University of Udine
Udine, Italy

Image Processing: Theories and Techniques

report

Author: Tibor Lukić, University of Novi Sad, Serbia.
Udine, October 2008 - January 2009.

PART I. Exercise 1. Tibor Lukić, University of Novi Sad.

SIGNALS

Introduction

A *signal* is any time-varying or spatial-varying quantity. If for a signal, the quantities are defined only on a discrete set of times, we call it a *discrete-time signal*. In other words, a discrete-time signal can be seen as a function from the set of integers to the set of numbers. A *continuous-time signal* is any function which is defined in an interval, most commonly an infinite interval.

We mention two examples. The first is a sound. It is a vibration of the air, therefore a sound signal associates a pressure value to every value of time. The second example is the image. The image assigns intensity value for each point of (two-dimensional) image domain. If the image is a digital, then it can be realized as a two-dimensional discrete signal too.

Many signals are periodic in nature. A *Periodic signal* always repeats itself, what can be described by a periodic function. A function f is periodic if it satisfies

$$f(t+T) = f(t)$$

for all t and some fixed T. The constant T is called as a *period* or a *wavelength* of f. A restriction of f onto a periodic length interval is called as one *wave* of f. A *frequency* is given by the inverse of the period, that is $1/T$.

Sinusoid is a well known periodic function. Let us consider its main characteristics. It is defined in general form by

$$A\,\sin(\omega\,t+\phi),$$

where t is the independent (real) variable, and the fixed parameters A, ω, and ϕ are all real constants. A meaning of parameters are the following:

$A-$ *Amplitude.* Half difference between the minimum and maximum values;

$\omega-$ *Frequency.* A corresponding period is $2\pi/\omega$;

$\phi-$ *Phase.* Phase is the measure of the starting point of one wave of sinusoid relative to another.

Experiments

Problem

"Create and visualize periodic signals of sinusoid, rectangular and triangular shapes with a given amplitude, frequency and phase."

Matlab codes

In the following we present Matlab codes for generation 3 types of signals: sinusoidal, rectangular and triangular.

```
function  y= sinus(amp,freq,fase,range,ncamp)
% Generation of a sinusoidal signal
% y    - signal function
% amp  - amplitude
% freq - frequency
% fase - phase
% range- signal function domain
% ncamp- number of discrete values in graph discretization
%
tmin=range(1,1); % minimum time value
tmax=range(1,2); % maximum time value
tper=abs(tmax-tmin)/freq; % signal period
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
y=amp*sin( (xt+fase)*2*pi/tper ); % calculate signal function values
plot(xt,y);    % plot the signal
grid;          % draw grid mesh
% adjust a graph display
ymax=max(y); ymin=min(y); xmin=min(xt);
xmax=max(xt); p=0.1;
delta = p*abs(ymax-ymin);
ymax=ymax+delta; ymin=ymin-delta;
axis([xmin xmax ymin ymax]);
%END

function y= rectangular(amp,freq,fase,range,ncamp)
% Generation of a rectangular signal
% y    - signal function
% amp  - amplitude
% freq - frequency
% fase - phase
% range- signal function domain
% ncamp- number of discrete values
```

```
%
tmin=range(1,1); % minimum time value
tmax=range(1,2); % maximum time value
tper=abs(tmax-tmin)/freq; % signal period
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
% determination of function values
for i=1:length(xt),
   if (mod(xt(i)+fase,tper)<tper/2)
        y(i)=amp;
   else y(i)=-amp;
   end;
end;
plot(xt,y); % plot the signal
grid;       % use grid mesh
%adjust a graph display
ymax=max(y); ymin=min(y);
xmin=min(xt); xmax=max(xt);
p=0.1; delta = p*abs(ymax-ymin);
ymax=ymax+delta; ymin=ymin-delta;
axis([xmin xmax ymin ymax]);
%END
```

```
function y= triangular(amp,freq,fase,range,ncamp)
% Generation of a triangular signal
% y    - signal function
% amp  - amplitude
% freq - frequency
% fase - phase
% range- signal function domain
% ncamp- number of discrete values
%
tmin=range(1,1);  % minimum time value
tmax=range(1,2);  % maximum time value
tper=abs(tmax-tmin)/freq; % signal period
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
% calcualte values of function
y=(amp*2/pi)*abs(asin(sin((xt+fase)*pi/tper )));
plot(xt,y);    % plot the signal
grid;          % use grid mesh
%adjust a graph display
ymax=max(y); ymin=min(y); xmin=min(xt);
xmax=max(xt);
p=0.1; delta = p*abs(ymax-ymin);
ymax=ymax+delta; ymin=ymin-delta;
axis([xmin xmax ymin ymax]);
%END
```

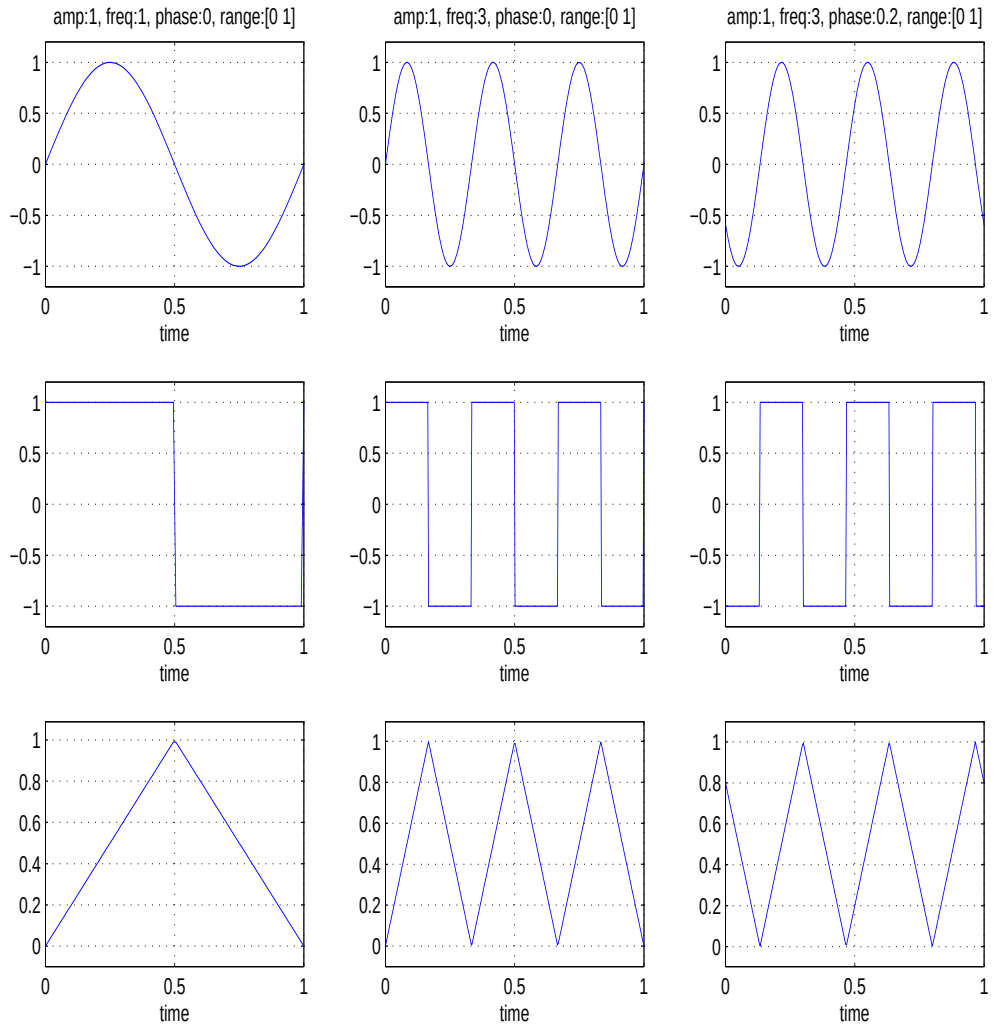

Figure 1: Signals.

```
%%%%%%%%%%%%%%%%%%%%%
% Plot signals
%%%%%%%%%%%%%%%%%%%%%
% ncamp-number of discrete samples
subplot(3,3,1); sinus(1,1,0,[0 1],100); % ncamp=100
title('amp:1, freq:1, phase:0, range:[0 1]'); xlabel('time');
subplot(3,3,2); sinus(1,3,0,[0 1],300); % ncamp=300
title('amp:1, freq:3, phase:0, range:[0 1]'); xlabel('time');
subplot(3,3,3); sinus(1,3,0.2,[0 1],300); % ncamp=300
title('amp:1, freq:3, phase:0.2, range:[0 1]');  xlabel('time');
subplot(3,3,4); rectangular(1,1,0,[0,1],100);  % ncamp=100
xlabel('time');
```

```
subplot(3,3,5); rectangular(1,3,0,[0,1],300); % ncamp=300
xlabel('time');
subplot(3,3,6); rectangular(1,3,0.2,[0,1],300); % ncamp=300
xlabel('time');
subplot(3,3,7); triangular(1,1,0,[0,1],100);  % ncamp=100
xlabel('time');
subplot(3,3,8); triangular(1,3,0,[0,1],300); % ncamp=300
xlabel('time');
subplot(3,3,9); triangular(1,3,0.2,[0,1],300); % ncamp=300
xlabel('time');
%END
```

Figure 1 shows 3 type of periodic signals generated by our codes: sinusoidal, rectangular and triangular. A parameter setting for each column is the same and it is showed above and below of each graph display. On the basis of our experiments we conclude that the number of discrete values, ncamp must be in accordance with the signal frequency. We suggest 100 values per period.

PART I. Exercise 2. Tibor Lukić, University of Novi Sad.

STATISTICAL SIGNALS

Introduction

Signals can be divide into a two large groups: *deterministic* and *statistical signals*. Let us consider a signal as a time variable, t function, X. The signal is deterministic if function values $X(t)$ are completely specified for every t. If function values $X(t)$ are random numbers with a given probability, then the signal is statistical.

Statistical signal has a correspondent probability, P given by

$$P(c < X(t) < d) = \int_c^d \varphi(x)\, dx,$$

where $c < d$ and φ is a *probability density function* (pdf). A probability function is completely determined by a corresponding pdf function. Depending on the pdf definition we distinguish different *probability distributions*.

The pdf of the *uniform distribution*, is:

$$\varphi_{a,b}(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b \\ 0, & elsewhere, \end{cases}$$

where parameters a, b defines a *distribution's support interval* $[a, b]$. This distribution is abbreviated by $\mathcal{U}(a, b)$.

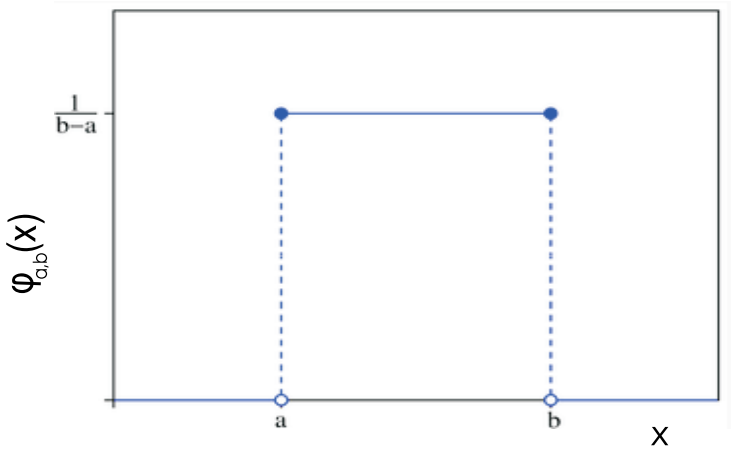

Figure 2: Probability density function of the uniform distribution, $\mathcal{U}(a, b)$.

The pdf of the *normal (Gaussian) distribution* is given by

$$\varphi_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where parameters μ and σ are mean value and standard deviation (variance), respectively. This distribution is abbreviated by $\mathcal{N}(\mu, \sigma^2)$. The *standard normal distribution* is the normal distribution with a mean of zero and a variance of one, $\mathcal{N}(0, 1)$.

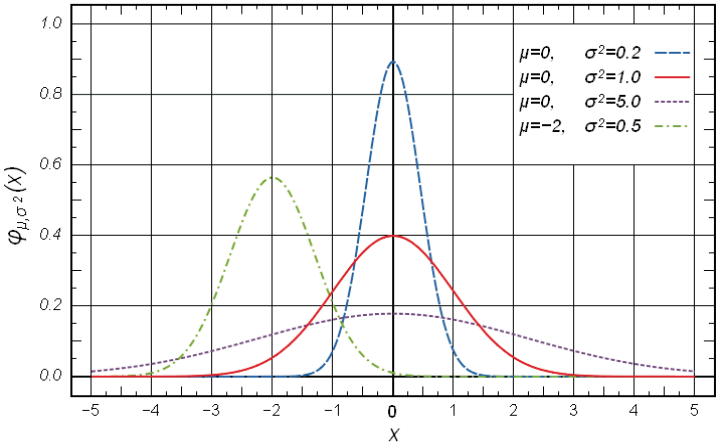

Figure 3: Probability density function of the normal distribution, $\mathcal{N}(\mu, \sigma^2)$. Illustration taken from Wikipedia.

Experiments

Problem

"Create a sequence of random numbers with uniform distribution (uniform noise) and then verify whether its distribution approximates the uniform distribution. Repeat the above exercise, but instead uniform noise use Gaussian noise (sequence of random numbers with normal/Gaussian distribution) with a given mean and variance."

Matlab codes

We create a sequence of random numbers with uniform distribution, $\mathcal{U}(a, b)$. After, by histogram we show the distribution of random numbers. The histogram should approximate the pdf of a corresponding uniform distribution. We will verify that by comparing an obtained histogram with the graph of the pdf function. The procedure is repeated analogically, but using a normal distribution, $\mathcal{N}(\mu, \sigma^2)$ instead a uniform one.

```
function y = pdf_uniform(x,a,b)
% Probability density function of uniform distribution,
% U(a,b)
%
for i=1:length(x)
    if (x(i)>=a) && (x(i)<=b),
```

```
        y(i)=1/(b-a);
    else
        y(i)=0;
    end;
end;
% end pdf_uniform


function y = pdf_gaussian(x,mu,sigma_sq)
% Probability density function of the normal distribution,
% N(mu, sigma_sq)
%
y=1/(sqrt(sigma_sq*2*pi))* exp(-(x-mu).^2/(2*sigma_sq));
% end pdf_gaussian

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verification of the distribution of a sequence
% of pseudorandom numbers generated by the Matlab
% function: rand.
% rand generate pseudorandom numbers of
% the uniform distribution, U(0,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=0; b=1; % specify the parameters of a un. dist., U(a,b)
range=[0 1]; tmin=range(1,1); tmax=range(1,2);
ncamp=3000; % number of the genretated pseudorandom numbers
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax;
y=(b-a)*rand(1,ncamp)+a; % generate pseudorandom numbers
subplot(1,2,1);
plot(xt,y); grid; title('Genarated pseudorandom
numbers (rand)'); xlabel(['ncamp: ' int2str(ncamp)]); ymax=max(y);
ymin=min(y); xmin=min(xt); xmax=max(xt); p=0.1;
delta=p*abs(ymax-ymin); ymax=ymax+delta; ymin=ymin-delta;
axis([xmin xmax ymin ymax]); % adjust a display
nbins=20; % number of used bins
[ord xout]=hist(y,nbins);
scale=(nbins)/(ncamp*(b-a)); % appropriate scale value
ord=ord*scale; % rescale ord
subplot(1,2,2);
bar(xout,ord); % plot the histogram
hold on;
% plot the pdf of the un. dist., U(a,b)
tmin=min(xout); tmax=max(xout);
p=0.5; delta=p*abs(tmax-tmin);
tmax=tmax+delta; tmin=tmin-delta; ncamp_un=200;
x_un=tmin:abs(tmax-tmin)/(ncamp_un-1):tmax;
y_un=pdf_uniform(x_un,a,b); % pdf of the un. dist.
```

```
plot(x_un,y_un,'LineWidth',3); title('Uniform distribution?');
% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Verification of the distribution of a sequence
% of pseudorandom numbers generated by the
% Matlab function: randn.
% randn generate pseudorandom numbers of
% the normal distribution, N(0,1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
range=[0 1]; tmin=range(1,1); tmax=range(1,2);
ncamp=30000; % number of the genretated pseudorandom numbers
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax;
y=randn(1,ncamp); % generate pseudorandom numbers
subplot(1,2,1);
plot(xt,y);  % plot the sequence of pseudorandom numbers
grid; title('Generated pseudorandom numbers (randn)');
xlabel(['ncamp: ' int2str(ncamp)]); ymax=max(y); ymin=min(y);
xmin=min(xt); xmax=max(xt); p=0.1; delta = p*abs(ymax-ymin);
ymax=ymax+delta; ymin=ymin-delta;
axis([xmin xmax ymin ymax]); % adjust the display
nbins=20; % number of used bins
[ord xout]=hist(y,nbins); ncamp_g=200; tmin=min(xout);
tmax=max(xout); x_g=tmin:abs(tmax-tmin)/(ncamp_g-1):tmax;
y_g=pdf_gaussian(x_g,0,1); % pdf of normal dist.
scale=max(y_g)/ord(int16(nbins/2)); % appropriate scale value
ord=ord*scale; % rescale ord
subplot(1,2,2);
bar(xout,ord);  % plot the histogram
hold on;
% plot the pdf of the normal distribution, N(0,1)
plot(x_g,y_g,'LineWidth',3); title('Normal distribution ?');
% END
```

Results

Figures 4 and 5 show a sequence of pseudorandom numbers generated by the Matlab function rand. In Figure 4 we have 3000 numbers, while in Figure 5 this number is 30000. Graphs on the right side show histograms and the pdf of the uniform distribution, $\mathcal{U}(0,1)$. Histograms are appropriately rescaled to be able to compare them with graphs of pdf. On the basis of our experiments we conclude that histograms approximate better the corresponding pdf with increasing the number of generated pseudorandom sequence elements (compare Figure 4 and 5). Also, we got a same conclusion in a case of normal distribution. Figures 6 and 7 show results of the experiments with a sequence of pseudorandom numbers with normal distribution, $\mathcal{N}(0,1)$.

