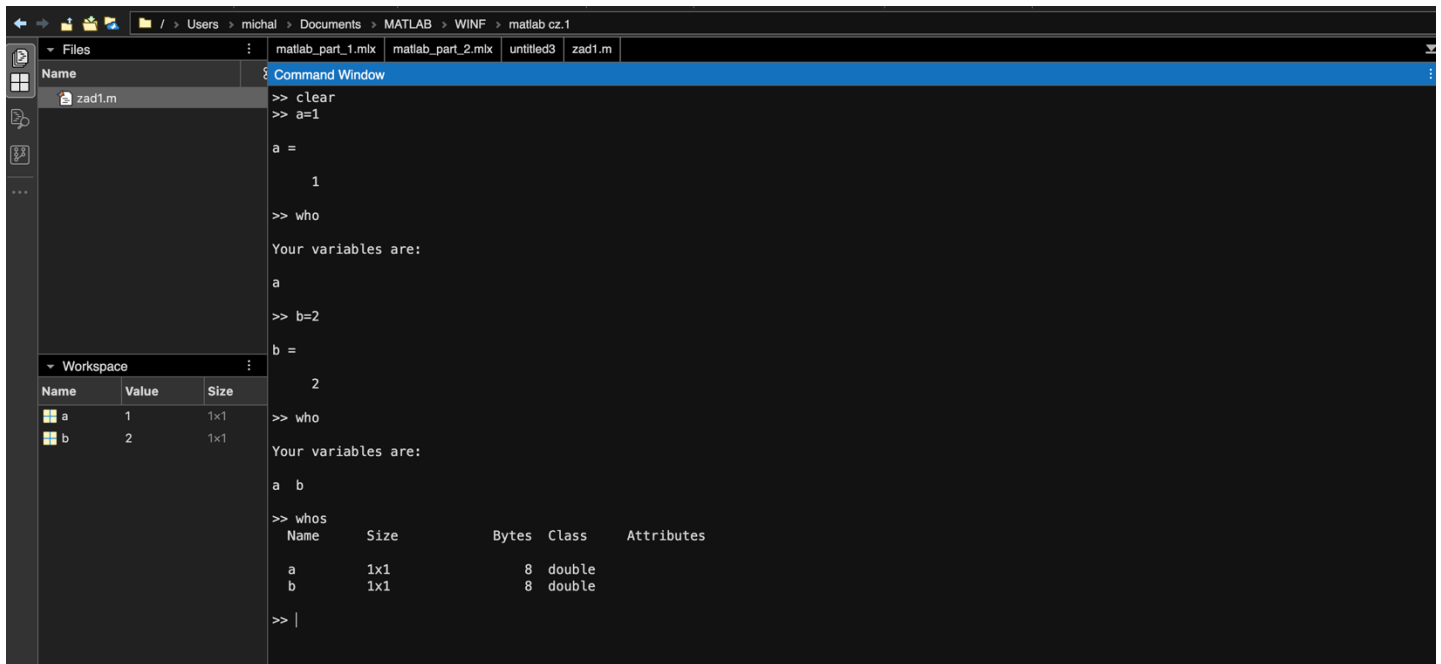


## Zad.1

Na potrzeby laboratorium stworzyłem nowy folder WINF, a w nim folder matlab cz.1. Następnie w oknie poleceń utworzyłem dwie zmienne a i b, przy okazji je wyświetlając, ponieważ po nadaniu im wartości nie dopisałem średnika. Następnie użyłem komend *who* i *whos*. Pierwsza wyświetla nazwy zmiennych obecnych w workspace'ie, a druga wyświetla ich nazwę, rozmiar, liczbę bajtów, typ zmiennych oraz ich atrybuty. O ile będziemy pracować w tym samym workspace oraz nie usuniemy/wyczyścimy zmiennych to informacje powinny być też dostępne w innym oknie.



The screenshot shows the MATLAB interface with the Command Window and Workspace. The Command Window displays the following commands and output:

```
>> clear
>> a=1
a =
    1
>> who
Your variables are:
a
>> b=2
b =
    2
>> who
Your variables are:
a b
>> whos
Name      Size      Bytes  Class  Attributes
a         1x1         8  double
b         1x1         8  double
```

The Workspace window shows the following variables:

Name	Value	Size
a	1	1x1
b	2	1x1

Następnie dodałem kilka innych zmiennych, zapisałem je do pliku ABC używając `save ABC a b c d` co spowodowało zapisanie zmiennych w pliku ABC.mat (co widać w lewym górnym rogu) czyli w skompresowanym formacie matlab'owym. Potem usunąłem zmienne poleceniem `clear` (jak widać po użyciu `who` i `whos` zmienne zostały usunięte) i wczytałem je z powrotem poleceniem `load ABC` (czego wynik również sprawdziłem `who` i `whos`).

