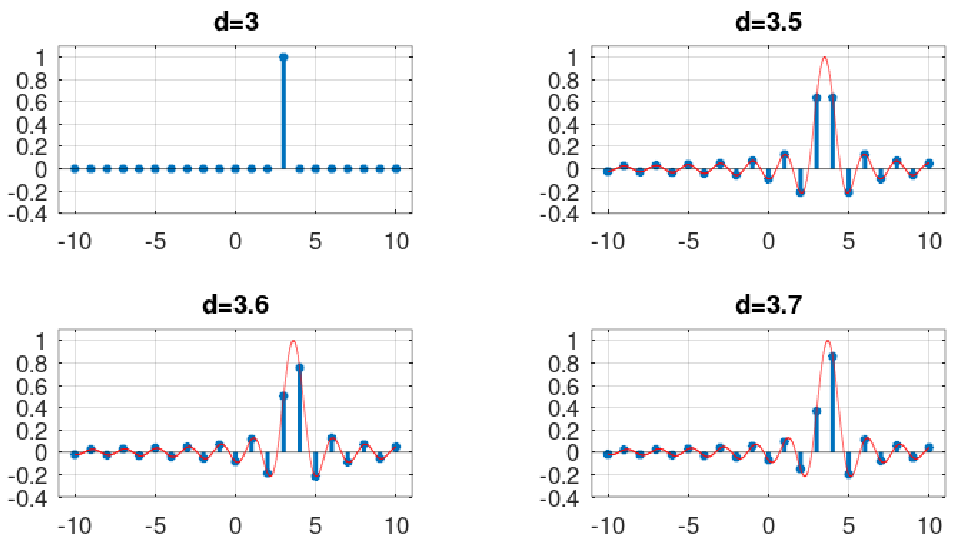
The transfer function of a simple delay z^{-d} is:

$$H(e^{j\omega}) = e^{-j\omega d},\, d \in \mathbb{Z}$$

what happens if, in $H(e^{j\omega}$ we use a non-integer $d \in \mathbb{R}$?

6.6.1.1. Impulse Response

$$\begin{aligned} h[n] &= IDFT\left\{e^{j\omega d}\right\} \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega d} e^{j\omega n} d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega(n-d)} d\omega \\ &= \frac{1}{\pi(n-d)} \frac{e^{j\pi(n-d)} - e^{-j\pi(n-d)}}{2j} \\ &= \frac{sin\pi(n-d)}{\pi(n-d)} \\ &= sinc(n-d) \end{aligned}$$



For now suffice it to say that we can actually interpolate in discrete time and find intermediate values of a discrete time sequence using just discrete times filters like the fractional delay

6.6.2. ONGOING The Hilbert Filter

- Demodulator

$$\begin{tikzcd}\cos(w_0 n) \arrow[r] & \boxed{H(jw)} \arrow[r] & \sin(w_0 n)\end{tikzcd}$$

can we build such a thing?

6.6.3. Implementing of Digital Filters

6.6.3.1. Leaky Integrator in C

```
using namespace std;

double leaky(double x) {
    static const double lambda = 0.9;
    static double y = 0;    // 1x memory cell
    // plus initialization
    // algorithm: 2x multiplication, 1x addition
    y = lambda * y + (1-lambda) *x;
    return y;
}

int main() {
    int n;
    for(n = 0; n <20; n++)
    {
        //call with delta signl
        printf("%.4f ", leaky(n==0 ? 1.0 : 0.0));
        //if(!((n+1)%10)) printf("\n");
    }
}
```

```
0.1000 0.0900 0.0810 0.0729 0.0656 0.0590 0.0531 0.0478 0.0430 0.0387 0.0349 0.0314 0.0282 0.0254 0.022
```

- we need a ”memory cell” to store previous state
- we need to initialize the storage before first use
- we need 2 multiplications and one addition per output sampel

6.6.3.2. Moving Average in C

BEGIN$_{\text{SRC}}$ C++ :exports both :flags -std=c++11

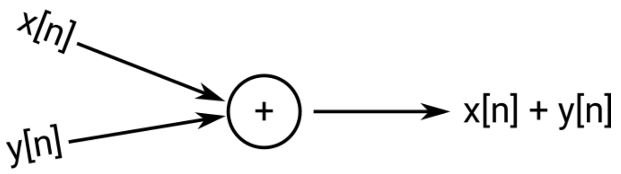
```
using namespace std;
double ma(double x) {
    static const int M = 5;
    static double z[M]; // Mx memory cells
    static int ix = -1;

    int n;
    double avg = 0;

    if(ix == -1) {      // initalize storage
        for(n=0; n<M; n++)
            z[n] = 0;
        ix = 0;
    }

    z[ix] = x;
    ix = (ix + 1) % M;  // circular buffer

    for(n=0; n<M; n++)  // Mx additions
        avg += z[n];

    return avg / M;     // 1x division
}

int main() {
    int n;
    for (n = 0; n<20; n++)
    {
        // call with delta signl
        printf("%.4f ", ma(n==0 ? 1.0 : 0.0));
        if(!((n+1)%10)) printf("\n");
    }
}
```

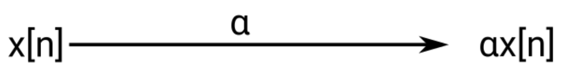
- we need M memory cells to store previous input values
- we need to initialize the storage before first use
- we need 1 division and M additions per output sample

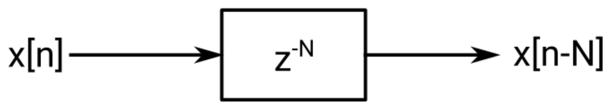
6.6.3.3. Programming Abstraction

With this three building blocks we can describe and Constant Coefficient Equation.

- Building Blocks

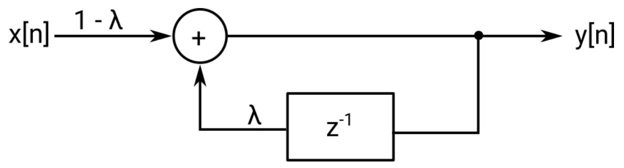






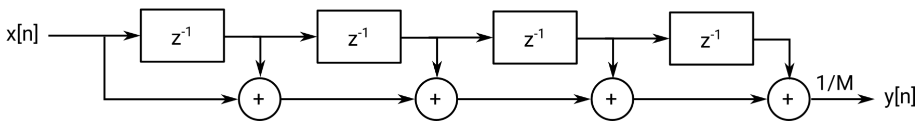
- Leaky Integrator

$$y[n] = \lambda y[n-1] + (1-\lambda)x[n]$$



- Moving Average

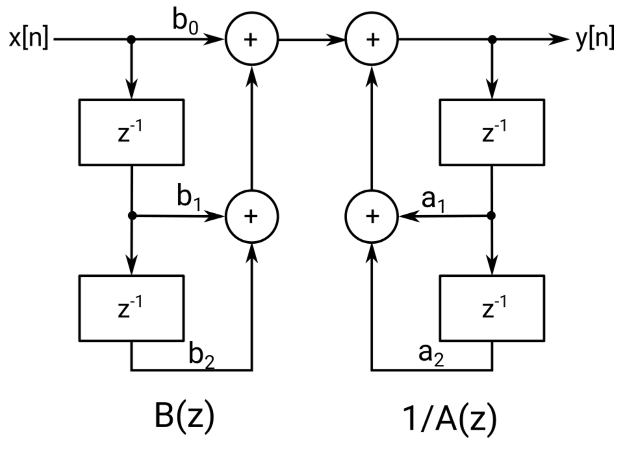
$$y[n] = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k]$$



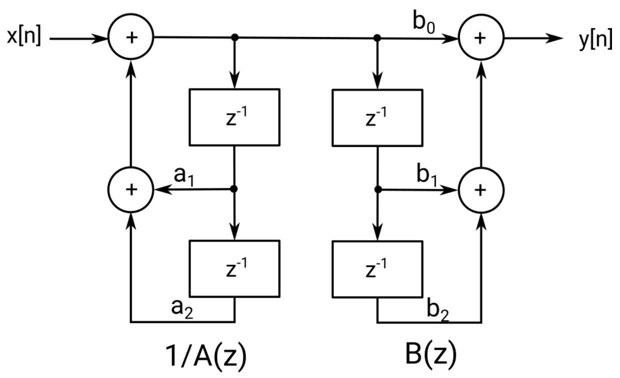
- The second-order section

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{B(z)}{A(z)}$$

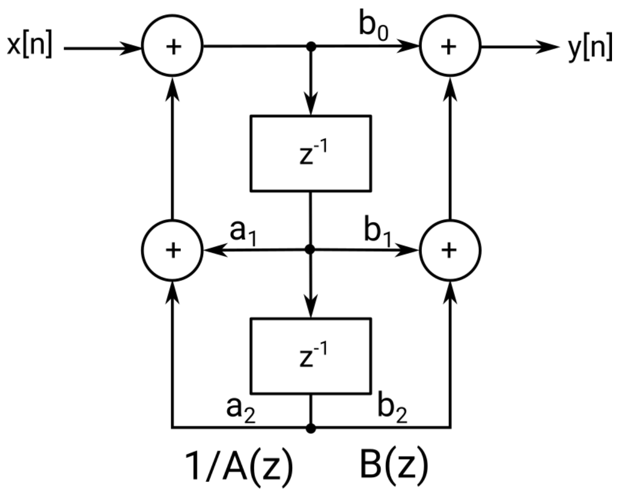
- Second-order section, direct form I



- Second-order section, inverted direct form I Because the convolution is commutative, numerator and denominator may be swapped.



- Second-order section, direct form II Since the content of the delay cells are exactly the same for all time, so we can lumb the delay cells together.



6.6.4. TODO Real-Time Processing

6.6.4.1. I/O and DMA Everything works in synch with a system clock of period T_s

- record a value $x_i[n]$
- process the value in a casual filter
- play the output $x_o[n]$

Everything needs to happen in at most T_s seconds!

Buffering:

- interrupt for each sample would be too much overhead
- soundcard consumes samples in buffers
- soundcard notifies when buffer used up
- CPU can fill a buffer in less time than soundcard can empty it

Double Buffering

- Delay $d = T_s \times \frac{L}{2}$ L: Lenght of the Buffer

- If CPU doesn't fill the buffer fast enough: **underflow**

Multiple I/O Processing

- Delay: $d = T_s \times L$
- usually start out process first

6.6.4.2. Implementation Framework

Low Level

- study soundcard data sheet
- write code to program soundcard via writes to IO Ports
- write an interrupt handler
- write the code to handle the data

High Level

- choose a good API
- write a callback function to handle the data

6.6.4.3. Callback Prototype

```
int Callback( const void *input,      // pointer to the input buffer
              void *output,           // pointer to the output buffer
              unsigned long samples, // length of samples
    );
{
    float* pIn = (float*)input;       // Convert the generic buffers
    float* pOut = (float*)output;     // to the right data type
    for (int n=0; n < samples; n++)  // Calls process for each sample in input the buffer
        *pOut++ = Process(*pIn++);    // and store the result into the output buffer.
}
```

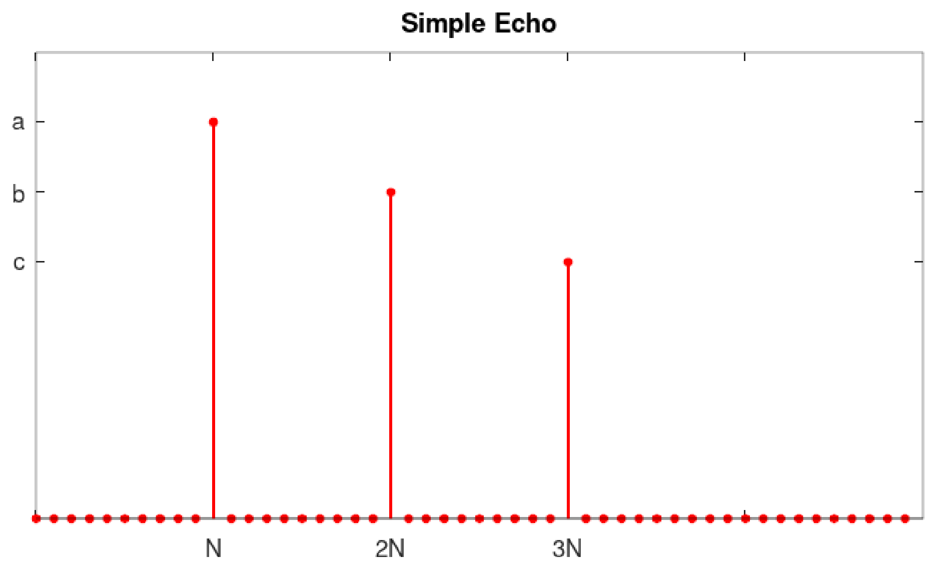
6.6.4.4. Processing Gateway

```
                                // 10 sec @ 24kHz
enum {BUF_LEN = 0x10000};       // Buffer length, power of 2
enum {BUF_MASK = BUF_LEN -1};   // Circular buffer mask
float m_pY[BUF_LEN];            // 2 Buffers
float m_pX[BUF_LEN];
int m_Ix;                       // 2 indixes into the buffers
int m_Iy;
float Process(float Sample)
{
    m_PX[m_Ix] = Sample;        // store the sample into input buffer
    float y = Effect();         // call Effect()
    m_pY[m_Iy] = y;             // store the output into output buffer
    m_Ix = (m_Ix + 1) & BUF_MAS; // Update indices with the ciruclar strategy
    m_IY = (m_Iy + 1) & BUF_MAS;
    return y;                   // return current output sample
}
```

6.6.4.5. Effect

Implementing the echo effect as a reflection of the original signal, scaled with a factor at subsequent points in time:

$$y[n] = \frac{ax[n] + bx[n-N] + cx[n-2N]}{a+b+c}$$



```
float Echo() {
    static float a = 0.85;              // the three scaling factors
    static float b = 0.6f;
    static float c = 0.45f;
    static float norm = 1.0f/(a+b+c);  // the normalisation factor
    static int N = (int)(0.3 * m_SR);  // delay between reflextion

    return norm * ( a * m_pX[m_Ix]
                    + b * m_pX[(m_Ix + BUF_LEN -N)   & BUF_MASK]
                    + c * m_pX[(m_Ix + BUF_LEN -2*N) & BUF_MASK]); }
```

6.6.5. TODO Derevereration and echo cancellation

Part VII.

Week 7 Module 5:

7. Sampling and Quantization

Interpolation describes the process of building a continuous-time signal $\mathbf{x(t)}$ from a sequence of samples $\mathbf{x[n]}$. In other words, interpolation allows moving from the discrete-time world to the continuous-time world. Interpolation raises two interesting questions:
The first one is how to interpolate between samples?

- In the case, of two samples, this is simple enough and there is there is a straight line that goes between these two samples.
- In the case of three samples, similarly, you have a parabola that goes through these 3 samples.
- If you have many samples, you can try to do the same and go through all samples but you see this is a trickier issue compared to what we have done with two or three samples.

The second question is:

- is there a minimum set of values you need to measure the function at so that you can perfectly reconstruct it.

Later on in the module, we are going to study sampling, i.e. the process of moving from a continuous-time signal to a sequence of samples. In other words, sampling allows moving from the continuous-time world to the discrete-time world. Suppose we take equally-spaced samples of a function $\mathbf{x(t)}$. The question is when is there a one-to-one relationship between the continuous-time function and its samples, i.e. when do the samples form a unique representation of the continuous-time function? To answer this question, we are going to use all the tools in the toolbox that we have looked at so far:

- Hilbert spaces
- projections
- filtering
- sinc functions
- and so on.

Everything comes together in this module to develop a profound and very useful result, the **sampling theorem**.
Before moving to the heart of the topic, let us briefly review its history. The Shannon sampling theorem has a very interesting history which goes back well before Shannon. Numerical analysts were concerned about interpolating tables of functions and the first one to proove a version of the sampling theorem was Whittaker in England in 1915. Harry Nyquist at Bell Labs came up with the Nyqvist criterion, namely that a function that has a maximum frequency F0F_{0F0} could be sampled at 2F0$2F_{02F0}$. In the Soviet Union, Kotelnikov proved a sampling theorem. The son of the first Whittaker further proved results on the sampling theorem. Then Herbert Raabe in Berlin wrote his PhD thesis about a sampling theorem that, wrong time wrong city, he got zero credit for. Denis Gabor worked on a version of the sampling theorem in the mid 1940s. Then Claude Shannon, the inventor of information theory, wrote a beautiful paper that is in the further reading for this class where the Shannon sampling theorem appears in the form that we use today. Last but not least, in 1949 Someya in Japan also proved the sampling theorem. You can see that it's a very varied history, it's a fundamental result where many people independently came up with this result.

