Systems

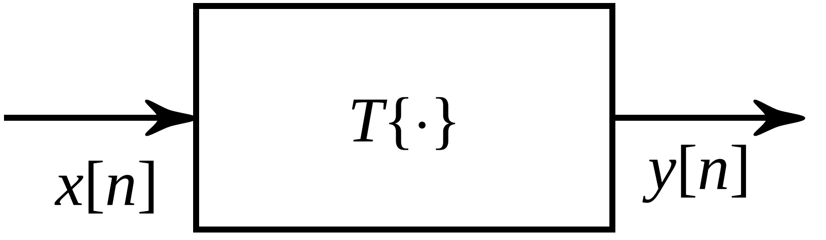
- A *system* is a transformation from one signal (called the input) to another signal (called the output or the response)
- Continuous-time systems with input signal ***x*** and output signal ***y*** (a.k.a. the response):
  - $y(t) = x(t) + x(t-1)$
  - $y(t) = x2(t)$

Squaring function can be used in sinusoidal demodulation

- Discrete-time system examples
  - $y[n] = x[n] + x[n-1]$
  - $y[n] = x2[n]$

The average of current input and delayed input is a simple filter

Systems

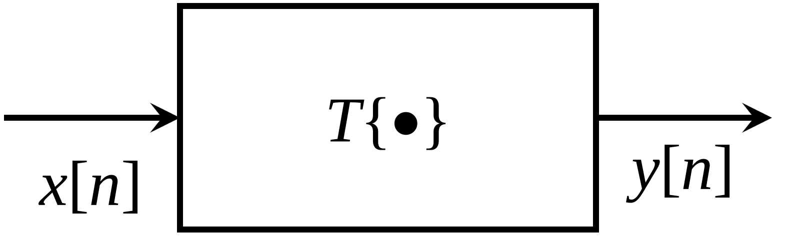


A discrete-time system is a transformation that maps an input sequence $x[n]$ into an output sequence $y[n]$.

System Characteristics

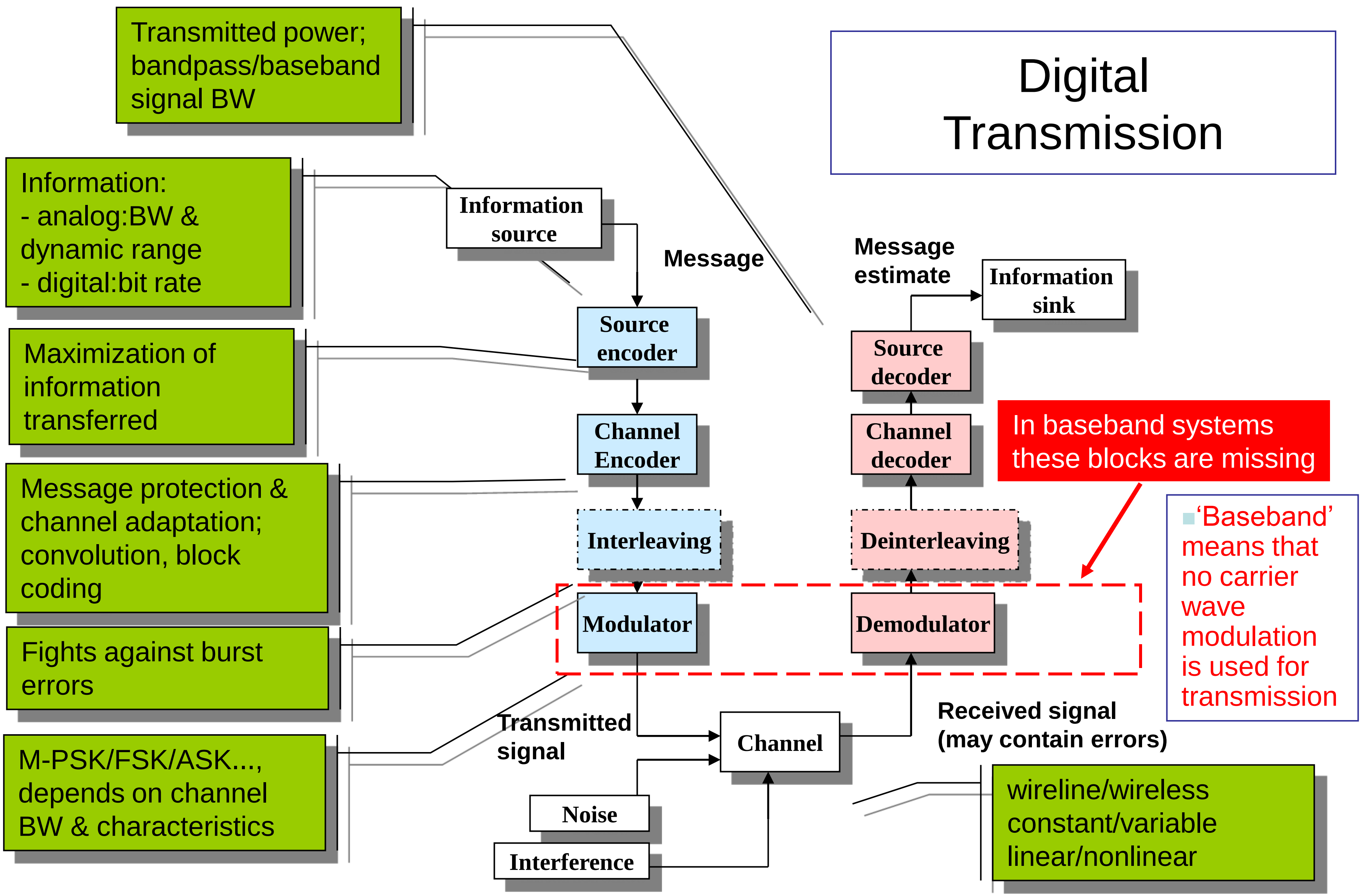
1. Linear vs. non-linear
2. Causal vs. non-causal
3. Time invariant

System Characteristics



1. Linear vs. non-linear
2. Time invariant vs. time variant
3. Causal vs. non-causal
4. Stable vs. unstable
5. Memoryless vs. state-sensitive
6. Invertible vs. non-invertible

Gambaran Sistem Komunikasi



Modulation/Coding Methods

- Digital PAM or Amplitude-Shift Keying (ASK)
- Phase Modulation
- Digital Phase Modulation or Phase-Shift Keying (PSK)
  - Binary PSK (BPSK)
  - Quadrature PSK (QPSK)
  - Differential PSK (DPSK)
  - Staggered Quadrature PSK (SQPSK)
- Quadrature Amplitude Modulation (QAM)
- Frequency-Shift Keying (FSK)
  - Continuous-Phase FSK (CPFSK)
- Amplitude Modulation (AM)
- Frequency Modulation (FM)
- Pulse Width Modulation (PWM)
- Pulse Position Modulation (PPM)
- Continuous-Phase Modulation (CPM)
- Minimum-Shift Keying (MSK)
- Pulse Amplitude Modulation (PAM)

- Fixed-Length Code Word
- Variable-Length Code Word
  - Entropy Coding → Huffman Coding
- Variable-to-Fixed Length Code Word
  - Lempel-Ziv Algorithm
- Temporal Waveform Coding
  - Pulse coded modulation (PCM)
    - Adaptive PCM (APCM)
  - Differential PCM (DPCM)
    - Adaptive DPCM (ADPCM)
  - Open-loop DPCM (D*PCM)
  - Delta modulation (DM) or 1-bit or 2-level DPCM
    - Linear DM (LDM)
    - Adaptive DM (ADM)
    - Continuously Variable Slope DM (CVSD)
- Model-Based Source Coding
  - Linear Predictive Coding (LPC)
- Spectral Waveform Coding
  - Subband Coding (SBC)
  - Transform Coding (TC)

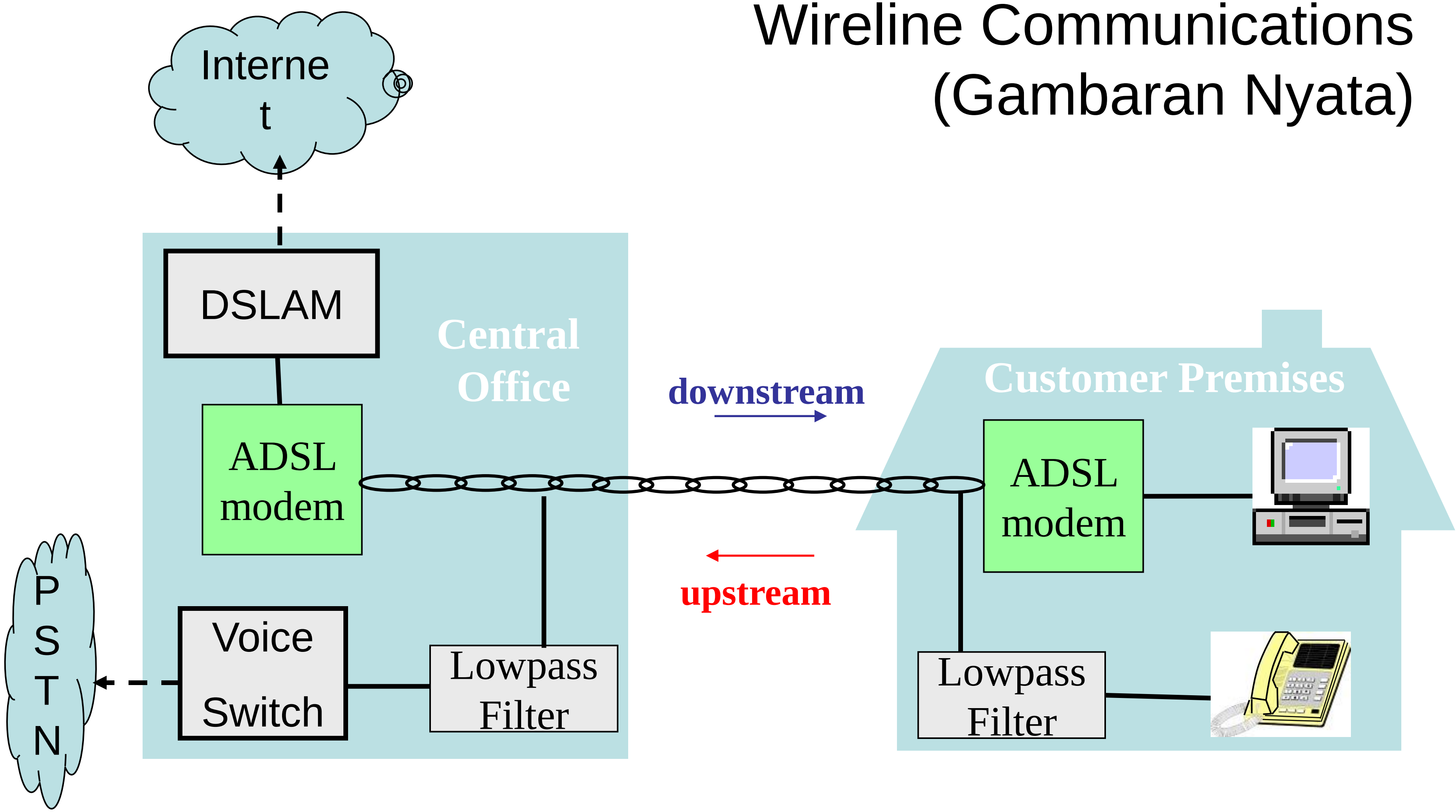
Communication Systems

- Voiceband modems (56k)
- Digital subscriber line (DSL) modems
  - ISDN: 144 kilobits per second (kbps)
  - Business/symmetric: HDSL and HDSL2
  - Home/asymmetric: ADSL and VDSL
- Cable modems
- Cell phones
  - First generation (1G): AMPS Analog
  - Second generation (2G): GSM, IS-95 (CDMA) Digital
  - Third generation (3G): cdma2000, WCDMA