The screenshot shows the MATLAB Command Window and Workspace. The Command Window contains the following commands and output:

```
>> a
>> b=2
b =
    2
>> who
Your variables are:
a b
>> whos
Name      Size      Bytes  Class  Attributes
a         1x1         8  double
b         1x1         8  double
>> c=[1;2;3];
>> d=zeros(2,2)
d =
     0     0
     0     0
>> save ABC a b c d
```

The Workspace window shows the following variables:

Name	Value	Size
a	1	1x1
b	2	1x1
c	[1;2;3]	3x1
d	[0,0;0,0]	2x2

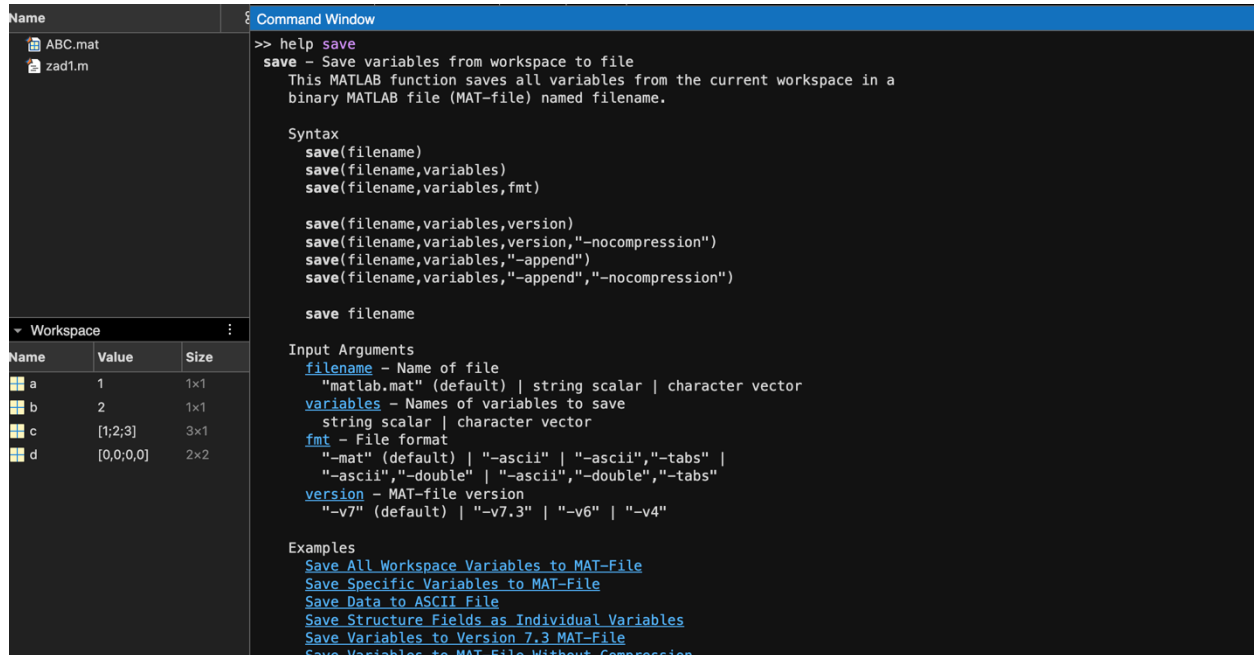
The screenshot shows the MATLAB Command Window and Workspace after clearing and loading variables. The Command Window contains the following commands and output:

```
>> clear variables
>> who
>> whos
>> load ABC.mat
>> who
Your variables are:
a b c d
>> whos
Name      Size      Bytes  Class  Attributes
a         1x1         8  double
b         1x1         8  double
c         3x1        24  double
d         2x2        32  double
>>
```

The Workspace window shows the following variables:

Name	Value	Size
a	1	1x1
b	2	1x1
c	[1;2;3]	3x1
d	[0,0;0,0]	2x2

W kolejnym kroku wyświetliłem pomoc do polecenia `save` (`help save`) oraz zapisałem tylko zmienną `c` do pliku `bb.mat` i `bb.txt` (tu należało użyć `save bb.txt c -ascii`) czego wyniki widać w lewym górnym rogu drugiego zrzutu ekranu.