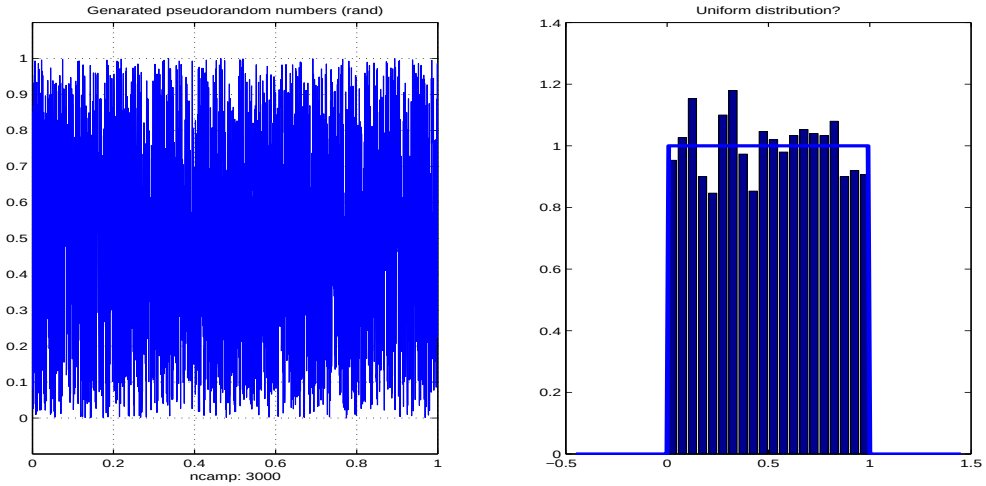

Figure 4: Sequence of 3000 random numbers with uniform distribution and its histogram.

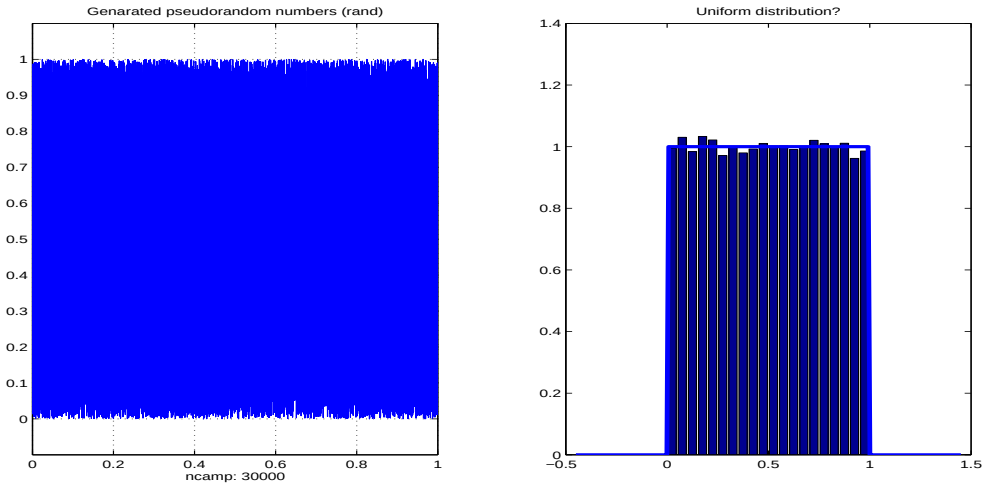

Figure 5: Sequence of 30000 random numbers with uniform distribution and its histogram.

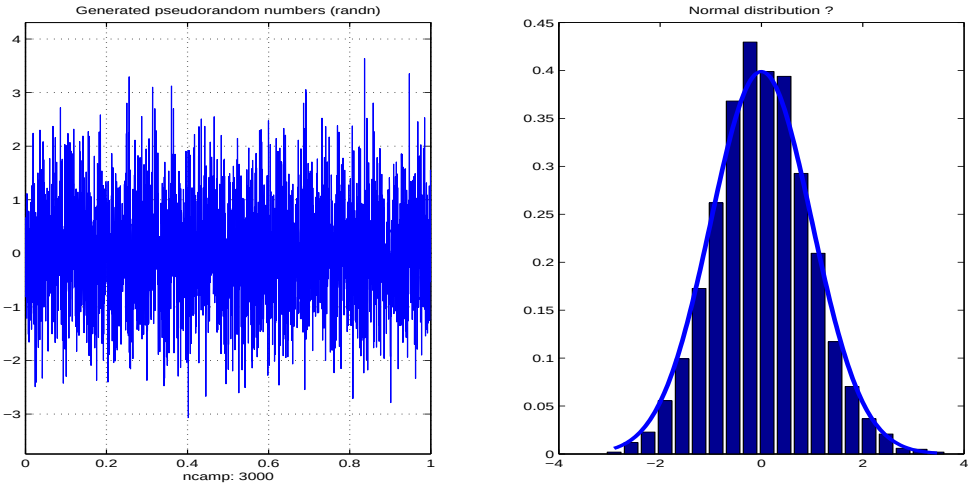

Figure 6: Sequence of 3000 random numbers with normal distribution and its histogram.

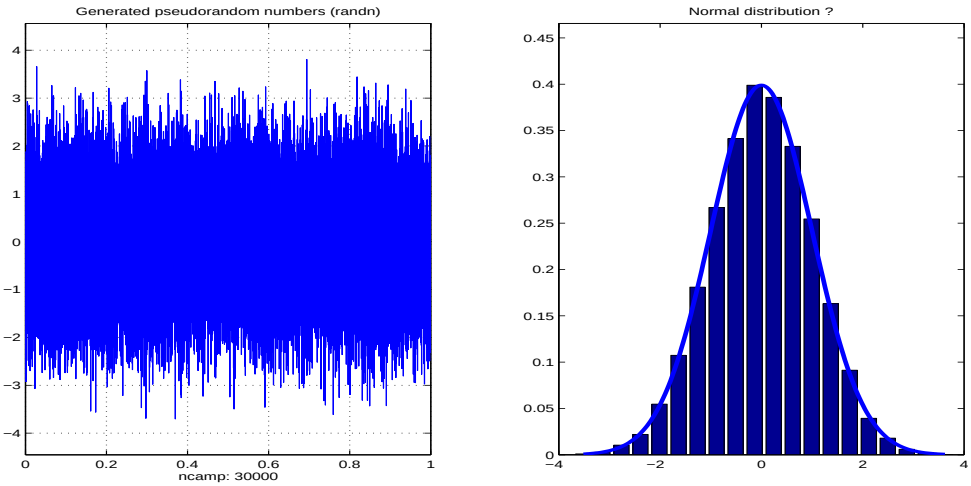

Figure 7: Sequence of 30000 random numbers with normal distribution and its histogram.

PART I. Exercise 3. Tibor Lukić, University of Novi Sad.

CORRELATION AND CONVOLUTION OF A SIGNAL

Introduction

In signal processing, *correlation* is a measure of similarity of two waveforms. Let f_1 and f_2 are continuous functions (signals). Correlation of f_1 and f_2 is a new function defined by

$$R_{12}(\tau) = R[f_1(t), f_2(t)](\tau) = \int_{-\infty}^{\infty} f_1^*(t) \cdot f_2(t+\tau)\, dt,$$

where $\tau \in \mathbb{R}$ and f_1^* is a complex-conjugate function of f_1. The formula essentially slides the function f_2 along the x-axis, calculating the integral of their product for each possible amount of sliding. When the functions match, the value of R_{12} is maximized. The maximum is a measure of similarity of two waveforms (f_1 and f_2).

It is easy to show, that the correlation satisfies the following property

$$R_{12}(\tau) = R_{21}^*(-\tau),\ \forall \tau \in \mathbb{R}. \tag{1}$$

This property is a well know symmetry of the correlation operation.

The *autocorrelation* is defined as the correlation integral of a given function (signal) with itself, at lag τ. Hence, the autocorrelation of f_1 is given by

$$R_{11}(\tau) = \int_{-\infty}^{\infty} f_1^*(t) \cdot f_1(t+\tau)\, dt\,. \tag{2}$$

An important property of the autocorrelation is its symmetry. Namely, if f_1 is real, then $R_{11}(\tau) = R_{11}(-\tau)$. The autocorrelation of a periodic function is, itself, periodic with the very same period. Therefore, it is a powerful mathematical tool for finding repeating patterns, such as the presence of a periodic signal which has been buried under noise. Finally, we note that the autocorrelation function reaches its peak at the origin, with zero lag.

Similar to correlation, *convolution* is an operation on two functions, producing a third function that is typically viewed as a modified version of one of the original functions. Definition of the convolution of functions f_1 and f_2 is given by

$$\rho_{12}(\tau) = \rho[f_1(t), f_2(t)](\tau) = \int_{-\infty}^{\infty} f_1(t) f_2(\tau - t)\, dt.$$

It is an integral of the product of the two functions after one (f_2) is reversed and shifted. From its properties we first mention that it is commutative, that is $\rho_{12}(\tau) = \rho_{21}(\tau),\ \forall \tau \in \mathbb{R}$. Let us suppose that f_1 and f_2 are real functions. The following relations between correlation and convolution hold

$$\rho_{12}(\tau) = R[f_1(t), f_2(-t)](\tau) = R[f_2(-t), f_1(t)](-\tau),\ \forall \tau \in \mathbb{R}.$$

Also, if f_2 is even function then

$$\rho_{12}(\tau) = R_{21}(-\tau),\ \forall\tau \in \mathbb{R} \tag{3}$$

and if f_2 is odd function then

$$\rho_{12}(\tau) = -R_{21}(-\tau),\ \forall\tau \in \mathbb{R}. \tag{4}$$

Experiments

Problem

"Calculate and visualize autocorrelations of signals from Exercise 1. Also, calculate the crossing correlations and convolutions of this signals."

Matlab codes

We create Matlab codes for calculation and visualization of the autocorrelation of three type of signals from exercise 1. Also we compare the convolution of this signals to its correlation. Two Matlab functions are used for calculation of correlation and convolution: **xcorr** and **conv**, respectively.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autocorrelations of the sinus, rectangular
% and triangular signals.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter settings for signal generation.
amp=1; fase=0; freq=3; range=[0 1]; ncamp=300;
% Autocerrelation of the sinus signal
subplot(3,2,1);
y_s=sinus(amp,freq,fase,range,ncamp); plot(y_s);
grid; title('Sinusoid signal'); xlabel('discrete samples');
subplot(3,2,2);
[c lags]=xcorr(y_s,y_s); plot(lags,c); grid;
title('Autocorrelations'); xlabel('lag');
% Autocorrelation of the rectangular signal
subplot(3,2,3);
y_r=rectangular(amp,freq,fase,range,ncamp);
plot(y_r); grid;
title('Rectangular signal'); xlabel('discrete
samples');
[c lags]=xcorr(y_r,y_r);
subplot(3,2,4); plot(lags,c);
```

```
grid; xlabel('lag');
% Autocorrelation of the triangular signal
subplot(3,2,5);
y_t=triangular(amp,freq,fase,range,ncamp);
plot(y_t); grid; title('Triangular signal'); xlabel('discrete
samples'); [c lags]=xcorr(y_t,y_t);
subplot(3,2,6);
plot(lags,c);grid; xlabel('lag');
% END