Multiplexing

- Time Division Multiplexing (TDM)
- Frequency Division Multiplexing (FDM)
- Code Division Multiplexing (CDM)
  - Code Division Multiple Access (CDMA) or Spread Spectrum Multiple Access (SSMA)
- Orthogonal Frequency Division Multiplexing (OFDM)

Wireline Communications (Gambaran Nyata)



- HDSL High bit rate 1.544 Mbps in both directions
- ADSL Asymmetric 1-10 Mbps downstream, 0.5-1 Mbps up
- VDSL Very high bit rate, 22 Mbps downstream, 3 Mbps up

REPRESENTASI MACAM-MACAM SINYAL

DI MATLAB

Signal Time Base

There are a variety of ways of generating waveforms. Most require that you begin with a vector that represents a time base. Consider generating data with a 1000 Hz sample frequency, for example.

```
>> t=(0:1000)/1000;
>> whos
  Name      Size                    Bytes  Class
  t         1x1001                   8008  double array
```

The colon operator creates a 1001-element row vector that represents time from zero to one second in steps of one millisecond. The command whos displays the name and size of each current variable.

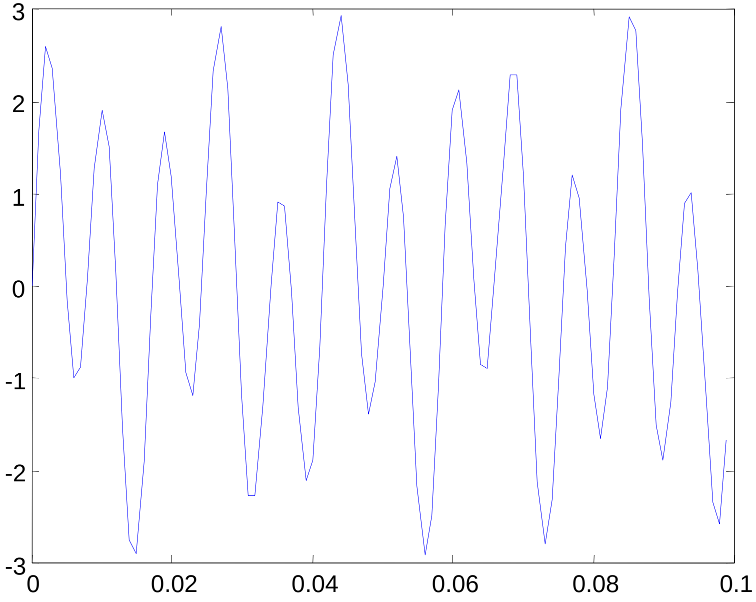
```
>> t=t';
>> whos
  Name      Size                    Bytes  Class
  t      1001x1                      8008  double array
```

The transpose operator (') changes the row vector to a column vector. The semicolon tells Matlab to compute but not to display the result.

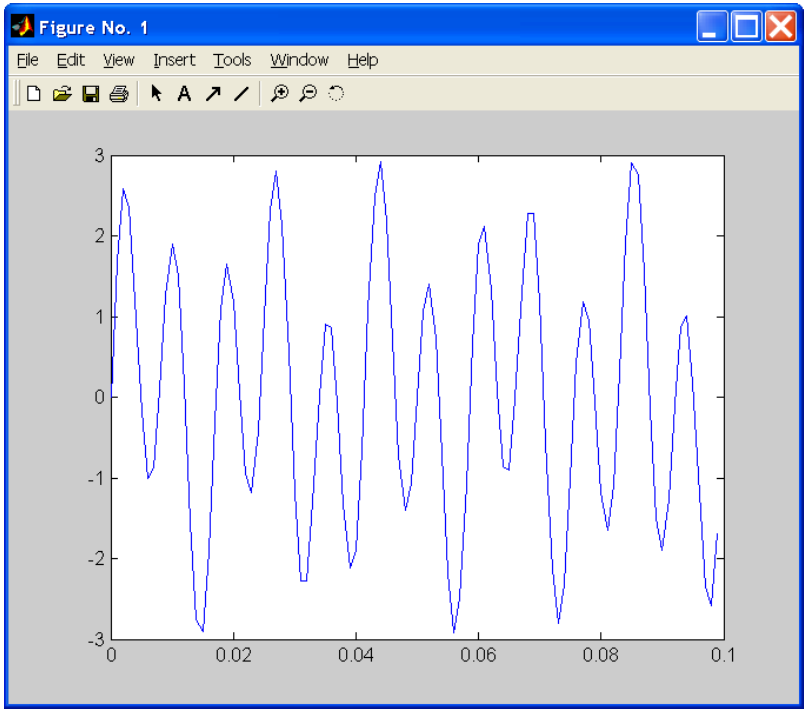
Basic Waveform Representation

Given *t* you can create a sample signal *y* consisting of two sinusoids, one at 50 Hz and one at 120 Hz with twice the amplitude.

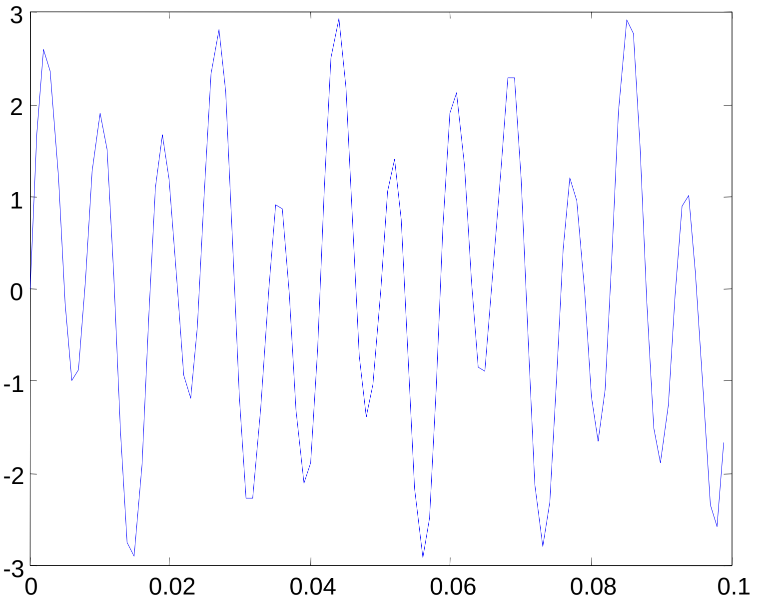
```
>> y = sin(2*pi*50*t) + 2*sin(2*pi*120*t);
>> plot(t(1:100),y(1:100));
```



Saving Plots



Edit > Copy Figure

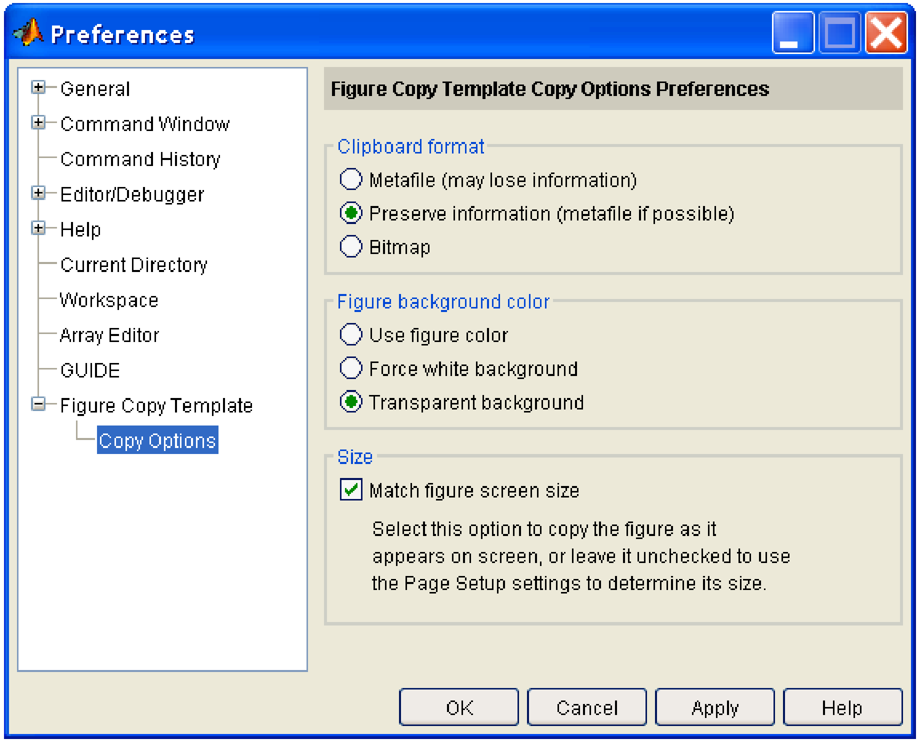


Matlab Figure as Powerpoint drawing object

Note: try to save plots as a *metafile*.