The screenshot shows the MATLAB Command Window with the command `>> help save` entered. The output displays the `save` function's purpose, syntax, input arguments, and examples. The workspace on the left contains variables `a`, `b`, `c`, and `d`.

```
>> help save
save - Save variables from workspace to file
This MATLAB function saves all variables from the current workspace in a
binary MATLAB file (MAT-file) named filename.

Syntax
save(filename)
save(filename,variables)
save(filename,variables,fmt)

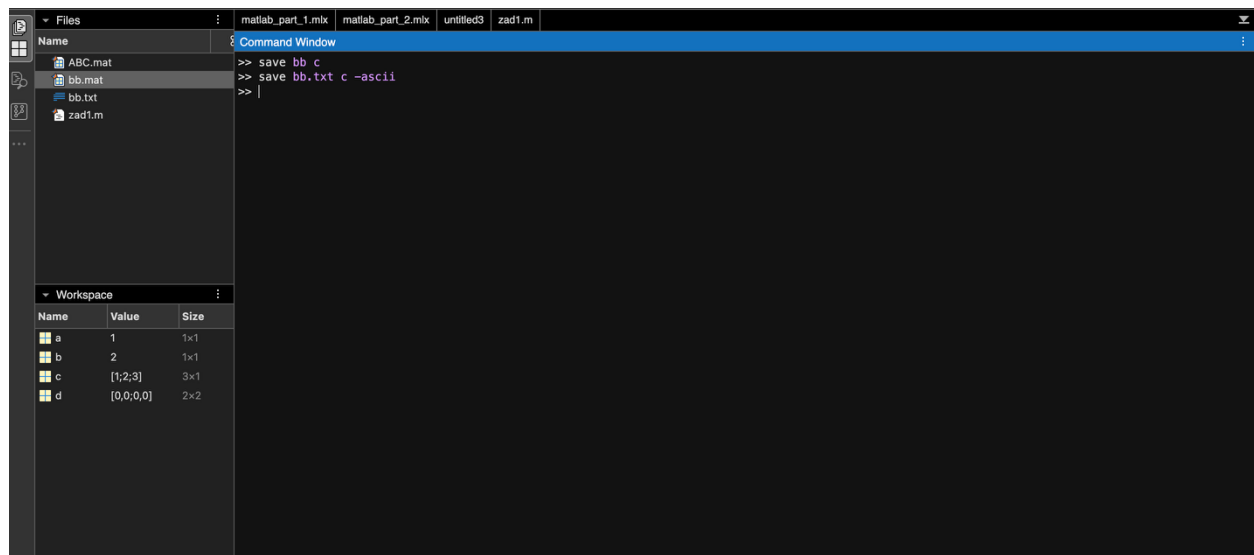
save(filename,variables,version)
save(filename,variables,version,"-nocompression")
save(filename,variables,"-append")
save(filename,variables,"-append","-nocompression")

save filename

Input Arguments
filename - Name of file
    "matlab.mat" (default) | string scalar | character vector
variables - Names of variables to save
    string scalar | character vector
fmt - File format
    "-mat" (default) | "-ascii" | "-ascii","-tabs" |
    "-ascii","-double" | "-ascii","-double","-tabs"
version - MAT-file version
    "-v7" (default) | "-v7.3" | "-v6" | "-v4"

Examples
Save All Workspace Variables to MAT-File
Save Specific Variables to MAT-File
Save Data to ASCII File
Save Structure Fields as Individual Variables
Save Variables to Version 7.3 MAT-File
Save Variables to MAT-File Without Compression
```

Name	Value	Size
a	1	1x1
b	2	1x1
c	[1;2;3]	3x1
d	[0,0;0,0]	2x2



The screenshot shows the MATLAB Command Window with the commands `>> save bb c` and `>> save bb.txt c -ascii` entered. The workspace on the left contains variables `a`, `b`, `c`, and `d`.

```
>> save bb c
>> save bb.txt c -ascii
```