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Correlation of the following signals:
% sinusoid-trinagular, triangular-sinusoid,
% rectangular-sinusoid, sinusoid-rectangular,
% rectangular-triangular, triangular-rectangular
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter settings for signal generation.
amp=1; fase=0; freq=3; range=[0,1]; ncamp=300;
% claculate values of signal functions
y_s=sinus(amp,freq,fase,range,ncamp);
y_r=rectangular(amp,freq,fase,range,ncamp);
y_t=triangular(amp,freq,fase,range,ncamp);
subplot(3,2,1);
[c lags]=xcorr(y_s,y_t);
plot(lags,c); grid; title('Correlation of
sinusoid and triangular'); xlabel('lag');
subplot(3,2,2);
[c lags]=xcorr(y_t,y_s);
plot(lags,c); grid; title('Correlation of
triangular and sinusoid'); xlabel('lag');
subplot(3,2,3);
[c lags]=xcorr(y_s,y_r);
plot(lags,c); grid; title('Correlation of
sinusoud and rectangular'); xlabel('lag');
[c lags]=xcorr(y_r,y_s);
subplot(3,2,4);
plot(lags,c); grid;
title('Correlation of rectangular and sinusoid'); xlabel('lag');
subplot(3,2,5);
[c lags]=xcorr(y_r,y_t); plot(lags,c); grid;
title('Correlation of rectangular and triangular'); xlabel('lag');
subplot(3,2,6);
[c lags]=xcorr(y_t,y_r); plot(lags,c); grid;
title('Correlation of triangular and rectangular'); xlabel('lag');
%END
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convolutions and correlations of following signals:
% sinusoid-trinagular, triangular-sinusoid,
% rectangular-sinusoid, sinusoid-rectangular,
% triangular-rectangular, rectangular-triangular
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% parameter settings for signal generation.
amp=1; fase=0; freq=3; range=[0,1]; ncamp=300;
y_s=sinus(amp,freq,fase,range,ncamp);
y_r=rectangular(amp,freq,fase,range,ncamp);
y_t=triangular(amp,freq,fase,range,ncamp);
subplot(3,2,1);
c=conv(y_t,y_s); plot(c); grid;
title('Convolution of triangular
and sinusoid');
subplot(3,2,2);
c=xcorr(y_s,y_t); plot(c); grid;
title('Correlation of sinusoid and triangular');
subplot(3,2,3);
c=conv(y_r,y_s); plot(c); grid;
title('Convolution of rectangular
and sinusoid');
subplot(3,2,4);
c=xcorr(y_s,y_r); plot(c); grid;
title('Correlation of sinusoid and rectangular');
subplot(3,2,5);
c=conv(y_r,y_t); plot(c); grid;
title('Convolution of rectangular and triangular');
subplot(3,2,6);
c=xcorr(y_t,y_r); plot(c); grid;
title('Correlation of triangular and rectangular');
% END
```

Results

Right column of Figure 8 shows autocorrelations of three different type of signals: sinusoid, rectangular and trinagular. These signals are presented in left column and they are also used as originals in Figure 9 and Figure 10. Obtained autocorrelations are symmetric (regarding the zero) what is in accordance with the relation (2). Figure 9 presents correlations between different signals. Each row shows correlation of two signals and their correlation with inverse queue. Presented examples are in accordance with the symmetry property described by the relation (1). Finally, in Figure 10 we compare operation of

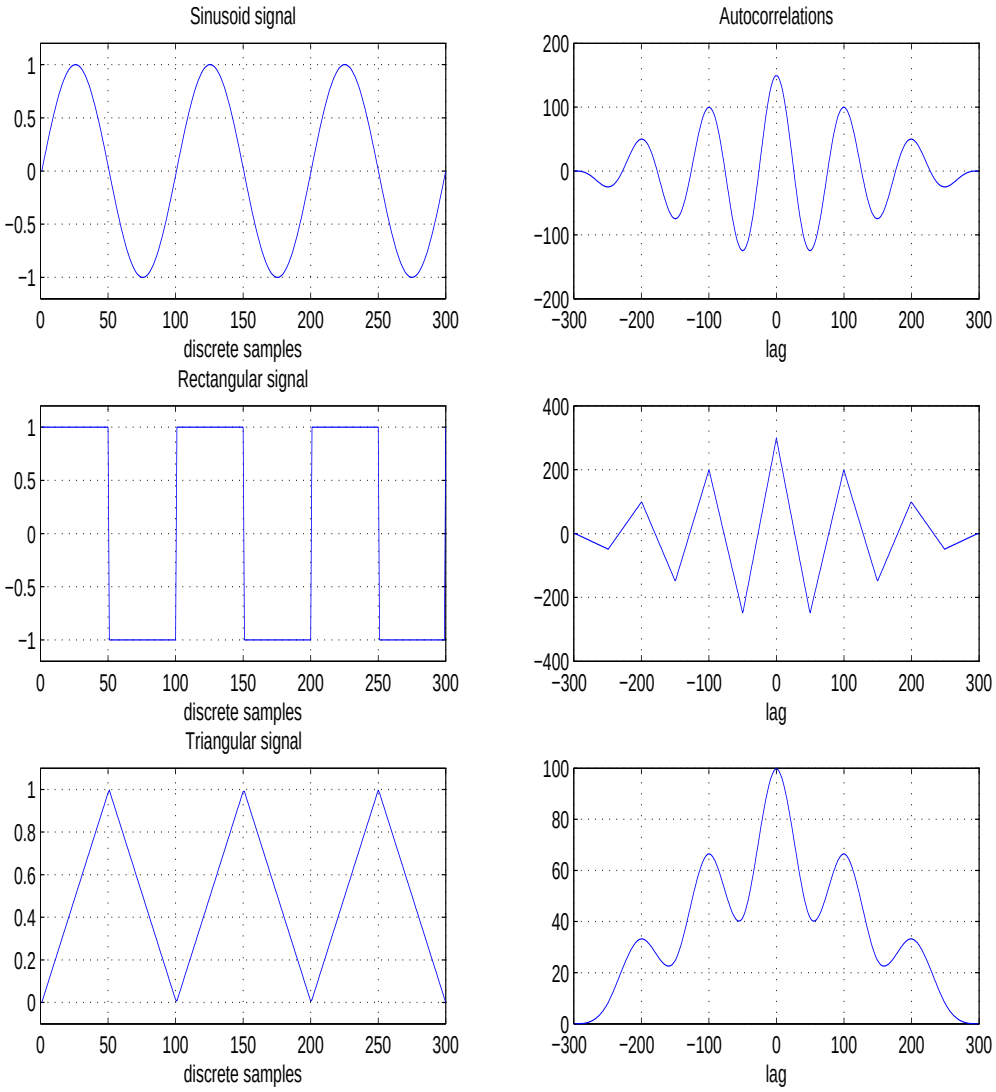

Figure 8: Autocorrelations of the rectangular, triangular and sinusoid signals.

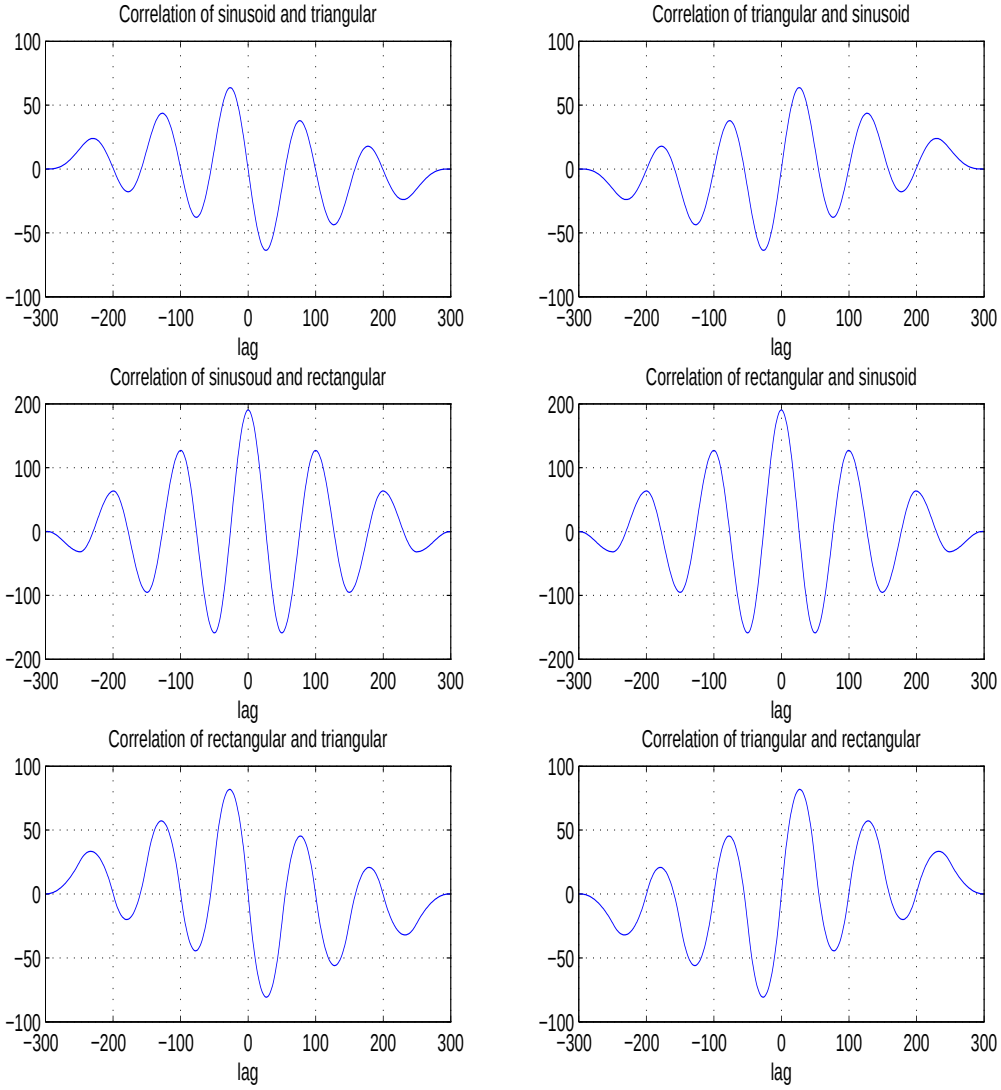

Figure 9: Correlations of the considered signals.

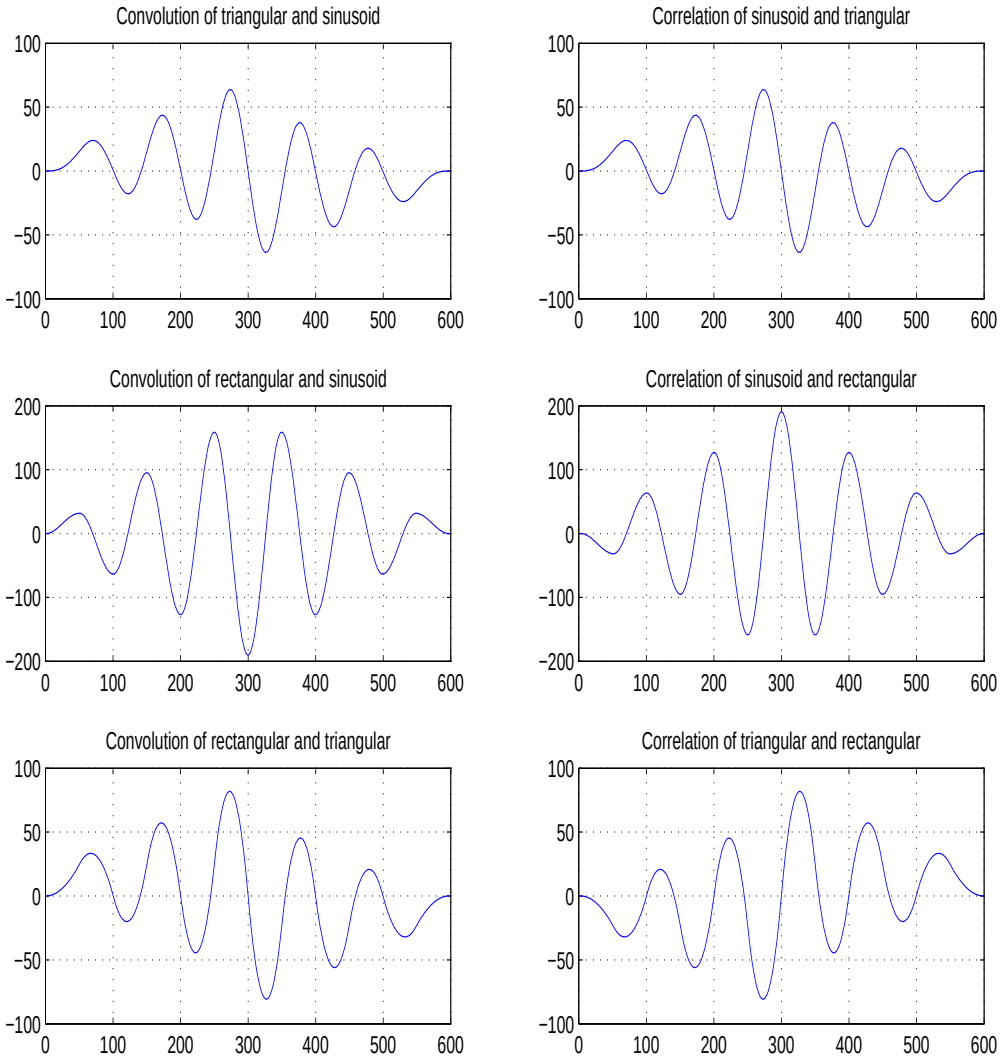

Figure 10: Convolutions and correlations of the considered signals.

correlation with operation of convolution. Sinusoid and rectangular functions (signals) are odd functions, while triangular function (signal) is an even function. Therefore, first and second rows demonstrate the relation (4), when third row demonstrates the relation (3).

PART I. Exercise 4. Tibor Lukić, University of Novi Sad.

AUTOCORRELATION OF A NOISY SIGNAL

Introduction

The sequence of random numbers with normal distribution and mean value zero, $\mathcal{N}(0, \sigma^2)$ is commonly called as *Gaussian white noise* or just *white noise*. The autocorrelation, $R(\tau)$ of a continuous white noise signal will have a strong peak at $\tau = 0$ and will be absolutely 0 for all other τ. This is a consequence of the fact that white noise has no periodicity. This correlation function can be represented by a *Dirac delta function*, designated by δ. It is a function representing an infinitely sharp peak bounding unit area:

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \quad \text{and} \quad \int_{-\infty}^{\infty} \delta(x)\, dx = 1.$$

One of the realizations of the Dirac delta function is a limit of the sequence of Gaussian density functions, that is

$$\delta_a(x) = \frac{1}{a\sqrt{\pi}} e^{-x^2/a^2}, \quad a \to 0.$$

Autocorrelation of a periodic signal is also periodic with the same frequency. This fact comes from the definition. Therefore, autocorrelation is useful for finding repeating patterns in a signal, such as determining the presence of a periodic signal which has been buried under noise.

Experiments

Problem

"Calculate and visualize the signals which contents only Gaussian white noise with different variance values and calculate their autocorrelations. Also, calculate and visualize the autocorrelations of noisy signals from Exercise 1 by adding Gaussian white noises (high and low) to the original signals."

We create a Matlab codes for consideration of white noisy sequences whit different variance values and their autocorrelations. Also, we add white noise to the deterministic signals (sinusoid, rectangular and triangular) and consider autocorrelations of such obtained noisy signals. This experiments are repeated by adding white noises with three different variance.

Matlab codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autocorrelation of three Gaussian white noises
% with different variances.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
ncamp=200; % signal discretzation
% set three variance values
variance_low=0.2; variance_medium=1; variance_high=5;
% calculate gaussian noises with zero mean and different variance
y_low=randn(1,ncamp)*sqrt(variance_low);
y_medium=randn(1,ncamp)*sqrt(variance_medium);
y_high=randn(1,ncamp)*sqrt(variance_high);
% plot noise data and corresponding autocorrelations
subplot(3,2,1);
plot(y_low); grid; title(['Gaussian noise with low
variance, ' num2str(variance_low) '.']);
subplot(3,2,2); [c lags]=xcorr(y_low); plot(lags, c); grid;
title('Autocorrelations');
subplot(3,2,3);
plot(y_medium); grid;
title(['Gaussian noise with medium variance, '
num2str(variance_medium) '.']);
subplot(3,2,4);
[c lags]=xcorr(y_medium); plot(lags,c); grid;
subplot(3,2,5);
plot(y_high); grid; title(['Gaussian noise with high variance, '
num2str(variance_high) '.']);
subplot(3,2,6); [c lags]=xcorr(y_high); plot(lags,c); grid;
% END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Autocorrelations of a sinus signal
% with added white noise.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter settings for sinus signal.
amp=1; fase=0; freq=3; range=[0,1];
ncamp=300; % signal discretization
y_s=sinus(amp,freq,fase,range,ncamp); % calculate the sinus signal
% set used variance values
variance_low=0.2; variance_medium=1; variance_high=5;
% calculate gaussian white noises with different variances
low_noise=randn(1,ncamp)*sqrt(variance_low);
```

```
medium_noise=randn(1,ncamp)*sqrt(variance_medium);
high_noise=randn(1,ncamp)*sqrt(variance_high);
% Add the noise to the original signal
y_low=y_s+low_noise; % low noise
y_medium=y_s+medium_noise; % medium noise
y_high=y_s+high_noise; % high noise
% plot signals
subplot(4,2,1); plot(y_s); grid; title('Sinusoid signal without
noise');
subplot(4,2,2);
[c lags]=xcorr(y_s); plot(lags, c); grid;
title('Autocorrelations');
subplot(4,2,3);
plot(y_low); grid;
title(['Sinusoid signal with low noise, var. '
num2str(variance_low) '.']);
subplot(4,2,4); [c lags]=xcorr(y_low); plot(lags, c); grid;
subplot(4,2,5);
plot(y_medium); grid; title(['Sinusoid signal with medium noise,
var. ' num2str(variance_medium) '.']);
subplot(4,2,6);
[c lags]=xcorr(y_medium); plot(lags,c); grid;
subplot(4,2,7);
plot(y_high); grid; title(['Sinusoid signal with high noise, var.
' num2str(variance_high) '.']);
subplot(4,2,8);
[c lags]=xcorr(y_high); plot(lags,c); grid;
% END
```

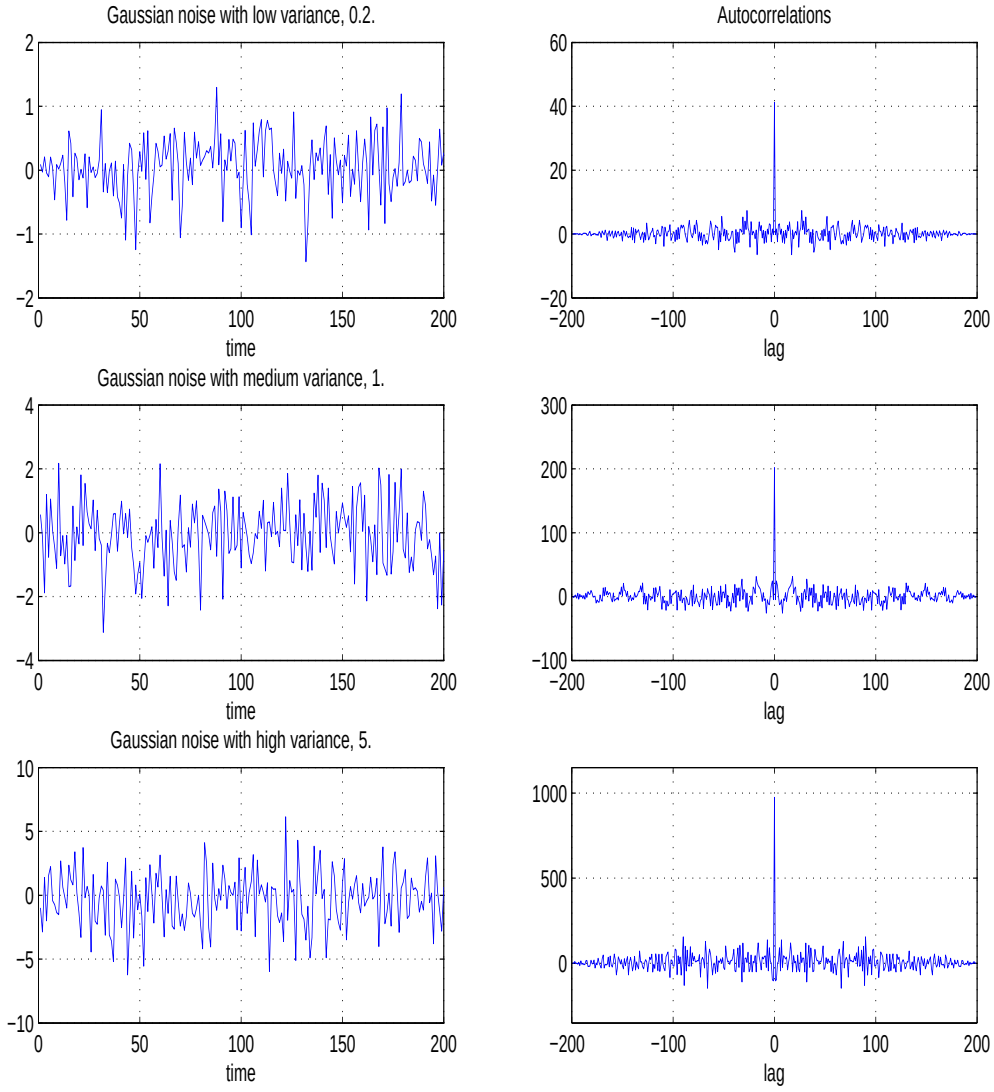

Figure 11: Autocorrelation of the Gaussian white noise.

Figure 11 shows Gaussian white noises with three different variances and their autocorrelations. Autocorrelations have strong peaks at 0. For larger variances peaks are higher. Figures 12, 13 and 14 shows autocorrelations of sinusoid, rectangular and triangular signals with added white noises, respectively. For low and medium variance noises (second and third rows) we can determine the frequency of the original signal from its autocorrelation. For too high variance noise (fourth rows) it is not possible.