Copy Preferences

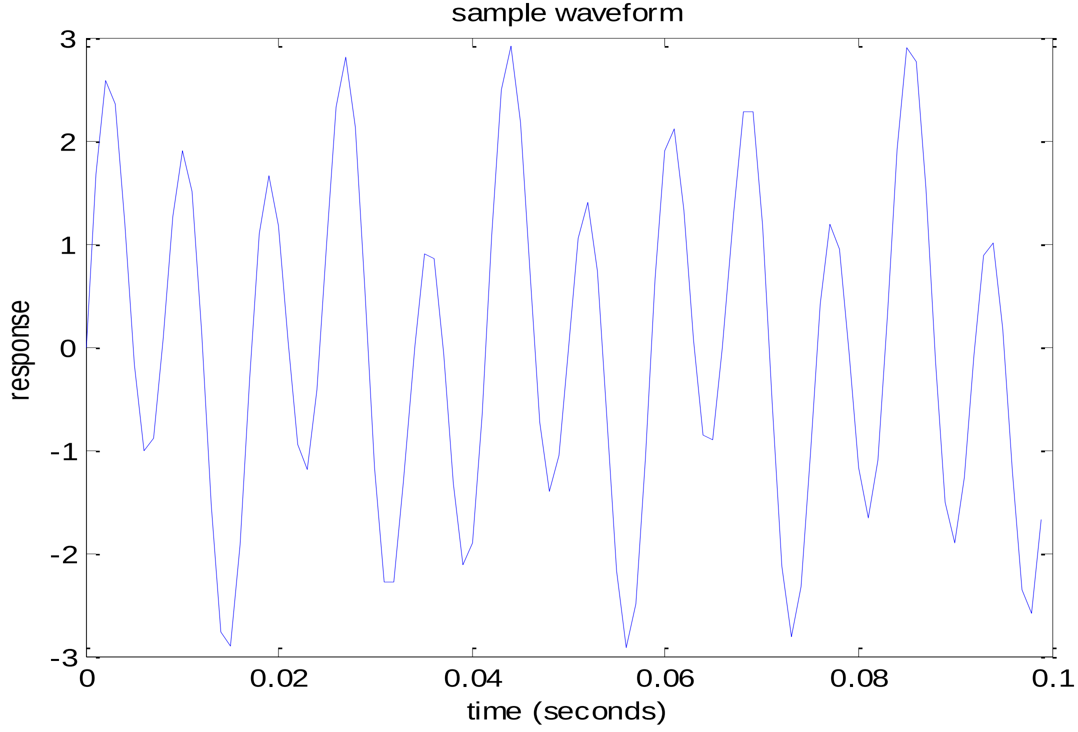
Edit > Copy Preferences



Labeling Plots

There are a variety of commands useful in labeling plots. Here are some examples:

```
>> xlabel('time (seconds)');
>> ylabel('response');
>> title('sample waveform');
```

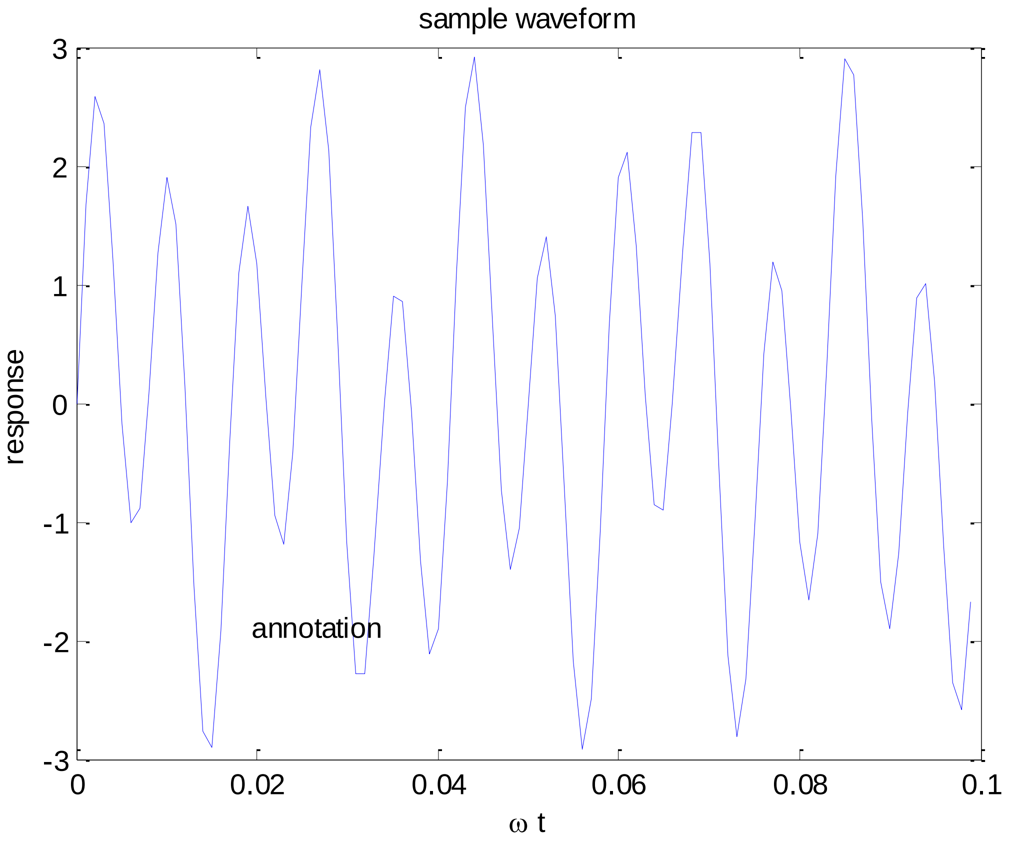


More on Labeling Plots

Greek letters and mathematical symbols can be included using TeX notation.

```
>> xlabel('\omega t');
>> gtext('annotation')
```

`gtext` activates a cross-hair that allows you to position the text on the plot.



Multi Channel Signals

Matlab represents ordinary one-dimensional sampled data signals, or sequences, as *vectors*. Vectors are 1-by-*n* or *n*-by-1 arrays, where *n* is the number of samples in the sequence.

Column orientation is preferable for single channel signals because it extends naturally to the multi channel case. For multi channel data, each column of the matrix represents one channel. Each row of such a matrix then corresponds to a sample point.

Suppose you define a five-element column vector as follows:

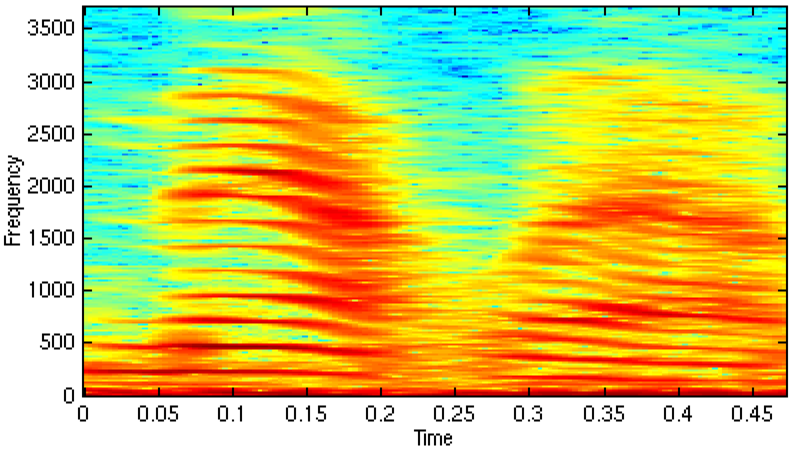
```
>> a = [1 2 3 4 5];
>> a = a';
```

To duplicate column vector *a* into a matrix, use the following method:

```
>> c = a(:,ones(1,3))
c =  1 1 1
     2 2 2
     3 3 3
     4 4 4
     5 5 5
```

Imported Signals: MAT file

```
>> load mtlb
>> specgram(mtlb,512,Fs,kaiser(500,5),475)
```



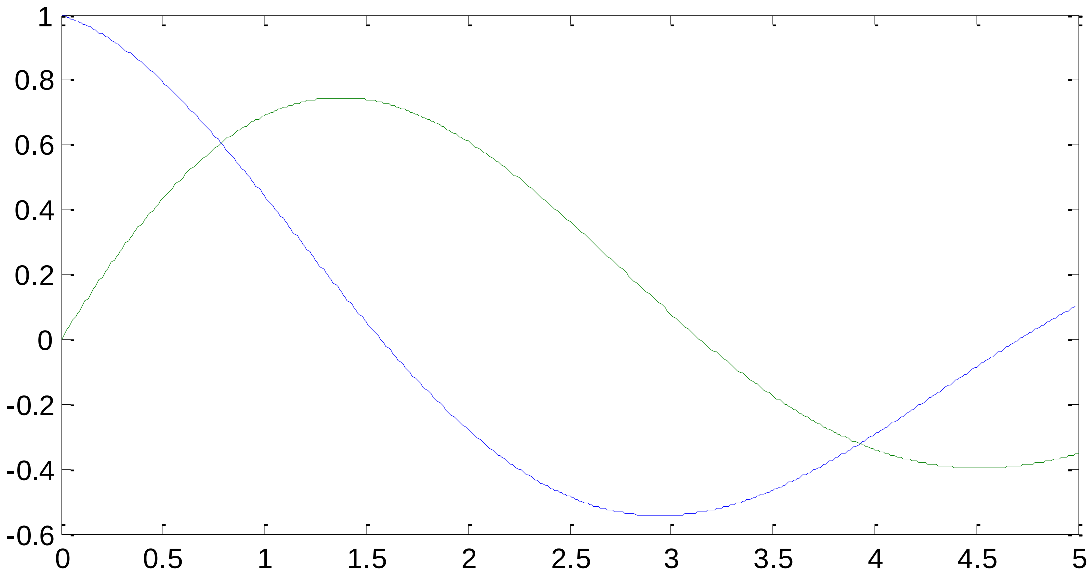
This is a digitized speech signal. You may play it using the command

```
>> sound(mtlb,Fs)
```

Imported Signal: ASCII file

The program **`datagen.cpp`** creates a file called *sample.dat*. The data for a exponentially damped sine and cosine are calculated and saved in three columns of numbers.

```
>> load sample.dat
>> x = sample(:,1);
>> y = sample(:,2);
>> z = sample(:,3);
>> plot(x,y,x,z)
```