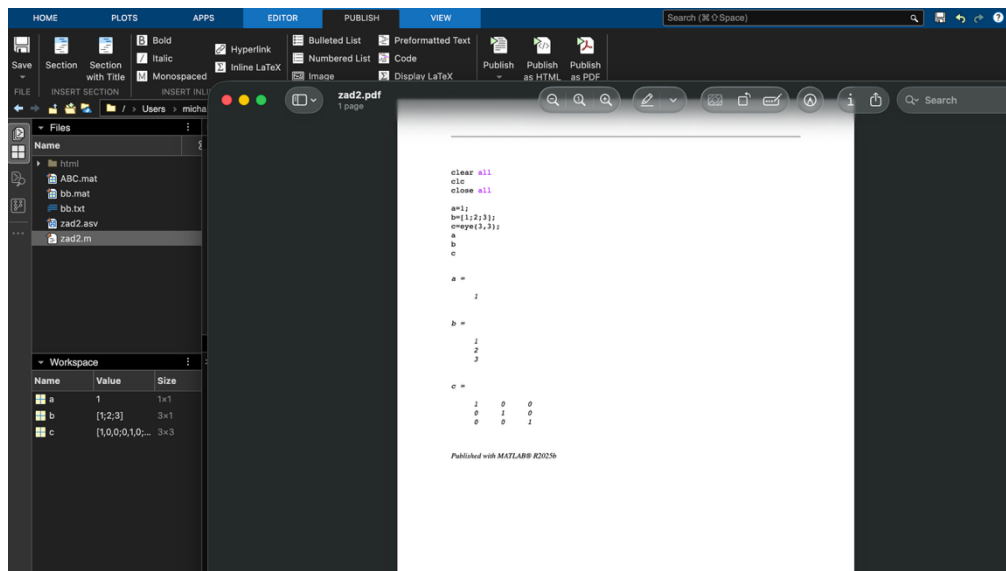
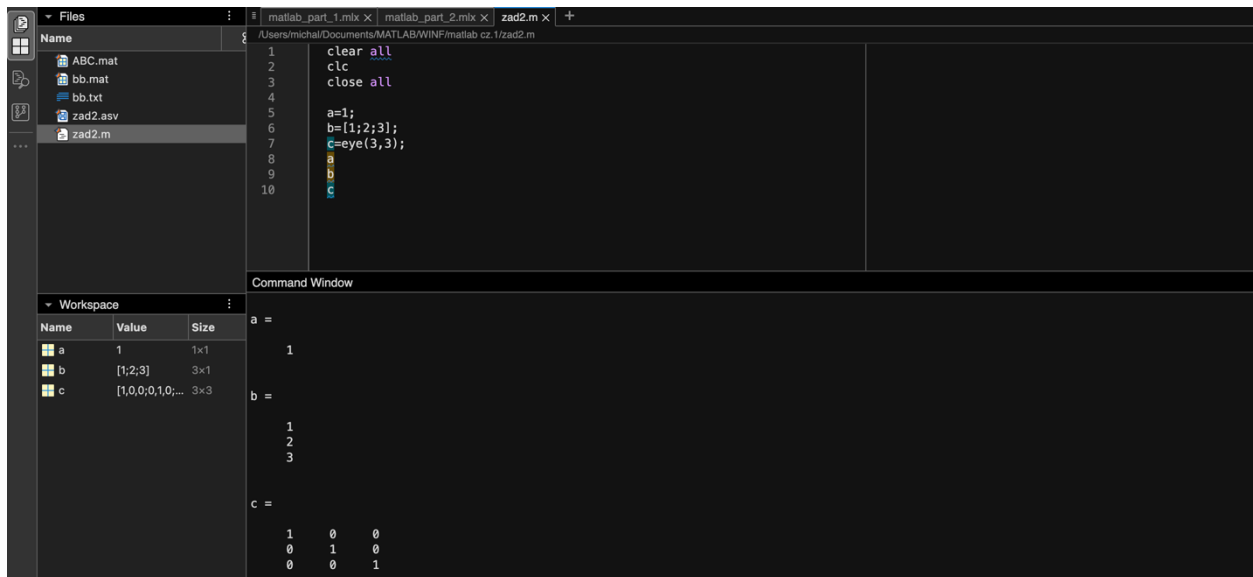
Name	Value	Size
a	1	1x1
b	2	1x1
c	[1;2;3]	3x1
d	[0,0;0,0]	2x2

## Zad.2

W tym zadaniu utworzyłem skrypt zad.2 . Po wyczyszczeniu poprzednich zmiennych, okna poleceń oraz zamknięciu ewentualnych wykresów(*clear all* , *clc* , *close all*) Wypisałem w nim kilka zmiennych różnych typów. Następnie w zakładce Publish przetestowałem funkcję Publish as PDF, która kod oraz wyniki z okna poleceń umieściła w nowym pliku pdf.

```
clear all
clc
close all

a=1;
b=[1;2;3];
c=eye(3,3);
a
b
c
```



## Zad.3

Wykonałem polecenia. Funkcja *dot* zwraca taką samą wartość co obliczenie iloczyn skalarnego pierwszym sposobem. Wartości sinusa dla wektora jest jeszcze wiele (aż do kolumny 1001), ale nie widzę sensu wszystkich ich tutaj umieszczać. Funkcja ma częstotliwość 1Hz co daje okres równy 1 s co widać po wartościach na drugim zrzucie ekranu: dla kolumny 51 wartość jest równa 0 , jest to połowa okresu sinusa czyli koniec jego pierwszej „górkę”. To, że jest to kolumna 51, a nie 50 wynika z tego, że indeksowanie w matlabie zaczyna się od 1 a nie od 0(jak np. w pythonie). Analogicznie pełny okres sinus przejdzie dla kolumny 101 gdzie wartość też jest równa 0 .