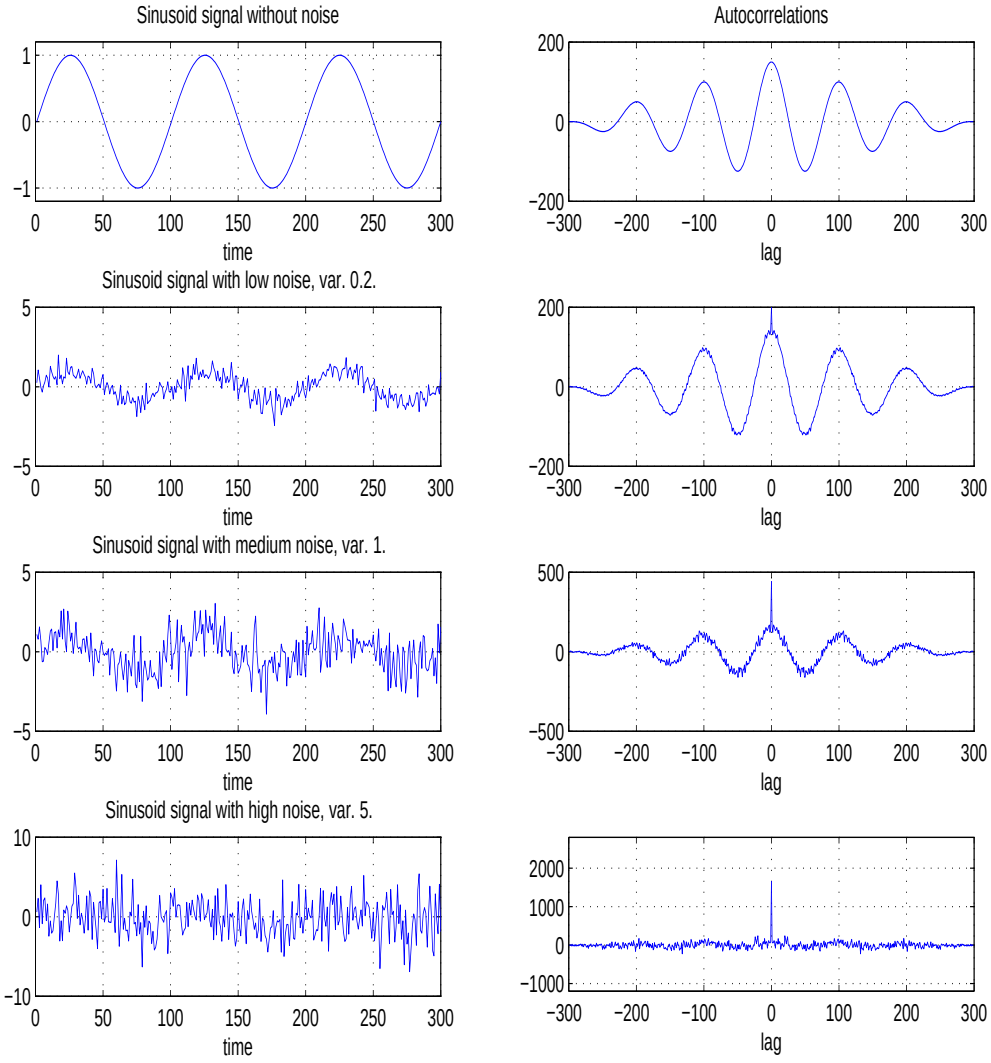

Figure 12: Autocorrelation of the noisy sinusoid signals.

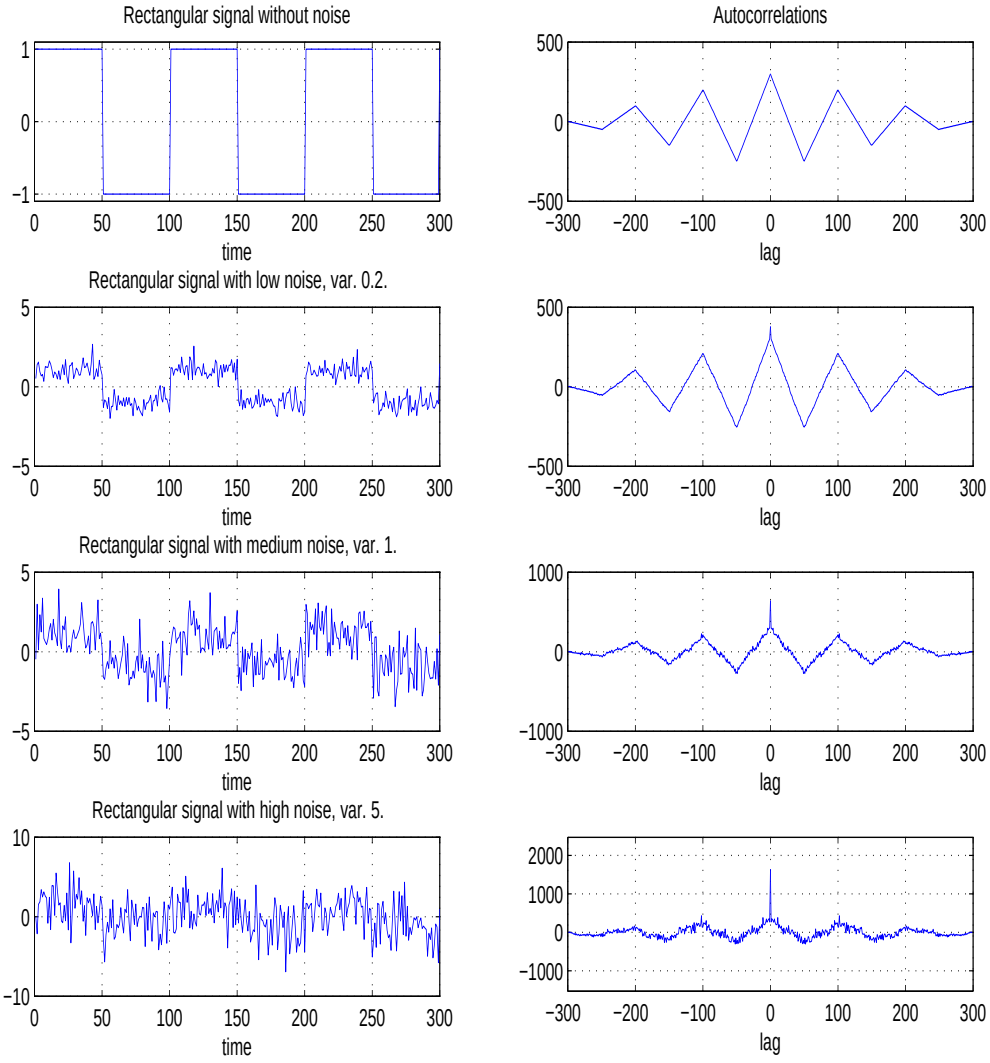

Figure 13: Autocorrelations of the noisy rectangular signals.

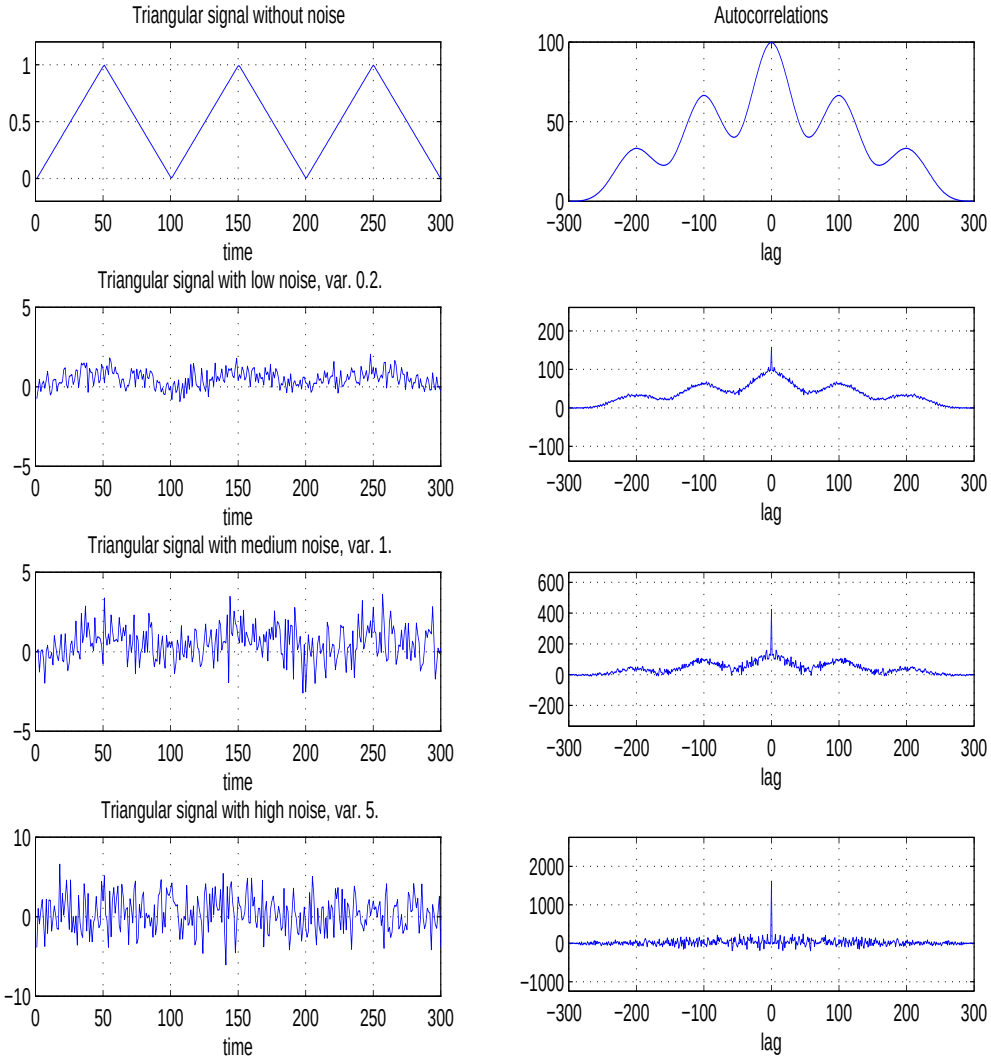

Figure 14: Autocorrelations of the noisy triangular signals.

PART II. Exercise 1. Tibor Lukić, University of Novi Sad.

FOURIER TRANSFORM OF A SIGNAL

Introduction

The study of *Fourier series* gave the motivation for the *Fourier transform*. At the beginning let us consider the Fourier series. The basic idea in the study of Fourier series is that complicated periodic functions are expressed as the weighted sum of simple waves of sines and cosines.

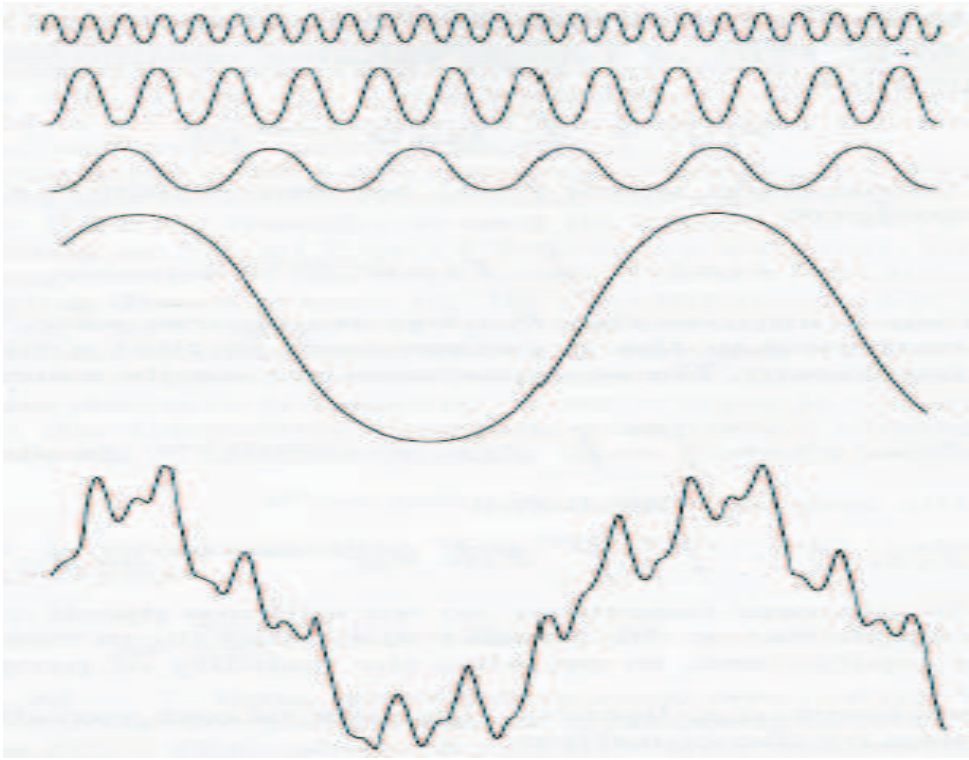

Figure 15: Illustration taken from R. C. Gonzales et al., *Digital Image Processing*, Pretince-Hall.

In Figure 15 the bottom function is the direct sum of the four functions above it. These four function are sines and cosines of different frequencies and amplitudes.

The Fourier series of the 2π-periodic function f on the interval $[-\pi, \pi]$ is defined by

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos nx + \sum_{n=1}^{\infty} b_n \sin nx,$$

where sequences

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx$$

and

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx$$

are *Fourier coefficients* of f. We note that coefficients a_n and b_n are amplitudes of sines and cosines of the frequency n in the Fourier series (sum).

Another formulation is given by Euler's formula

$$e^{in} = \cos nx + i\sin nx,$$

where i is the imaginary unit. Let suppose that

$$f(x) = \sum_{n=-\infty}^{\infty} A_n e^{inx}$$

holds for corresponding complex coefficients A_n. In the following we will determine these coefficients. Now examine

$$\begin{aligned}\int_{-\pi}^{\pi} f(x)e^{-imx}\,dx &= \int_{-\pi}^{\pi}\left(\sum_{n=-\infty}^{\infty} A_n e^{inx}\right)e^{-imx}\,dx \\ &= \sum_{n=-\infty}^{\infty} A_n \int_{-\pi}^{\pi}[(\cos(n-m)x + i\sin(n-m)x]dx \\ &= 2\pi A_m,\end{aligned}$$

hence

$$A_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx\,.$$

It is easy to shown that the relation between complex and real Fourier coefficients is

$$A_n = \begin{cases} \frac{1}{2}(a_{-n} + ib_{-n}), & n < 0 \\ \frac{1}{2}a_0, & n = 0 \\ \frac{1}{2}(a_n - ib_n), & n > 0\,. \end{cases}$$

In a general case, for a periodic function with period L on the interval $[-L/2, L/2]$ we have the following Fourier formulas

$$f(x) = \sum_{n=-\infty}^{\infty} A_n e^{i(2\pi nx/L)} \tag{5}$$

and

$$A_n = \frac{1}{L}\int_{-L/2}^{L/2} f(x)e^{-i(2\pi nx/L)}\,dx\,. \tag{6}$$

The Fourier transform is a generalization of the complex Fourier series. Introducing a substitution $n/L = \omega$, replacing the discrete A_n with continuous $F(\omega)$ and letting $L \to \infty$ from (5) and (6) we get

$$f(x) = \int_{-\infty}^{\infty} F(\omega)e^{i2\pi\omega x}d\omega \tag{7}$$

and

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx, \tag{8}$$

where $dk \sim \triangle\omega = (n+1)/L - n/L = 1/L$. The function F defined by (8) is the *Continuous Fourier transform* of f. The argument ω is a frequency of f. The transform F, often called the *frequency domain* representation of the original function, describes which frequencies are present in the original function f. Relation (7) defines the *inverse Continuous Fourier transform*. It makes possible to completely reconstruct a function (signal) from its Fourier transform, which is an important property. Values of the Fourier transform for each ω, $F(\omega)$ are complex numbers, hence we can represent they in exponential form

$$F(\omega) = |F(\omega)| \cdot e^{i\phi(\omega)},$$

where modulus (magnitude) $|F(\omega)|$ is called as *real* or *Fourier spectrum* of f and $\phi(\omega)$ is *phase angle*.

For practical application of the Fourier transform, especially for digital signal processing its discretization is needed. Let sequence $x(t_d)$, $t_d = 0, \ldots, N-1$ represents a time-digital signal. The *discrete Fourier transform* of $x(t_d)$ is defined by

$$X(\omega_d) = \sum_{t_d=0}^{N-1} x(t_d) e^{-i\frac{2\pi}{N} t_d \omega_d}, \quad \omega_d = 0, \ldots, N-1$$

and its inverse transform is given by

$$x(t_d) = \sum_{\omega_d=0}^{N-1} X(\omega_d) e^{i\frac{2\pi}{N} t_d \omega_d}, \quad t_d = 0, \ldots, N-1\,.$$

Unfortunately, the discrete Fourier transform is not so efficient, especially in real-time applications. The reason is the number of complex multiplications and additions required which is proportional to N^2. To overcome this disadvantage the *fast Fourier transform* (FFT) is developed. In our experiments we will calculate the Fourier transform using the FFT algorithm implemented in MATLAB environment.

Experiments

Problem

"Calculate and visualize the real spectrum and phase angle of rectangular and triangular signals, first without noise, then with added Gaussian white noise (mean value is zero)."