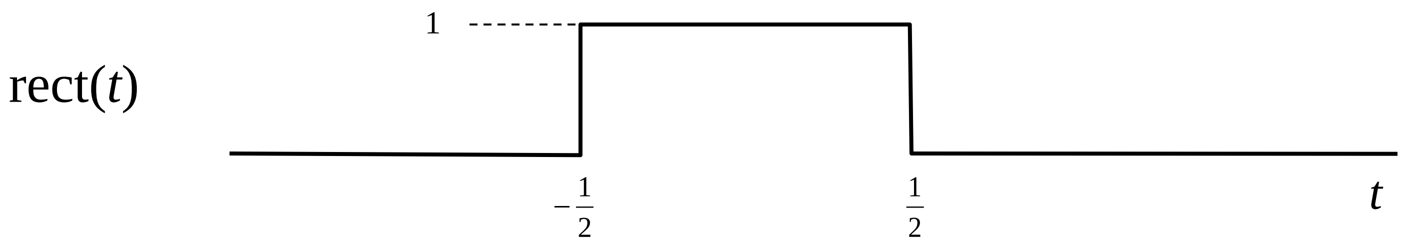


Source code: **`datagen.cpp`**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int i;
    double x, y, z, ampl, phi;
    const double pi = 4.0*atan(1.0);
    FILE *fp;
    char *filename;
    filename = (argc<2? "sample.dat": argv[1]);
    printf("output file: %s\n",filename);
    fp = fopen(filename,"wt");
    if (!fp) {
        printf("error opening output file\n");
        return -1;
    }
    for (i=0; i<=500; i++) {
        x = 0.01*i;
        ampl = exp(-0.2*x);
        phi = 2.0*pi*x;
        y = ampl*cos(phi);
        z = ampl*sin(phi);
        fprintf(fp,"%g %g %g\n",x,y,z);
    }
    fclose(fp);
    return 0;
}
```

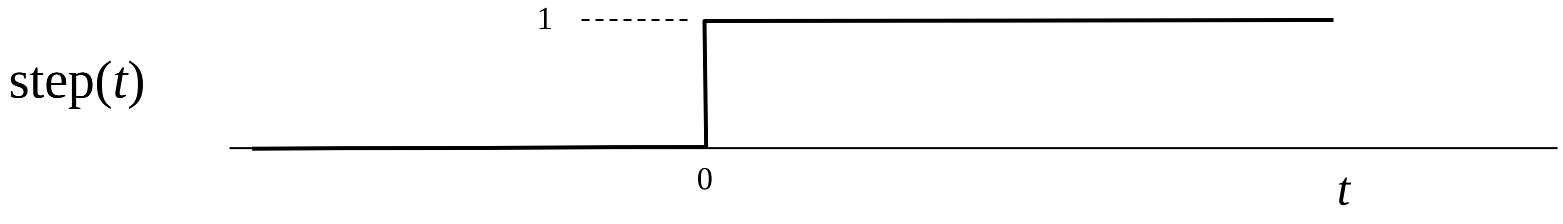
Matlab rect function

```
function y=rect(x)
%RECT  function.
%   RECT(X) returns a matrix whose elements are the rect of the elements
%   of X, i.e.
%        y = 1      if |x| < 0.5
%          = 0.5   if |x| = 0.5
%          = 0      if |x| > 0.5
%   where x is an element of the input matrix and y is the resultant
%   output element.

%   Author: John Loomis 25 Feb 2000

y=zeros(size(x));
i=find(abs(x)==0.5);
y(i) = 0.5;
i=find(abs(x)<0.5);
y(i)=1.0;
```



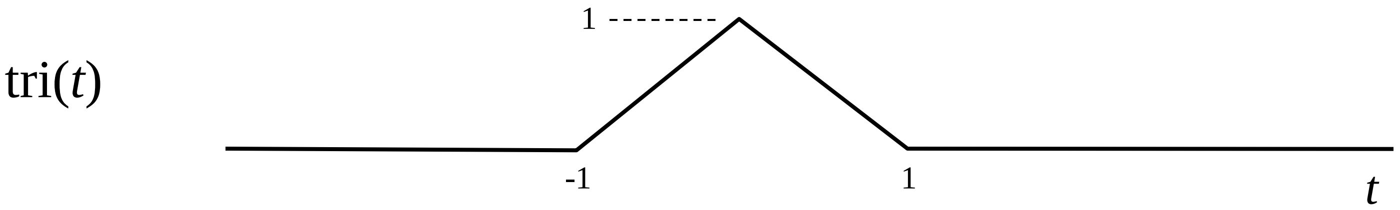
Matlab step function

```
function y=step(x)
%STEP  function.
%   STEP(X) returns a matrix whose elements are the step of the elements
%   of X, i.e.
%       y  = 1     if x > 0
%          = 0.5   if x = 0
%          = 0     if x < 0
%   where x is an element of the input matrix and y is the resultant
%   output element.

%   Author: John Loomis 25 Feb 2000

y=zeros(size(x));
idx=find(x==0);
y(idx) = 0.5;
idx=find(x>0);
y(idx)=1.0;
```



Matlab tri function

```
function y=tri(x)
%TRI  function.
%   TRI(X) returns a matrix whose elements are the tri of the elements
%   of X, i.e.
%         y = 1 - |x| if |x| < 1
%           = 0     if x >= 1
%   where x is an element of the input matrix and y is the resultant
%   output element.

%   Author: John Loomis 12 Jan 2003

y=zeros(size(x));
idx=find(abs(x)<1.0);

y(idx) = 1-abs(x(idx));
```



Matlab sinc function

```
function y=sinc(x)
%SINC Sin(pi*x)/(pi*x) function.
%   SINC(X) returns a matrix whose elements are the sinc of the elements
%   of X, i.e.
%        y = sin(pi*x)/(pi*x)    if x ~= 0
%          = 1                   if x == 0
%   where x is an element of the input matrix and y is the resultant
%   output element.
%

y=ones(size(x));
i=find(x);
y(i)=sin(pi*x(i))./(pi*x(i));
```

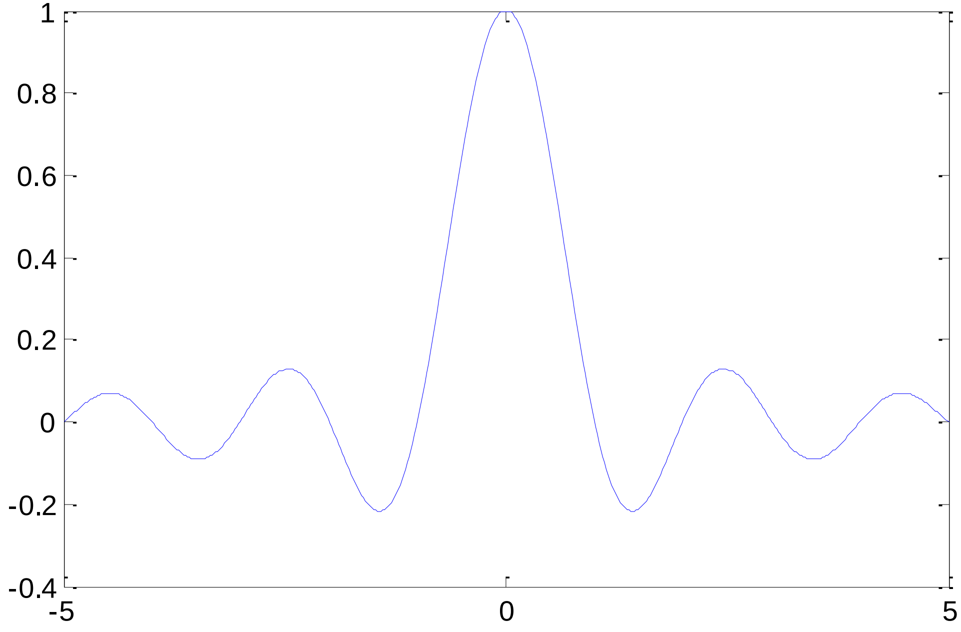
found in signal processing toolbox

Sinc Function

The function **`sinc`** computes the mathematical sinc function

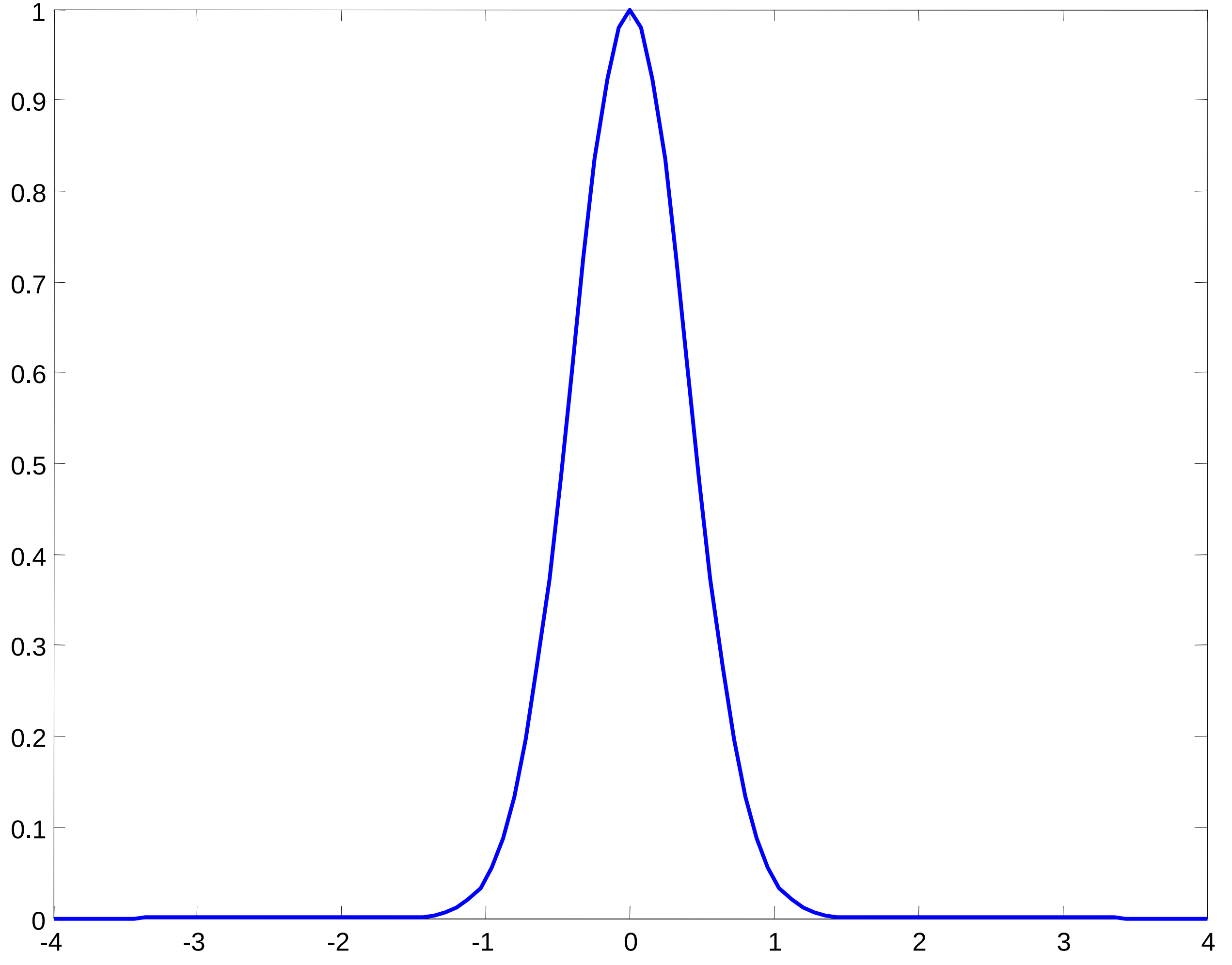
```
>> t = linspace(-5,5,500);
>> yc = sinc(t);
>> plot(t,yc);
```

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$



Gaus Function

$$\mathrm{gaus}(x) = e^{-\pi x^2}$$

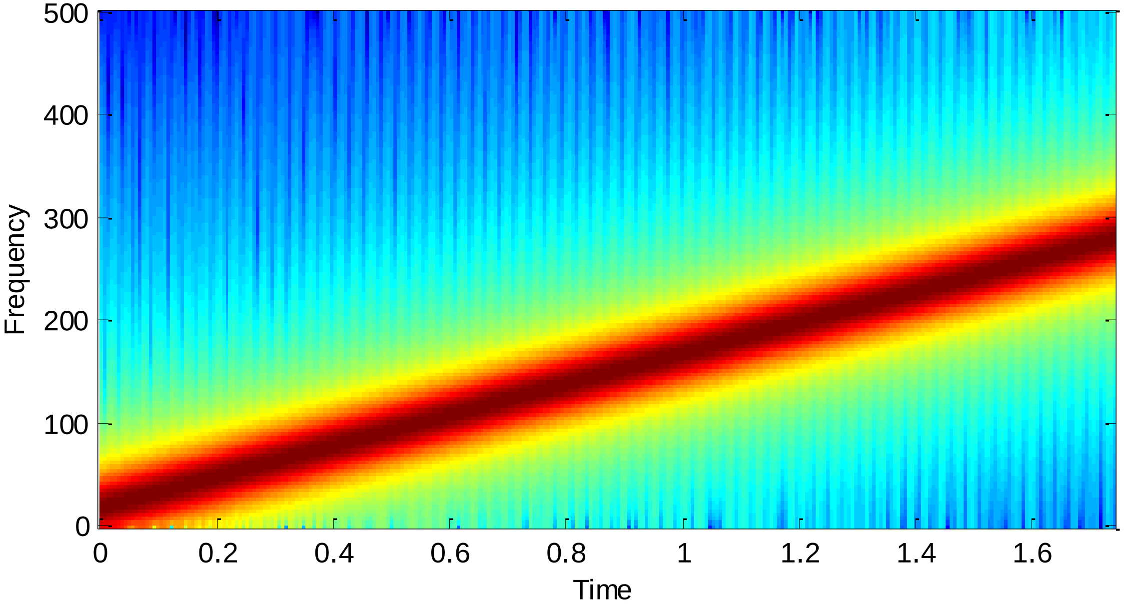


Chirp Function

The **`chirp`** function generates a linear swept-frequency cosine signal. An optional parameter specifies alternative sweep methods. An optional parameter **`phi`** allows the initial phase to be specified in degrees.

To compute 2 seconds of a linear chirp signal with a sample rate of 1 kHz, that starts at DC and crosses 150 Hz at 1 second, use

