EXERCISE 1 – INTRODUCTION TO MATLAB

Objective: Getting to know basic MATLAB functions related to discrete-time signals, ways to display a discrete-time signal as well as how MATLAB functions are written.

Example 1

a) Create the signal $s(n) = \cos(n\pi/10) + \sin(n\pi/5)$ for $0 \leq n \leq 39$. What is the duration of this signal?

b) Create the signal $a^n \cdot (u(n+5) - u(n-35))$. Assume that $a = 1.2$, $a = 0.5$, $a = -1.2$ and $a = -0.5$. Compare the obtained signals.

c) Create the signal $p(n)$ with the following 4 non-zero samples: $p(0) = 1$, $p(1) = 2.5$, $p(2) = 4$, $p(3) = -1$. Then create the signal $q(n) = p(n-2)$ and the signal $r(n) = 0.5q(n) + p(n)$. Display all three signals on the same diagram.

Example 2

a) Create the complex sinusoid $f(n) = e^{j\alpha n}(u(n) - u(n-40))$ for $\alpha = \pi/8$. Then display the real and the imaginary part of $f(n)$, as well as its magnitude and phase.

b) Create the signal $g(n) = f^*(n)$ (asterisk denotes complex conjugate), and then create the sum $s(n) = f(n) + g(n)$ and the difference $d(n) = f(n) - g(n)$. Display the real and the imaginary part of the obtained signals. What can be observed?

Example 3

Write the MATLAB function which will implement the operation of complex signal conjugation by using elementary arithmetic operations. Write the function in the MATLAB editor.

SOLUTIONS:

Example 1

a)

Signal generation

A one-dimensional signal in MATLAB is represented as a vector (ordered array of numbers), and thus the meaning of time, i.e. which value corresponds to which time instant, is relative and has to be defined by the user. The user should first generate the vector indicating the time instants at which particular samples will be positioned, and then the vector containing actual values of the signal. In this example the moments at which the signal is positioned range from 0 to 39, thus the duration of the signal is ___ . Mathematically speaking, this does not mean that the signal is undefined out of this range (the signal, as a mapping from **Z** to **R** has to be defined for *each* integer value of *n*). This only means that we usually assume that the signal is equal to zero out of this range, and for that reason we use such a vector representation only for finite duration signals. A notable exception to this is when we represent periodic signals by representing only one of its periods as a vector. It will be seen later that some other classes of infinite duration signals can also be represented in MATLAB, although not as arrays of samples in time domain.

```
>> n=0:39;
>> s=cos(pi*n/10) + sin(pi*n/5);
```

Graphical display of a signal

The function **stem** is used to display a vector in a way common for discrete-time signals. As its first parameter this function takes the vector of time instants where the signal is positioned (which is thus a vector of consecutive integers), and as its second parameter, it takes the vector of signal values. The dimensions of these two vectors must match. Functions **title** and **xlabel** are used to define the names for the diagram (quantity on the *y*-axis) and the *x*-axis, respectively.

```
>> stem(n,s);
>> title('Sum of two sinusoidal signals'); xlabel('discrete time n')
```

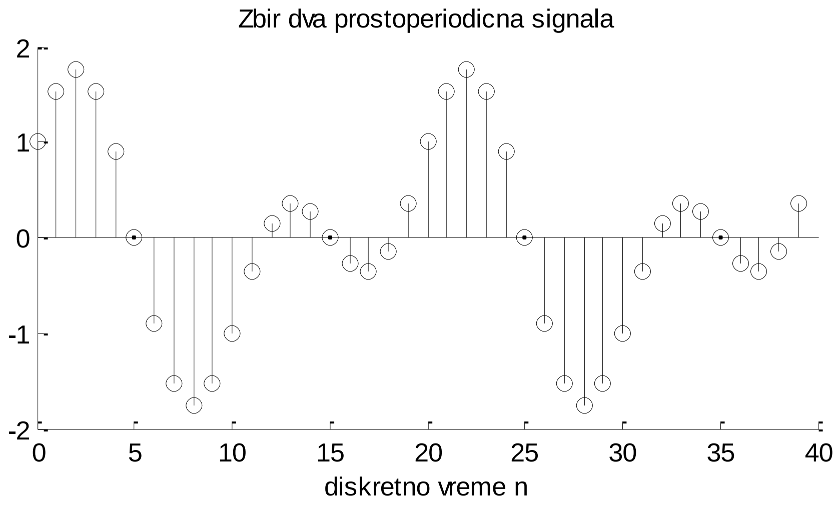


Figure 1.1 A discrete-time signal displayed using the function **stem**[1]