7.1. The Continous-Time World

7.1.1. Introduction

The continuous-time world is the world we live in, the physical reality of the world, in contrast with the discrete-time world, the world inside a computer. We are first going to look at models of the world and compare digital with analog views of the world. Then we are going to study continuous-time signal processing in greater details. Furthermore, we will introduce the last form of Fourier transform we have not yet encountered in this class, the continuous-time Fourier transform.

7.1.2. The continous-time paradigm

Two views of the world

Table 7.1.: Two views of the world 1

Digital World	Analog World
arithmetic	calculus
combinatorics	distributions
computer sience	system theory
DSP	electronics

Table 7.2.: Two views of the world 2

Digital World	Analog World
countable integer index M	real-valued time t [sec]
sequences $x[n] \in \ell_2(\mathbb{Z})$	function $x(t) \in L_2(\mathbb{R})$
frequency $\omega \in [-\pi, \pi]$	frequency $\Omega \in \mathbb{R}(rad/sec)$
DTFT: $\ell_2(\mathbb{Z}) \rightarrow L_2[-\pi, \pi]$	FT: $L_2(\mathbb{R}) \rightarrow L_2(\mathbb{R})$

$\ell_2(\mathbb{Z})$	Square-Summable infinite sequences
$L_2([a,b])$	Square-integrable functions over an interval
Sampling	$x(t) \rightarrow x[n]$
Interpolation	$x[n] \rightarrow x(t)$

7.1.3. Continuous-time signal processing

time	real variable t
signal x(t)	complex function of areal variable
finite energy	$x(t) \in L_2(\mathbb{R})$
inner product in $L_2(\mathbb{R})$	$\langle x(t), y(t)\rangle = \int_{-\infty}^{\infty} x^*(t)\, y(t)$
energy	$\lvert\lvert x(t)\rvert\rvert^2 = \langle x(t), x(t)\rangle$

7.1.3.1. Analog LTI filters

$$\begin{aligned} y(t) &= (x * h)t \\ &= \langle h^*(t-\tau), x(\tau)\rangle \\ &= \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau \end{aligned}$$

7.1.3.2. Fourier analysis

- in discrete time max angular frequency is $\pm\pi$
- in continous time no max frequency: $\Omega \in \mathbb{R}$
- concept is the same:

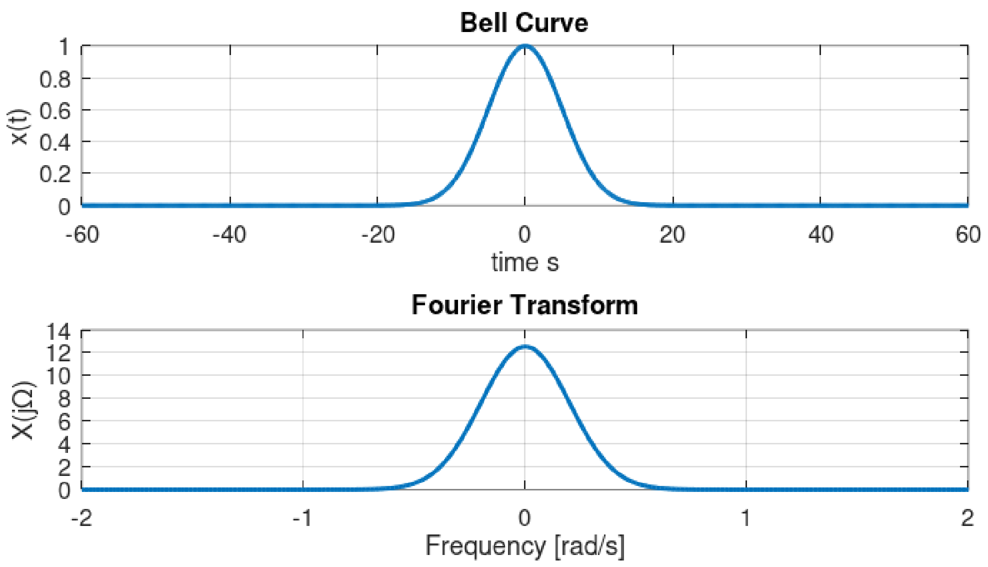
$$X(j\Omega) = \int_{-\infty}^{\infty} e^{-j\Omega\, t}\, dt$$

$$x(t) = \frac{1}{2\,\pi} \int_{-\infty}^{\infty} X(j\Omega)\, e^{j\Omega\, t} dt$$

7.1.3.3. Real-world frequency

- Ω expresse in rad/s
- $F = \dfrac{\Omega}{2\,\pi}$, expressed in Hertz (1/s)
- period $T = \dfrac{1}{F} = \dfrac{2\,\pi}{\Omega}$

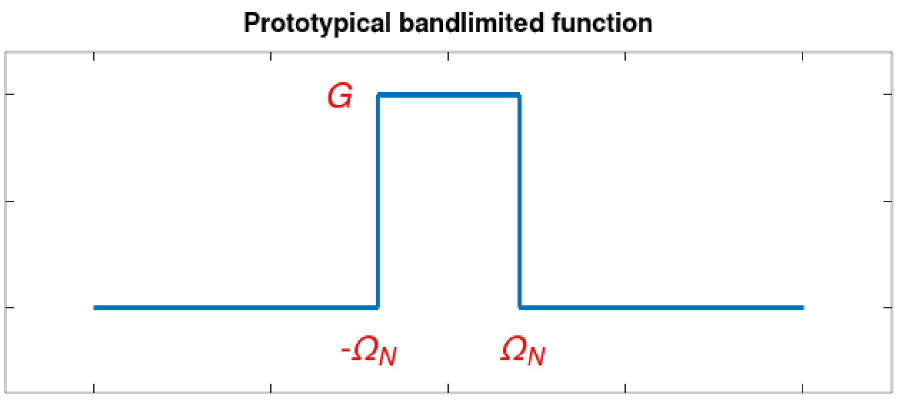
7.1.3.4. Example



7.1.3.5. Convolution theorem

$$Y(j\,\Omega) = X(j\Omega)\; H(j\Omega)$$

7.1.3.6. Prototypical Bandlimited Functions



$$\Phi(j\ \Omega) = G\ rect(\frac{\Omega}{2\ \Omega_N})$$

The time domain function can be determinded by means of its **Inverse Fourier Transform**

$$\begin{aligned}\phi(t) &= \frac{1}{2\ \pi}\int_{-\infty}^{\infty}\Phi(j\Omega)e^{j\Omega\ t}d\Omega\\ &= G\frac{\Omega_N}{\pi}sinc(\frac{\Omega_N}{\pi}\ t)\end{aligned}$$

The time domain function is up to a scaling, one of these sinc functions. We will normalize this sinc function, so that the area is equal to 2π in the Frequency Domain. Then the inverse continuous time Fourier Transform will have a maxmimum of 1 at the origin.

normalization	$G = \frac{\pi}{\Omega_n}$
totoal bandwith	$\Omega_B = 2\ \Omega_N$
define	$T_s = \frac{2\ \pi}{\Omega_B} = \frac{\pi}{\Omega_N}$

This leads to the normalized prototypical bandlimted function:

Frequency Domain

$$\Phi(j\ \Omega) = \frac{\pi}{\Omega_N}\ rect\left(\frac{\Omega}{2\ \Omega_N}\right)$$

Time Domain

$$\phi(t) = sinc\left(\frac{t}{T_s}\right)$$

7.1.4. TODO Plot Normalized prototypicale bandlimited function

7.2. Interpolation

Main Task	$x[n] \Rightarrow x(t)$
Gaps	fill the gaps between samples

7.2.1. Interpolation requirements

- decide on T_s
- make sure $x(nT_s) = x[n]$
- make sure x(t) is smooth

7.2.2. Why smoothness

- jumps (1st order discontinuities) would require the signal to move ”faster than light”
- 2nd order discontinuities would require infinite acceleration
- the interpolation should be infinitely differentiable
- ”natural” solution: polynomial interpolation

7.2.3. Polynomial interpolation

- N points ⇒ polynomial of degree (N-1)
- $p(t) = a_0 + a_1 t + a_2 t^2 a ... + a_{N-1} t^{(N-1)}$
- ”naive” approach

$$\begin{cases} p(0) & = x[0] \\ p(T_s) & = x[1] \\ p(2T_s) & = x[2] \\ ...... & \\ p((N-1)T_s) & = x[N-1] \end{cases}$$

Without loss of generality:

- consider a symmetric interval $I_N = [-N...N]$
- set $T_s = 1$

$$\begin{cases} p(-N) & = x[-N] \\ p(-N+1) & = x[-N+1] \\ ...... & \\ p(0) & = x[0] \\ p(N) & = x[N] \end{cases}$$

7.2.4. Lagrange interpolation

The natural solution to this interpolation problem is given by Lagrange interpolation

- P_N : space of degree-2N polynominals over $\mathrm{I_N}$
- a basis for $\mathrm{P_N}$ is the family of 2N + 1 Lagrange polynominals

$$L_n^{(N)}(t) = \prod_{k=-N}^{N} \frac{t-k}{n-k} \text{ for } M = -N,...,N$$

The formula:

$$\boxed{p(t) = \sum_{n=-N}^{N} x[n] L_n^{(N)}(t)}$$

The Lagrange interpolation is the sought-after polynominal interpolation:

- polynominal of degree 2N through 2N+1 points is unique
- the Lagrangian interpolator satisfies

$$p(N) = x[N] \text{ for } -N \leq M \leq N$$

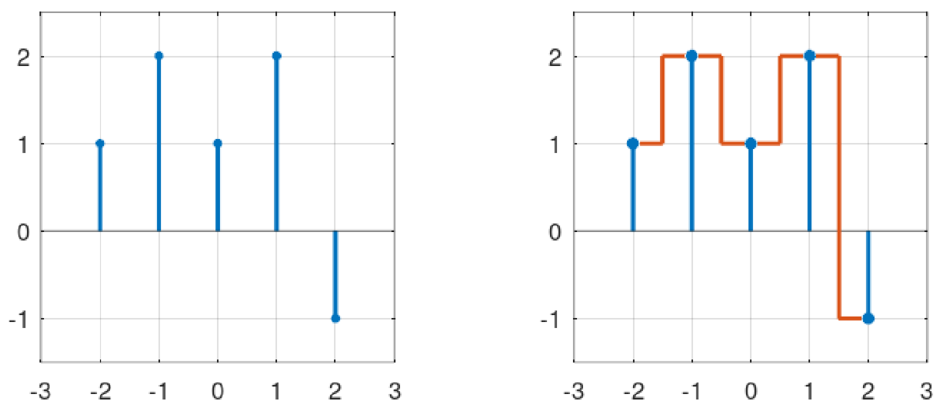
since

$$L_n^{(N)}(N) = \begin{cases} 1 \text{ if } M = N \\ 0 \text{ if } M \neq N \end{cases} \quad -N \leq M, N \leq N$$

key property	maximmally smooth (infinitely many continuous derivatives)
drawback	interpolation "bricks" depend on N

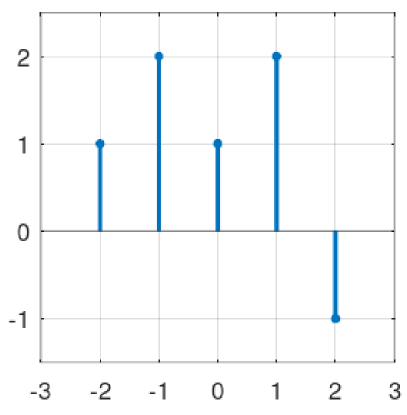
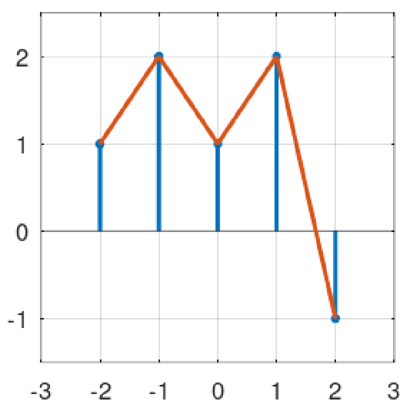
7.2.5. Local Interpolation

7.2.5.1. Zero-order hold "Box Function"



- $x(t) = x[t + 0.5]$ for $-N \leq t \leq N$
- $x(t) = \displaystyle\sum_{n=-N}^{N} x[n]\ rect(t-n)$
- interpolation kernel: $i_0(t) = rect(t)$
- $i_0(t)$: zero-ordre hold
- interpolation support is 1
- interpolation is not even continous

7.2.5.2. First-order piece-wise linear "Hat Function"

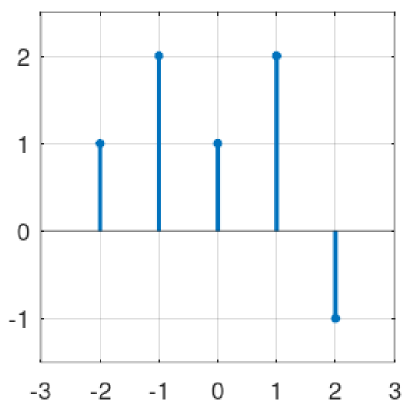
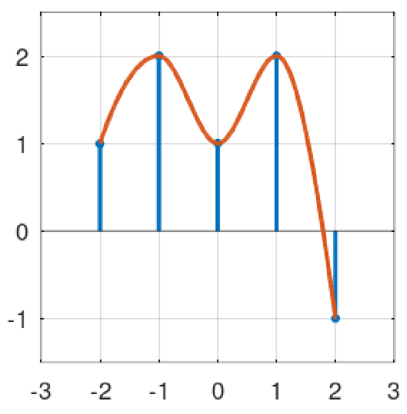



- connect the dots strategy
- $x(t) = \sum_{n=-N}^{N} x[n]\, i_1(t-n)$
- interpolation kernel:

$$i_1(t) = \begin{cases} 1 - |t| & |t| \leq 1 \\ 0 & otherwise \end{cases}$$

- interpolation support is 2
- interpolation is continuous but derivative is not

7.2.5.3. Third-order interpolation

- $x(t) = \sum_{n=-N}^{N} x[n]\, i_3(t-n)$
- interpolation kernel obtained by splicing two cubic polynominals
- interpolation support is 4
- interpolation is continuous up to second derivative

7.2.5.4. Local Interpolation schemes

$$x(t) = \sum_{n=-N}^{N} x[n]\ ic(t-n)$$

Interpolator's requirements:

- $\mathrm{i_c}$: interpolation kernel
- $\mathrm{i_c}(0) = 1$
- $\mathrm{i_c}(\mathrm{t}) = 0$

Key property	same interpolating function independently of N and of location
drawback	lack of smoothness

7.2.6. Sinc interpolation formula

A remarkable result:

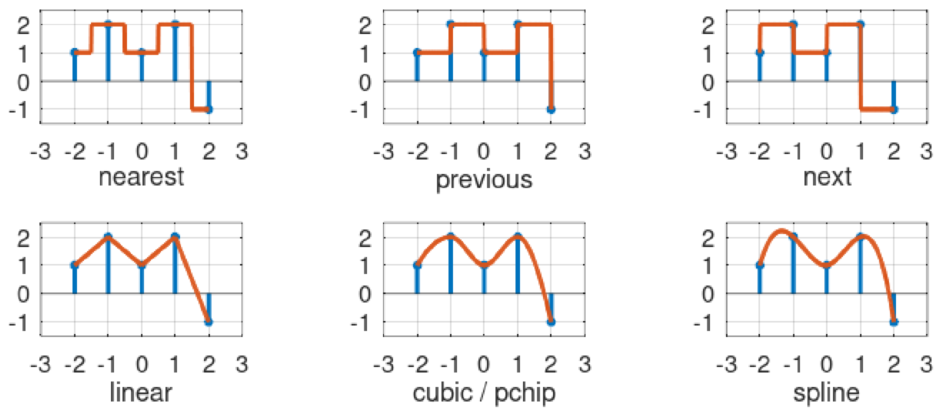
$$\lim_{N\to\infty} L_n^{(N)}(t) = sinc(t-n)$$

In the limit, local and global interpolation are the same!