```
>> t = (0:2000)/1000;
>> y = chirp(t,0,1,150);
>> specgram(y,256,1000,256,250);
```

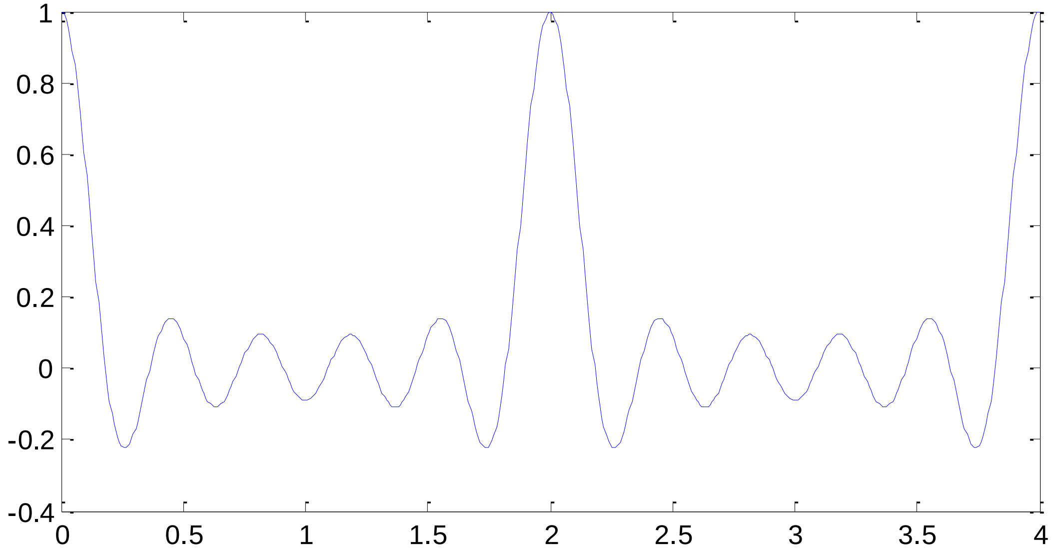


Dirichlet Function

The function **`diric`** computes the Dirichlet function, sometimes called the *periodic sinc* or *aliased sinc* function.

```
>> t = linspace(0,4,500);
>> yd = diric(pi*t,11);
>> plot(t,yd);
```

$$\text{diric}(x,n) = \begin{cases} -1^{\frac{x}{2\pi}(n-1)} & x = 0, \pm 2\pi, \pm 4\pi, \ldots \\ \dfrac{\sin(nx/2)}{n\sin(x/2)} & \text{otherwise} \end{cases}$$



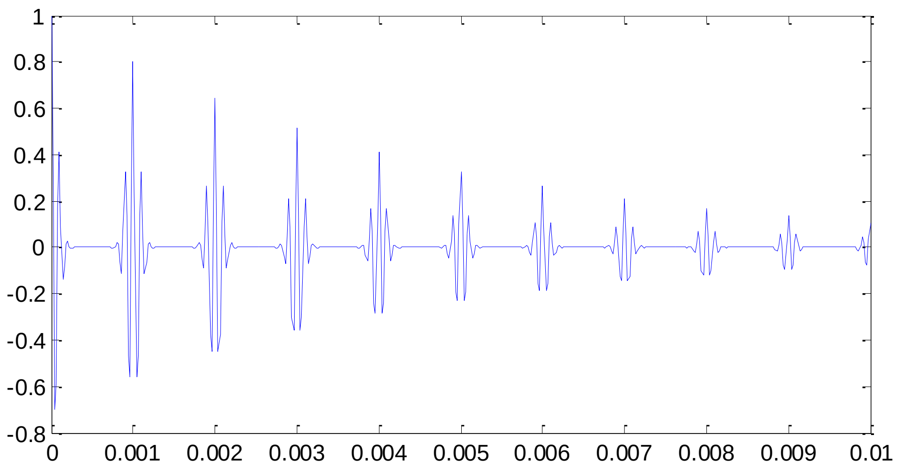
Pulstran Function

The **`pulstran`** function generates pulse trains from either continuous or sampled prototype pulses.

The following example generates a pulse train consisting of the sum of multiple delayed interpolations of a Gaussian pulse. The pulse train is defined to have a sample rate of 50 kHz, a length of 10 msec, and a pulse repetition rate of 1KHz. The array *d* specifies the delay to each pulse repetition in column 1 and an optimal attenuation for each repetition in column 2.

The pulse train is constructed by passing the name of the gauspuls function to pulstran, along with additional parameters that specify a 10 kHz Gaussian pulse with 50% bandwidth.