Instead of the vector *n* any other vector of 40 consecutive integers could have been used, which would lend a different interpretation to the signal.

```
>> m = 10:0.25:19.75;
>> stem(m,s);
>> title('The same signal with a different time axis'), xlabel('m')
```

[1] In the figures the names will sometimes be given in Serbian since these exercises are translations from the Serbian original versions. The author kindly apologizes for this inconvenience.

b)

Signal generation

First we need to define the points (time instants) where the non-zero portion of the signal should be positioned. As $u(n)$ denotes the Heaviside signal, $u(n+5)$ denotes the Heaviside signal shifted to the right/left (underline the correct option) by ___ samples, so the time range of interest is:

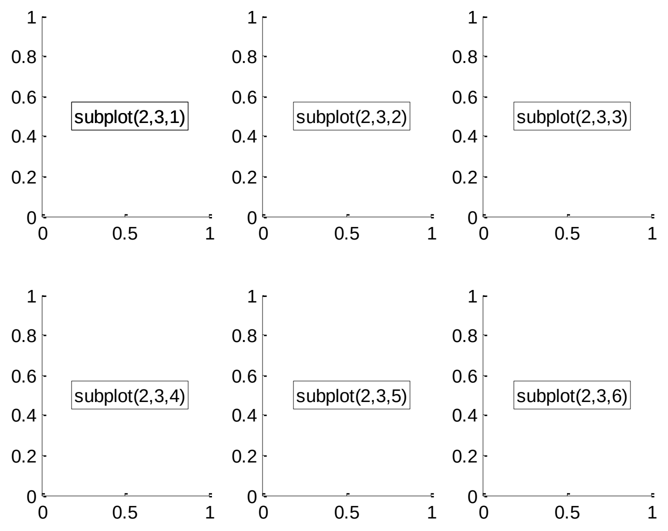
```
>> n=-5:34;
```

After the time range has been defined, what is left is to generate the desired signals. They are assigned to variables `s1`, `s2`, `s3` and `s4`. The use of parentheses `()` is mandatory since raising a number to the power of n has greater priority than multiplication.

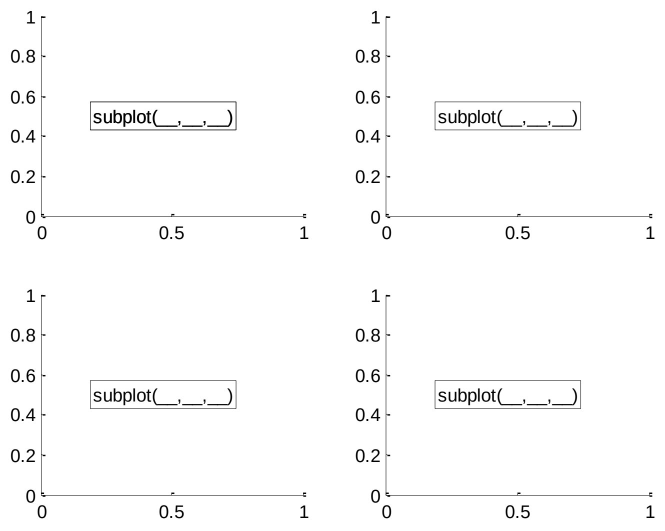
```
>> s1= 1.2.^n; s2= 0.5.^n; s3 = (-1.2).^n; s4 = (-0.5).^n;
```

Simultaneous display of multiple signals

The function **subplot** is used for simultaneous display of multiple diagrams within the same figure (the same window). The standard form of this function is `subplot(m,n,p)`, where `m` is the number of diagrams on the vertical axis (number of rows), `n` is the number of diagrams on the horizontal axis (number of columns), and `p` indicates the diagram to be displayed (see Fig. 1.2.a). Add the missing values in Fig. 1.2.b.