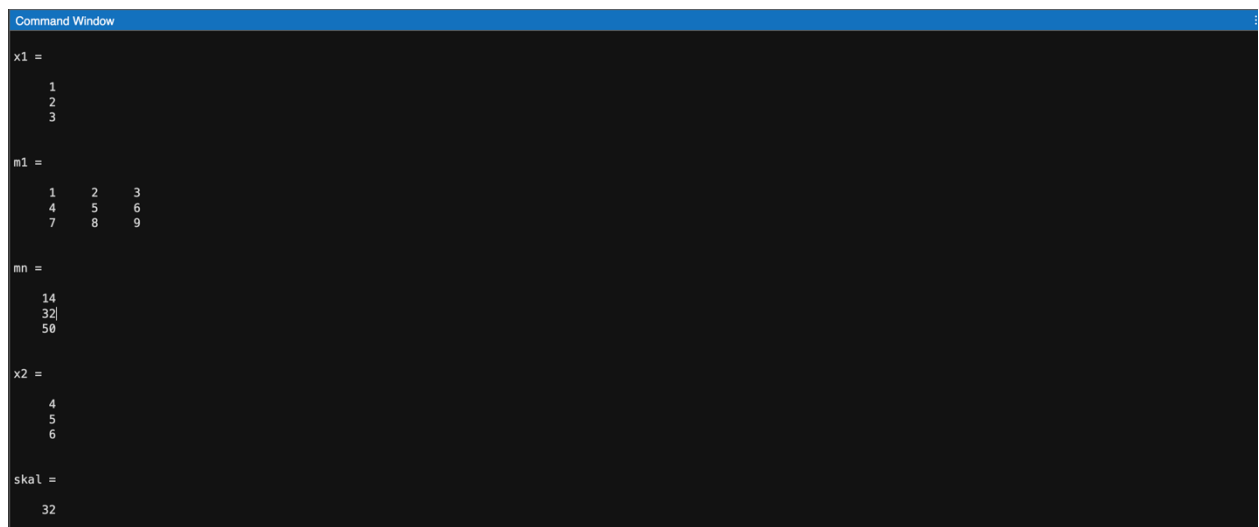
---

```
clear all
clc
close all

a=1;
b=[1;2;3];
c=eye(3,3);
a
b
c
%%
clear all
clc
close all

x1=[1;2;3]
m1=[1,2,3;4,5,6;7,8,9]
mn=m1 * x1
x2=[4;5;6]
skal=x1.'*x2
skal2=dot(x1,x2)
zewn=x1*x2.'
t=[0:0.01:10];
f=1;
s=sin(2*pi*f*t)
```

---



The screenshot shows the MATLAB Command Window with the following output:

```
Command Window

x1 =
     1
     2
     3

m1 =
     1     2     3
     4     5     6
     7     8     9

mn =
    14
    32
    50

x2 =
     4
     5
     6

skal =
    32
```

```

skal2 =
    32

zewn =
     4     5     6
     8    10    12
    12    15    18

s =
Columns 1 through 17
     0     0.0628     0.1253     0.1874     0.2487     0.3090     0.3681     0.4258     0.4818     0.5358     0.5878     0.6374     0.6845     0.7290     0.7705     0.8090     0.8443
Columns 18 through 34
    0.8763    0.9048    0.9298    0.9511    0.9686    0.9823    0.9921    0.9980    1.0000    0.9980    0.9921    0.9823    0.9686    0.9511    0.9298    0.9048    0.8763
Columns 35 through 51
    0.8443    0.8090    0.7705    0.7290    0.6845    0.6374    0.5878    0.5358    0.4818    0.4258    0.3681    0.3090    0.2487    0.1874    0.1253    0.0628    0.0000
Columns 52 through 68
   -0.0628   -0.1253   -0.1874   -0.2487   -0.3090   -0.3681   -0.4258   -0.4818   -0.5358   -0.5878   -0.6374   -0.6845   -0.7290   -0.7705   -0.8090   -0.8443   -0.8763
Columns 69 through 85
   -0.9048   -0.9298   -0.9511   -0.9686   -0.9823   -0.9921   -0.9980   -1.0000   -0.9980   -0.9921   -0.9823   -0.9686   -0.9511   -0.9298   -0.9048   -0.8763   -0.8443