$$x(t) = \sum_{n=-N}^{N} x[n] sinc\left(\frac{t-nT_s}{T_s}\right)$$

7.2.7. Octave Interpolation Overview

Octave manual Chapter 29.1 One-dimensional Interpolation



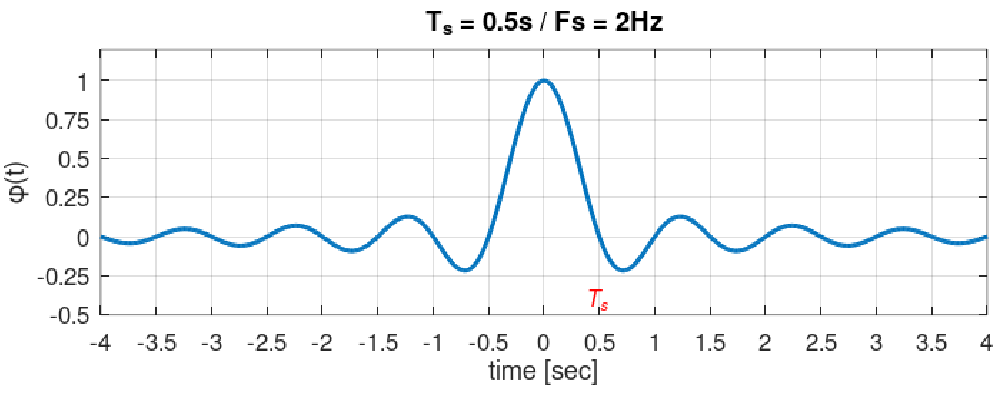
$$x(t) \sum_{n=-N}^{N} x[n] i_c(t-n)$$

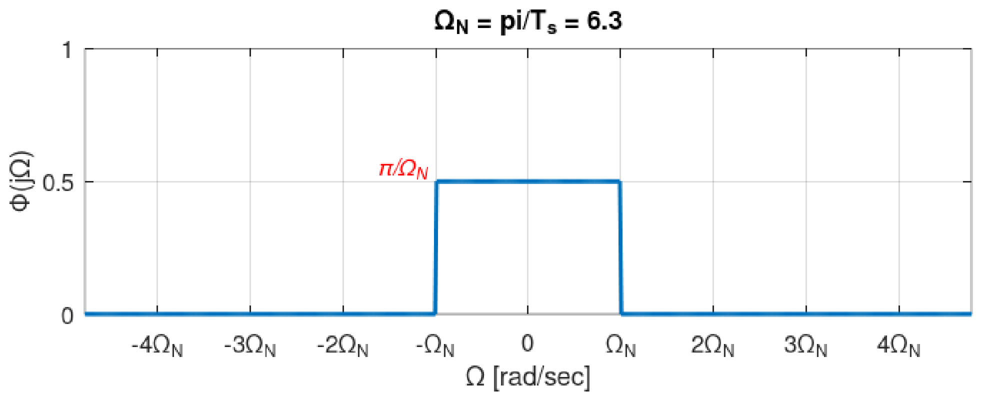
7.3. Sampling of bandlimited functions

7.3.1. Key Facts about the sinc

$$\phi(t) = sinc\left(\frac{t}{T_s}\right) \quad \longleftrightarrow \quad \Phi(j\Omega) = \frac{\pi}{\Omega_N} rect\left(\frac{\Omega}{2\Omega_N}\right)$$

$$T_s = \frac{\pi}{\Omega_n} \qquad \Omega_N = \frac{\pi}{T_s}$$





7.3.2. The spectrum of interpolated signals

What is the spectrum of the

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] sinc\left(\frac{t - nT_s}{T_s}\right)$$

$$\begin{aligned}
X(j\Omega) &= \int_{-\infty}^{\infty} x(t)\, e^{-j\Omega t} dt \\
&= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[n] sinc\left(\frac{t-nT_s}{T_s}\right)\, e^{-j\Omega t} dt \\
&= \sum_{n=-\infty}^{\infty} x[n] \int_{-\infty}^{\infty} sinc\left(\frac{\boxed{t-nT_s}}{T_s}\right)\, e^{-j\Omega t} dt \\
&= \sum_{n=-\infty}^{\infty} x[n] \left(\frac{\pi}{\Omega_N}\right) rect\left(\frac{\Omega}{2\Omega_N}\right) \boxed{e^{-jnT_s\Omega}} \\
&= \left(\frac{\pi}{\Omega_N}\right) rect\left(\frac{\Omega}{2\Omega_N}\right) \sum_{n=-\infty}^{\infty} x[n] e^{-j(\pi/\Omega_N)\Omega_n}
\end{aligned}$$

$$= \begin{cases} \left(\frac{\pi}{\Omega_N}\right) X(e^{j\pi(\Omega/\Omega_N)}) & |\Omega| \leq \Omega_N \\ 0 & otherwise \end{cases}$$

Spectrum of Sinc-Sampling

The spectrum of $x(t)$ is equal to the scaled version of the DTFT of the sequence between $-\Omega_N$ and Ω_N.

Pick interpolation period T_s:

- $X(j\Omega)$ is $\Omega_N - bandlimited$, with $\Omega_N = \pi/T_s$
- fast interpolation ($_s$ small) $\Rightarrow$ wider spectrum
- slow interpolation ($_s$ large) $\Rightarrow$ narrower spectrum

7.3.3. The space of bandlimited functions

Claims:

- the space of $\Omega_N - bandlimited$ functions is a Hilbert space
- the functions $\phi^{(n)}(t) = sinc((t-n)$, with n $\in \mathbb{Z}$ form a bais for the space
- if $x(t)$ is $\pi - BL$ the sequence x[n] = x(n), with n $\in \mathbb{Z}$, is a sufficient representation, i.e. we can recunstruct x(t) from x[n]

The space $\pi - BL$

- is a a vector space because $\pi - BL \subset L_2(\mathbb{R})$
- inner product is standard inner product in $L_2(\mathbb{R})$
- completeness... that's more delicate

Basis for $\pi - BL$

The sync function is an orthornormal basis for the $\pi - BL$ space.

Inner product:

$$\langle x(t), y(t)\rangle = \int_{-\infty}^{\infty} x^*(t)y(t)dt$$

Convolution:

$$(x * y)(t) = \langle x^*(\tau), y(t-\tau)\rangle$$

A basis for the $\pi - BL$ space

$$\phi^{(M)}(t) = sinc(t-n), \text{ for } M \in \mathbb{Z}$$

$$FT sinc(t) = rect\left(\frac{\Omega}{2\pi}\right)$$

$$(sinc * sinc)(m-n) = \begin{cases} 1 \text{ for } m = n \\ 0 \text{ otherwise} \end{cases}$$

7.3.4. The sampling Theorem

7.3.4.1. Sampling as a basis expansion

To see sampling as an orthonormal expansion, we take our sample of orthonormal vectors $\phi^(n)$, taking a product with x and we look what comes out.

Analysis Formula

$$x[n] = \langle sinc\left(\frac{t-nT_s}{T_s}\right), x(t)\rangle = T_s x(nT_s)$$

Synthesis Formula

$$x(t) = \frac{1}{T_s}\sum_{n=-\infty}^{\infty} x[n] sinc\left(\frac{t-nT_s}{T_s}\right)$$

- the space of $\Omega_n - bandlimited$ functions is a Hilbert space
- set $T_s = \pi/\Omega_N$
- the functions $\phi^{(n)}(t) = sinc((t-nT_s)/T_s)$ form a bais for the space
- for any $x(t) \in \Omega_N - BL$ the coefficients in the sinc basis are the (scaled) samples $T_s x(nT_s)$

Corollary

for any $x(t) \in \Omega_N - BL$, a sufficient representation is the sequence $x[n] = x(nT_s)$

The sampling theorem in Hertz

Any signal x(t) bandlimited to F_N Hz can be sampled with no loss of information using a sampling frequency $F_s \geq 2F_N$ (i.e. sampling period $T_s \leq 1/2\ F_N$)

7.4. Sampling of nonbandlimited functions

7.4.1. Raw Sampling

Raw sampling is when we don't care about first taking the inner product with the sinc function. So we just take x(t) and every $\mathrm{T_s}$ seconds, we take a sample.

The continous-time complex exponential

Table 7.3.: Aliasing

sampling period	digital frequency	$\{\hat{x}\}$
$T_s < \pi/\Omega_0$	$0 < \omega_o < \pi$	$e^{j\Omega_0}$
$\pi/\Omega_0 < T_s < 2\pi/\Omega_0$	$\pi < \omega_0 < 2\pi$	$e^{j\Omega_1}$: $\Omega_1 = \Omega_0 - 2\pi/T_s$
$T_s > 2\pi/\Omega_0$	$\omega_0 > 2\pi$	$e^{j\Omega_2}$: $\Omega_2 = \Omega_0 mod(2\pi/T_s)$

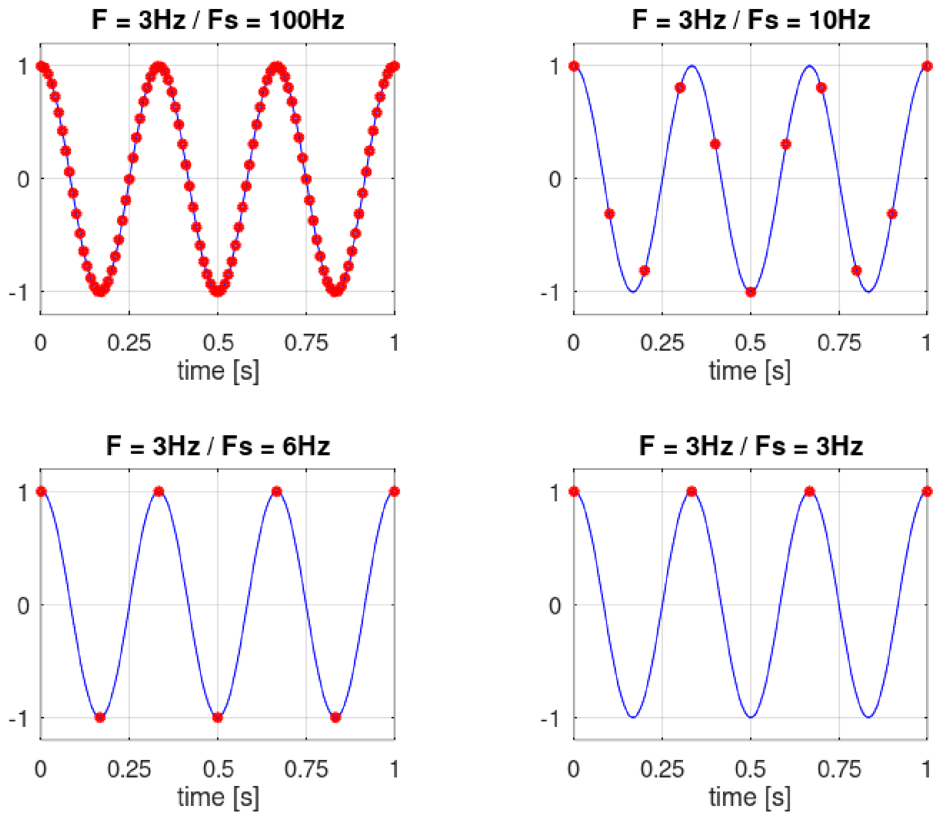
7.4.2. Sinusoidal Aliasing

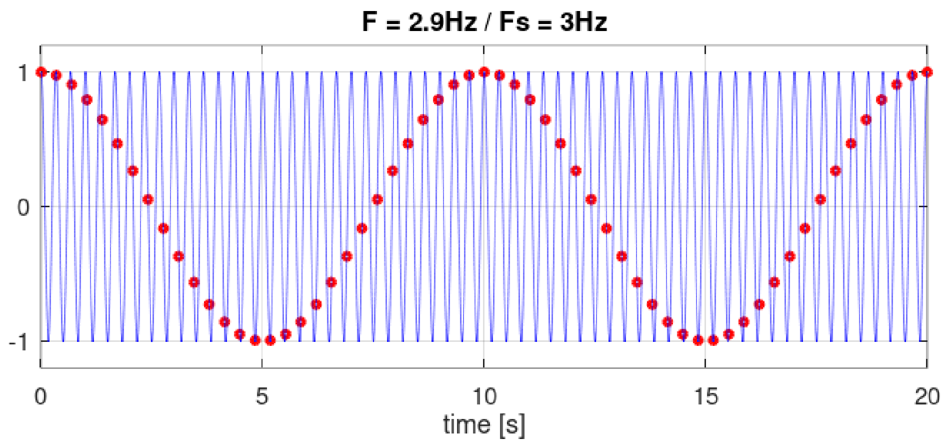
$$x(t) = cos(2_o t)$$
$$x[n] = x(nT_s) = cos(\omega_0 n)$$

with

$$F_s = \frac{1}{T_s}$$
$$\omega_o = 2\pi(\frac{F_0}{F_s})$$

7.4.2.1. Aliasing: Sampling a Sinusoid





7.4.3. Aliasing for arbitrary spectra

A contiuous time signal x_c sampled every T_s seconds gives a sequence x[n]. Which is equal to the contious time signals at multiples of the sampling intervals T_s.

- $x_c(t) \Rightarrow x[n] = x_c(nT_s)$

In Fourier Transform domain we have a spectra of the continuous time signal $X_c(j\Omega)$. And at the output we have a discrete time Fourier Transform of the sequence $X(j\omega)$. What is that going to be in genaral? And how is it going to be related to the input spectrum?

- $X(j\Omega) \Rightarrow X(j\omega) = ?$

The key idea:

- pick T_s and set $\Omega_N = \pi/T_s$
- pick $\Omega_= < \Omega_N$

$$\begin{aligned}
e^{j\Omega_0 t} &\rightarrow e^{j\Omega_0 T_s n}\\
e^{j(\Omega_0+2\Omega_N)t} &\rightarrow e^{j(\Omega_0+2\Omega_N)T_s n}, \text{ add } 2\Omega_N\\
e^{j(\Omega_0+2\Omega_N)t} &\rightarrow e^{j(\Omega_0 T_s n+2\Omega_N T_s n)}, \text{ expand this product}\\
e^{j(\Omega_0+2\Omega_N)t} &\rightarrow e^{j(\Omega_0 T_s n+\frac{2\pi}{T_s}T_s n)}\\
e^{j(\Omega_0+2\Omega_N)t} &\rightarrow e^{j(\Omega_0 T_s n+2)}, \; e^{j2} \text{ is equal to one}\\
e^{j(\Omega_0+2\Omega_N)t} &\rightarrow e^{j\Omega_0 T_s n}, \text{ the same discrete time sequence as before}
\end{aligned}$$

So we do not see the higher frequency complex exponential, it simply looks like the lower frequency exponential Ω_0.

So in general, if we have two frequencies sampled, the higher frequency is aliased back onto the lower frequency and we simply see the sum of these two.

- Spectrum of raw-sampled signals
  - start with the inverse Fourier Transform

$$x[n] = x_c(nT_s) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X_c(j\Omega)e^{j\Omega M T_s}d\Omega$$

- frequencies $2\Omega_N$ apart will be aliased, so split the integration interval

$$x[n] = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\int_{(2k-1)\Omega_N}^{(2k+1)\Omega_N} X_c(j\Omega)e^{j\Omega M T_s}d\Omega$$

- with a change of variable and using $e^{j(\Omega+2k\Omega_N)T_s M} = e^{j\Omega T_s M}$

$$x[n] = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\int_{-\Omega T_s M}^{\Omega T_s M} X_c(j(\Omega - 2k\Omega_N)e^{j\Omega M T_s}d\Omega$$

- interchange summation and integral

$$x[n] = \frac{1}{2\pi}\int_{-\Omega T_s M}^{\Omega T_s M}\left[\sum_{k=-\infty}^{\infty} X_c(j(\Omega - 2k\Omega_N)\right]e^{j\Omega M T_s}d\Omega$$

- periodization of the spectrum; define

$$X_c(j\Omega) = \sum_{k=-\infty}^{\infty} X_c(j(\Omega - 2k\Omega_N)$$

- so that

$$x[n] = \frac{1}{2\pi}\int_{-\Omega T_s M}^{\Omega T_s M} X_x(j\Omega)e^{j\Omega M T_s}d\Omega$$

- set $\omega = \Omega T_s$

$$\begin{aligned} x[n] &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\frac{1}{T_s}X_c(j\frac{\omega}{T_s})e^{j\omega M}d\omega \\ &= IDTFT\ \frac{1}{T_s}X_c(j\frac{\omega}{T_s}) \\ X(e^{j\omega}) &= \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X_c\left(j\frac{\omega}{T_s} - j\frac{2\pi k}{T_s}\right) \end{aligned}$$

$$X(e^{j\omega}) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X_c\left(j\frac{\omega}{T_s} - j\frac{2\pi k}{T_s}\right)$$

- **TODO** Example: signal bandlimited to Ω_0 and $\Omega_\mathrm{N} > \Omega_0$
- **TODO** Example: signal bandlimited to Ω_0 and $\Omega_\mathrm{N} = \Omega_0$
- **TODO** Example: signal bandlimited to Ω_0 and $\Omega_\mathrm{N} < \Omega_0$
- **TODO** Example: non-bandlimited signal

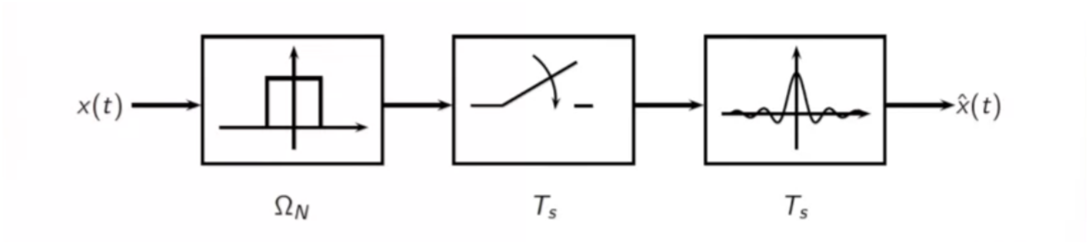
7.4.4. Sampling strategies

given a sampling period T_s

- if the signal is bandlimited to π/T_s or less, raw sampling is fine i.e. equivalent to sinc sampling up to scaling factor T_s.
- if the signal is not bandlimited, two choices:
  - bandlimit via lowpass filter in the *continuous-time domain* before sampling i.e. sinc sampling
  - or raw sample the signal an incur aliasing
- aliasing sounds horrible, so usualle we choose to bandlimit in continuous time

- Sinc Sampling and Interpolation

$$\hat{X}[n] = \langle sinc\left(\frac{t-nT_s}{T_s}\right), x(t)\rangle = (sinc_{Ts} * x)(nT_s)$$
$$\hat{X}[n] = \sum_n x[n]\ sinc\left(\frac{t-nT_s}{T_s}\right)$$



7.5. Quantization

7.5.1. Stochastic signal processing

7.5.1.1. Terminology (from W.Smith)

Mean

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i = (x_0 + x_1 + x_2 + ... + x_{N-1})/N$$

In electronics, the mean is commonly called the **DC** (direct current) value. Likewise, **AC** (alternationg current) refers to how the signal fluctuates around the mean value. For simple repetitive waveform, its excursion can be described by its peak-to peak value. If the signal has a random nature, a more generalized method must be used.