a) Each diagram shows the way to reference it using the function **subplot**.



b) Add the missing values of parameters of the **subplot** function needed to reference each diagram

Figure 1.2 Illustration of the use of the function **subplot**

Organize diagrams in two rows and two columns. Display the signal `s1` in the upper left diagram, signal `s2` in the upper right diagram, signal `s3` in the lower left diagram, and signal `s4` in the lower right diagram. The axes of each diagram should be scaled so as to indicate the exact range of actual signal values (function **axis**, option **tight**).

```
>> subplot(2,2,1), stem(n,s1), title('s1, a=1.2'), axis tight
>> subplot(2,2,2), stem(n,s2)
>> title('s2, a=0.5')
>> axis tight
>> subplot(2,2,4), stem(n,s4), title('s4, a=-0.5'), axis tight
>> subplot(2,2,3), stem(n,s3), title('s3, a=-1.2'), axis tight
```

In the previous set of command lines, the lines 2, 3 and 4 indicate that it is not necessary to state all the parameters of a particular diagram in a single command line, as was the case in lines 1, 2 and 6. The line 5 references the upper/lower left/right diagram (underline the correct option) and the line 6 references the upper/lower left/right diagram (underline the correct option), to illustrate the fact that diagrams do not have to be referenced in a particular order.

Figure 1.3 shows four exponential signals for different values of the base. It can be seen (as expected) that:

- if the base is greater than 1, the signal increases/decreases (underline the correct option) with *n*;
- if the base is a positive real number smaller than 1, the signal increases/decreases (underline the correct option) with *n*;
- if the base is a negative real number greater than −1, the signal alternates between positive and negative values, and its absolute value increases/decreases (underline the correct option) with *n*;
- if the base is smaller than −1, the signal alternates between positive and negative values, and its absolute value increases/decreases (underline the correct option) with *n*.

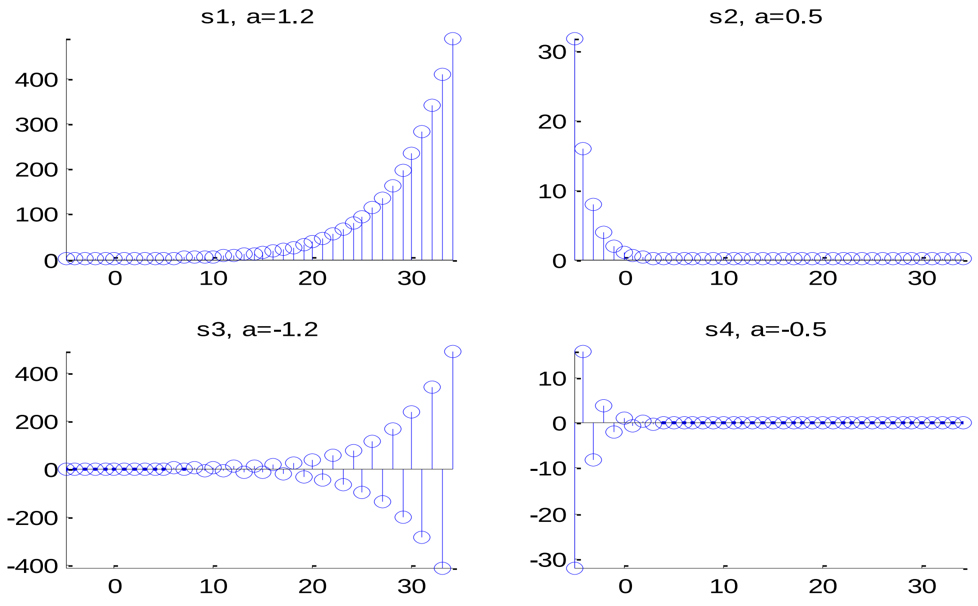


Figure 1.3 Segments of the signal a^n for $a = 1.2$, $a = 0.5$, $a = -0.5$ and $a = -1.2$ (40 samples)

c) As has already been said, the notion of time in MATLAB is relative. MATLAB treats each object as a vector and its interpretation is left to the user. This somewhat complicates the time-shifting operation.