```

## Zad.4

Na początku stworzyłem dwie figury z dwoma różnymi wykresami. Po przypisaniu plot'ów do figur po wpisaniu ponownie *figure(1)* lub *figure(2)* w oknie wyświetlającym wykresy zmieniamy się na konkretną figurę.

```

Command Window
>> x=[1:1:10]

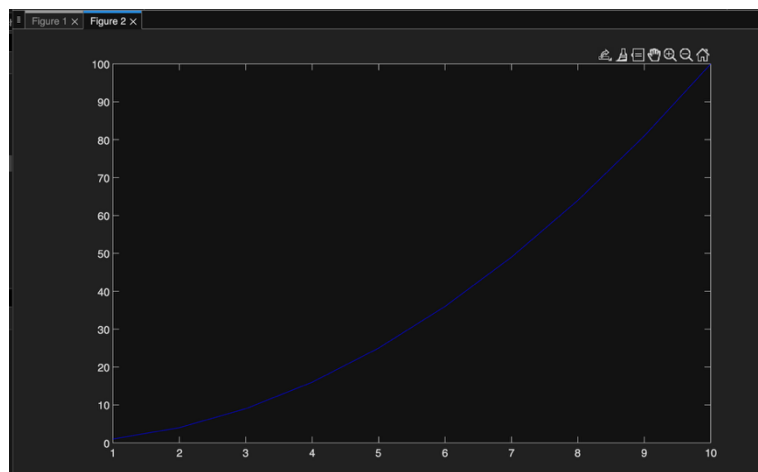
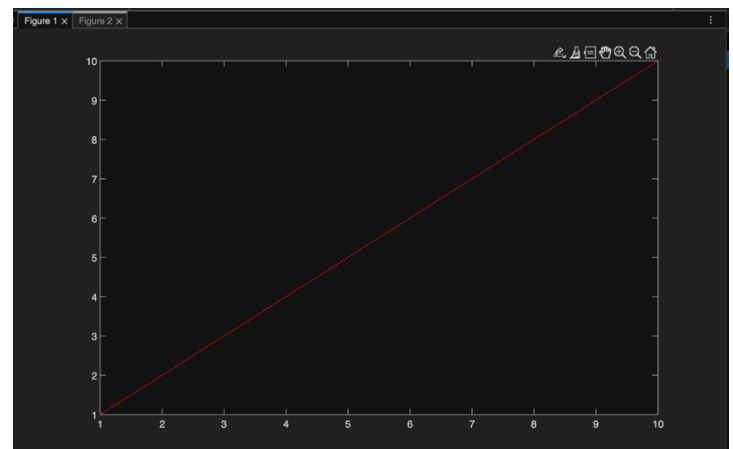
x =
     1     2     3     4     5     6     7     8     9    10

>> y=x;
>> figure(1)
>> plot(x,y,'r')
>> figure(2)
>> z=x.^2

z =
     1     4     9    16    25    36    49    64    81   100

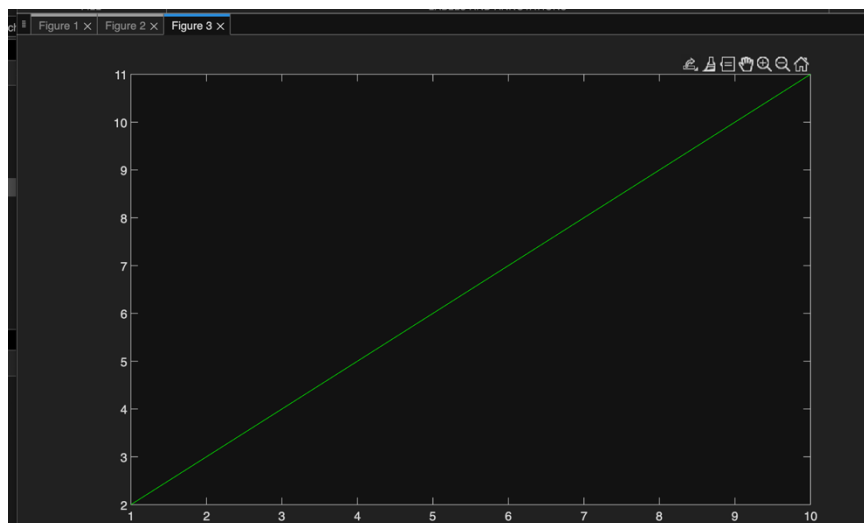
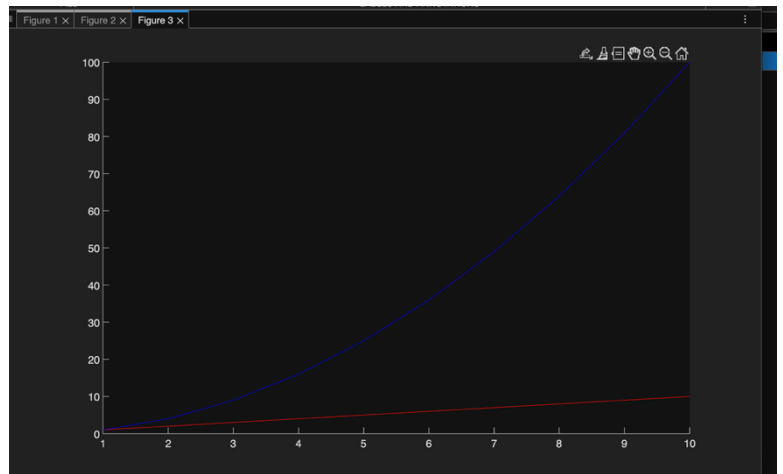
>> plot(x,z,'b')
>> figure(1)
>> |

```



Domyślnie gdy w figurze, w której znajduje się już jakiś wykres i narysujemy kolejny to pierwszy wykres zostanie zastąpiony. Aby w jednej figurze wyświetlić dwa wykresy możemy użyć komendy *hold on*. Aby wyłączyć tę funkcjonalność możemy wpisać *hold off*.

```
Command Window
>> figure(3)
hold on
plot(x,y,'r')
plot(x,z,'b')
hold off
k=x+1;
>> plot(x,k,'g')
>> |
```



---

```
clear all
clc
close all
t = 0:0.01:10;
figure(1);
s=5*sin(2*pi*t);
plot(t,s);
grid()
xlabel('Czas')
ylabel('wartosc')
title('sinus')
ylim([-6 6])

figure(2);
plot(t, t, 'b');    % f(x) = x
hold on
plot(t, t.^2, 'r'); % f(x) = x^2
plot(t, t.^3, 'g'); % f(x) = x^3
hold off

figure(3);
loglog(t,t.^2+1); % f(x) = x^2+1 obie osie w skali logarytmicznej

figure(4);
plot(t,t.^2+1); % f(x) = x^2+1 tylko oś y w skali logarytmicznej
yscale("log")

figure(5);
subplot(3,1,1);
plot(t, t, 'b');    % f(x) = x
xlabel('Czas')
ylabel('wartosc')
title('x')
subplot(3,1,2)
plot(t, t.^2, 'r'); % f(x) = x^2
xlabel('Czas')
ylabel('wartosc')
title('x^2')
subplot(3,1,3);
plot(t, t.^3, 'g'); % f(x) = x^3
xlabel('Czas')
ylabel('wartosc')
title('x^3')
figura_5=figure(5);
savefig(figura_5,"Figura_nr_5");
```

