Metodyka i Techniki Programowania II

Katedra Telekomunikacji, **EiT**

dr inż. Jarosław Bułat (c)

kwant@agh.edu.pl

Plan prezentacji

- Matlab licencja TAH - Total Academic Headcount License
- Podstawowe elementy języka Matlab
- Dokumentacja
- Funkcje
- Operacje dyskowe
- Generowanie sygnału (DSP)
- Wizualizacja
- Rozwiązywanie układu równań
- Speed

Podcast

Lex Fridman

Bjarne Stroustrup: C++ | Artificial Intelligence

Podstawowe elementy języka

przypomnienie

Matlab - podstawy

```
clear;
```

- Zawsze kasuj “workspace”

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
x = [ 1,2,3; pi, -1i*3, 3+1i*4 ];
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof
- Wszystko jest wektorem

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
x = [ 1,2,3; pi, -1i*3, 3+1i*4 ];
x(:, 1)
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof
- Wszystko jest wektorem
- Slicing (indeksowanie macierzy)

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
x = [ 1,2,3; pi, -1i*3, 3+1i*4 ];
x(:, 1)

x = 1:3:10
% x =
%   1    4    7   10
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof
- Wszystko jest wektorem
- Slicing (indeksowanie macierzy)
- Tworzenie wektorów

Matlab - podstawy

```
clear;

x = 0;
disp( x );
x

x = 'acb';
x = [ 1,2,3; pi, -1i*3, 3+1i*4 ];
x(:, 1)

x = 1:3:10
% x =
%   1    4    7   10

if x==0
    disp('zero');
end
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof
- Wszystko jest wektorem
- Slicing (indeksowanie macierzy)
- Tworzenie wektorów
- Warunek

Matlab - podstawy

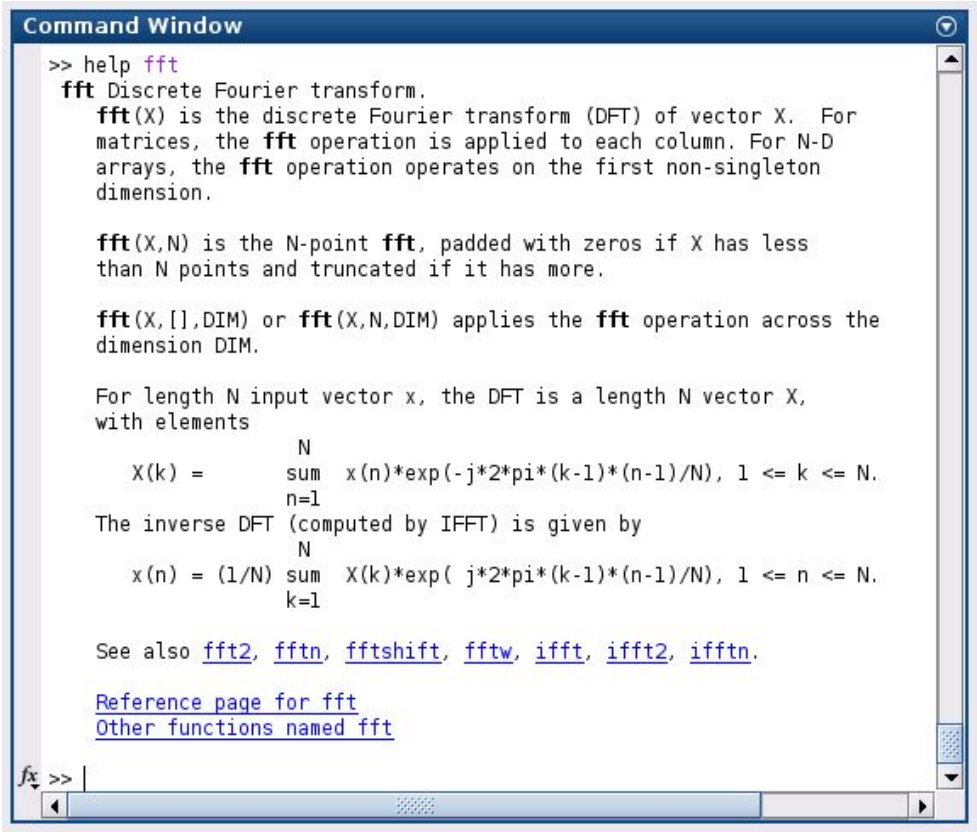
```
clear;