Signal generation

Let us create the signal $p(n)$ as a row vector, by explicitly defining it as follows:

```
>> p=[1 2.5 4 -1];
```

Simultaneous display of multiple signals

The signal $q(n)$ has the same shape as the signal $p(n)$, but it is shifted to the left/right (underline the correct option) by ___ samples. Time shifting can be easily effected by just using different vectors to denote the position of the signal in time (vectors `n1` i `n2` in this example). The function **stem**, on its own, does not offer the possibility of displaying multiple vectors in one figure and thus we need to use the function **hold** with the parameter **on**. In this way the contents of the diagram will be added to instead of simply replaced. The third parameter used by the function **stem** defines the colour of the line that will be used to indicate the data (`r` for red, `g` for green and `b` for blue).

```
>> n1 = [0:3]; n2 = n1+2;
>> hold on; stem(n1,p,'r');stem(n2,p,'g');
>> xlabel('time axis n');
```

The black and white version of the resulting diagram in Fig. 1.4.

Time shifting

The generation of the signal *r(n)* causes a problem since it requires time-shifted signals to be added. Since MATLAB allows only addition of matrices (vectors) of the same dimensions unless one of them is a _______ , vectors p and q need to be expanded so as to achieve appropriate time shifting, namely, _____ zeros have to be appended to the left of the vector q, as well as at the right of vector p (to avoid dimension mismatch). This, of course, did not affect the mathematical interpretation of these signals.

```
>> q = [0 0 p];
>> p = [p 0 0];
>> r = 0.5*q+p;
>> n = 0:5;
>> hold on; stem(n,p,'r');stem(n,q,'g');stem(n,r,'b'); xlabel('time axis n');
```

The black and white version of the resulting diagram in Fig. 1.5.

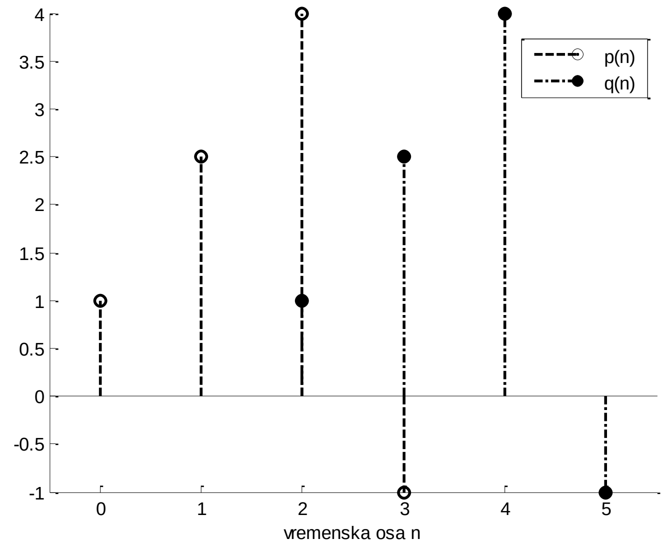


Figure 1.4 Simultaneous display of multiple signals using the function **hold**

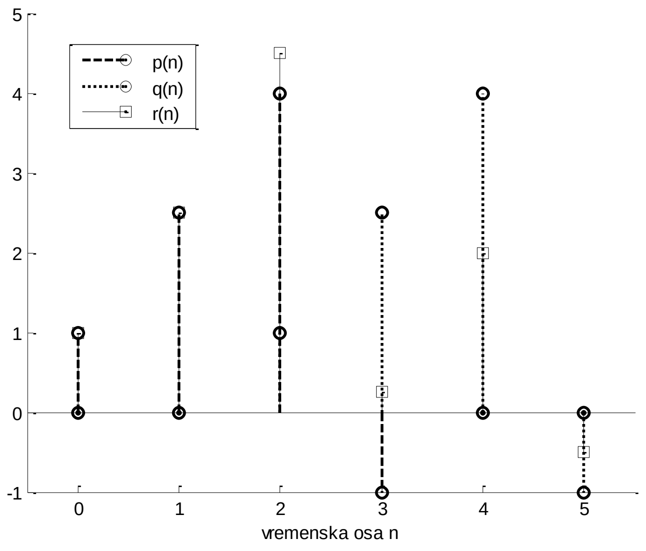


Figure 1.5 Signals *p*(*n*), *q*(*n*) and *r*(*n*)