Standard Deviation

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2)} = \sqrt{(x_0-\mu)^2 + (x_1-\mu)^2 + ... + (x_{N-1}-\mu)^2/(N-1)}$$

$|x_i - \mu|$ describes how far the i^{th} sample **deviates** (differs) from the mean. The **average deviation** of a signal is found by summing the deviations of all the individual samples, and then dividing by the number of samples N. We take the absolute value of each deviation before summation; otherwise the positive and the negative termss would average to zero.

The **standard deviation** is similar to the average deviation, except the averaging is done with power instead of amplitude.

The standard deviation is a measure of how far the signal fluctuates from the mean.

Variance

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2)$$

The variance represents the power of signal fluctuation from the mean.

RMS Root Mean Square The standard deviation measures only the AC portion of a signal, while rms value measures both the AC and DC components. If a signal has no DC component, its rms value is identical to its standard deviation.

SNR Signal to Noise Ratio

$$snr = \frac{mean}{standarddeviation} = \frac{\mu}{\sigma} = \frac{\frac{1}{N}\sum_{i=0}^{N-1} x_i}{\sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2)}}$$

CV Coefficent Variation

$$CV = \frac{standard\ deviation}{mean} \times 100 = \frac{\sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2)}}{\frac{1}{N}\sum_{i=0}^{N-1} x_i} \times 100$$

7.5.1.2. TODO Deterministic vs. stochastic

7.5.1.3. A simple discrete-time random signal generator

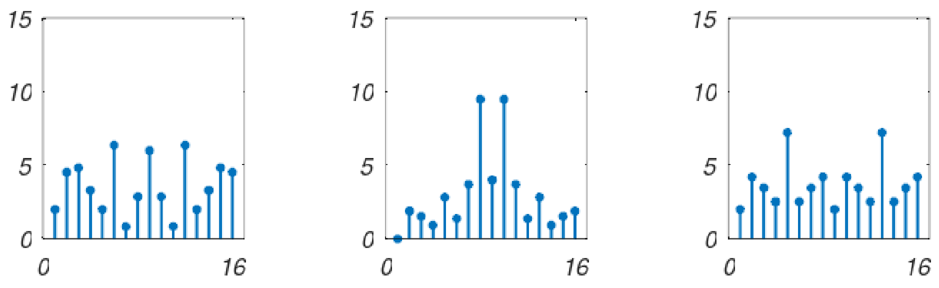
For each new sample, toss a fair coin:

$$x[n] = \begin{cases} +1 & \text{if the outcome of the n-th toss is head} \\ -1 & \text{if the outcome of the n-th toss is tail} \end{cases}$$

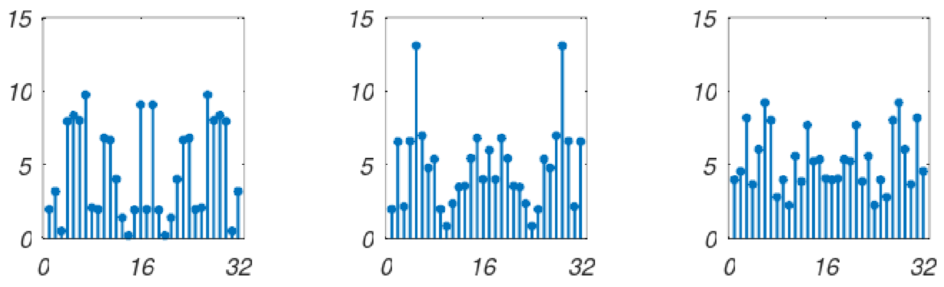
- each sample is independet from all others
- each sample value has 50% probability
- every time we turn on the generator we obtain a different *realization* of the signal
- we know the "mechanics" behind each instance
- but how can we analyze a random signal?

7.5.1.4. Spectral Properties

- let's try with the DFT of a finite set of random samples



- every time it's a different
- try with more data



- no clear pattern

7.5.1.5. Averaging

- when faced with random data an intuitive response is to take ”averages”
- in probability theory the average is across realizations and it’s called Expectation
- Expectation for the coin-toss signal

$$\boxed{\text{E[x[n]]} = \text{-1} \times P[\text{n-th toss is tail}] + 1 \times P[\text{n-th toss is head}] = 0}$$

so the average value for each sample is zero....
as a consequence, averaging the DFT will not work
$E[x[n]] = 0$
however the signal ”moves”, so its energy over power must be nonzero

7.5.1.6. TODO Averaging the DFT

7.5.1.7. Energy and power

- the coin-toss signal has infinite energy

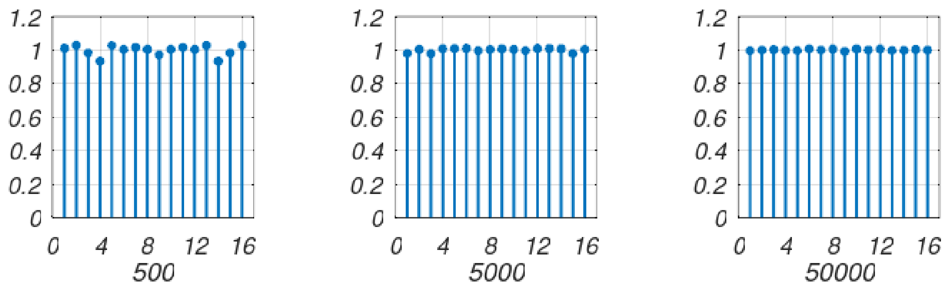
$$E_x = \sum_{k=-\infty}^{\infty} |x[n]|^2 = \lim_{N\to\infty} = \infty$$

- however it has finite power over any interval:

$$P_x = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2 = 1$$

7.5.1.8. Averaging the DFT’s square magnitude, normalized

- pick an interval length N
- pick an number of iterations M
- run the signal generator M times and obtain M N-point realizations
- compute the DFT of each realizations
- average their square magnitude divided by N



7.5.1.9. Power spectral density

$$P[k] = E\left[|X_N[k]|^2 / N \right]$$

- it looks very much as if P[k] = 1
- if $|X_N[k]|^2$ tends to the *energy* distribution in frequnec....
- ... $|X_N[k]|^2 / N$ tends to the *power* distribution (aka **density**) in frequency

PSD

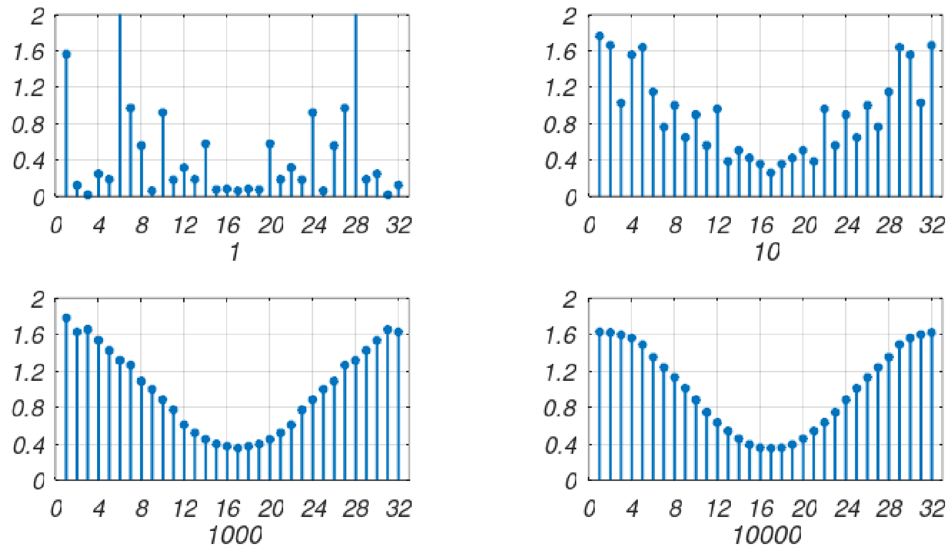
The frequency-domain representation for stochastic processes is the power spectral density: $P[k] = \frac{1}{N} |X_N[k]|^2$

7.5.1.10. Power spectral density: Intuition

- P[k] = 1 means that the power is equally distributed over all frequencies
- i.e. we cannot predict the signal moves "slowly" or "super-fast"
- this is because each sample is independent of each other: we could have a realization of all ones or a realization in which the sign changes every other sample or anything in between.

7.5.1.11. Filtering a random process

- let's filter the process with a 2-point Moving Average filter
- y[n] = (x[n] + x[n-1])/2
- what is the power spectral density
- pick an interval length N
- pick an number of iterations M
- run the signal generator M times and obtain M N-point realizations
- filter all M-realization
- compute the DFT of each filtered realizations
- average their square magnitude divided by N

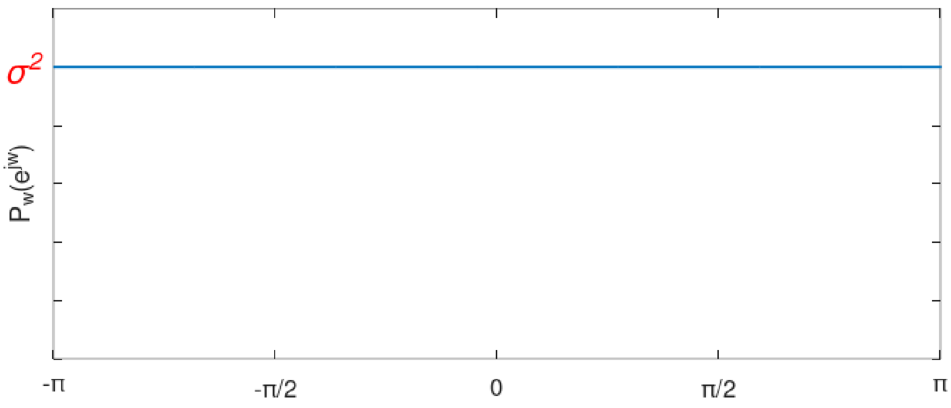


7.5.1.12. TODO Noise

7.5.1.13. White noise

- white indicates uncorrelated samples
- $r_w[n] = \sigma^2\delta[n]$: The autocorrleation is zero except at zero where it will take the value of the variant
- $P_w(e^{j\omega} = \sigma^2$: The power spectral density is the constant σ^2 wher σ is the variance of the stochastic signal.

Graphically the power spectral density of a white signal couldn't be any simpler.



- the PSD is independent of the probability distribution of the single samples (depends only on the variance)
- distribution is important to estimate bounds for the signal
- very often a Gaussioan distribution models the experimental data the best
- **AWGN**: additive white Gaussian noise

7.5.1.14. Summary

- a stochastic process is characterized by its power spectral density (PSD)
- it can be shown (see text book) that the PSD is

  $$P_x(e^{j\omega}) = DTFTr_x[n]$$

  where

  $$r_x[n] = E\left[x[k]x[n+k]\right]$$

  is the autocorrelation of the proess
- for a filtered stochastic process $y[n] = \mathfrak{H}x[n]$, it is:

  $$P_x(e^{j\omega}) = |H(e^{j\omega})|^2 P_x(e^{j\omega})$$

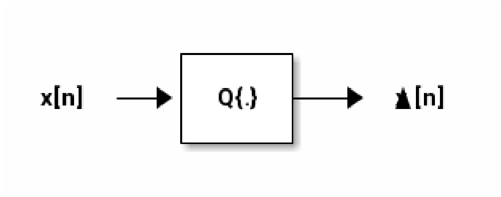
In Words

The power spectral density of the output is equal to the power spectral density of the input times the frequency response in magnitude square.

7.5.2. Quantization

7.5.2.1. Quantization schemes

- digital devices can only deal with integers (b bits per sample)
- we need to map the range of a signal onto a finite set of values
- irreverible loss of information $\longrightarrow$ Quantization Noise



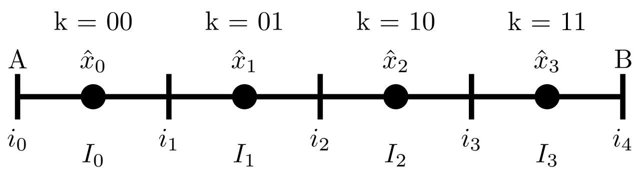
Several factors at play:

- storage budget (bits per sample)
- storage scheme (fixed point, floating point)
- properties of the input (input $\in \mathbb{C} \rightarrow$ output $\in \mathbb{N}$

7.5.2.2. Scalar quantization

The simplest quantizer:

- each sample is encoded individually (hence scalar)
- each sample is quantized independently (memoryless quantization)
- each sampe is encoded using R bits



- what are the optimal interval boundries I_k ?
- what are the optimal quantization values $\hat{x}_k$?

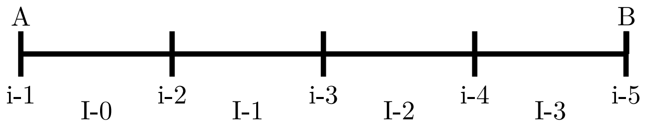
7.5.2.3. Quantization Error

$$e[n] = Q\{x[n]\} - x[n] = \hat{x} - x[n]$$

- model x[n] as a stochastic process
- model error as a white noise sequence
  - error samples are uncorrelated
  - all error samples have the same distribution
- we need statistics of the input to study the error

7.5.2.4. Uniform quantization

- simple but very general case
- range is split into 2^R equal intervals of width $\Delta = (B - A)2^{-R}$
- **With a Bit-Rate R** of 2 bits is a region split into 4 equally spaced intervals



- filters designed for deterministic signals work (in magnitude) in the stochaistic case
- we lose the concept of phase since we don't know the shape of a realization in advance.

Part VIII.

Week 8 Module 6:

8. Digital Communication Systems

Part IX.

Week 8 Module 7:

9. Image Processing

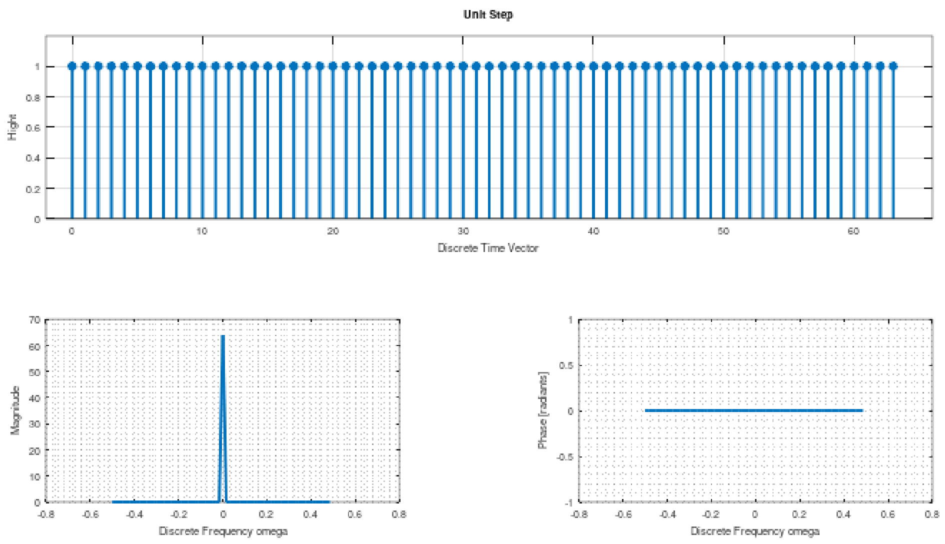
Part X.