---

Skrypt tworzy kilka figur i rysuje funkcje zadane w poleceniu do zadania, podpisując osie i zapisując figure(5) do pliku Figura\_nr\_5.fig. W figurze 1 za pomocą `ylim(-6 6)` dodatkowo określiłem na jakim zakresie osi y ma być wyświetlony wykres (wcześniej ekstrema były słabo widoczne). Następnie w oknie poleceń za pomocą komendy `openfig(Figura_nr_5)` możemy wczytać figurę z pliku.



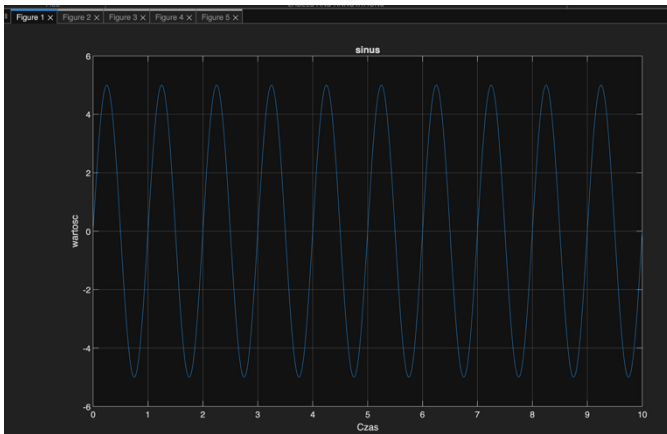


Figura nr 1

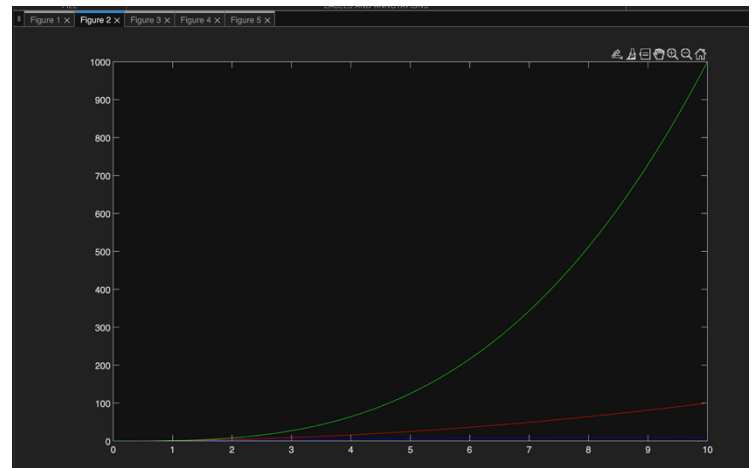


Figura nr 2

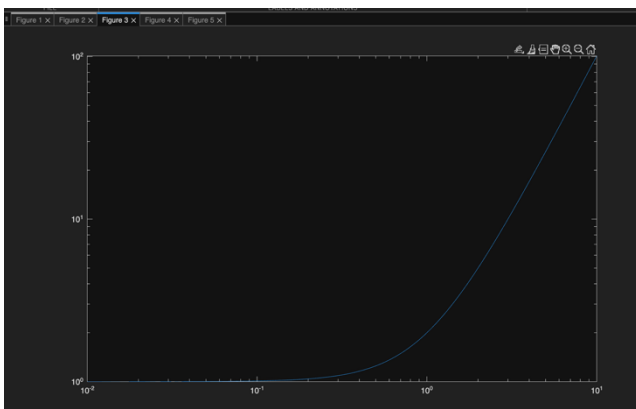


Figura nr 3

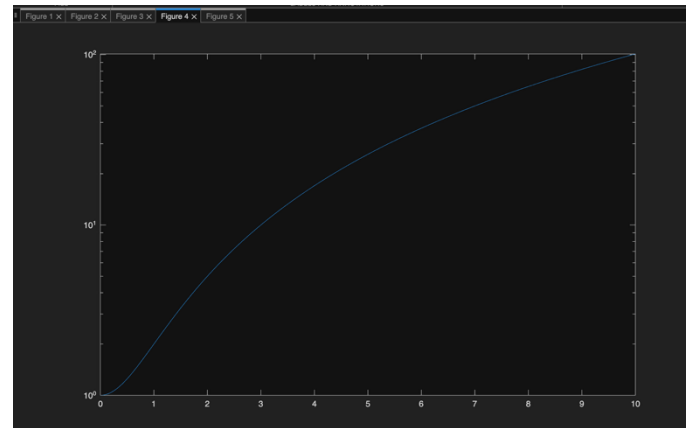


Figura nr 4

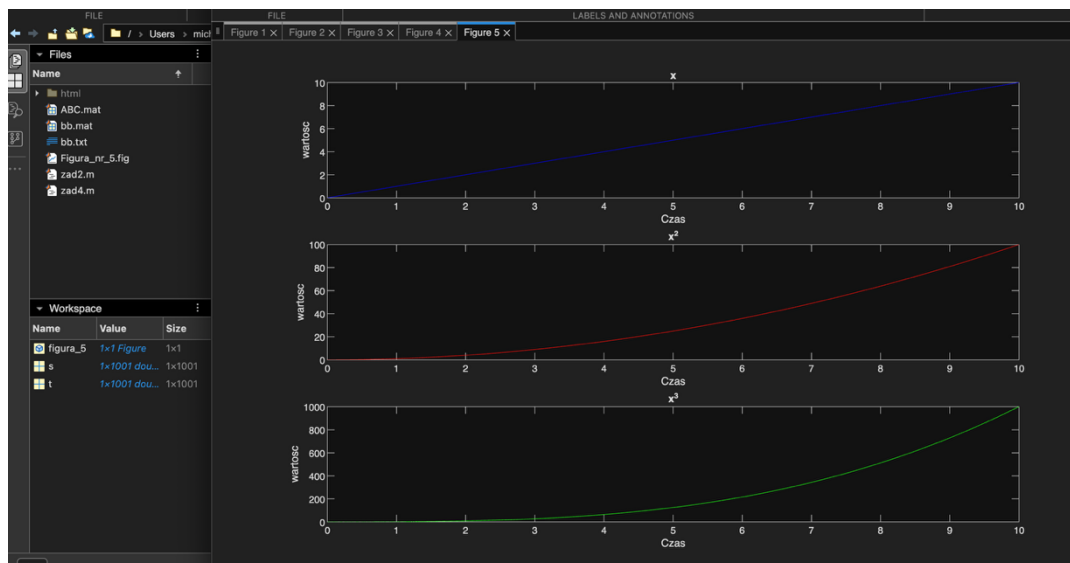


Figura nr 5

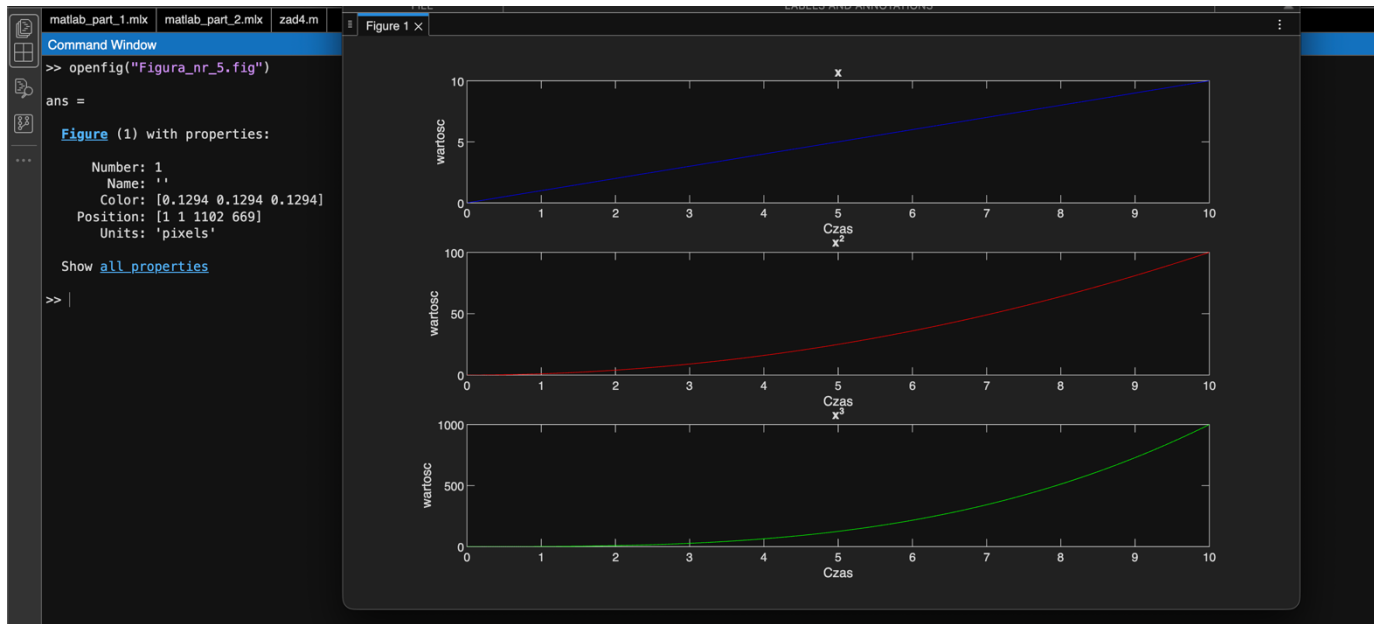


Figura wczytana z pliku