```
>> Fs = 50e3; % 50 kHz
>> length = 10e-3; % 10 msec
>> t = 0:1/Fs:length;
>> d = [ (0:10)/1e3; 0.8.^(0:10)]';
>> yf = pulstran(t,d,'gauspuls',10e3,0.5);
>> plot(t,yf);
```



UKURAN SINYAL

Signal Energy

How much energy a signal has and specifically how much energy is required to produce the signal is an important concept. Energy is a non-negative property usually calculated from the square of something. Electric energy is the square of the electric field (or voltage). For a sound wave it is the pressure squared.

$$E = \int_{-\infty}^{\infty} \left| s(t) \right|^2 dt$$

Mathematical signals can extend to infinity and may have infinite energy. Realistic signals associated with real events have a finite duration and a finite energy. A concert "signal" is associated with a scheduled start and finish time. A music CD has a start and a stop. Speech starts and stops, and so forth.

$$E = \sum_n \left| x[n] \right|^2 \Delta t$$

We often leave this out

Noise and Clutter

Signals are often associated with *noise* and *clutter*. Background noise is a continuously present background level, usually assumed to be a stationary (independent of time) random (or stochastic) process. The sound of an air conditioner or the hum of florescent lights could be background noise. Clutter is an unwanted signal. The sound of a cell phone conversation during a movie is audio clutter.

In calculating signal duration and energy, we want to edit out the clutter and either cancel the background noise or subtract the mean noise level.

Mean Time and Duration

If we consider $|s(t)|^2$ as a density in time, the average time can be defined as.

$$< t > = \int_{-\infty}^{\infty} t \left| s(t) \right|^2 dt$$

The mean time tells us approximately when the energy is localized in time.
Duration can be measured from the standard deviation:

$$\sigma_t^2 = \int_{-\infty}^{\infty} \left(t - < t > \right)^2 \left| s(t) \right|^2 dt$$

$$= < t^2 > - < t >^2$$

Duration can be calculated from *order* statistics as the range: t_{max}-t_{min}

Average Power

$$|s(t)|^2$$

is the energy per unit time at time *t*, or instantaneous power

$$P_{t_1 \rightarrow t_2} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} |s(t)|^2 \, dt$$

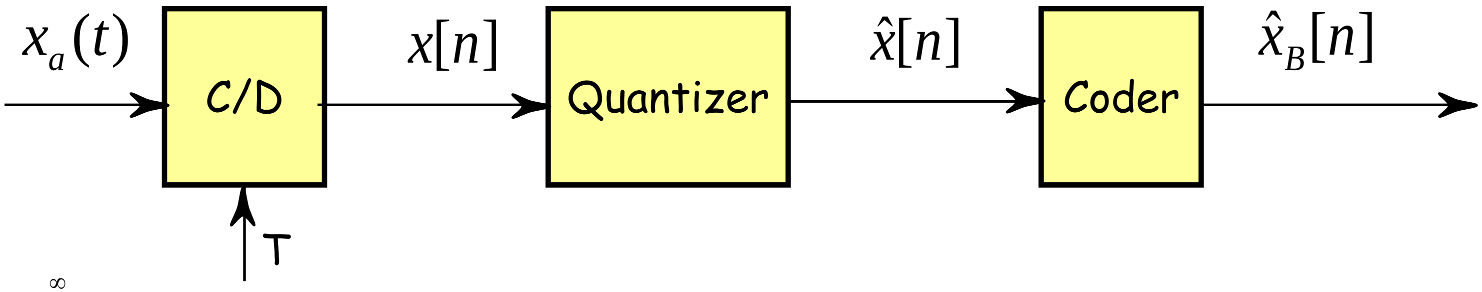
is the average power over the given interval

For a sequence

$$P_{n_1 \rightarrow n_2} = \frac{1}{n_2 - n_1 + 1} \sum_{n=n_1}^{n_2} |s[n]|^2$$

KUANTISASI TERHADAP SINYAL

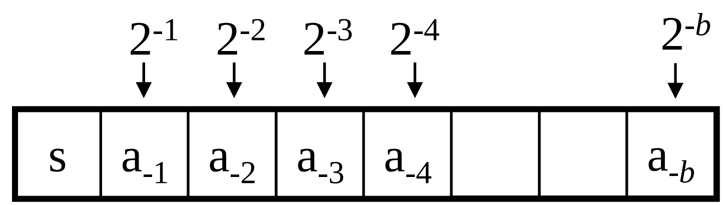
Conceptual Representation of A/D Conversion



$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

$$x[n] = x_a(t) \cdot s(t)$$

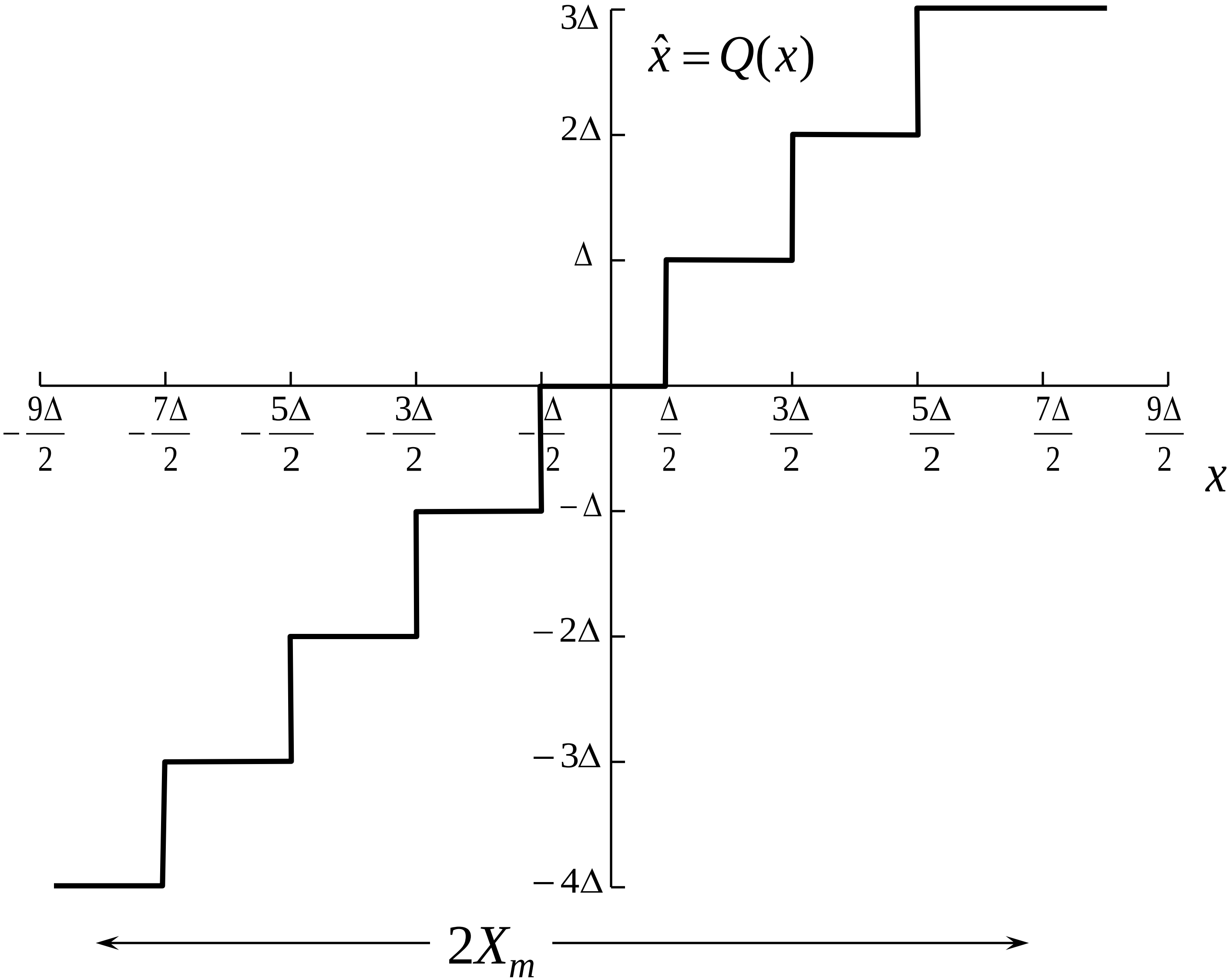
0.11	3/4
0.10	1/2
0.01	1/4
0.00	0
1.11	-1/4
1.10	-1/2
1.01	-3/4
1.00	-1



Signed, (b+1)-bit fixed-point fraction

$$\hat{x} = -s + a_{-1}2^{-1} + a_{-2}2^{-2} + \cdots + a_{-b}2^{-b}$$

Typical Quantizer for A/D Conversion



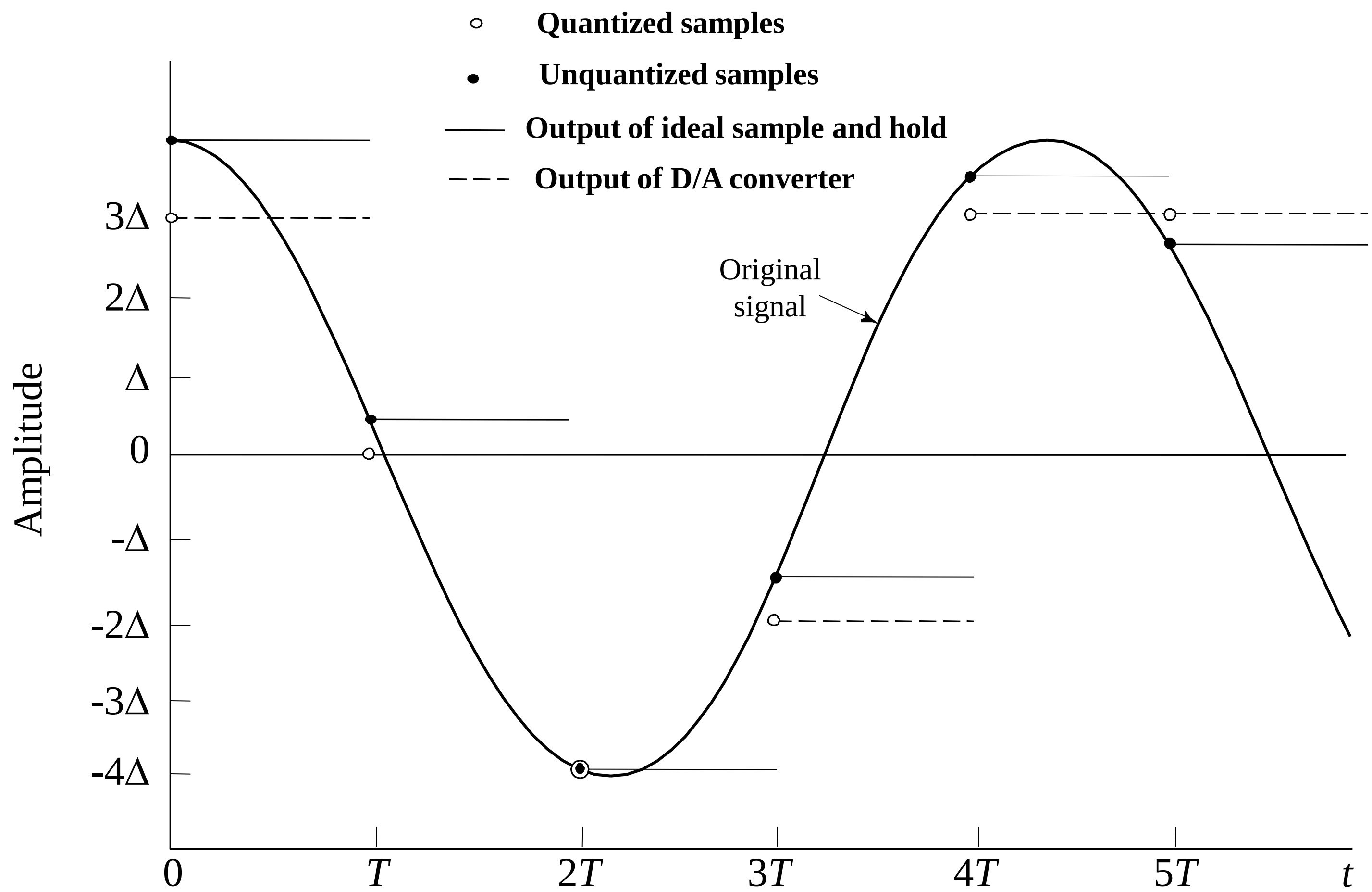
Two's-complement code	Offset binary code
011	111
010	110
001	101
000	100
111	011
110	010
101	001
100	100

Possible Numeric Interpretations

Binary symbol	Numeric value	Symmetric value
0.11	3/4	7/8
0.10	1/2	5/8
0.01	1/4	3/8
0.00	0	1/8
1.11	-1/4	-1/8
1.10	-1/2	-3/8
1.01	-3/4	-5/8
1.00	-1	-7/8

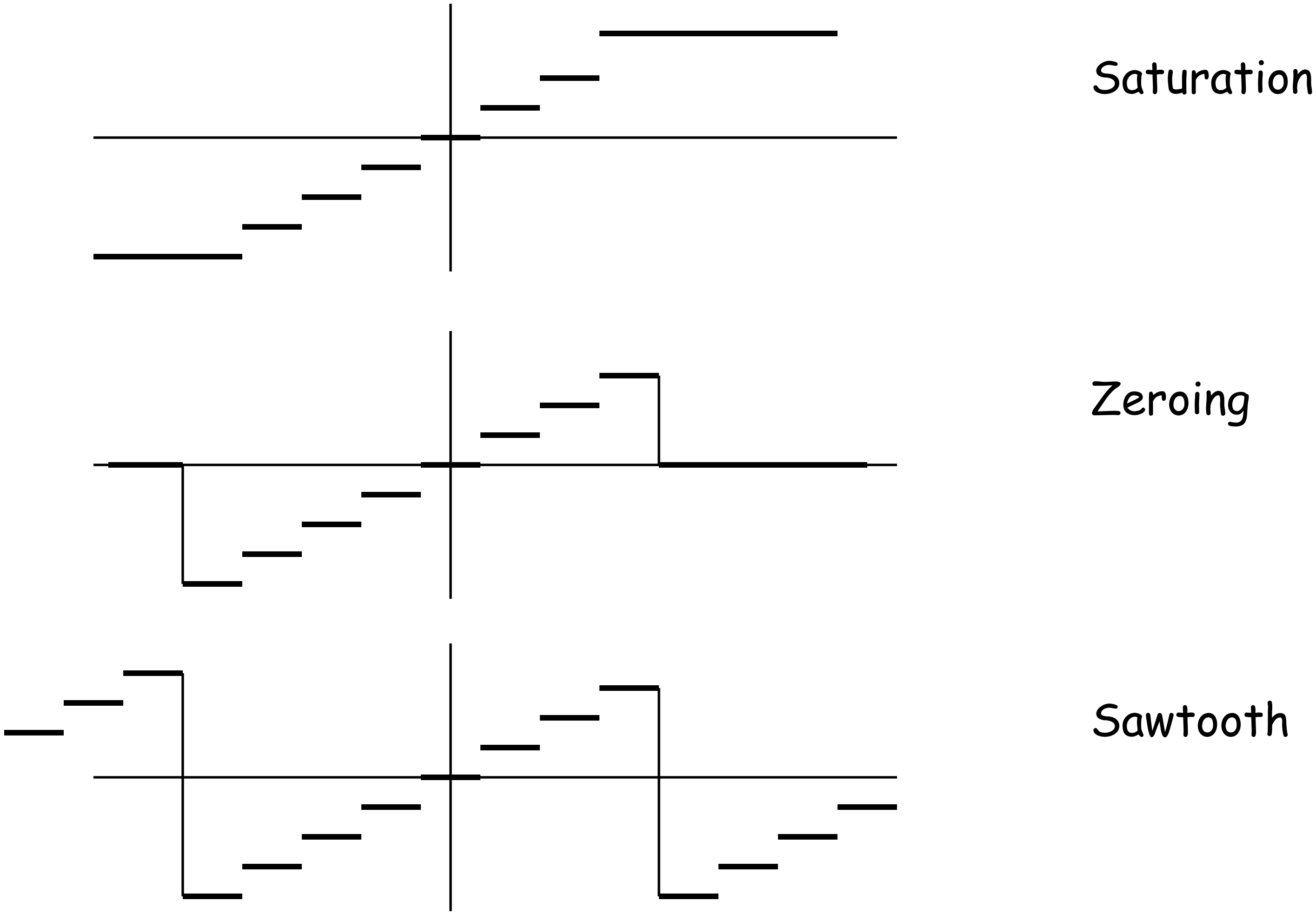
No zero-step value

Example 3-Bit Quantizer



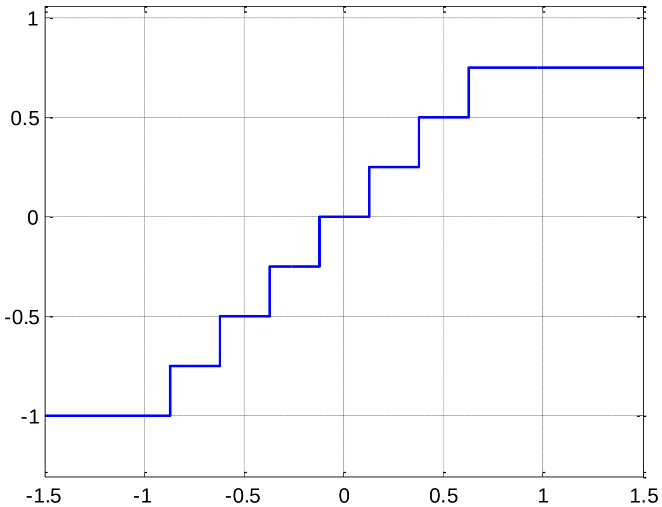
$\hat{x}_B[n]$	011	000	100	110	011	011
	3	0	-4	-2	3	3

Overflow Characteristics



Quantization