Appendix

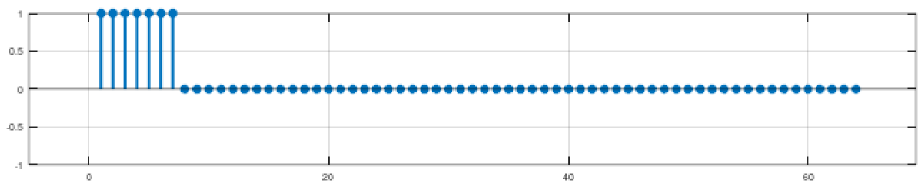
10. DFT Plots

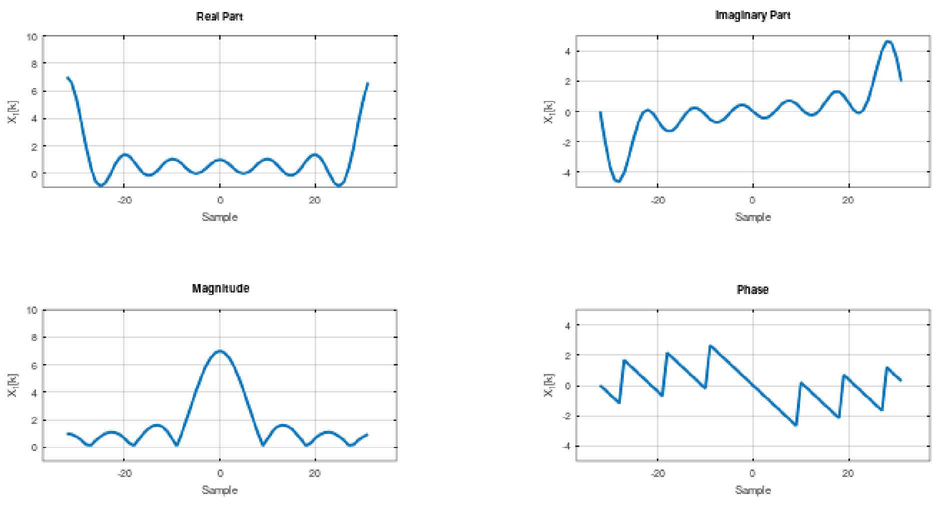
10.1. DFT Unit Step

- FFT Tutorial University Rhode Island

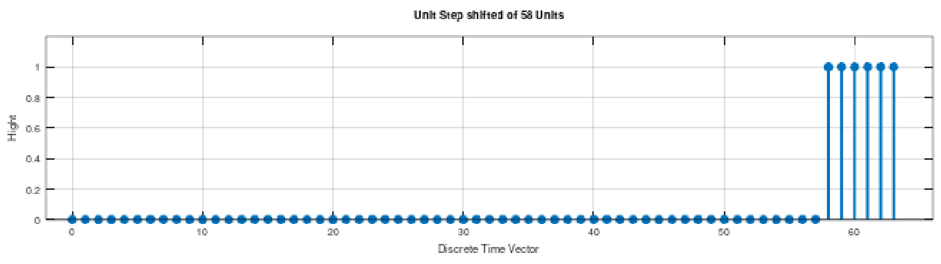


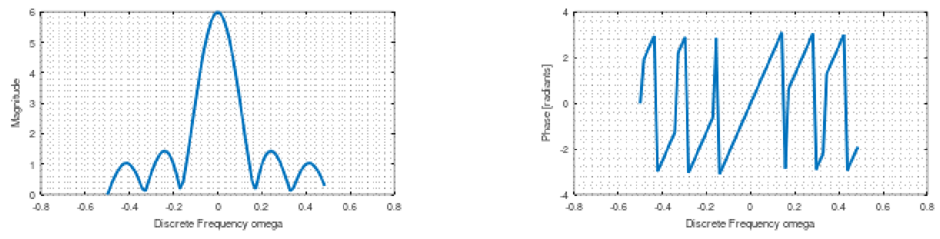
10.2. DFT Pulse Function



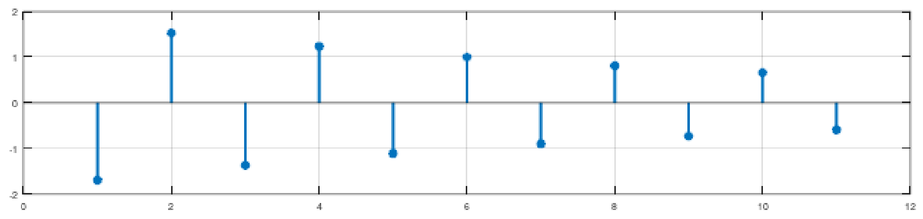


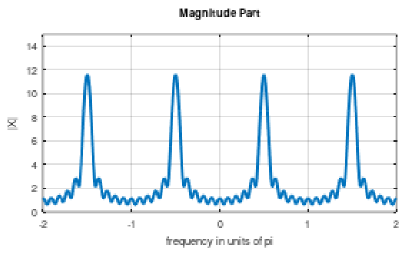
10.3. DFT Shifted Pulse Function

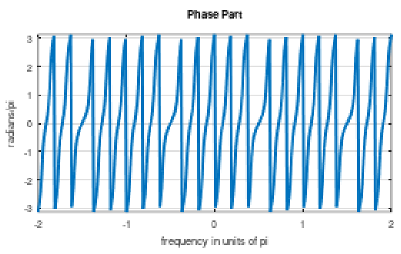




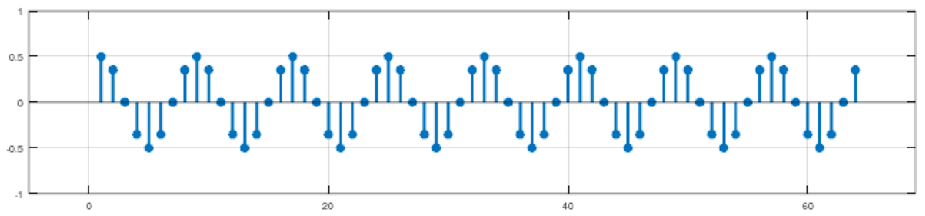
10.4. DFT Complex Exponential

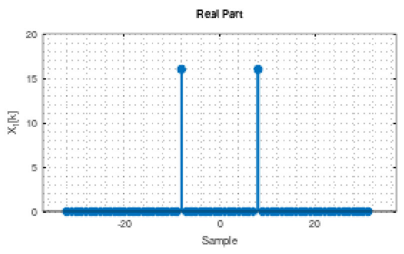


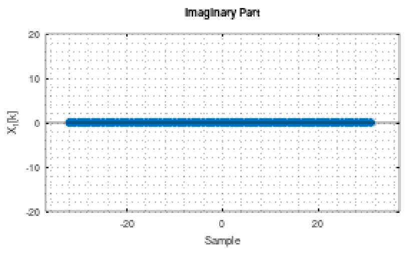




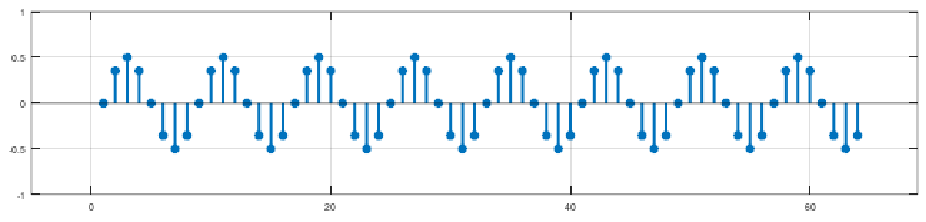
10.5. DFT Cosine

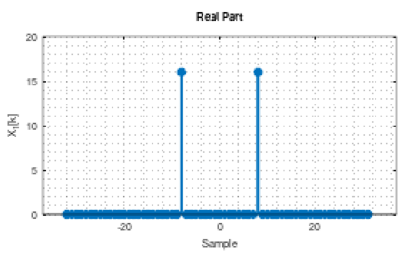


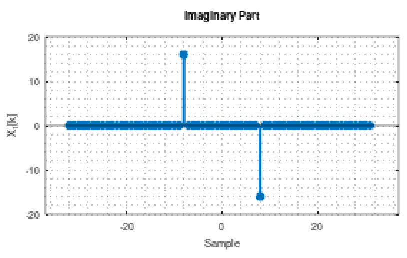




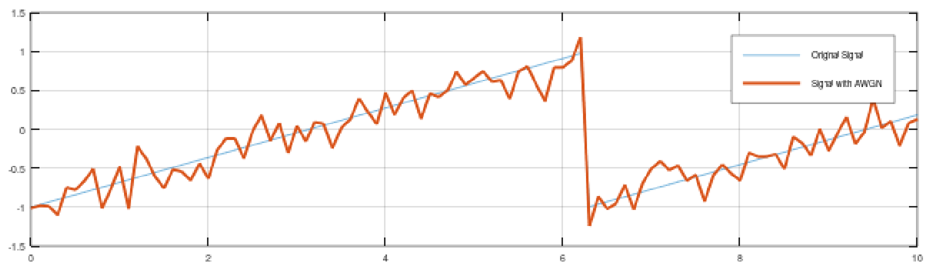
10.6. DFT Sinusoid Sine





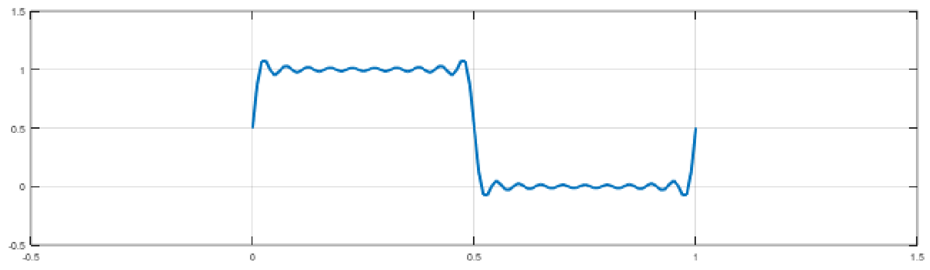


10.7. Noise



10.8. Normalized Pulse

ω_s



10.9. Butterworth Filter

DSP Related.COM

11. Plotting the DFT

11.1. With Python

11.1.1. Plotting the DFT

In this notebook we will look at the practical issues associated to plotting the DFT and in particular the DFT of real-world signals. We will examine how to map the DFT coefficients to real-world frequencies and we will investigate the frequency resolution of the DFT and the effects of zero padding.
As a quick reminder, the definition of the DFT for a length-NN signal is:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk},\ k = 0, 1, ..N-1$$

FFT Module

In Python, we will use the fft module in Numpy to compute the DFT

```
# First the usual bookkeeping
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
from scipy.io.wavfile import write
```

```
# Sets the size of the output plots
plt.rcParams["figure.figsize"] = (14,4)
```

Typically, we will take a vector of data points, compute the DFT and plot the magnitude of the result. For instance, consider the DFT of a linear ramp:

```
x = np.arange(0, 10.2, 0.2) - 5
#draw figure
plt.stem(x);
```

```
X = np.fft.fft(x);
plt.stem(abs(X));
```

11.1.2. Positive and negative frequencies

The coefficient number `k` indicates the contribution (in amplitude and phase) of a sinusoidal component of frequency

$$\omega_k = \frac{2\pi}{N}k$$

Because of the rotational symmetry of complex exponentials, a positive frequency ω between π and 2π is equivalent to a negative frequency of $\omega - 2\pi$; this means that half of the DFT coefficients correspond to

negative frequencies and when we concentrate on the physical properties of the DFT it would probably make more sense to plot the coefficients centered around zero with positive frequencies on the right and negative frequencies on the left.
The reason why this is not usuall done are many, including

- convenience
- since we are manipulating finite-length signals, the convention dictates that we start at index zero
- when dealing with real-valued data, the DFT is symmetric in magnitude, so the first half of the coefficients is enough
- if we're looking for maxima in the magnitude, it's just easier to start at zero.

There is also another subtle point that we must take into account when shifting a DFT vector: **we need to differentiate between odd and even length signals**. With $k = 0$ as the center point, odd-length vectors will produce symmetric data sets with $(N1)/2$ points left and right of the oring, whereas even-length vectors will be asymmetric, with one more point on the positive axis; indeed, the highest positive frequency for even-length signals will be equal to $\omega_{N/2} = \pi$. Since the frequencies of π and $-\pi$ are identical, we can copy the top frequency data point to the negative axis and obtain a symmetric vector also for even-length signals. Here is a function that does that:

```python
def dft_shift(X):
    N = len(X)
    if (N % 2 == 0):
        # even-length: return N+1 values
        return np.concatenate((X[(N/2):], X[:(N/2)+1]))
    else:
        # odd-length: return N values
        return np.concatenate((X[int((N+1)/2):], X[:int((N-1)/2)]))
```

```python
plt.stem(abs(dft_shift(X)));
```

While the function does shift the vector, the indices are still from zero to N—1N—1. Let's modify it so that we return also the proper values for the indices:

```python
def dft_shift(X):
    N = len(X)
    if (N % 2 == 0):
        # even-length: return N+1 values
        return (np.arange(-int(N/2), int(N/2) + 1),
                np.concatenate((X[int(N/2):], X[:int(N/2)+1])))
    else:
        # odd-length: return N values
        return (np.arange(-int((N-1)/2), int((N-1)/2) + 1),
                np.concatenate((X[int((N+1)/2):], X[:int((N+1)/2)])))
```

```python
n, y = dft_shift(X)
plt.stem(n, abs(y));
```

11.1.3. Mapping the DFT index to real-world frequencies

The next step is to use the DFT to analyze real-world signals. As we have seen in previous examples, what we need to do is set the time interval between samples or, in other words, set the "clock" of the system. For audio, this is equivalent to the sampling rate of the file.

```python
# import IPython  not used in orgmode
from scipy.io import wavfile
Fs, x = wavfile.read("./sound/piano.wav")
# IPython.display.Audio(x, rate=Fs) not used in orgmode
```

Play piano sound
In order to look at the spectrum of the sound file with a DFT we need to map the digital frequency ”bins” of the DFT to real-world frequencies.
The $k-th$ basis function over $\mathbb{C}^N$ completes $k-periods$ over N samples . If the time between samples is $1/Fs$, then the real-world frequency of the $k-th$ basis function is periods over time, namely $k(F_s/N)$.
Let’s remap the DFT coefficients using the sampling rate:

```python
def dft_map(X, Fs, shift=True):
    resolution = float(Fs) / len(X)
    if shift:
        n, Y = dft_shift(X)
    else:
        Y = X
        n = np.arange(0, len(Y))
    f = n * resolution
    return f, Y
```

```python
# let's cut the signal otherwise it's too big
x = x[:32768]
X = np.fft.fft(x);
f, y = dft_map(X, Fs)
plt.plot(f, abs(y));
```

The plot shows what a spectrum analyzer would display. We can see the periodic pattern in the sound, like for all musical tones. If we want to find out the original pitch we need to zoom in in the plot and find the first peak. This is one of the instances in which shifting the DFT does not help, since we’ll be looking in the low-frequency range. So let’s re-plot withouth the shift, but still mapping the frequencies:

```python
X = np.fft.fft(x);
f, y = dft_map(X, Fs, shift=False)
plt.plot(f[:2000], abs(y[:2000]));
```

We can see that the first peak is in the vicinity of 200Hz; to find the exact frequency (to within the resolution afforded by this DFT) let’s find the location

```python
dft_resolution = float(Fs)/ len(x)
print("DFT resolution is", dft_resolution, "Hz")

# let's search up to 300Hz
max_range = int(300 / dft_resolution)
ix = np.argmax(abs(y[:max_range]))
pitch = f[ix]
print("the note has a pitch of", pitch, "Hz")
```

Concert Pitch

So the note is a A, half the frequency of concert pitch.