Matlab codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate and visualize Fourier spectrum and phase
% of the centered fast Fourier transformation
% of triangular and rectangular signal
% (without any noise).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter settings
amp=1; fase=0; freq=1; range=[0,1]; ncamp=freq*100;
freq_dom=zeros(1,ncamp); % frequency domain scale
% create the centered frequency domain scale
j=1; for i=ncamp/2:-1:0
    freq_dom(j)=-i;
    j=j+1;
end; for i=1:(ncamp/2-1)
    freq_dom(i+ncamp/2+1)=i;
end; subplot(2,3,1);
% calculate and plot the rectangular signal
y_r=rectangular(amp,freq,fase,range,ncamp); title('Rectangular
signal'); xlabel('time');
subplot(2,3,2);
spec=fft(y_r); % fast Fourier trasform of rectangular signal
spec=fftshift(spec); % moving the zero frequency to the center
realspec_r=abs(spec); % real spectrum of triangular signal
phasesp_r=angle(spec); % phase of triangular signal
plot(freq_dom,realspec_r); grid; title('Real spectrum');
xlabel('frequency');
subplot(2,3,3);
plot(freq_dom,phasesp_r); grid; title('Phase');
xlabel('frequency');
subplot(2,3,4);
% calculate and plot the triangular signal
y_t=triangular(amp,freq,fase,range,ncamp); title('Triangular
signal'); xlabel('time');
subplot(2,3,5);
spec=fft(y_t); spec=fftshift(spec);
realspec_t=abs(spec); phasesp_t=angle(spec);
plot(freq_dom,realspec_t); grid;
title('Fourier spectrum');
xlabel('frequency');
subplot(2,3,6); plot(freq_dom,phasesp_t);
grid; title('Phase'); xlabel('frequency');
% END
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate and visualize Fourier spectrum and phase
% of the rectangular signal
% whit added Gaussian white noise.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter settings for rectangular signal.
amp=1; fase=0; freq=50; range=[0,1]; ncamp=200;
freq_dom=zeros(1,ncamp); % frequency domain scale
% create the centered frequency domain scale
j=1; for i=ncamp/2:-1:0
    freq_dom(j)=-i;
    j=j+1;
end; for i=1:(ncamp/2-1)
    freq_dom(i+ncamp/2+1)=i;
end;
tmin=range(1,1); % minimum time value
tmax=range(1,2); % maximum time value
tper=abs(tmax-tmin)/freq; % signal period
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
% set used variance values
variance_low=1;      % low varinace value
variance_medium=10;  % medium varinace value
variance_high=100;   % high varinace value
% calculate gaussian nosise with zero mean and different variance
low_noise=randn(1,ncamp)*sqrt(variance_low);
medium_noise=randn(1,ncamp)*sqrt(variance_medium);
high_noise=randn(1,ncamp)*sqrt(variance_high);
% calculate the rectangular signal
y_r=rectangular(amp,freq,fase,range,ncamp);
subplot(3,3,1);
y_low=y_r+low_noise; % add low noise
plot(xt,y_low); grid; title(['Noisy rec. , freq.: ' num2str(freq)
', variance: ' num2str(variance_low) '.']); xlabel('time');
subplot(3,3,2);
spec=fft(y_low); % fast Fourier transform
spec=fftshift(spec); % moving the zero frequency
                     % component to the center
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Real
spectrum'); xlabel('frequency'); subplot(3,3,3);
phasesp=angle(spec); plot(freq_dom,phasesp); grid; title('Phase');
xlabel('frequency');
subplot(3,3,4);
y_medium=y_r+medium_noise; % add medium noise
plot(xt,y_medium); grid; title(['Noisy rec., freq.: '
num2str(freq) ', variance: ' num2str(variance_medium) '.']);
```

```
xlabel('time');
subplot(3,3,5);
spec=fft(y_medium); % fast Fourier transform
spec=fftshift(spec); % moving the zero frequency
                     % component to the center
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Real
spectrum'); xlabel('frequency');
subplot(3,3,6);
phasesp=angle(spec); plot(freq_dom,phasesp); grid; title('Phase');
xlabel('frequency');
subplot(3,3,7);
y_high=y_r+high_noise; % add high noise
plot(xt,y_high); grid; title(['Noisy rec. , freq.: ' num2str(freq)
', variance: ' num2str(variance_high) '.']); xlabel('time');
subplot(3,3,8);
spec=fft(y_high); % fast Fourier transform
spec=fftshift(spec); % moving the zero frequency
                     % component to the center
realspec=abs(spec); plot(freq_dom,realspec); grid;
title('Fourier spectrum'); xlabel('frequency');
subplot(3,3,9);
phasesp=angle(spec); plot(freq_dom,phasesp); grid;
title('Phase'); xlabel('frequency');
% END
```

The code for noisy triangular signal is very similar to the above presented, therefore we omit it.

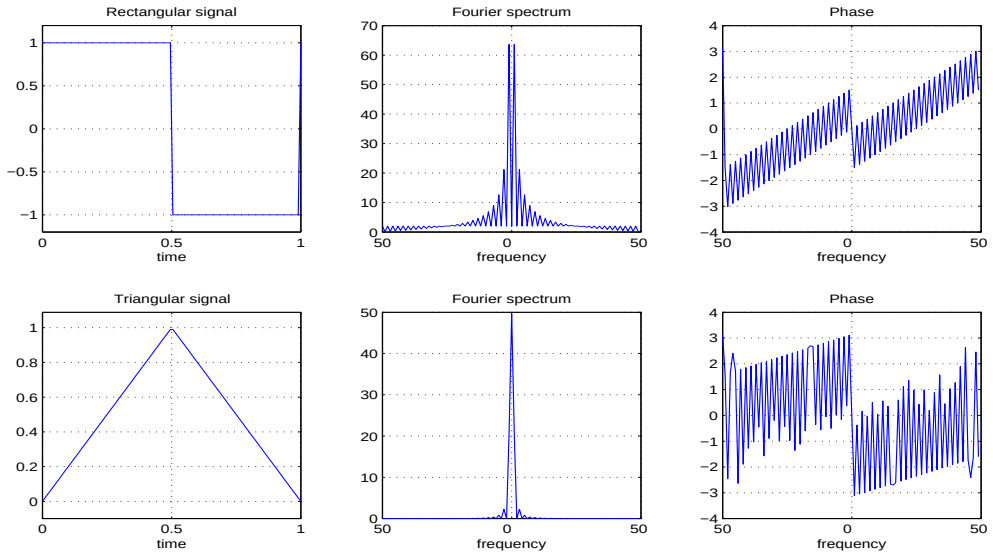

Figure 16: Fourier spectrum and phase of the rectangular and triangular shape signals.

First column of Figure 16 shows one wave of two signals: rectangular and triangular. Second and third rows show real spectrum and phase of Fourier transforms of the given two signals. In both cases the dominant frequencies are very low and around zero (see real spectrum). This is because high frequencies correspond to signals with high frequencies or to signals with sudden and large changes in intensity.

Figures 17 and 18 show noised rectangular and triangular signals respectively. The noisy signals are obtained by adding a Gaussian white noise to original data. Fourier transform of noisy signal allows to determine the frequency of original signal even it is not 'visible' from the noisy signal. This we may observe in first and second row of presented figures. However, signals in third rows are to much noisy, so the original frequency cannot be determine by Fourier transforms.

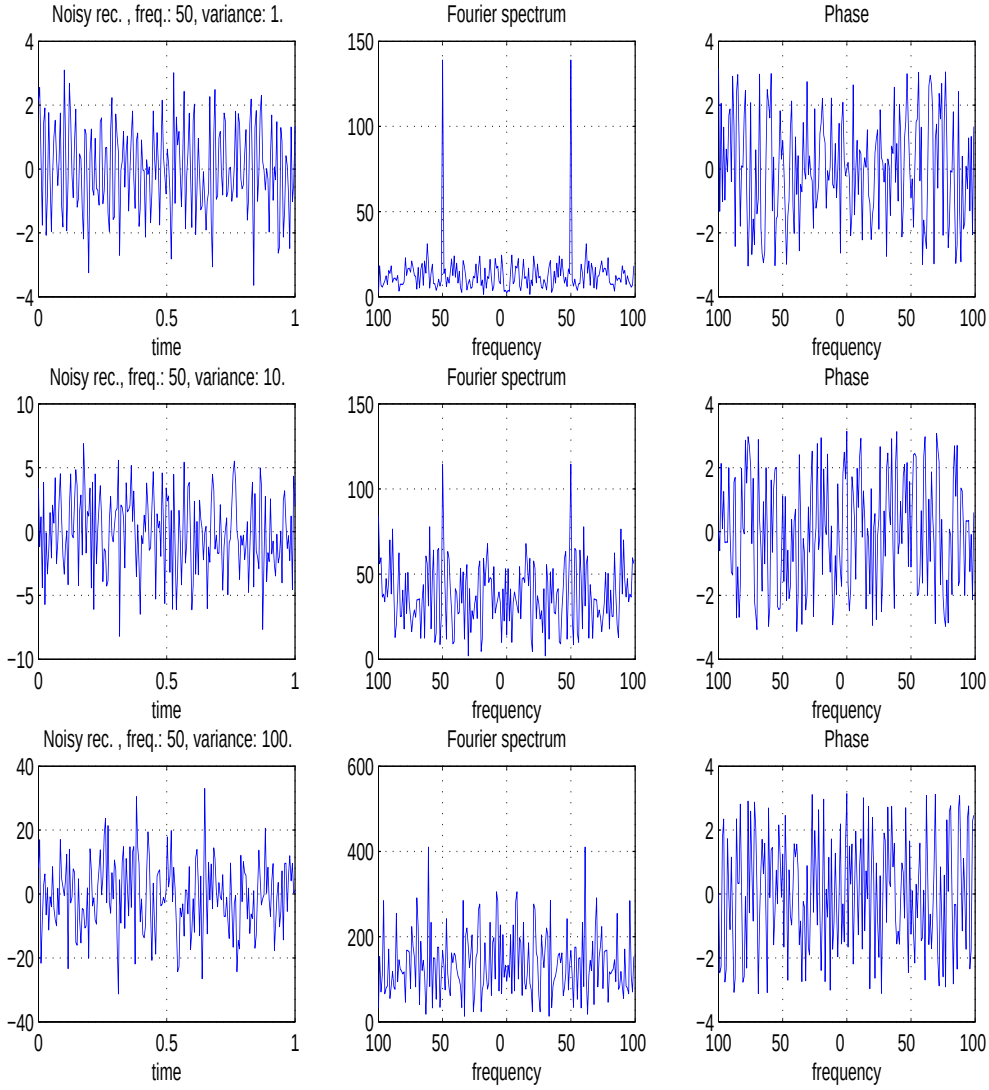

Figure 17: Rectangular noisy signal.

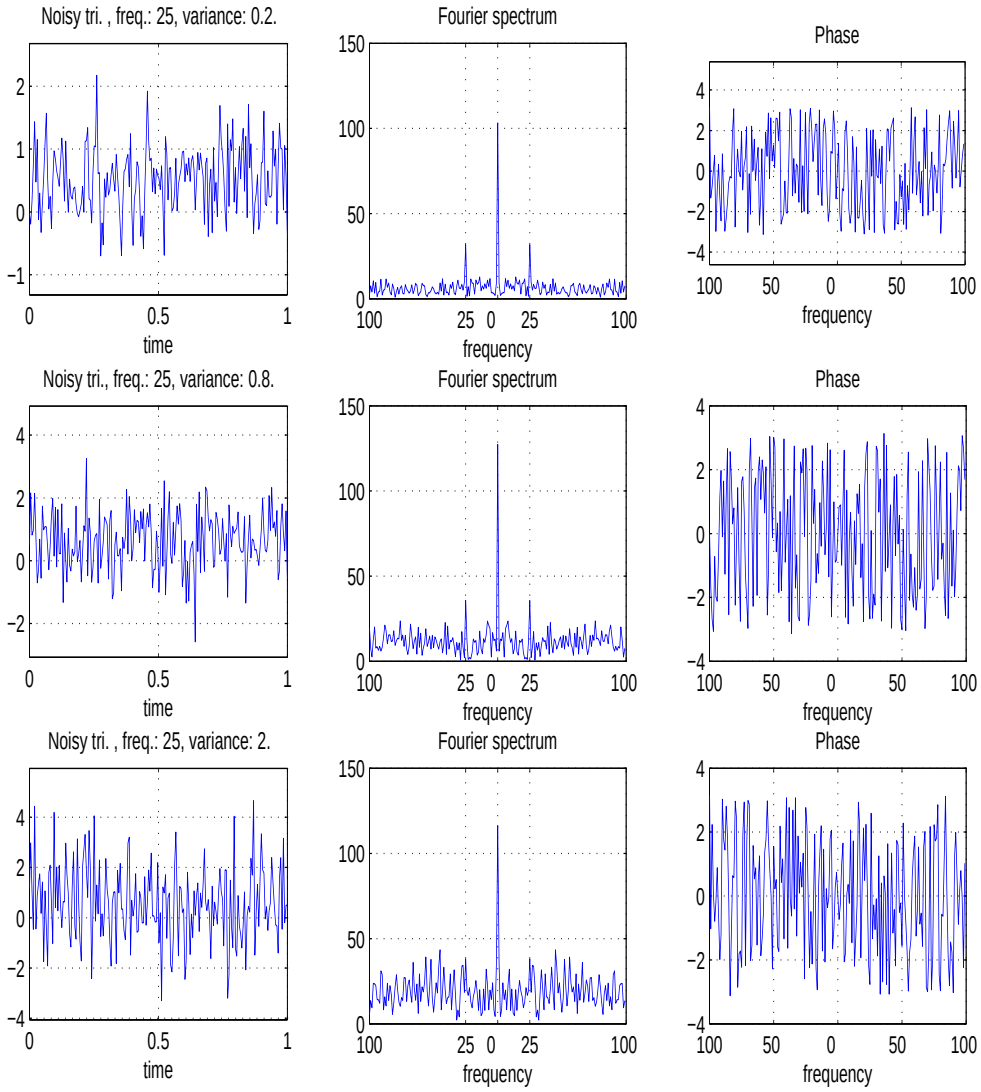

Figure 18: Triangular noisy signal.

PART II. Exercise 2. Tibor Lukić, University of Novi Sad.

FOURIER SPECTRUM OF A SIGNAL

Introduction

See the introduction of the previous Exercise.

Experiments

Problem

"Calculate and visualize the Fourier spectrums of sinusoid signals with different frequencies. After repeat the experiment with the same signals but with added Gaussian white noise."

Matlab codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculation of Fourier spectrum
% of sinusoid signals with different frequencies
% and without and with added Gaussian white noise.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
% parameter settings for sinus signal.
amp=1; fase=0; range=[0,1]; ncamp=100;
freq_dom=zeros(1,ncamp); % frequency domain scale
% create the centered frequency domain scale
j=1; for i=ncamp/2:-1:0
    freq_dom(j)=-i;
    j=j+1;
end; for i=1:(ncamp/2-1)
    freq_dom(i+ncamp/2+1)=i;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% WITHOUT NOISE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure; subplot(3,2,1); freq=1;
y_s_1=sinus(amp,freq,fase,range,ncamp); % calculate the sinus signal
```

```
title(['Signal freq.: ' num2str(freq)]); xlabel('time');
subplot(3,2,2);
spec=fft(y_s_1); % fast Fourier transform
%spec=fftshift(spec); % moving the zero frequency
                     % component to the center
realspec=abs(spec); plot(realspec); grid; title('Fourier
spectrum'); xlabel('frequency'); subplot(3,2,3); freq=10;
y_s_10=sinus(amp,freq,fase,range,ncamp); % calculate the sinus signal
title(['Signal freq.: ' num2str(freq)]); xlabel('time');
subplot(3,2,4); spec=fft(y_s_10);
% spec=fftshift(spec);
realspec=abs(spec); plot(realspec); grid; title('Fourier
spectrum'); xlabel('frequency'); subplot(3,2,5); freq=20;
y_s_20=sinus(amp,freq,fase,range,ncamp); % calculate the sinus signal
title(['Signal freq.: ' num2str(freq)]); xlabel('time');
subplot(3,2,6); spec=fft(y_s_20); spec=fftshift(spec);
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Fourier
spectrum'); xlabel('frequency');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% WITH GAUSSIAN NOISE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tmin=range(1,1); % minimum time value
tmax=range(1,2); % maximum time value
xt=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
variance=1;  % varinace of noise
noise=randn(1,ncamp)*sqrt(variance); % Gaussian
                                     % white noise data
y=y_s_1+noise; % add the noise to the signal
figure; subplot(3,2,1); plot(xt,y); grid; title('Signal freq.:
1'); xlabel('time'); subplot(3,2,2);
spec=fft(y); % fast Fourier transform
spec=fftshift(spec); % moving the zero frequency component to the center
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Fourier
spectrum'); xlabel('frequency'); subplot(3,2,3);
y=y_s_10+noise; % add the noise to the signal
plot(xt,y); grid; title('Signal freq.: 10'); xlabel('time');
subplot(3,2,4); spec=fft(y); spec=fftshift(spec);
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Fourier
spectrum'); xlabel('frequency'); subplot(3,2,5);
y=y_s_20+noise; % add the noise to the signal
plot(xt,y); grid; title('Signal freq.: 10'); xlabel('time');
subplot(3,2,6); spec=fft(y); spec=fftshift(spec);
realspec=abs(spec); plot(freq_dom,realspec); grid; title('Fourier
spectrum'); xlabel('frequency');
% END
```

Figure 19 shows sinus signals and their real spectrums of the corresponding Fourier transforms. For each signal biggest real spectrum value agrees with the signal frequency. In Figure 20 we consider noisy signals and their real Fourier spectrums. The noisy signals are obtained by adding a Gaussian white noise with variance 1 to the sinusoid signals considered in Figure 19. The dominant frequencies in real spectrums agree with the frequencies of the original signals, even they are not evident from noisy signals.

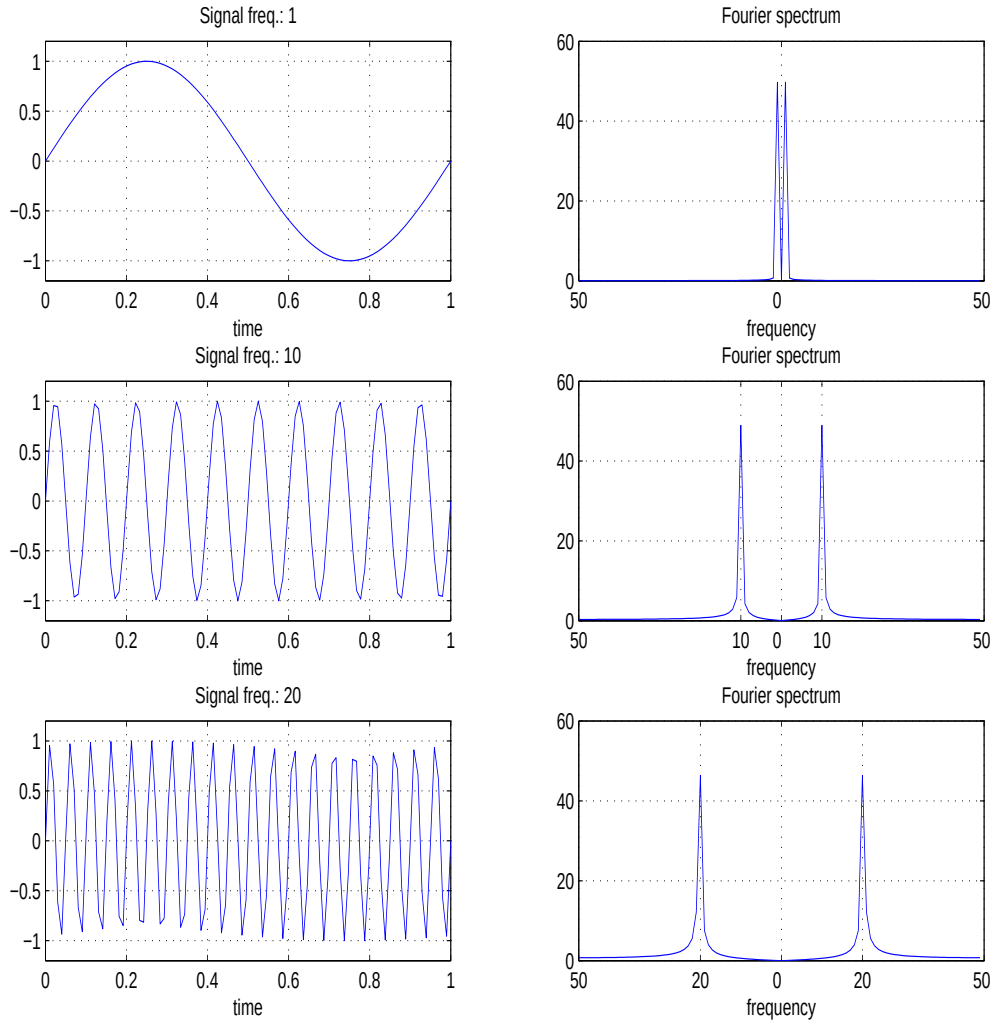

Figure 19: Fourier spectrums of signals without noise.