```
clear
N = 3;   % number of bits
in = -1.5 : .01 : 1.5;
out = qtz(in,N);
stairs(in,out);
grid
axis equal
```



This quantizer clips out-of-range values. (*saturation*)

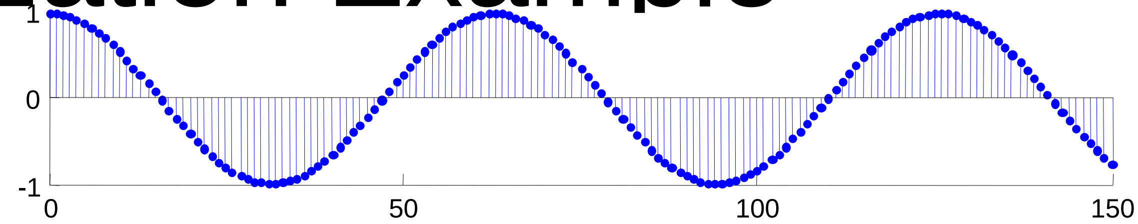
```
function y = qtz(in,N)

n = 2^(N-1);
y = round(in*n)/n;

% clip output at limits

max = 1 - 1/n;
idx = find(y>max);
y(idx)=max;
idx = find(y<-1);
y(idx)=-1;
```

Change **`round`** to **`floor`** for truncation.

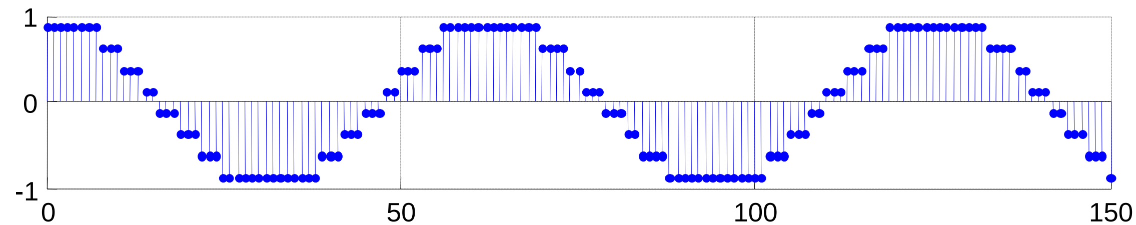
```
>> unique(out)