11.1.4. Zero-padding

Since the resolution of a DFT depends on the length of the data vector, one may erroneously assume that, by artificially extending a given data set, the resulting resolution would improve. Note that here we're not talking about collecting more data; rather, we have a data set and we append zeros (or any other constant value) to the end of it. This extension is called zero-padding.
The derivation of why zero-padding does not increase the resolution is detailed in the book. Here we will just present a simple example.

```
N = 256
Delta = 2*np.pi / N
n = np.arange(0, N)

# main frequency (not a multiple of the fundamental freq for the space)
omega = 2*np.pi / 10

x = np.cos(omega * n) + np.cos((omega + 3*Delta) * n)
plt.plot(abs(np.fft.fft(x))[:100]);
```

we can tell the two frequencies apart and, if you zoom in on the plot, you will see that they are indeed three indices apart. Now let's build a signal with two frequencies that are less than $\Delta\Delta$ apart:

```
x = np.cos(omega * n) + np.cos((omega + 0.5*Delta) * n)
plt.plot(abs(np.fft.fft(x))[:100]);
```

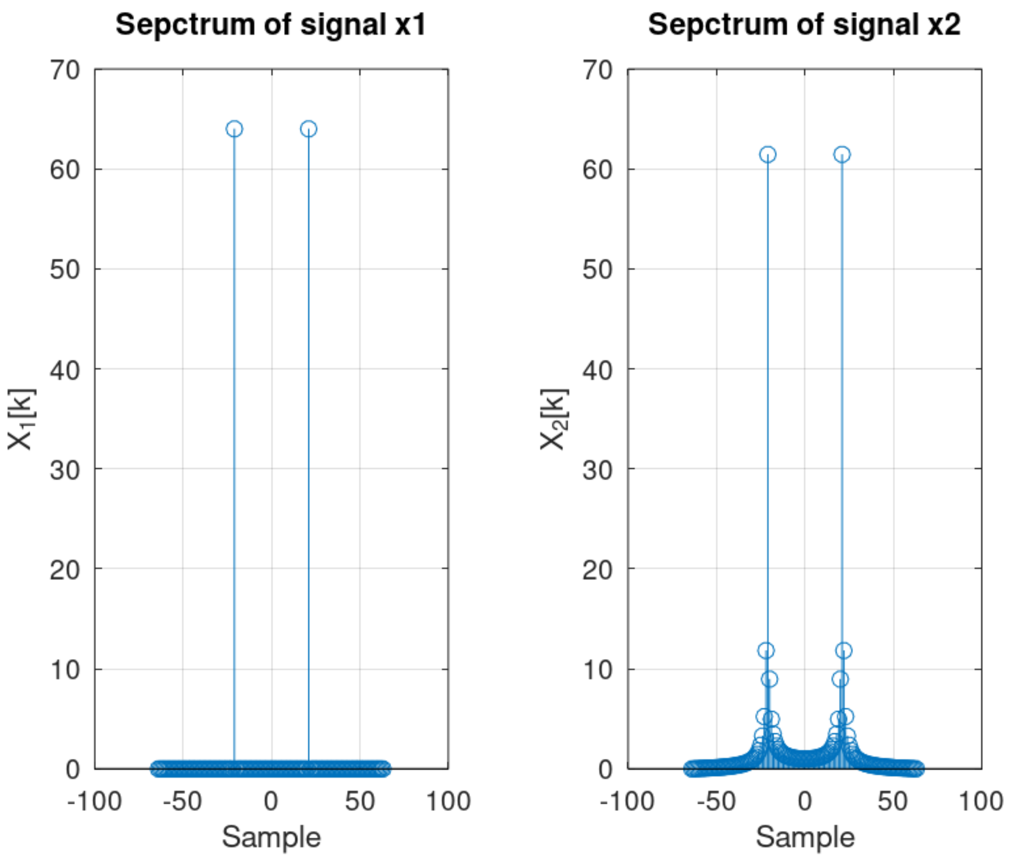
The two frequencies cannot be resolved by the DFT. If you try to increase the data vector by zero padding, the plot will still display just one peak:

```
xzp = np.concatenate((x, np.zeros(2000)))
plt.plot(abs(np.fft.fft(xzp))[:500]);
```

11.2. With Matlab/Octave

```
N=128;
fo1=21/128;
fo2=21/127;
n=0:N-1;
x1=cos(2*pi*fo1*n);
x2=cos(2*pi*fo2*n);
X1=fft(x1);                               # Compute the dft of X1 using FFT algorithmw
X2=fft(x2);                               # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )              # Do not open the graphic window in org
subplot(1,2,1),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid on;
xlabel("Sample");
ylabel("X_1[k]");
title("Sepctrum of signal x1");
subplot(1,2,2), stem(n-N/2,fftshift(abs(X2)))
grid on;
xlabel("Sample");
ylabel("X_2[k]");
title("Sepctrum of signal x2");

# Org-Mode specific setting
```

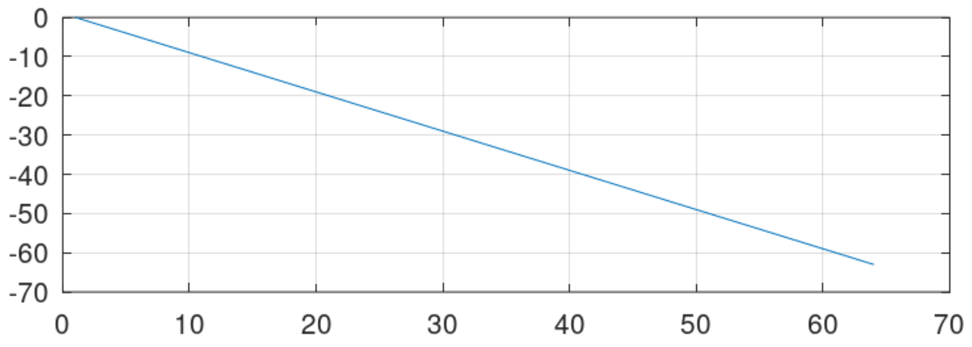
```
print -dpng ./image/oct_fft.png;
ans = "./image/oct_fft.png";
```

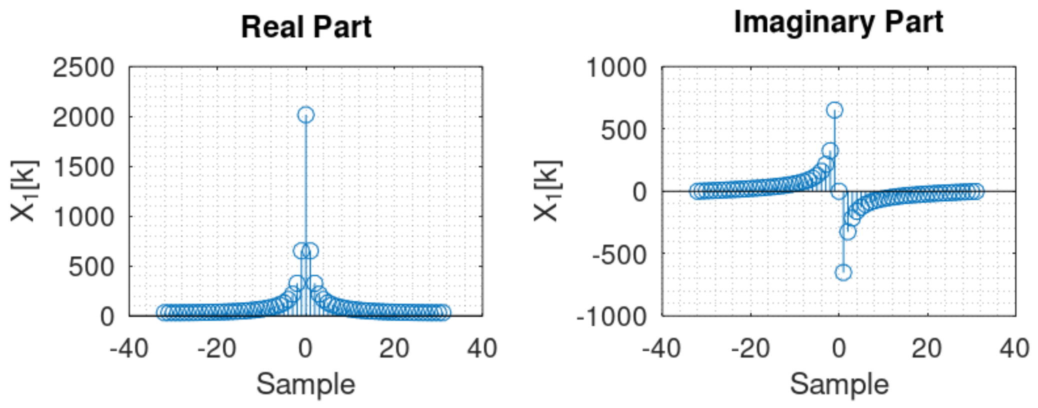


11.2.1. Homework 4

```
N=64;
n=0:N-1;

x1=ones(N,1);
X1=fft(x1);                                   # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                  # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
```

```
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw4a_fft.png;
ans = "./image/hw4a_fft.png";
```





```
x1=[1 2 3 4 5]
X1=fft(x1);                                     # Compute the dft of X1 using FFT algorithmw
X2=fft(X1);


ans = X2/5;
```

```
1    5    4    3    2
```

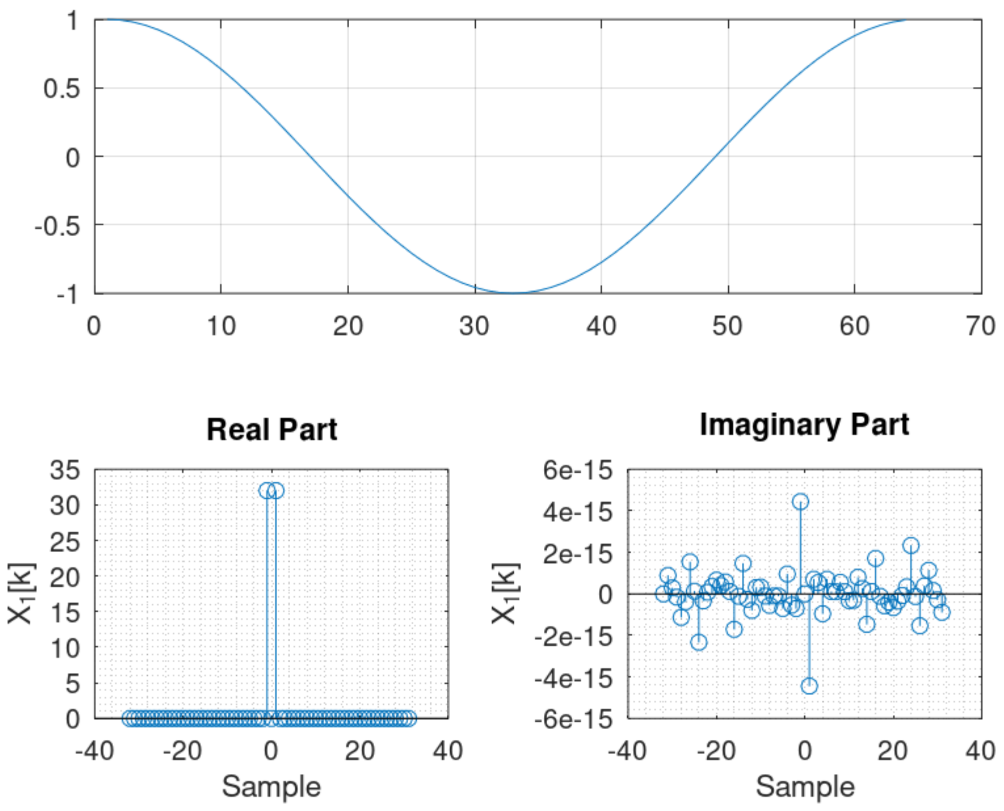
11.2.2. Homework 6

```
N=64;
fo1=4*60/64;
n=0:N-1;

x1=(-1)
X1=fft(x1);                                     # Compute the dft of X1 using FFT algorithmw
```

```
# Graphik
figure( 1, "visible", "off" )                # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6a_fft.png;
ans = "./image/hw6a_fft.png";
```

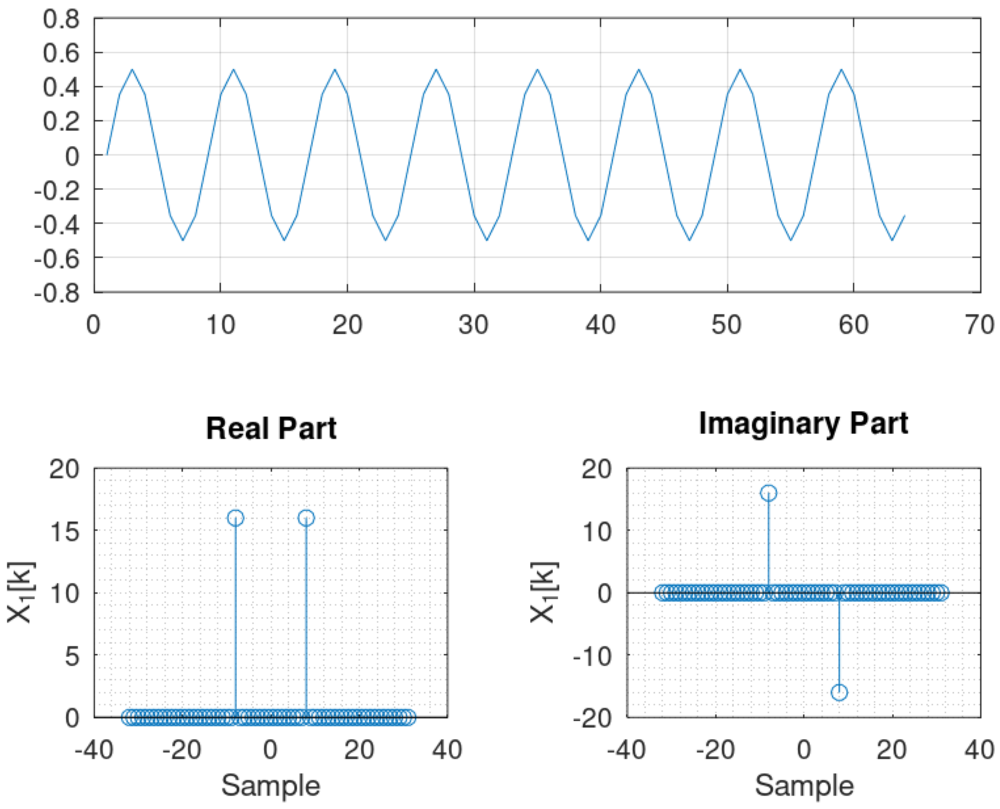
```
N=64;
fo1=8;
n=0:N-1;

x1=0.5*sin(2*pi/N*fo1*n);
X1=fft(x1);                                 # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6b_fft.png;
```

```
ans = "./image/hw6b_fft.png";
```



```
N=64;
fo1=4;
n=0:N-1;

x1=2*cos(2*pi/N*fo1*n);
X1=fft(x1);                                    # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                   # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1)
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
```

```
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6d_fft.png;
ans = "./image/hw6d_fft.png";
```



12. Dual-tone multi-frequency (DTFM) signaling

DTMF signaling is the way analog phones send the number dialed by a user over to the central phone office. This was in the day before all-digital networks and cell phones were the norm, but the method is still used for in-call option selection (”press 4 to talk to customer service”…).
The mechanism is rather clever: the phone’s keypad is arranged in a $4x3$ grid and each button is associated to two frequencies according to this table:

	1209 Hz	336 Hz	477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

The frequencies in the table have been chosen so that they are ”coprime”; in other words, no frequency is a multiple of any other, which reduces the probability of erroneously detecting the received signals due to interference. When a button is pressed, the two corresponding frequencies are generated simultaneously and sent over the line. For instance, if the digit ’1’ is pressed, the generated signal will be:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk},\ k = 0, 1, ..N-1$$

FFT Module

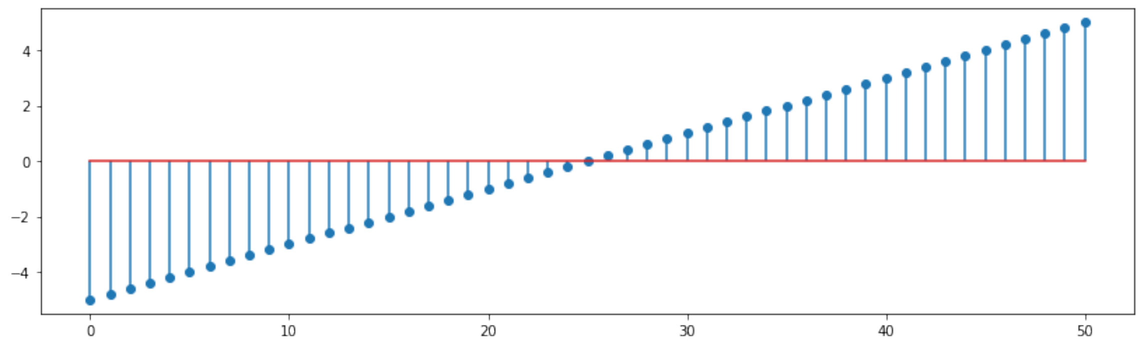
In Python, we will use the fft module in Numpy to compute the DFT

```
# First the usual bookkeeping
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
from scipy.io.wavfile import write
```

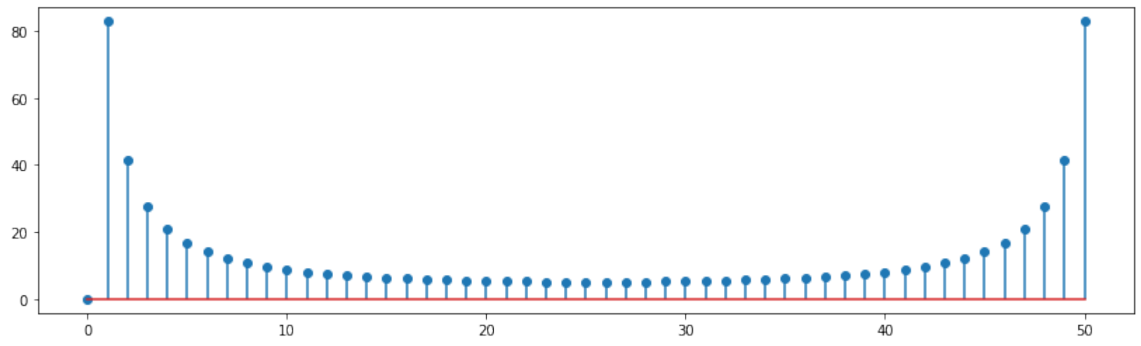
```
# Sets the size of the output plots
plt.rcParams["figure.figsize"] = (14,4)
```