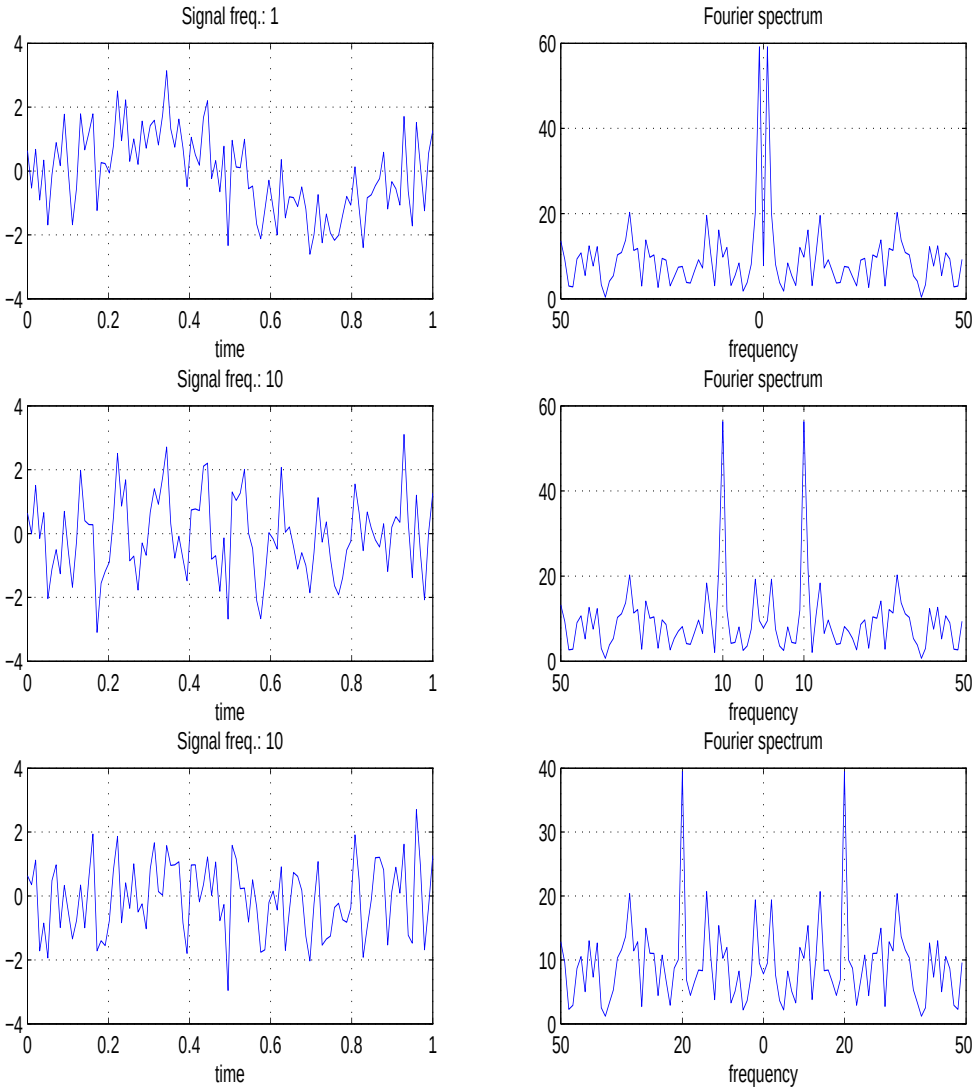

Figure 20: Fourier spectrums of noisy signals.

PART II. Exercise 3. Tibor Lukić, University of Novi Sad.

SIGNAL FILTERING

Introduction

Signal filtering refers to an attempt to extract the important part of the received signal/data while eliminating random contributions called "noise" or other unwanted features which obscure the ones that matter. In a fact, filter operator is signal transform operator with certain properties. If Θ denotes the filter operator then we have

$$\Theta[f_I(t)] = f_O(t),$$

when f_I and f_O are input and output time-signal functions, respectively. In the following we will consider *Linear Time-Invariant* (LTI) signal filters. The LTI filer, Θ_{LTI} has two important property:

1) It is linear, that is, if $f_{O1}(t) = \Theta_{LTI}[f_{I1}(t)]$ and $f_{O1}(t) = \Theta_{LTI}[f_{I1}(t)]$ then

$$\Theta_{LTI}[c_1 f_{I1}(t) + c_2 f_{I2}(t)] = c_1 f_{O1}(t) + c_2 f_{O2}(t),$$

where c_1 and c_2 are constant numbers.

2) It is time-invariant, that is,

$$\Theta_{LTI}[f_I(t + \tau_D)] = f_O(t + \tau_D),$$

where τ_D is time-delay.

Convolution is a commonly used operation in LTI filtering. The convolution operation defined by

$$f_O(t) = \int f_I(t) I(\tau - t)\, dt$$

satisfies the above two properties of the LTI filter. The function I is called as *impulse response* function. The properties of the impulse response function determine the output signal. This kind of filtering is also called as *time domain filtering*. However, filtering can be done directly in the frequency domain too, by operating on the signal's Fourier transform. This is called as *frequency domain filtering* and it is realized by the following formula

$$F_O(\omega) = F_I(\omega) \cdot T(\omega),$$

where F_0 and F_I are output and input signals Fourier transforms, respectively. Function T is a *filter* function which suppresses certain frequencies in the transform while leaving others unchanged. The output signal, f_O is obtained as an inverse Fourier transform of F_O.

The relationship between the time and frequency domain filtering is given by the *convolution theorem*. It states that under suitable conditions the Fourier transform of a convolution is the point-wise product of Fourier transforms. Let express it more precisely. Let f and g be two functions with convolution $f * g$ and $\mathcal{F}$ denote the Fourier transform operator. Then by the convolution theorem the following equations hold

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$$

and

$$\mathcal{F}(f \cdot g) = \mathcal{F}(f) * \mathcal{F}(g).$$

Experiments

Problem

"Add Gaussian white noise to the sinus signal and then filter the obtained noisy signal. Do filtering using time and frequency domain filtering. Visualize the input and output signal as well as impulse response and filter functions. "

Matlab codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SIGNAL FILTERING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
% parameter settings for rectangular signal.
amp=1; fase=0; freq=5; range=[-1,1]; ncamp=200;
figure; subplot(1,2,1);
y=sinus(amp,freq,fase,range,ncamp); % sinus signal
plot(y); grid;% visualize the original signal
title('Original signal'); xlabel('discrete samples');
variance=0.2;      % noise varinace
% calculate gaussian nosise with zero mean
noise=randn(1,ncamp)*sqrt(variance);
% add noise and create the input signal
y_I=y+noise;  % input signal
subplot(1,2,2);
plot(y_I); grid; % visualize the input signal
title('Input (noisy) signal'); xlabel('discrete samples');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TIME DOMAIN FILTERING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
irf_variance=3; % inpuls response function (irf)/mask
tmin=-1; % min for Gaussian irf
tmax=1;  % max for Gaussian irf
ncamp_g=10; % number of discret values
xtg=tmin:abs(tmax-tmin)/(ncamp_g-1):tmax; % time discretization
I_f=pdf_gaussian(xtg,0,irf_variance); % irf/mask
figure; subplot(1,2,1);
plot(I_f); grid; % visualize the irf/mask
title('Impulse response function'); xlabel('discrete samples');
y_O=conv(y_I,I_f); % convolution calculation
subplot(1,2,2);
plot(y_O); grid; % visualize the output signal
title('Output signal'); xlabel('discrete samples');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FREQUENCY DOMAIN FILTERING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filter_variance=0.01;
tmin=-1; % minimum time value
tmax=1;  % maximum time value
xtg=tmin:abs(tmax-tmin)/(ncamp-1):tmax; % time discretization
T_f=pdf_gaussian(xtg,0,filter_variance); % filter function
figure; subplot(2,2,2);
plot(T_f); grid; title('Filter function');
xlabel('discrete samples');
spec_I=fft(y_I); F_I=fftshift(spec_I);
F_O=T_f.*F_I; % Fourier transform (FT) of the output signal
spec_O=ifftshift(F_O); % re-shifting of the FT
y_O=real(ifft(spec_O));
subplot(2,2,4); plot(y_O); grid;
title('Output (filtered) signal'); xlabel('discrete samples');
freq_dom=zeros(1,ncamp); % frequency domain scale
% create the centered frequency domain scale
j=1; for i=ncamp/2:-1:0
    freq_dom(j)=-i;
    j=j+1;
end; for i=1:(ncamp/2-1)
    freq_dom(i+ncamp/2+1)=i;
end;
% visualize the input and output signal's Fourier spectrums
subplot(2,2,1);
plot(freq_dom,abs(F_I)); grid; % show input signal's F. spectrum
xlabel('frequency'); title('Fourier spectrum of the input
signal');
subplot(2,2,3);
plot(freq_dom,abs(F_O));  grid; % show output signal's spectrum.
```

```
xlabel('frequency'); title('Fourier spectrum of the output
signal');
% END
```

Results

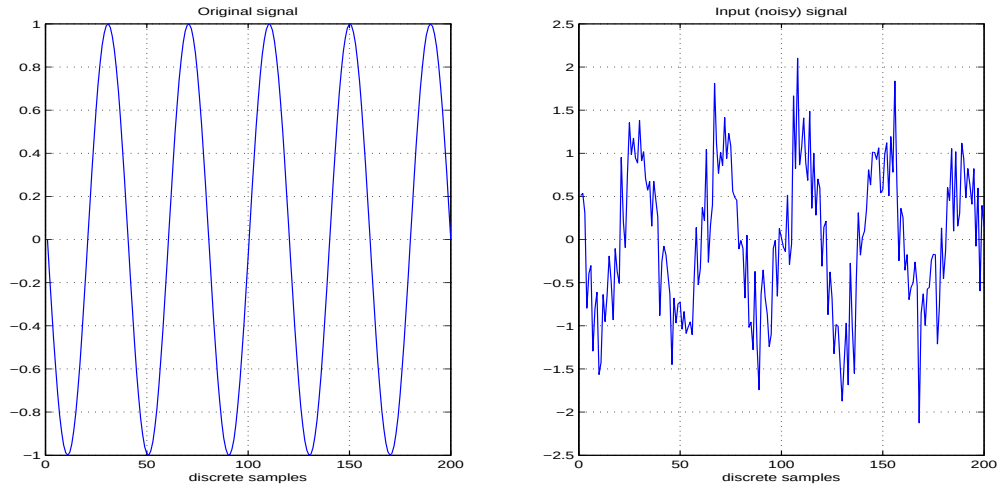

Figure 21: Original and input (noisy) signals.

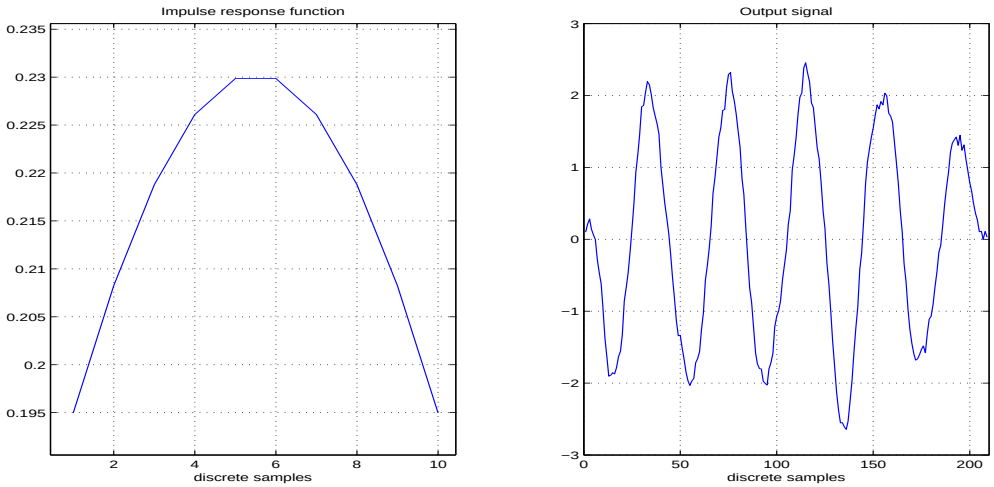

Figure 22: Time domain filtering.

Within this exercise we filter the given noisy sinus signal. The noisy signal is obtained by adding a Gaussian white noise to the original signal, see Figure 21. First, we filtering in time domain. The used impulse response function is the Gaussian probability density function with zero mean and 0.1 variance. On Figure 22 we can see the effect of this filtering. The output signal is less noisy and smoother than the input signal. We can conclude that the Gaussian impulse response function has a 'smoothing' effect in filtering. Similar goal is achieved by filtering in frequency domain. The Figure 23 shows the effect of such filtering. The used filter function is the Gaussian probability density function with zero mean and 0.01 variance. The smoothing effect of this filtering is achieved by removing high frequencies in signal's frequency domain (see Figure 23). High frequencies correspond with rapid signal intensity changes, when low frequencies correspond with constant or smooth signal intensity.

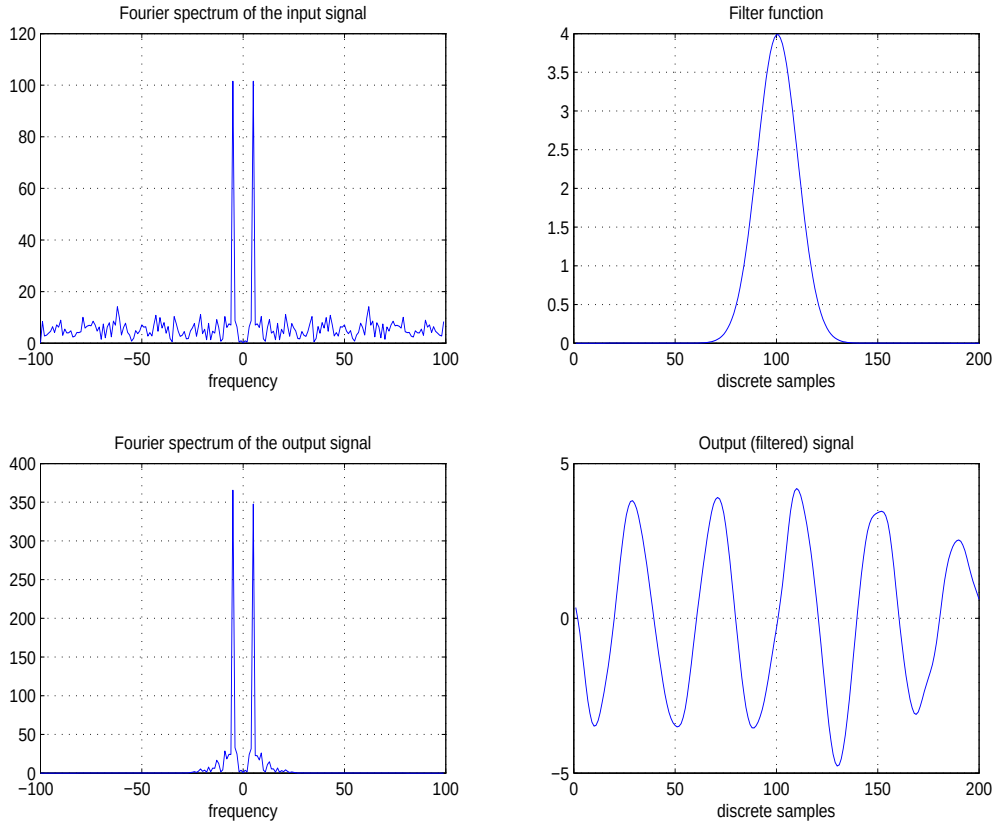

Figure 23: Frequency domain filtering.

PART III. Exercise 1. Tibor Lukić, University of Novi Sad.

FOURIER TRANSFORM OF THE IMAGE

Introduction

The Fourier transform of a single variable (see Fourier transform of a signal) is straightforward extended to two variables. The Fourier transform of a continuous two variable function, $f(x, y)$ is defined by

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-i2\pi(ux+vy)} \, dxdy$$

and its *inverse* transform is given by

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{-i2\pi(ux+vy)} \, dudv.$$

Now, variable pair (u, v) belongs to the *two-dimensional frequency domain* (plane or part of plane) and (x, y) belongs to the *space domain* of f. Similarly, as in one dimensional case, values of the Fourier transform for each (u, v), are complex numbers, therefore we can represent them in exponential form

$$F(u, v) = |F(u, v)| \cdot e^{i\phi(u,v)},$$

where modulus (magnitude) $|F(\omega)|$ is called as *real* or *Fourier spectrum* of f and $\phi(u, v)$ is called as *phase angle*.

Often in practical applications, for instance when $f(x, y)$ represents a digital image, the discrete form of the Fourier transformation and its inverse is used. The *two-dimensional discrete Fourier transform* of a discrete function $f(m, n)$ of size $M \times N$ is given by

$$F(p, q) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) e^{-i2\pi(pm/M+qn/N)}, \qquad \begin{matrix} p = 0, 1, 2, \ldots, M-1 \\ q = 0, 1, 2, \ldots, N-1 \end{matrix}$$

and its inverse

$$f(m, n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} F(p, q) e^{i2\pi(pm/M+qn/N)}, \qquad \begin{matrix} m = 0, 1, 2, \ldots, M-1 \\ n = 0, 1, 2, \ldots, N-1 \, . \end{matrix}$$

The variables p and q are frequency variables and m and n are spacial variables. We note that the value of the transform at $(p, q) = (0, 0)$,

$$F(0, 0) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n)$$

is the average of $f(m,n)$. If $f(m,n)$ represents an image, the value of discrete Fourier transform at $(0,0)$ is equal to the average gray level of the image.

The MATLAB function fft2 implements the fast Fourier transform algorithm for computing the two-dimensional discrete Fourier transform and MATLAB function ifft2 computes the inverse two-dimensional discrete Fourier transform. In visualizations and data processing it is common practice to rearrange the frequency domain in such way that the zero-frequency component takes place in center of the domain. This rearranging in MATLAB is realized by the function fftshift.

Experiments

Problem

"Calculate and visualize the Fourier spectrum of the given synthetic image with and without added Gaussian white noise. Compare the Fourier spectrums of images with and without noise."

Matlab codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FOURIER TRANSFORM OF THE IMAGE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create a synthetic image
f=zeros(30,30); f(5:24,13:17)=1;
subplot(1,3,1);
imshow(f,[]); % original image
title('Original image'); ft=fft2(f); fts=fftshift(ft);
subplot(1,3,2);
imshow(log(1+abs(fts))); title('Fourier
spectrum'); subplot(1,3,3); mesh(log(1+abs(fts)),[]); title('3D
visualization');
% add Gaussian white noise to the image
noise_variance=0.01;
f_noise=imnoise(f,'gaussian',0,noise_variance); figure;
subplot(1,3,1);
imshow(f_noise,[]); title('Noisy image');
ft=fft2(f_noise); fts=fftshift(ft); subplot(1,3,2);
imshow(log(1+abs(fts)),[]); title('Fourier spectrum');
subplot(1,3,3);
mesh(log(1+abs(fts))); title('3D visualization');
% END
```