ans =

   -1.0000   -0.7500   -0.5000   -0.2500
         0    0.2500    0.5000    0.7500
```

Quantization Example

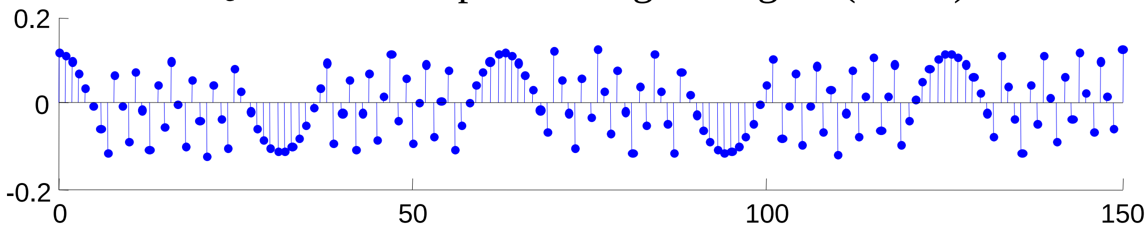
```
clear
n = 0:150;
x =
0.99*cos(n/10);
subplot(4,1,1);
stem(n,x);
subplot(4,1,2);
y = qtz(x,3);
stem(n,y);
grid;
subplot(4,1,3);
stem(n,x-y);
subplot(4,1,4);
y = qtz(x,8);
stem(n,x-y);
```



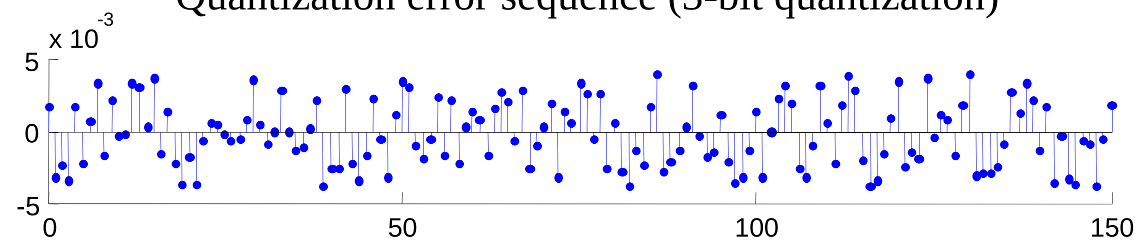
Unquantized samples of signal 0.99 cos(*n*/10)



Quantized samples of original signal (3-bits)



Quantization error sequence (3-bit quantization)



Quantization error sequence (8-bit quantization)

Alan V. Oppenheim and Ronald W. Schafer with John R. Buck, *Discrete-Time Signal Processing*, Second Edition, Prentice-Hall, 1999. 194-195

Analysis of Quantization Errors

- The difference between the quantized sample $\hat{x}[n]$ and true sample value $x[n]$ is the quantization error:

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

- If a linear round-off $(B+1)$-bit quantizer is used, then

  $$-\Delta/2 < e[n] \leq \Delta/2$$

  which holds whenever

  $$(-X_m - \Delta/2) < x[n] \leq (X_m - \Delta/2)$$

  where Δ is step size of the quantizer:

  $$\Delta = X_m/2^B$$

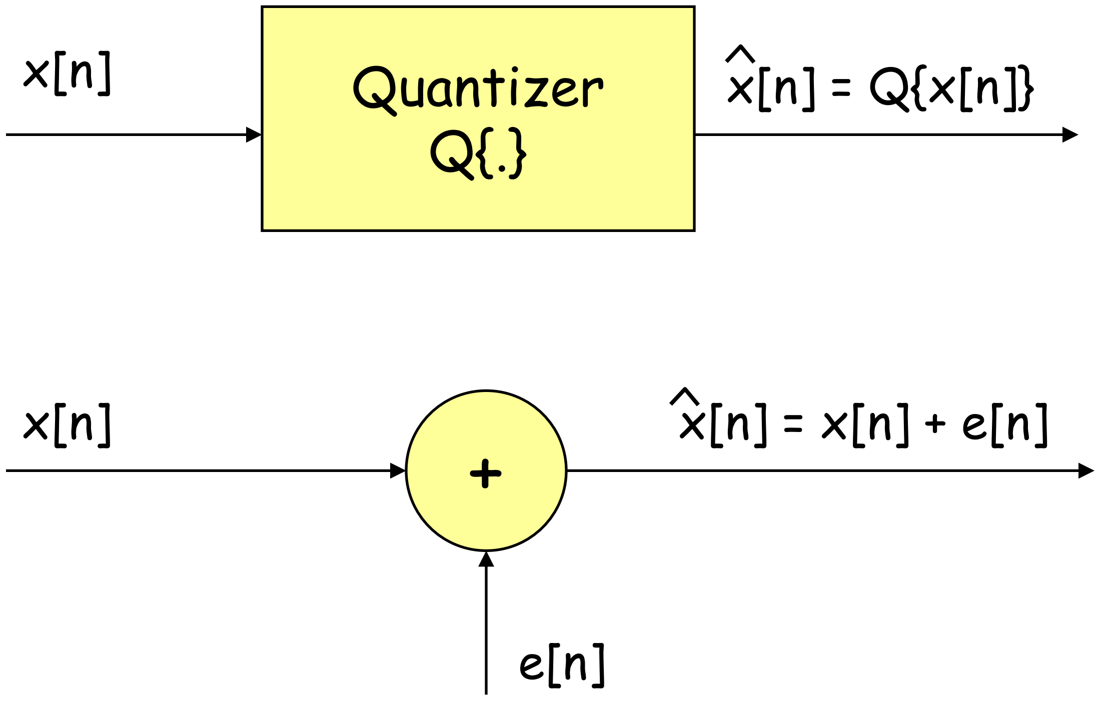
- If $x[n]$ is outside the range mentioned above, then the quantization error is larger in magnitude than $\Delta/2$ and such samples are said to be *clipped*.

Analysis of Quantization Errors 2

The statistical representation of quantization errors is based on the following assumptions:

- The error sequences $e[n]$ is a sample sequence of a stationary random process.
- The error sequence is uncorrelated with the sequence $x[n]$.
- The random variables of the error process are uncorrelated; i.e., the error is a white-noise process.
- The probability distribution of the error process is a uniform over the range of quantization error.

Additive Noise Model for Quantizer



Quantization SNR

$$\text{SNR} = 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) \qquad \text{where } \sigma_x^2 \text{ is the variance of the signal}$$

$$= 10\log_{10}\left(12 \cdot 2^{2B}\, \frac{\sigma_x^2}{X_m^2}\right) \qquad \text{for a rounding quantizer}$$

$$= 6.02B + 10.8 - 20\log_{10}\left(\frac{X_m}{\sigma_x}\right)$$

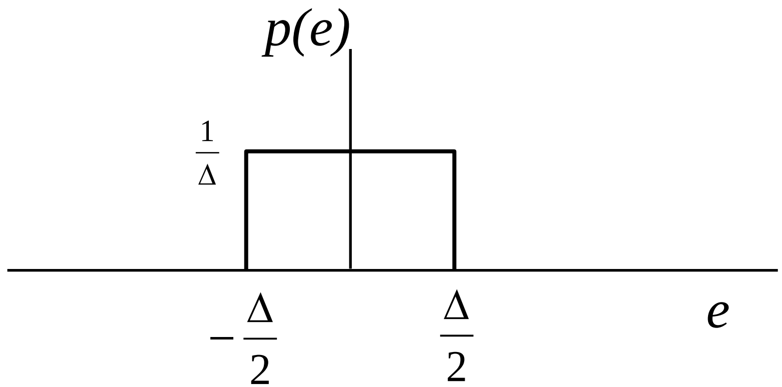
- The SNR ratio increases approximately 6 dB for each bit added to the word length of the quantized samples.

If we set the range of the signal to four times the signal variance to avoid clipping the peaks, then $X_m = 4\ \sigma_x$

$$\text{SNR} \approx 6B - 1.25$$

Quantization Error Observations

- In low number-bit case, the error signal is highly correlated with the unquantized signal.
- The quantization error for high number-bit quantization is assumed to vary randomly and is uncorrelated with the unquantized signal.



$$\sigma_e^2 = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 \frac{1}{\Delta} de = \frac{\Delta^2}{12}$$

For a (B+1)-bit quantizer with full-scale value X_m the noise variance or power is

$$\sigma_e^2 = \frac{2^{-2B} X_m^2}{12}$$

Range vs. Resolution

- The trade-off between peak signal amplitude and the absolute size of the the quantization noise is a fundamental design decision.
- For analog signals such as speech or music, the distribution of amplitudes tends to be concentrated about zero and falls off rapidly with increasing amplitude.
  - The probability that the magnitude of a sample will exceed 3 or 4 times the RMS value is very low.
  - For example, obtaining a signal-to-noise ratio of about 90~96 dB for use in high quality music recording and playback requires 16-bit quantization.
  - But it should be remembered that such performance is obtained only if the input signal is carefully matched to the full-scale range of the A/D converter.

Overview of Finite-Precision Numerical Effects

- Format of Number Representation :
  - Sign and Magnitude: $X = 1 .b_1b_2…b_B$ for $X \le 0$, - 20 → 10010100
  - One's Complement: $X = 1 .b_1b_2…b_B$ for $X \le 0$, - 20 → 11101011
  - Two's Complement*: $X = 1 .b_1b_2…b_B + 0 .00…01$ for $X < 0$, - 20 → 11101100
- Output samples from an A/D converter are quantized and thus can be represented by fixed-point binary numbers.
  - A real number can be represented with infinite precision : such as in two's complement form

$$x = X_m \left(-b_0 + \sum_{i=1}^{\infty} b_i 2^{-i} \right)$$

where X_m is an arbitrary scale factor and the b_i's are either 0 or 1. The quantity b_0 is referred to as the *sign bit* .

Overview of Finite-Precision Numerical Effects 2

- Limitation of the finite word lengths for operation
  - Example: overflow effect

Addition

Binary	Decimal
0.1101	0.8125
0.1001	0.5625
1.0110	1.3750

Multiplication

Binary	Decimal
0.1101	0.8125
0.1001	0.5625
0.01110101	0.45703125

- If a finite number of bits (B+1) are used in quantization, then the representation must be :

$$\hat{x} = Q_B[x] = X_m\left(-b_0 + \sum_{i=1}^{B} b_i 2^{-i}\right) = X_m \hat{x}_B$$

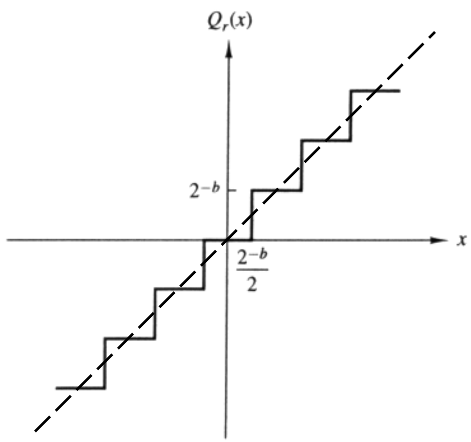
Overview of Finite-Precision Numerical Effects 3

- Limitation of the finite word lengths for quantization
  - the smallest difference between numbers is

$$\Delta = X_m 2^{-B}$$

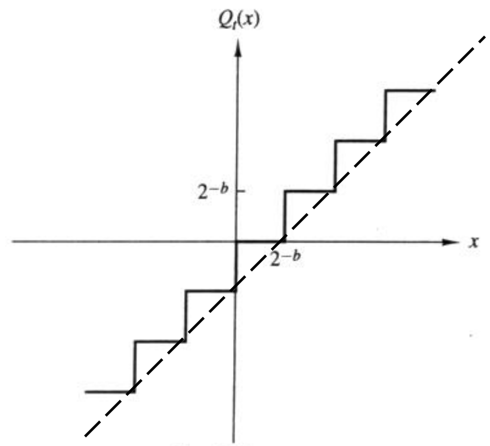
  - the quantization error : $e = Q_B[x] - x$
- Quantization Forms:
  - Rounding
  - Value truncation
  - Magnitude truncation
- Overflow Characteristics :
  - Saturation or Clipping
  - Zeroing
  - 'Sawtooth' or Natural overflow

Quantization errors and its statistical characterization in rounding, truncation in 2's complement, and truncation in sign-magnitude quantizer.



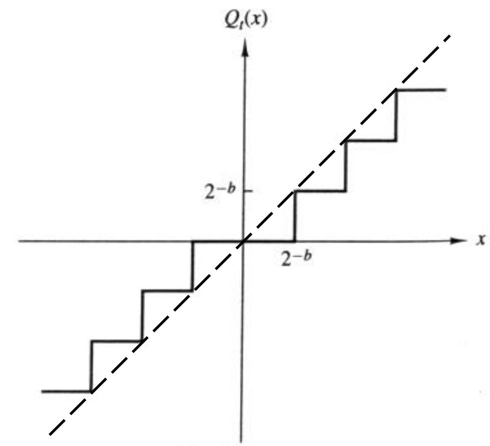
$$E_r = Q_r(x) - x$$
$$-\frac{1}{2}\cdot 2^{-b} \le E_r \le \frac{1}{2} 2^{-b}$$

Rounding



$$E_t = Q_t(x) - x$$
$$2^{-b} \le E_t \le 0$$

Truncation in 2's complement



$$E_t = Q_t(x) - x$$
$$-2^{-b} \le E_t \le 2^{-b}$$

Truncation in sign-magnitude