Typically, we will take a vector of data points, compute the DFT and plot the magnitude of the result. For instance, consider the DFT of a linear ramp:

```
x = np.arange(0, 10.2, 0.2) - 5
#draw figure
plt.stem(x);
```



```
X = np.fft.fft(x);
plt.stem(abs(X));
```



13. Goethe's temperature measurement

- Key Characteristic of a Digital Signal
    1. Series of measurements, taken at regular interval
    2. Each sample has a discrete amplitut
- Moving Average

$$\begin{aligned} y[n] &= \frac{1}{N}\sum_{m=0}^{N-1} x[n-m] \\ &= \frac{1}{N}x[n] + \underbrace{\frac{1}{N}\sum_{m=1}^{N-1} x[n-m] + \frac{1}{N}x[n-N]}_{y[n-1]} - \frac{1}{N}x[n-N] \\ &= y[n-1] + \frac{1}{N}(x[n] - x[n-N]) \end{aligned}$$

- n: average of the last capital N obervations
- N: window of observation over which average is computed
- Engineers Scientist Guide Chapter 15

14. Karplus-Strong Algorithm

The Karplus-Strong algorithm is a simple digital feedback loop with an internal buffer of samples. The buffer is filled with a set of initial values and the loop, when running, produces an arbitraryly long output signal. Although elementary, the K-S loop can be used to synthesize interesting musical sounds. Let's start with a basic implementation of K-S loop:

```python
def KS_1(x, N):
    # given the initial buffer x, produce a N-sample output
    #  by concatenating identical copies of the buffer
    y = x
    while len(y) < N:
        # keep appending until we reach or exceed the required length
        y = np.append(y, x)
    # trim the excess
    y = y[0:N+1]
    return y
```

First we need to include the necessary Python libraries:

```python
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
from scipy.io.wavfile import write
```

Set the size of the output plots

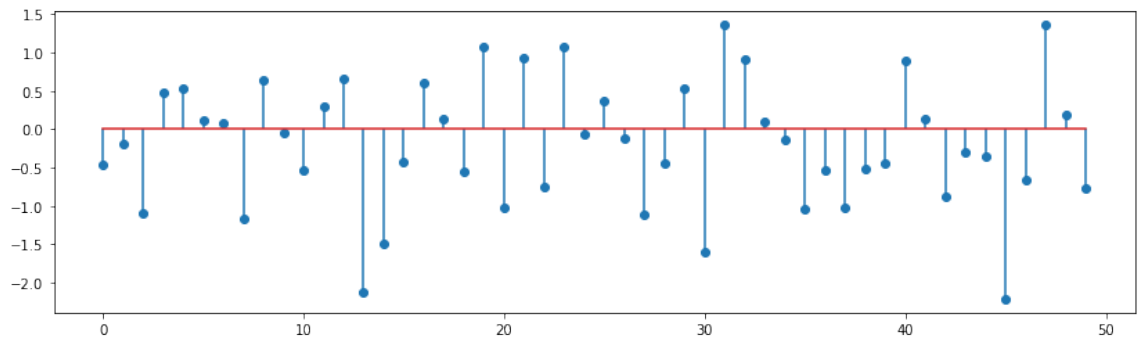
```python
plt.rcParams["figure.figsize"] = (14,4)
```

Then, since we're playing audio, we need to set the internal "clock" of the system, aka the sampling rate:

```python
Fs = 16000 # 16 KHz sampling rate
```

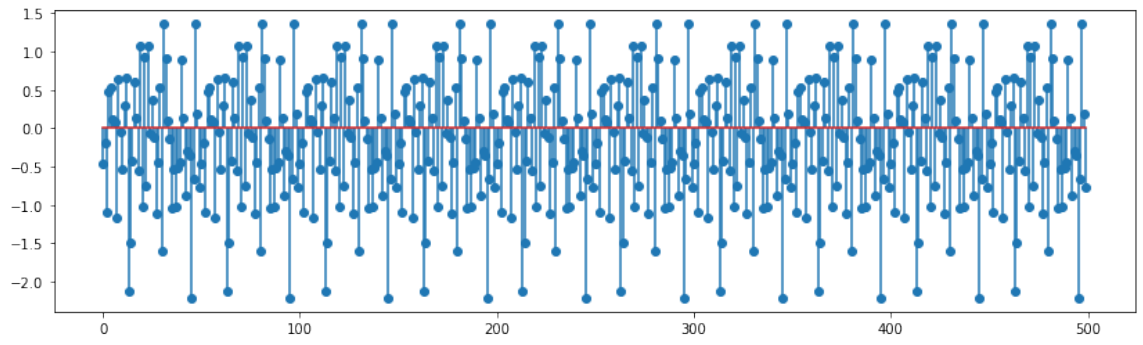
With this sampling rate, since the period of the generated signal is equal to the length of the inital buffer, we will be able to compute the fundamental frequency of the resulting sound. For instance, if we init the K-S algorithm with a vector of 50 values, the buffer will fit 16000/50=320 times in a second's worth of samples or, in other words, the resulting frequency will be 320Hz, which corresponds roughly to a E4 on a piano.
We still haven't talked about what to use as the initial values for the buffer. Well, the cool thing about K-S is that we can use pretty much anything we want; as a matter of fact, using random values will give you a totally fine sound. As a proof, consider this initial data set:

```python
b = np.random.randn(50)

#draw figure
plt.stem(b);
```



Let's now generate a 2-second audio clip:

```
y = KS_1(b, Fs * 2)

# we can look at a few periods:
plt.stem(y[0:500]);
```



```
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test1.w
```

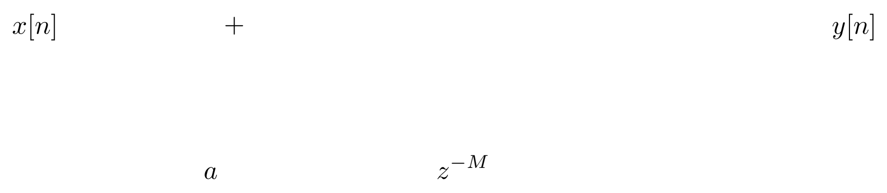
Play wave 1

```
# let's play an octave lower: just double the initial buffer's length
b = np.random.randn(100)
y = KS_1(b, Fs * 2)
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test2.w
```

Play wave 2

The Block Diagram

OK, so the K-S algorithm works! From the signal processing point of view, we can describe the system with the following block diagram.



The output can be expressed as

$$y[n] = x[n] + x[n-M]$$

assuming that the signal is the finite support signal

$$x[n] = \begin{cases} 0 & \text{for } n < 0 \\ b_n & \text{for } 0 \le n \le M \\ 0 & \text{for } n \ge M \end{cases}$$

Let's implement the K-S algorithm as a signal processing loop

```
def KS_2(x, N):
    # length of the input
    M = len(x)
    # prepare the output
    y = np.zeros(N)
    # this is NOT an efficient implementation, but it shows the general principle
    # we assume zero initial conditions (y[n]=0 for n < 0)
    for n in range(0, N):
        y[n] = (x[n] if n < M else 0) + (y[n-M] if n-M >= 0 else 0)
    return y
```

```
# let's play an octave lower: just double the initial buffer's length
b = np.random.randn(50)
y = KS_2(b, Fs * 2)
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test3.w
```

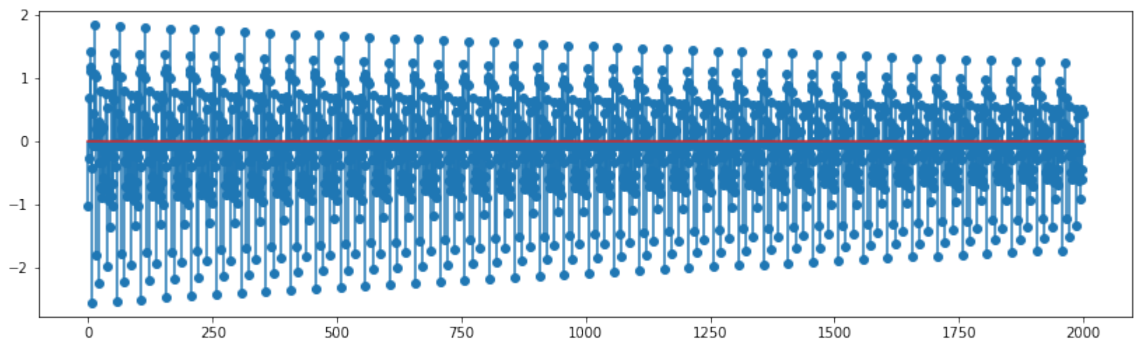
Play wave 3
By looking at block diagram we can see a simple modification that adds a lot of realism to the sound: by setting to a value close to but less that one, we can introuce a decay in the note that produces guitar-like sounds:

$$y[n] = x[n] + \alpha y[n-M]$$

```
def KS_3(x, N, alpha = 0.99):
  M = len(x)
  y = np.zeros(N)
  #
  for n in range(0, N):
      y[n] = (x[n] if n < M else 0) + alpha * (y[n-M] if n-M >= 0 else 0)
  return y
```

If we now plot the resulting K-S output, we can see the decaying envelope:

```
y = KS_3(b, Fs * 2)

# we can look at a few periods:
plt.stem(y[0:2000]);
```



```
# let's play an octave lower: just double the initial buffer's length
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test4.w
```

Play wave 4
There is just one last detail (the devil's in the details, here as everywhere else). Consider the output of a dampened K-S loop; every time the initial buffer goes through the loop, it gets multiplied by so that we can write

$$y[n] = \alpha^{\frac{n}{M}} x[n] + \alpha y[n - M]$$

(think about it and it will make sense). What that means is that the decay envelope is dependent on both and or, in other words, the higher the pitch of the note, the faster its decay. For instance:

```
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test5.w
```

Play wave 5

```
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test6.w
```

Play wave 6
This is no good and therefore we need to compensate so that, if α is the same, the decay rate is the same. This leads us to the last implementation of the K-S algorithm:

```
def KS(x, N, alpha = 0.99):
    # we will adjust alpha so that all notes have a decay
    #  comparable to that of a buf len of 50 samples
    REF_LEN = 50
    M = len(x)
    a = alpha ** (float(M) / REF_LEN)
    y = np.zeros(N)
    #
    for n in range(0, N):
        y[n] = (x[n] if n < M else 0) + a * (y[n-M] if n-M >= 0 else 0)
    return y
```

```
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test7.w
```

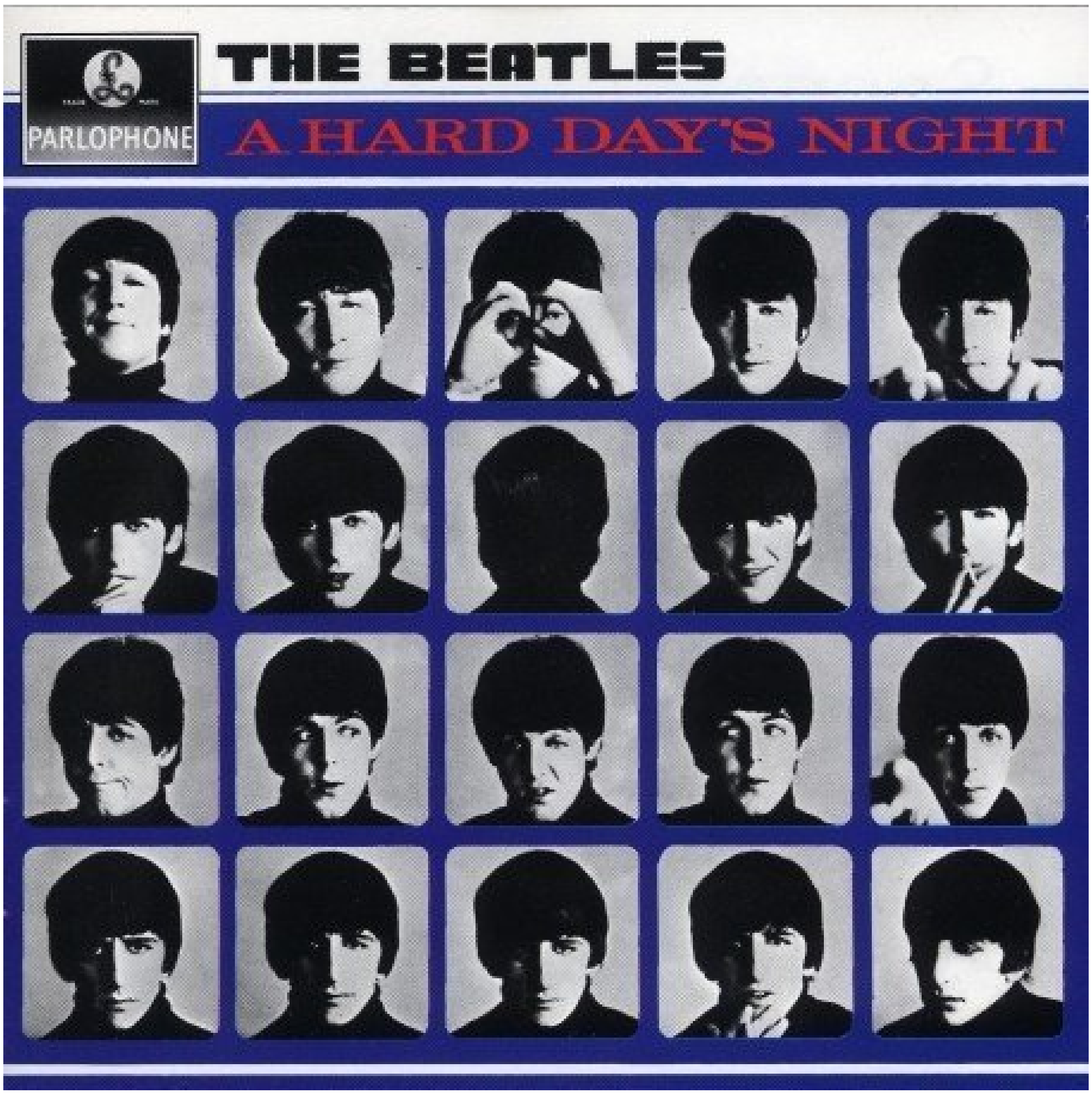
Play wave 7

```
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test8.w
```

Play wave 8

15. Playing Music

Let's now play some cool guitar and, arguably, no guitar chord is as cool as the opening chord of "A Hard Day's Night", by The Beatles.



Much has been written about the chord (which, in fact, is made up of 2 guitars, one of which a 12-string, a piano and a bass) but to keep things simple, we will accept the most prevalent thesis which states that the notes are $D_{3,F3,G3,G4,A4,C5}$ and G_5. To give it a "wider" feeling we will add another D_2 below.

In Western music, where equal temperament is used, A_4 is the reference pitch at a frequency at 440Hz. All other notes can be computed using the formula $f(n) = A_4 \times 2n/12$ where is the number of half-tones between A_4 and the desired note. The exponent n is positive if the note is above A_4 and negative otherwise.