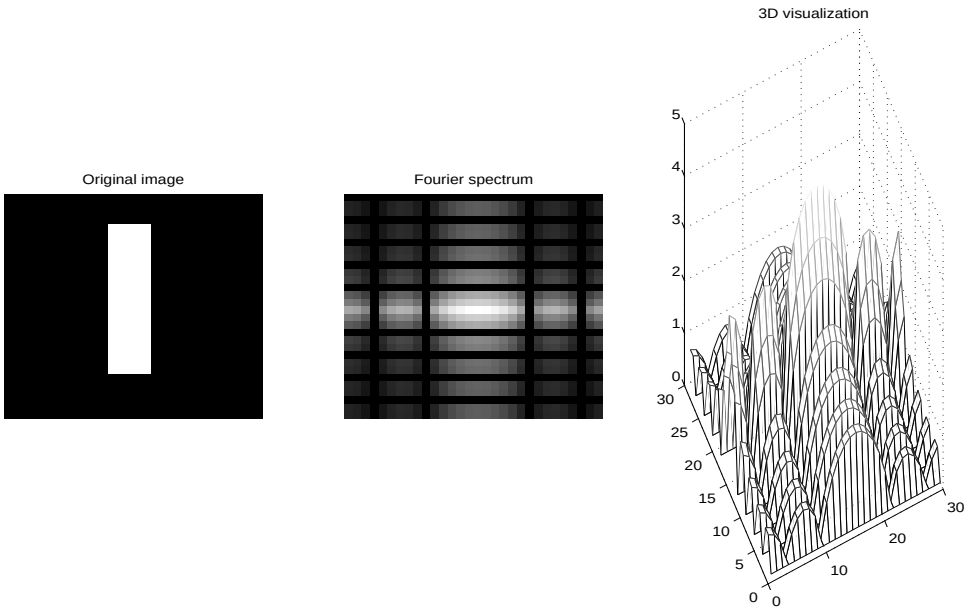

Figure 24: Fourier spectrum of the original image.

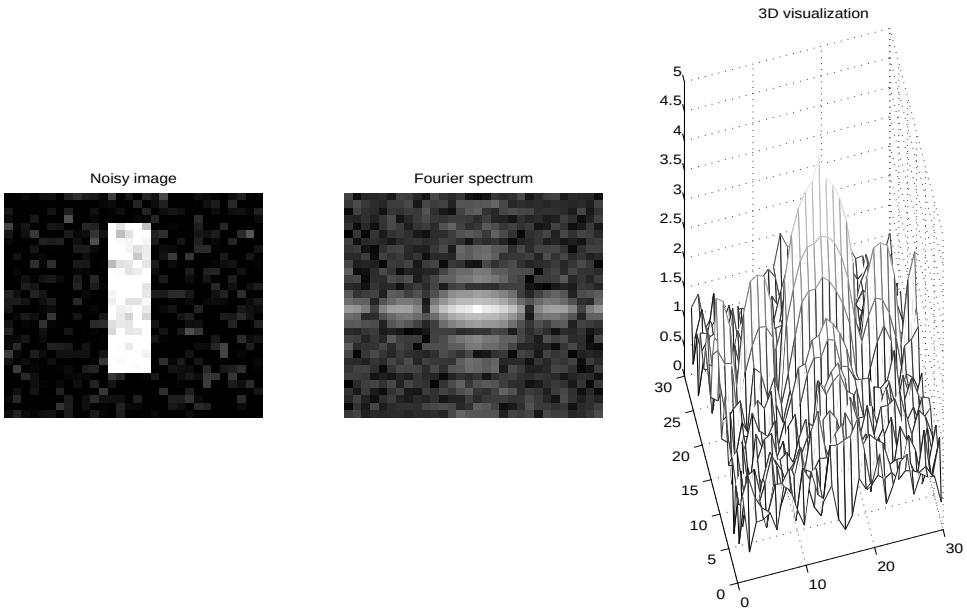

Figure 25: Fourier spectrum of the noisy image.

Let us consider the Figure 24. It contains a white rectangular placed vertically on black background. Fourier spectrum (centered) shows the intensities (energy) of the frequency components. Low frequency components have high intensities which agree with the white area of the spectrum's center. The plot also shows that the spectrum has more energy at high horizontal frequencies than at high vertical frequencies. This reflects the fact that horizontal cross sections of the image are "narrow pulses", while vertical cross sections are "broad pulses". Narrow pulses have more high-frequency content than broad pulses. Figure 25 shows a noisy image and its Fourier spectrum. The consequences of the noise are the present of more high-frequency energies in all directions in the spectrum. This is the main difference in compare with the spectrum of the original image (see Figure 24). Removing this high-frequencies (reflected by light-gray and white pixels in the spectrum) located between spectrum's center and its border, but keeping the low-frequencies in spectrum's center unchanged is a way for reconstruction of the original image from noisy data. More about this you can find in next exercise (Image filtering).

PART III. Exercise 2. Tibor Lukić, University of Novi Sad.

IMAGE FILTERING

Introduction

Image filtering is one of the most common way to process digital images. Similarly as in signal filtering image filtering refers to an attempt to eliminating random contributions called "noise" or other unwanted features which obscure the ones that matter. We note that the determination of unwanted features is very much problem oriented and there exist a large number of filtering types with different effect on the given image. Image filtering techniques fall into two broad categories: *spatial domain filtering* and *frequency domain filtering*.

Spacial domain filtering refers to the approaches based on the direct manipulation of pixels in an image. The method consists of moving the *filter mask* from point to point in an image. At each point the filter's *response* is commonly defined as a sum of products of the filter coefficients (weights) and the corresponding image pixels intensities in the area spanned by the mask. The scheme of spacial filtering is illustrated in Figure 26. Convolution is a commonly used operator in spacial domain filtering. If f and ψ are functions of two discrete variables, then the formula for the two-dimensional discrete convolution, c is given by

$$c(u,v) = \sum_m \sum_n f(m,n) \cdot \psi(u-m, v-n).$$

In image filtering convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. If f represents an input image then c is the filtered or output image. The function of weights is given by the ψ and called as convolution mask or *point spread function* (PSF). The PSF determines the character of the filtering (smoothing, sharpening etc.) and there exist a large number of such functions. One of the well known example for PSF is the Gaussian probability density function, which has a smoothing or noise reduction effect.

The frequency domain of an image is a space defined by values of its Fourier transform and correspondent frequency variables. Filtering in frequency domain refers to approaches which manipulate with values of Fourier transform. Similarly as in case of signal filtering, the Fourier transform of the output image is given by

$$F_O(u,v) = F_I(u,v) \cdot T(u,v),$$

where u and v are frequency variables, and F_I is the Fourier transform of the input image, and T is a given *filter* function. The output image is obtained as an inverse Fourier transform of $F_O(u,v)$, see Figure 27. The filter function, T suppresses certain frequencies in the transform while leaving others unchanged. Two well know categories of filters are *lowpass* and *highpass* filters. Lowpass filters attenuate high frequencies, while leave low ones unchanged. Gaussian probability density function is commonly used such

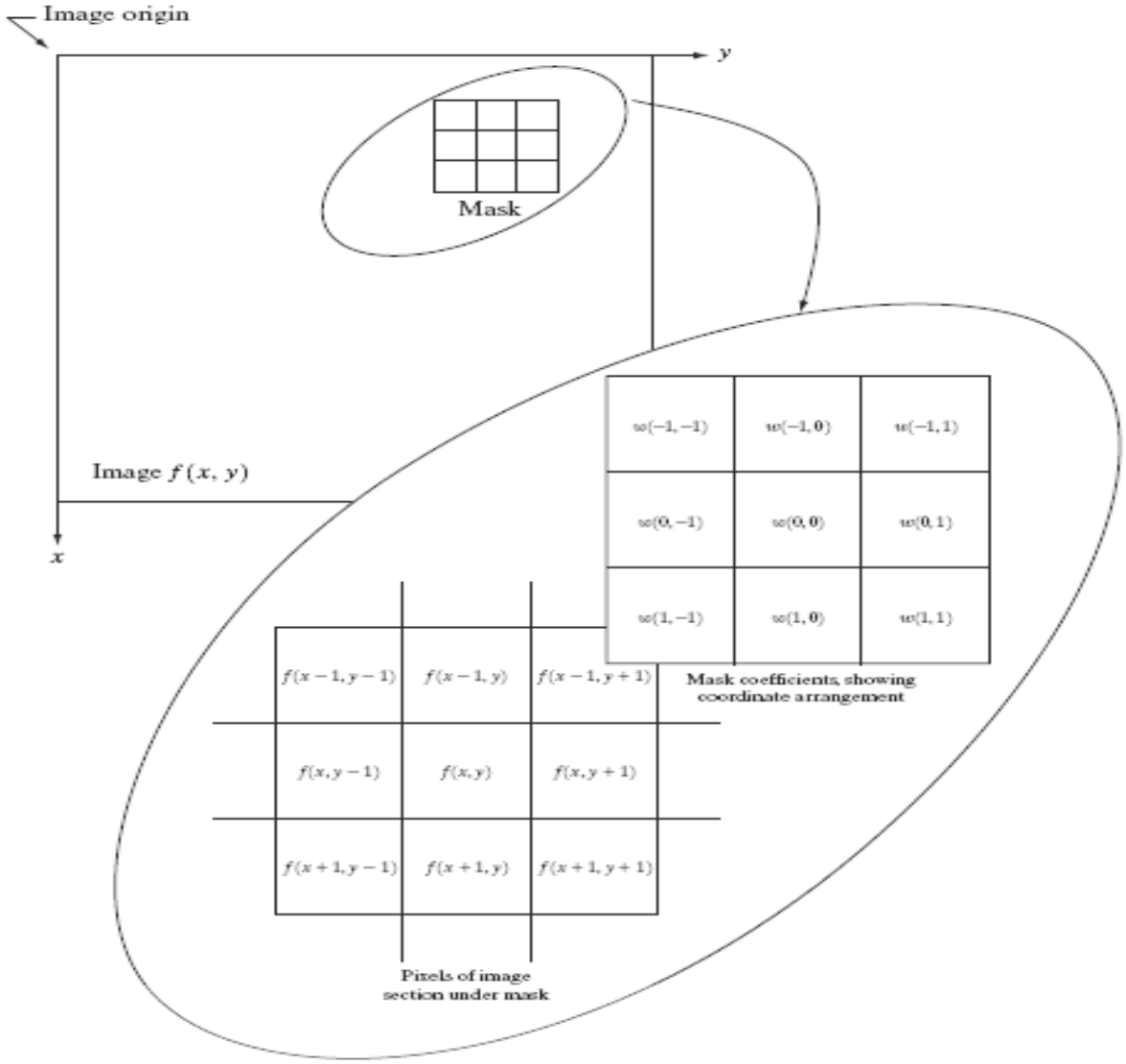

Figure 26: The scheme of spacial domain filtering with a mask of size 3×3. The response of the filter at point (x, y) is $R = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \ldots + w(0,0)f(x,y) + \ldots + w(1,0)f(x+1,y+1) + w(1,1)f(x+1,y+1)$. Illustration is taken from R. C. Gonzales et al., *Digital Image Processing*, Pretince-Hall.

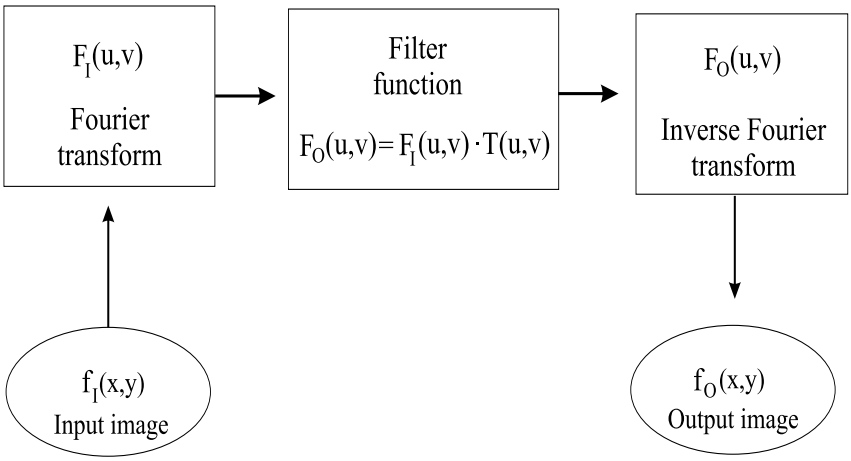

Figure 27: The scheme of frequency domain filtering.

filter function. The highpass filters have exactly opposite characteristics in compare with lowpass ones. Low frequencies correspond to the slowly varying components of an image (like smooth areas), while high frequencies are responsible for components with abrupt gray level changes (like edges or noise). Therefore, an effect of lowpass filtering is image smoothing, while highpass filtering has sharpening effect. It is obvious that the filter choice is hardly problem dependent.

Correspondence between spacial and frequency domain filtering is given by the convolution theorem. This theorem is a straightforwardly generalization of the convolution theorem for functions of one variable and reader can find it within "Signal Filtering" exercise.

Experiments

Problem

"Filtrate the synthetic image with Gaussian noise from exercise 1. Make filtration in both, spacial and frequency domains. Use the Gaussian probability density function as a PSF and filter function. Compar the Fourier spectrums of input and output (filtered) images."