Example 2

a) Exponential signal is created using the function **exp**. The function **real**, in the general case, returns the matrix composed of real parts of elements of the input matrix. Similarly, the function **imag** returns the imaginary parts, the function **abs** returns the magnitudes of complex values and the function **angle** their phases expressed in radians, ranging from –π to π. Instead of the semicolon operator (;) a simple comma (,) can be used to delimit the functions.

```
>> n=0:39;
>> f=exp(j*pi/8*n);
>> subplot(2,2,1), stem(n,real(f)), title('Real part of a complex sinusoid')
>> subplot(2,2,2), stem(n,imag(f)), title('Imaginary part of a complex sinusoid')
>> subplot(2,2,3), stem(n,abs(f)), title('Magnitude of a complex sinusoid')
>> subplot(2,2,4), stem(n,angle(f)), title('Phase of a complex sinusoid')
```

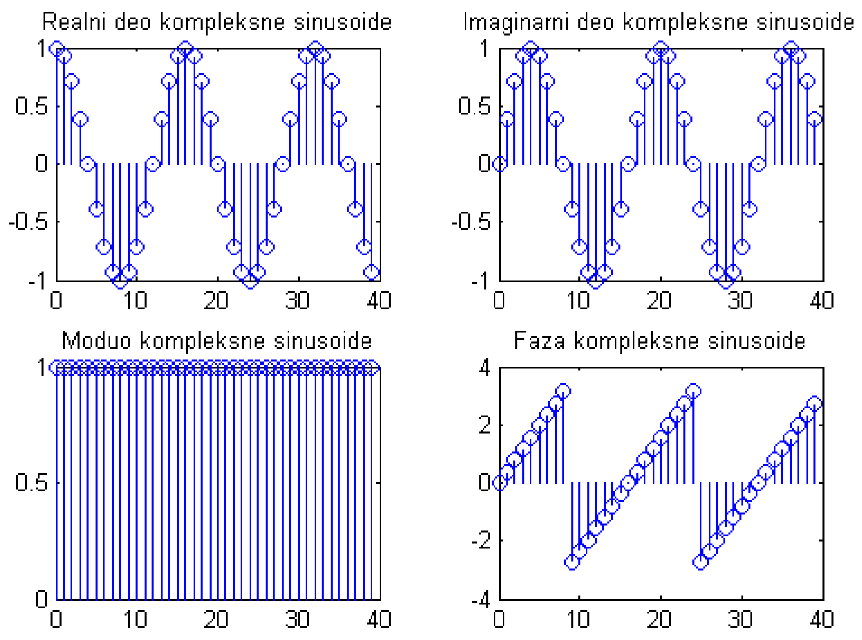


Figure 1.6 The display of a complex sinusoid

The result is shown in Fig. 1.6. The real part of the complex sinusoid $e^{j\frac{\pi}{8}n}$ is a sine/cosine function (underline the correct option) of angular frequency ____, while its imaginary part is a sine/cosine function (underline the correct option) of the same angular frequency. The magnitude is equal to ______ for all *n*, and the phase is linear. However, since the phase is displayed in the range from ____ to ____, there are phase discontinuities which occur whenever the phase reaches its maximum value.

b) The complex conjugate signal can be created using the function **conj**.

```
>> g=conj(f);
>> s=f+g; d=f-g;
>> subplot(2,2,1), stem(n,real(s)), title('Real part of the sum f(n)+conj(f(n))')
>> subplot(2,2,2), stem(n,imag(s)), title('Imaginary part of the sum f(n)+conj(f(n))')
>> subplot(2,2,3), stem(n,real(d)), title('Real part of the difference f(n)-conj(f(n))')
>> subplot(2,2,4), stem(n,imag(d)), title('Imaginary part of the difference f(n)-conj(f(n))')
```

The result is shown in Fig. 1.7. By adding two complex numbers which compose a complex conjugate pair, we obtain their real/imaginary part (underline the correct option) multiplied by ____ . On the other hand, by subtracting them, we obtain their real/imaginary part (underline the correct option) multiplied by ____ . The same holds for signals.