Each note is generated using a separate Karplus-Strong algorithm. We try to mix the different "instruments" by assigning a different gain to each note. Also, we sustain Paul's D note on the bass a bit longer by changing the corresponding decay factor.

```
  def freq(note):
    # general purpose function to convert a note  in standard notation
    #  to corresponding frequency
    if len(note) < 2 or len(note) > 3 or \
        note[0] < 'A' or note[0] > 'G':
        return 0
    if len(note) == 3:
        if note[1] == 'b':
            acc = -1
        elif note[1] == '#':
            acc = 1
        else:
            return 0
        octave = int(note[2])
    else:
        acc = 0
        octave = int(note[1])
    SEMITONES = {'A': 0, 'B': 2, 'C': -9, 'D': -7, 'E': -5, 'F': -4, 'G': -2}
    n = 12 * (octave - 4) + SEMITONES[note[0]] + acc
    f = 440 * (2 ** (float(n) / 12.0))
    #print note, f
    return f


def ks_chord(chord, N, alpha):
    y = np.zeros(N)
    # the chord is a dictionary: pitch => gain
    for note, gain in chord.items():
        # create an initial random-filled KS buffer the note
        x = np.random.randn(int(np.round(float(Fs) / freq(note))))
        y = y + gain * KS(x, N, alpha)
    return y
```

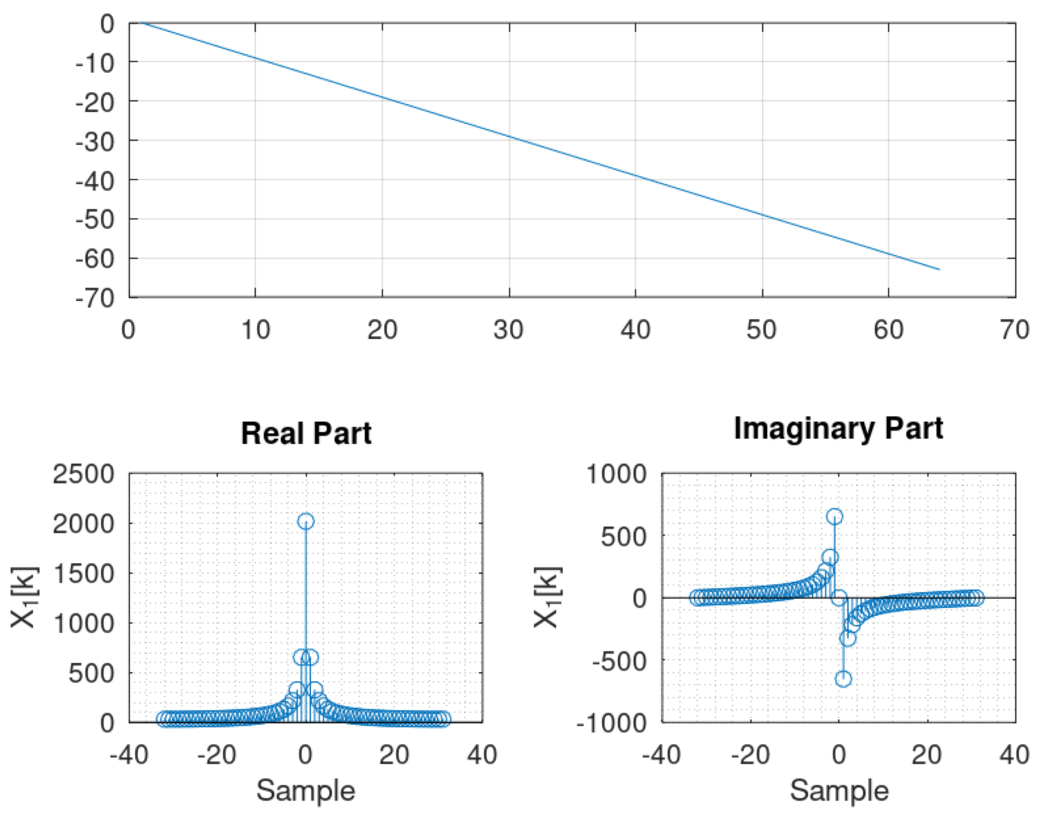
```
  # A Hard Day's Night's chord
hdn_chord = {
    'D2' : 2.2,
    'D3' : 3.0,
    'F3' : 1.0,
    'G3' : 3.2,
    'F4' : 1.0,
    'A4' : 1.0,
    'C5' : 1.0,
    'G5' : 3.5,
}

# write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/test4
write('/home/christian/Daten/learning/Digital-Signal-Processing/sp4comm/orgmode/appendix/sound/hdn.wav
```

A Hard Day's Night openeing chord

16. Homework Module 3

16.1. Excersice 4

```
N=64;
fo1=4*60/64;
n=0:N-1;

x1=(-1)*n
X1=fft(x1);                                # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )               # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw4a_fft.png;
ans = "./image/hw4a_fft.png";
```



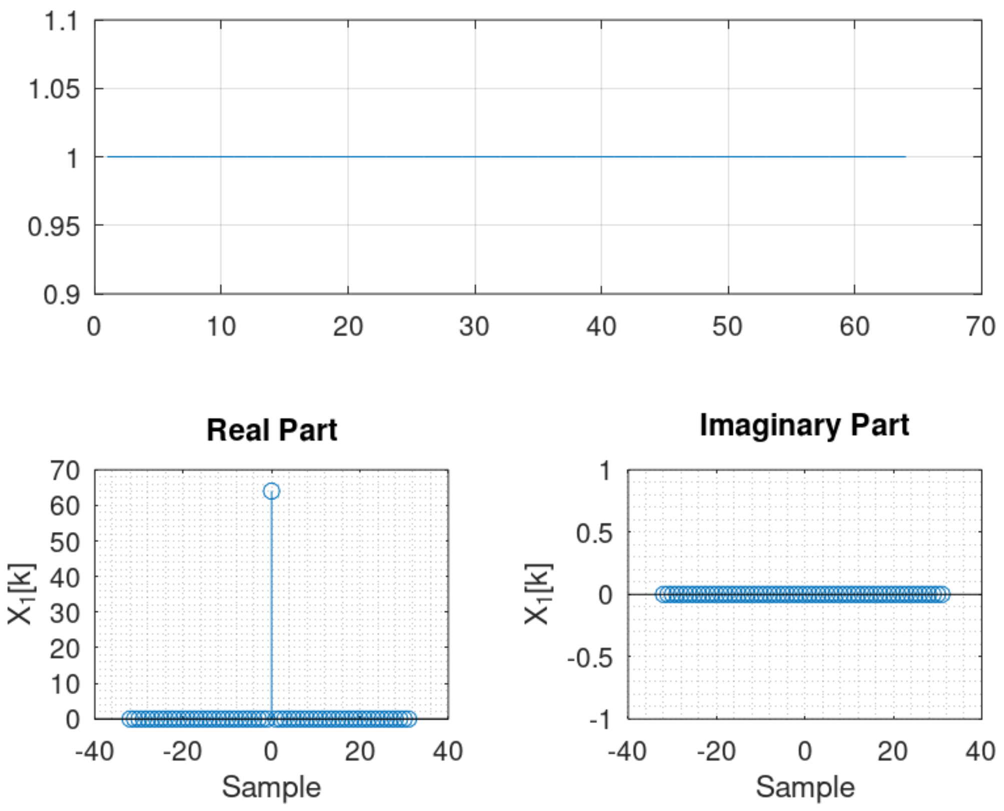
```
x1=[1 2 3 4 5]
X1=fft(x1);                                     # Compute the dft of X1 using FFT algorithmw
X2=fft(X1);


ans = X2/5;
```

```
1   5   4   3   2
```

16.2. Excersice 6

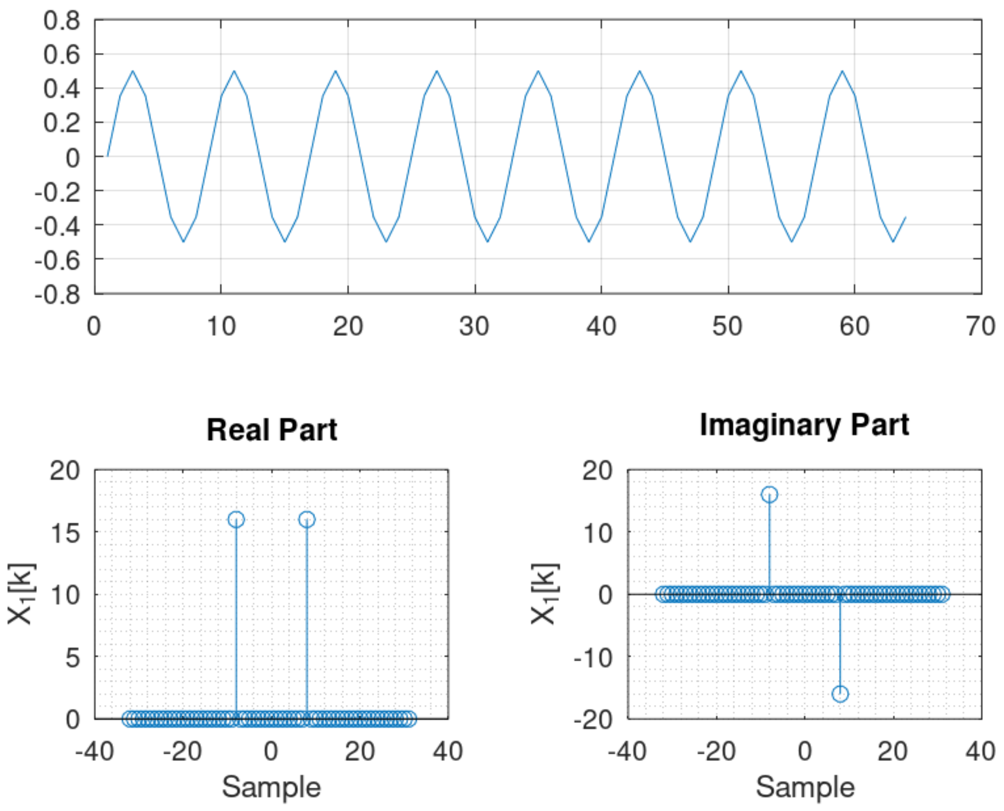
```
N=64;
n=0:N-1;

x1=ones(N,1);
X1=fft(x1);                                     # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                    # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
```

```
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6a_fft.png;
ans = "./image/hw6a_fft.png";
```



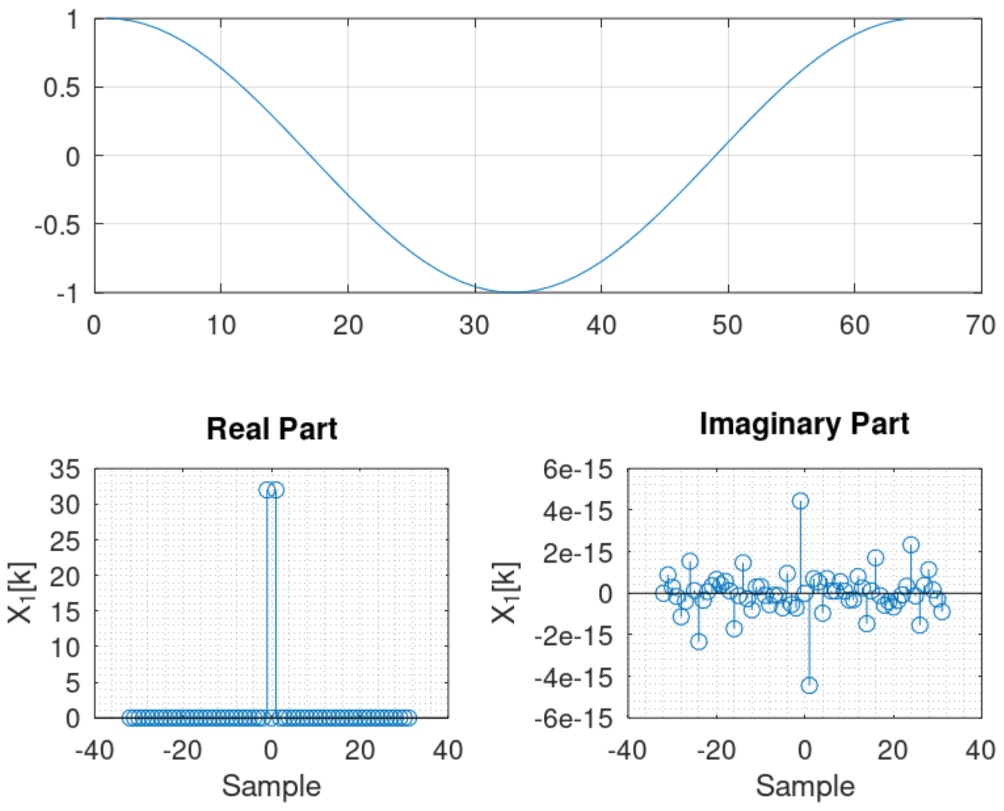
```
N=64;
fo1=8;
n=0:N-1;

x1=0.5.*sin(2*pi*fo1*n/N);
X1=fft(x1);                                   # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                  # Do not open the graphic window in org

subplot(2,2,[1,2])
```

```
plot(x1)
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6d_fft.png;
ans = "./image/hw6d_fft.png";
#ans = X1'
```



```
N=64;
fo1=8;
n=0:N-1;

x1=0.5*sin(2*pi/N*fo1*n);
```

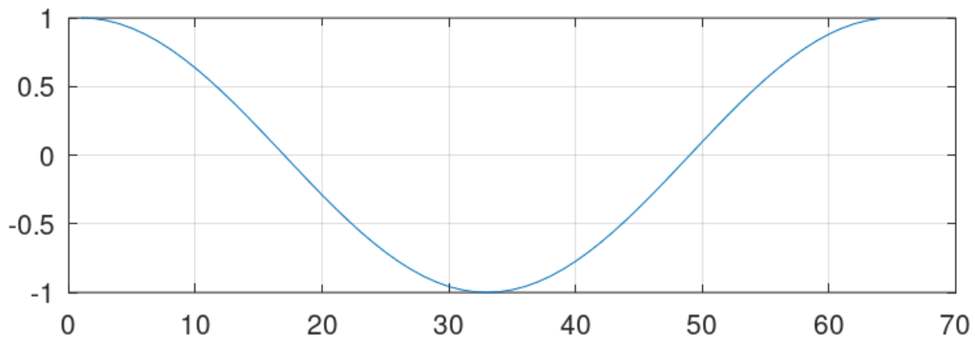
```
X1=fft(x1);                                   # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )                  # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6b_fft.png;
ans = "./image/hw6b_fft.png";
```

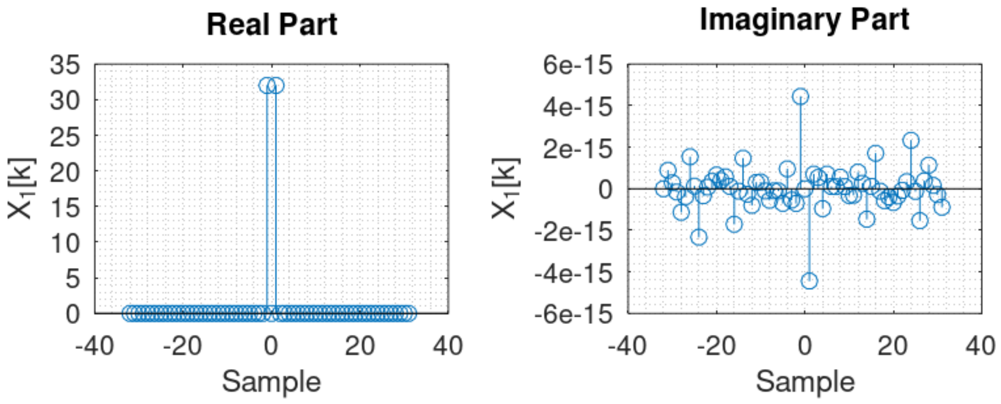


```
N=64;
fo1=8;
fo2=4;
n=0:N-1;

f1 = @(n) 0.5*sin(2*pi/N*fo1*n);
f2 = @(n) 2*cos(2*pi/N*fo1*n);
f3 = @(n) 1

norm1 = sum( f1([0:63]) .* f1([0:63]));
norm2 = sum( f2([0:63]) .* f2([0:63]));
norm3 = sum( f3([0:63]) .* f2([0:63]));

# ans = norm1;
# ans = norm2;
# ans = norm3;
ans = (norm1 + norm2 + norm3);
```

```
135.9999999999999 135.9999999999999 135.9999999999999 135.9999999999999 511.9999999999997 31.99999999999998
127.9999999999999 8.000000000000004 32.00000000000001 32
```

```
N=64;
L=8;
M=8
n=0:N-1;

fo = L/M
x1=cos(2*pi/N*fo*n);
X1=fft(x1);                                # Compute the dft of X1 using FFT algorithmw

# Graphik
figure( 1, "visible", "off" )               # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6b_fft.png;
ans = "./image/hw6b_fft.png";
```


1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70


Real Part
$X_1[k]$
Sample
Imaginary Part
$X_1[k]$
Sample

17. Links

- DSP Professor Murugan Pallikonda Rajasekaran
- Software Defined Radio with HackRF
- VSB - Modulation (MVB)
- Phase-Shift Method-Based Optical VSB
- Understanding the "Phasing Method" of Single Sideband Demodulation
- Einseitenbandmodulation

17.1. All About Circuits

17.1.1. Filter Design

- Practical FIR Filter Design: Part1 - Design with Octave - see also m-files/fir-filter.m
- Practical FIR Filter Design: Part2 Implementing Your Filter
- Design of FIR Filters Using the Frequency Sampling Method
- FIR Filter Design by Windowing: Concepts and the Rectangular Window
- From Filter Specs to Window Parameters in FIR Filter Design
- Design Examples of FIR Filters Using the Window Method

17.1.2. Software Defined Radio

- Practical Guide to Radio-Frequency Analysis and Design
- Introduction to Software-Defined Radio

17.1.3. Digital Modulation

- Digital Modulation: Amplitude and Frequency
- Digital Signal Processing in Scilab: How to Decode an FSK Signal
- How to Use I/Q Signals to Design a Robust FSK Decoder

18. Books

- Flatland: A Romance of Many Dimensions

Emacs 24.3.1 (Org mode 8.2.5h)