Matlab code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IMAGE FILTERING
% IN SPACE AND FREQUENCY DOMAINS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create the original image
```

```
f=zeros(30,30); f(5:24,13:17)=1; subplot(1,2,1);
imshow(f); % visualize the original image
title('Original image');
% add Gaussian white noise to the image f
noise_variance=0.05;
f_noise=imnoise(f,'gaussian',0,noise_variance); subplot(1,2,2);
imshow(f_noise); title('Input (noisy) image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FREQUENCY DOMAIN FILTERING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure; subplot(2,2,1);
% calculte the Fourirer transform (FT) of the input image
SpectImg=fft2(f_noise); SpectImg=fftshift(SpectImg);
subplot(2,2,1);
% visualize the Fourirer spectrum of the input image
imshow(log(1+abs(SpectImg)),[]); title('Fourier spectrum of the
input image');
% create image filter
ncamp_f=14; % filter size
filter_variance=3;
filter=fspecial('gaussian',ncamp_f,filter_variance);
% padd filter with zeros
filter_pad=padarray(filter,[(size(SpectImg,1)-ncamp_f)/2
(size(SpectImg,2)-ncamp_f)/2]);
subplot(2,2,2);
mesh(filter_pad); % visualize the filter surface
title('Filter function');
% calculate the Fourier transform of the output image (FTO)
SpectOut=SpectImg.*filter_pad; subplot(2,2,3);
imshow(log(1+abs(SpectOut)),[]); % visualize the FTO
title('Fourier spectrum of the output image');
SpectOut=ifftshift(SpectOut); % re-shifting of the FTO
% calculate the output image as inverse Fourier transform
f_O=ifft2(SpectOut);
subplot(2,2,4);
imshow(real(f_O),[]); % visualize of the output image
title('Output image');
%%%%%%%%%%%%%%%%%%%%%%%%
% SPACE DOMAIN FILTERING
%%%%%%%%%%%%%%%%%%%%%%%%
figure;
ncamp_psf=5; % PSF size
psf_variance=3; psf=fspecial('gaussian',ncamp_psf,psf_variance);
subplot(2,2,1);
% visualize the Fourirer spectrum of the input image
```

```
imshow(log(1+abs(SpectImg)),[]); title('Fourier spectrum of the
input image');
subplot(2,2,2);
mesh(psf); % visualize the point spread function (PSF)
title('Point spread function');
f_O=conv2(f_noise,psf); % calculate the imput image
SpectOut=fft2(f_O); SpectOut=ifftshift(SpectOut); subplot(2,2,3);
imshow(log(1+abs(SpectOut)),[]); % visualize the Fourier spectrum
                                 % of the output image
title('Fourier spectrum of the output image'); subplot(2,2,4);
imshow(f_O,[]); title('Output image');
% END
```

Results

Within these experiments we attempt to demonstrate the image filtering as a noise reduction tool. Figure 28. shows the original image and its noisy version, which is obtained by adding Gaussian white noise with variance of 0.05. This noisy image is used as an input image in our experiments.

Space domain filtering of the input image we present in Figure 29. The used PSF or convolution mask is defined by Gaussian pdf with zero mean and variance 3 and the sampling size is 5×5. Hence the mask gives a highest importance to the pixel in the center. The output image is a smoothed and less noisy version of the input image. We note that the size of the output image is bit larger than the input image's size. This is the consequence of the calculation by convolution operator. Namely, if the input image has size of $M \times N$ and the size of PSF is $a \times b$ then the filtered image has a size $[M+a-1, N+b-1]$. Therefore, the output (filtered) image has an added layer to border, which we can simply cut. It is interesting to compare the Fourier spectrums of the input and output images. The output image has less high frequency components, while the low frequency components are enhanced.

Frequency domain filtering of the input image is illustrated in Figure 30. The using filter function is defined by Gaussian pdf. Considering a plot we can conclude that it is a typical lowpass filter function: attenuate high frequencies (it is zero far away from the center), while leave low ones unchanged (its highest values are located around the center). The Fourier spectrum of the output image is in the accordance with this property. The output image is less noisy and smoother than the input image and similar with the output image obtained by space domain filtering.

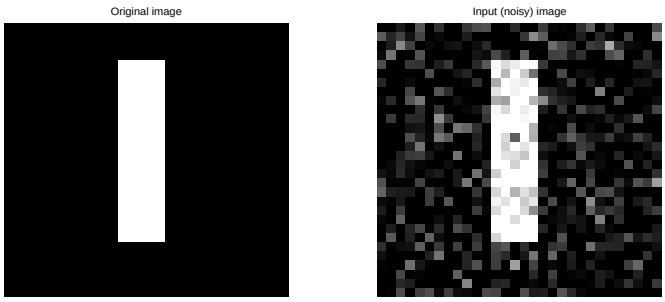

Figure 28: Original and the input image with added Gaussian white noise.

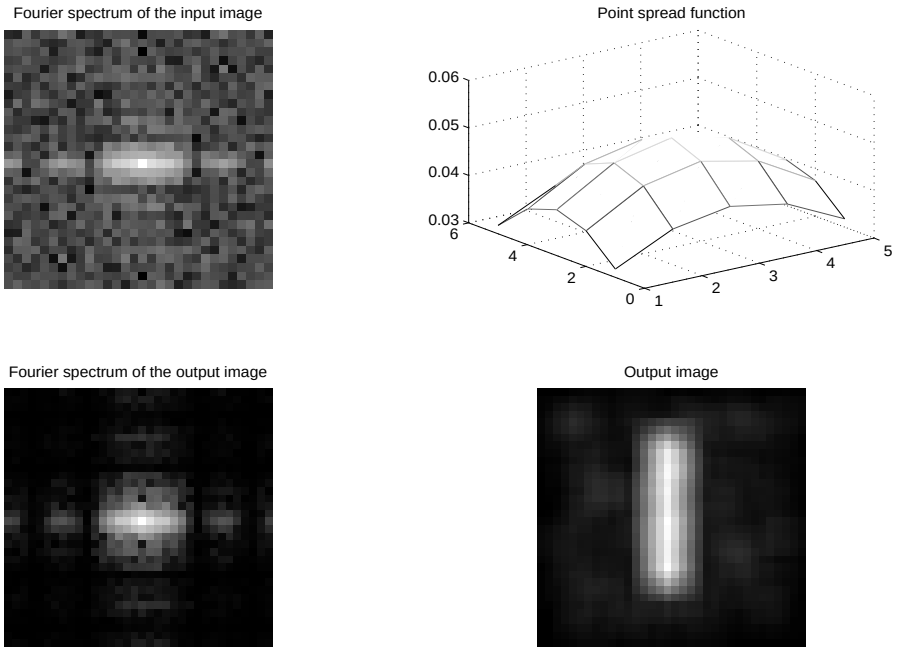

Figure 29: Filtering of the input image in space domain.

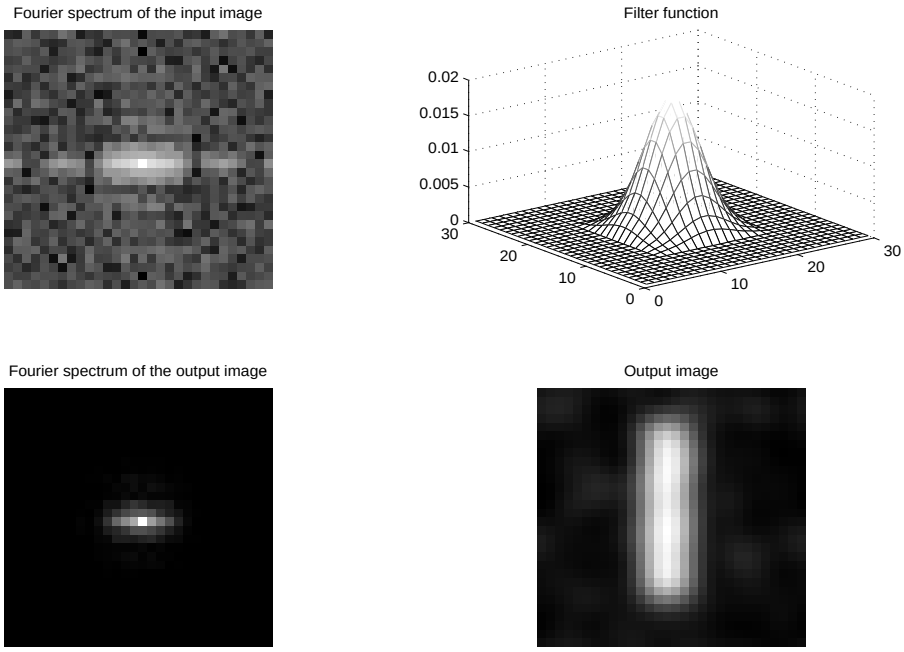

Figure 30: Filtering of the input image in frequency domain.

PART III. Exercise 3. Tibor Lukić, University of Novi Sad.

EDGE DETECTION

Introduction

Edge is an image feature which plays an important role in many applications of image processing, especially in image segmentation (subdividing an image into its constituent regions). We can define it as a set of connected pixels at which the image intensity (gray-level) changes sharply. Often, edges lie on the boundary between two image regions, but in general it could not be always true. The boundary is a "global" concept always related to a corresponding image region, while edge is a "local" concept based on a measure of image intensity changes at a point. Figure 31 shows an ideal and ramp digital edges and they gray-level profiles.

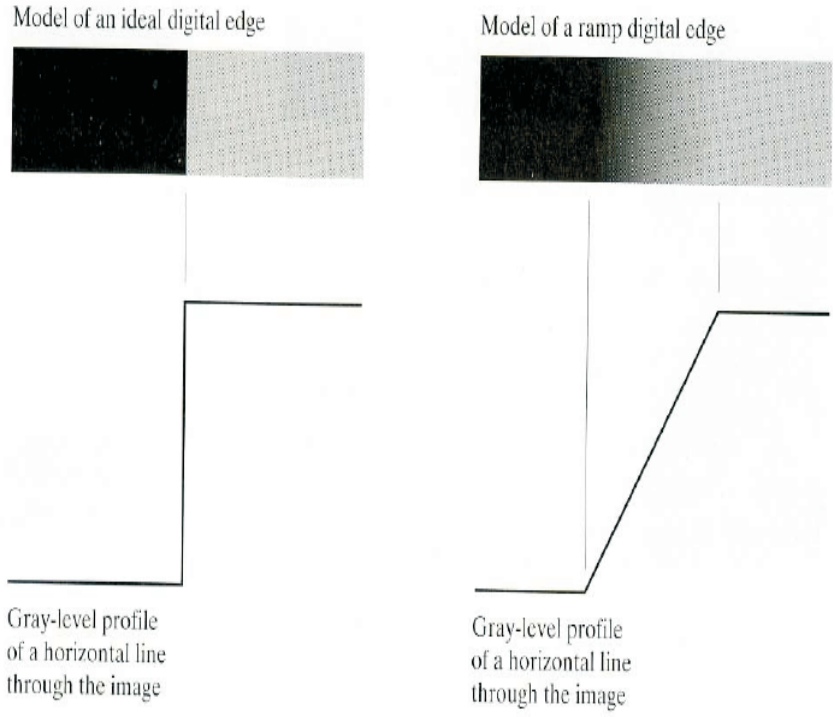

Figure 31: Ideal and ramp digital edges and their gray-level profiles. Illustration taken from R. C. Gonzalez et al., *Digital Image Processing*, Pretince-Hall.

A well know operator used in image processing to measure intensity changes at a point is the *gradient* operator. It comes from the mathematical analysis. The gradient of a differentiable function $f(x, y)$ at a point (x, y) is defined as the vector

$$\text{grad} f = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right]^T .$$

The magnitude of this vector,

$$|\text{grad}f| = \sqrt{\frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2} \tag{9}$$

is a measure of the function's change at a point. If f is a digital image function then the magnitude of the gradient, $|\text{grad}f|$ (commonly called *image gradient* or just gradient) is a measure of the image intensity changes at the considered point. The image gradient is zero in homogeneous and small in smooth image areas and its highest values achieves at locations with rapid changes in intensity, like edges. Therefore, the image gradient is a useful tool in edge detection and the gradient image (image with image function $|\text{grad}f|$) emphasizes rapid changes, that is edges, of the considered image. However, image function is not differentiable, hence the gradient cannot be calculated analytically and some approximation is need. There are various approximations of image gradient. We consider two well known such approximations called as *Prewitt* and *Sobel* operators.

Let consider a 3×3 image area as it presented in Figure 32. The Prewitt operators approximates the partial derivatives at point f_5 by the following way

$$G_x = (f_7 + f_8 + f_9) - (f_1 + f_2 + f_3)$$

and

$$G_y = (f_3 + f_6 + f_9) - (f_1 + f_4 + f_7),$$

where G_x and G_y approximates the derivatives in x and y directions, respectively. This expressions can be implemented in space domain filtering using a corresponding convolution mask presented in Figure 32 (bottom).

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

Prewitt masks

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1
G_x			G_y		

Sobel masks

-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1
G_x			G_y		

Figure 32: **Top:** A 3×3 image area, the f's are intensity values. **Bottom:** Prewitt and Sobel masks used to compute the gradient at point f_5.

The Sobel operators are given by

$$G_x = (f_7 + 2f_8 + f_9) - (f_1 + 2f_2 + f_3)$$

and

$$G_y = (f_3 + 2f_6 + f_9) - (f_1 + 2f_4 + f_7).$$

The corresponding convolution masks are shown in Figure 32 (bottom). The Sobel operators give more importance to the center point (weight value of 2) which is used to achieve some smoothing effect.

As we can see in equation (9) the computation of the gradient requires that the components G_x and G_y have to be combined. The well known approach in the image gradient approximation is given by

$$|\text{grad} f| \approx \sqrt{G_x^2 + G_y^2}.$$

Experiments

Problem

"It is given a synthetic image with Gaussian noise. Determine the edges by space domain filtering using Sobel and Prewitt gradient masks. Visualize the image gradient and the directional gradients too."

Matlab code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EDGE DETECTION
% BY SPACE DOMAIN FILTERING
% USING PREWITT AND SOBEL GRADIENT MASKS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create the original image
f=zeros(30,30); f(5:24,13:17)=1;
subplot(1,2,1);
imshow(f,[]); % visualize the original image
title('Original image');
% add Gaussian white noise to the image f
noise_variance=0.05;
f_noise=imnoise(f,'gaussian',0,noise_variance);
subplot(1,2,2);
imshow(f_noise,[]); title('Input (noisy) image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOBEL GRADIENT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
subplot(1,3,1);
% create Sobel image mask
h_sobel= fspecial('sobel');
% space domain filtering with h_sobel mask
```

```
f_filtered_x=conv2(h_sobel,f_noise); imshow(abs(f_filtered_x),[]);
title('Sobel |Gx| image');
subplot(1,3,2);
% space domain filtering with h_sobel' mask
f_filtered_y=conv2(f_noise, h_sobel'); % conv2(f_noise, h')
imshow(abs(f_filtered_y),[]); title('Sobel |Gy| image');
% Sobel image gradient
f_filtered=sqrt(f_filtered_x.^2+f_filtered_y.^2);
subplot(1,3,3);
imshow(f_filtered,[]); title('Sobel gradient image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREWITT GRADIENT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;
subplot(1,3,1);
% create Prewitt image mask
h_prewit= fspecial('prewit');
f_filtered_x=conv2(h_prewit,f_noise);
imshow(abs(f_filtered_x),[]); title('Prewitt |Gx| image');
subplot(1,3,2);
f_filtered_y=conv2(f_noise, h_prewit');
imshow(abs(f_filtered_y),[]); title('Prewitt |Gy| image');
% Prewitt image gradient
f_filtered=sqrt(f_filtered_x.^2+f_filtered_y.^2); subplot(1,3,3);
imshow(f_filtered,[]); title('Prewitt gradient image');
% END
```

Results

Within this exercise we determine edges of the given input image by calculating Sobel and Prewitt image gradients. The input image is obtained by adding Gaussian white noise with variance 0.05 to the original image, both images are presented in Figure 33. Sobel and Prewitt gradients are implemented by using the convolution operator of the input image and the corresponding Sobel and Prewitt masks (see Figure 32). Figures 34 and 35 shows the gradient images of Sobel and Prewitt operators and also the gradient component images in x and y directions ($|G_x|$, $|G_y|$). Both gradient images emphasize the edges of the rectangular on the input image. Sobel gradient image is bit smoother than Prewitt one. This is a consequence of the fact that Sobel operator gives more importance to the center point in the mask (weight 2, see Figure 32).

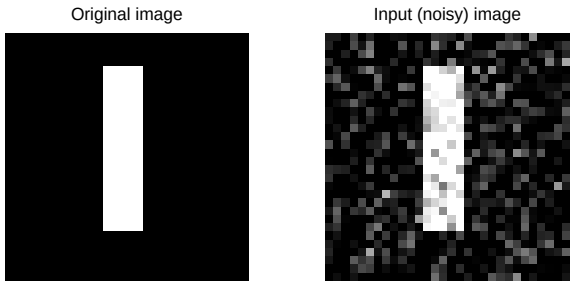

Figure 33: Original and the noisy input image obtained by adding a Gaussian white noise.

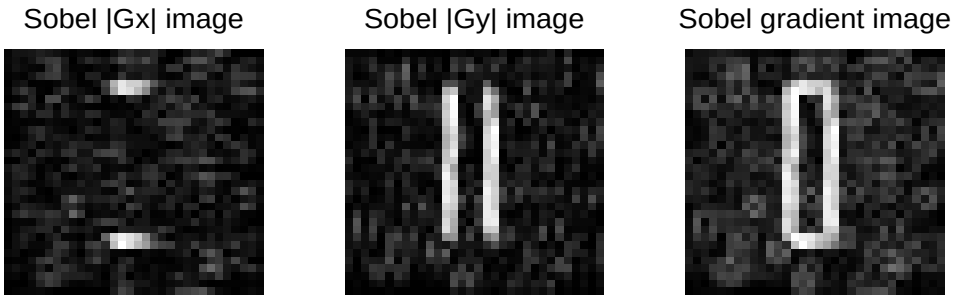

Figure 34: Edge detection by Sobel gradient.

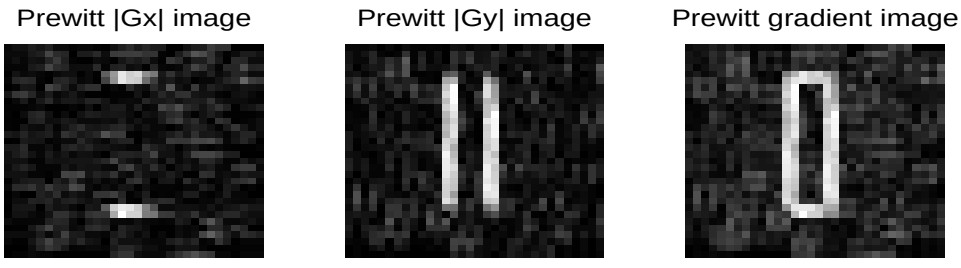

Figure 35: Edge detection by Prewitt gradient.

References

[1] Oppenheim A. V. and Shafer R. W., *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[2] Bracewell R., *The Fourier Transform and Its Applications*, McGraw Hill, 2000.

[3] Gonzalez R. C. and Richard E. W., *Digital Image Processing*, Pretince-Hall, 2006.