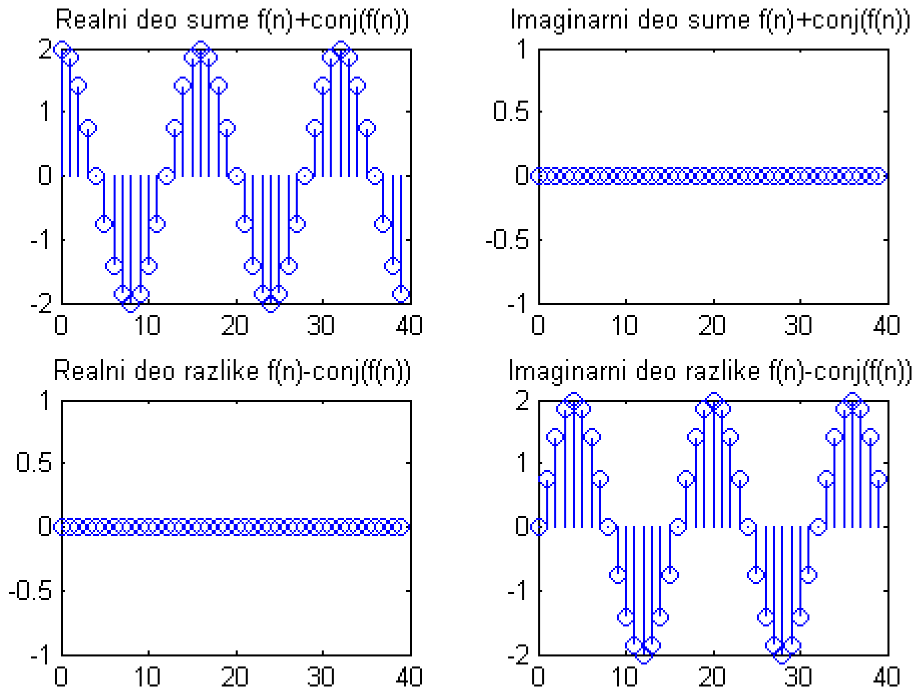


Figure 1.7 The sum and the difference of the signal *f(n)* and its complex conjugate $f^*(n)$

Example 3

MATLAB, as a programming language, supports user defined functions. The code of a function, i.e the array of command lines which implement it, is stored in text format in a file with extension .m, so called m-file. The name of the file should be identical as the name of the function. MATLAB provides its own text editor for writing user functions, but a function can be written in any other ASCII text editor.

The first line of an m-file contains the definition of the function call in the following form:

```
function [output1, output2, outputM] = function_name(param1, param2, paramN)
```

function is a reserved word, `output`***i*** refers to the name of the *i*-th output variable, `param`***i*** refers to the name of *i*-th input variable. The number of input and output variables is not limited and it is left to the user to define it. If there is only one output variable, angular brackets are not mandatory. Unlike some other programming languages MATLAB does not require the type of a variable to be defined, as it treats them all as matrices.

A good programming practice is to explain what the function does in a brief comment which immediately follows the function call definition.

The remainder of the function is its body, composed of command lines which contain MATLAB functions (as ones previously mentioned) or other user defined functions. These command lines have the identical form as command lines stated directly in MATLAB workspace. MATLAB offers many other possibilities as well, but for the purposes of this course they will not be needed.

Writing a function

Start the MATLAB editor by selecting *File/New/M-File* in the workspace menu. Give the name `conjugate` to the function to be written. The function that performs the conjugation of a complex signal has __ input and __ output parameter(s). Thus, the first line of this m-file should have the following form:

```
function y = conjugate(x)
```

After the function call definition it is desirable to explain what the function does.

```
%calculates the complex conjugate of the input parameter x
```

This is followed by the body of the function, which amounts to the extraction of the real and the imaginary part of the input parameter, and multiplication of the imaginary part with −j.

```
y = real(x) - j*imag(x);
```

Save the file under the name ________________ using *File/Save As...* in the editor window, and then test how the function works:

```
>> g1=conjugate(f);
>> g1 == g
```

The equality operator == in the general case compares the corresponding elements of two matrices (elements at corresponding locations) and returns 1 if the values are the same or 0 if they are different.