x = 1:3:10
% x =
%   1     4     7    10

for xx=x
    sprintf('iter=%d', xx)
end
% ans =
%     'iter=1'
% ans =
%     'iter=4'
% ans =
%     'iter=7'
```

- Zawsze kasuj “workspace”
- Zmienne nie trzeba deklarować
- Zmienne zmieniają typ
- String == pojedynczy apostrof
- Wszystko jest wektorem
- Slicing (indeksowanie macierzy)
- Tworzenie wektorów
- Warunek
- Pętla

Dokumentacja

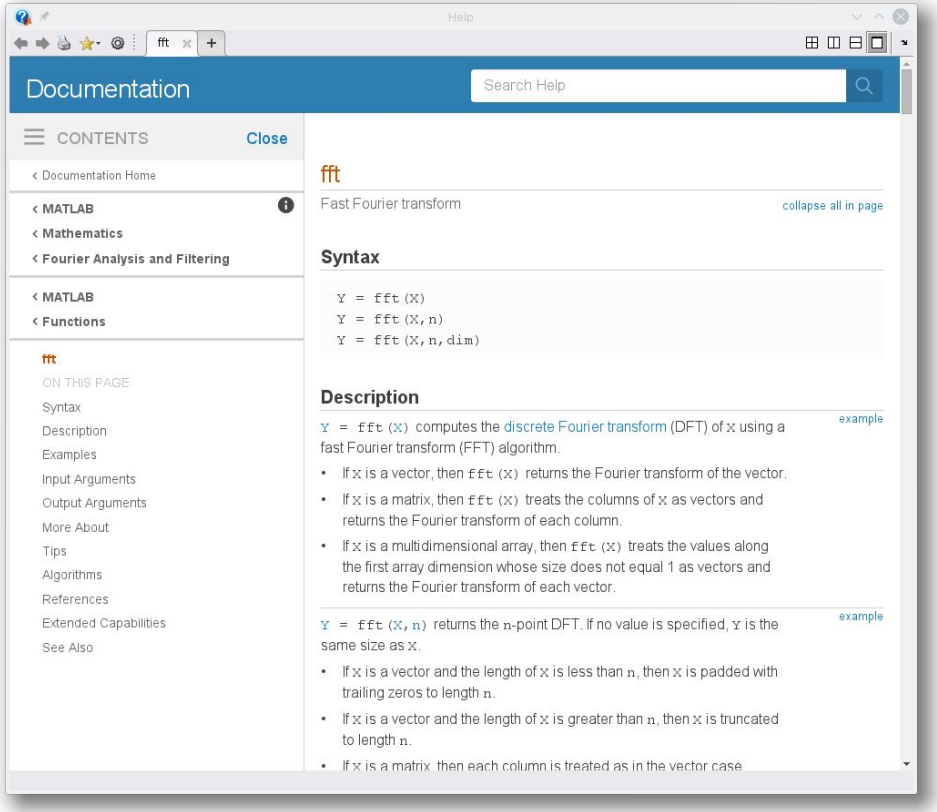
help, doc

Matlab - dokumentacja



» **help xxx:**
dokumentacja inline

Matlab - dokumentacja



- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki

Matlab - dokumentacja

Discrete Fourier Transform of Vector

`Y = fft(X)` and `X = ifft(Y)` implement the Fourier transform and inverse Fourier transform, respectively. For `X` and `Y` of length `n`, these transforms are defined as follows:

$$Y(k) = \sum_{j=1}^{n} X(j)\, W_n^{(j-1)(k-1)}$$

$$X(j) = \frac{1}{n} \sum_{k=1}^{n} Y(k)\, W_n^{-(j-1)(k-1)},$$

where

$$W_n = e^{(-2\pi i)/n}$$

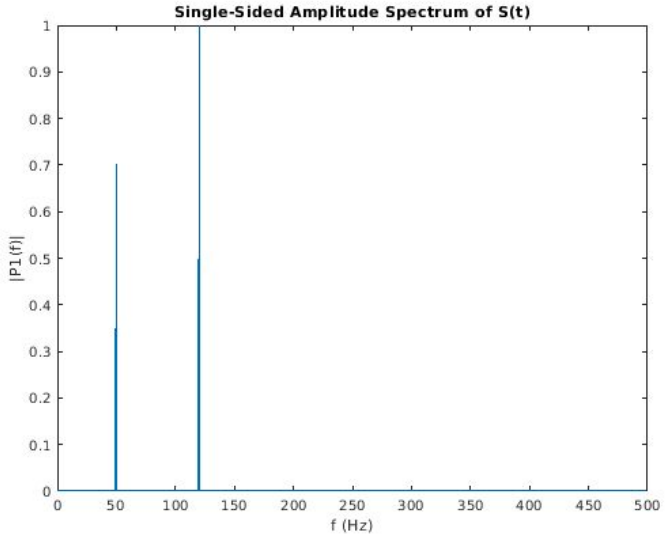
is one of n roots of unity.

- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu

Matlab - dokumentacja

Now, take the Fourier transform of the original, uncorrupted signal and retrieve the exact amplitudes, 0.7 and 1.0.

```
Y = fft(S);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

plot(f,P1)
title('Single-Sided Amplitude Spectrum of S(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```



- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu
  - przykłady

Matlab - dokumentacja

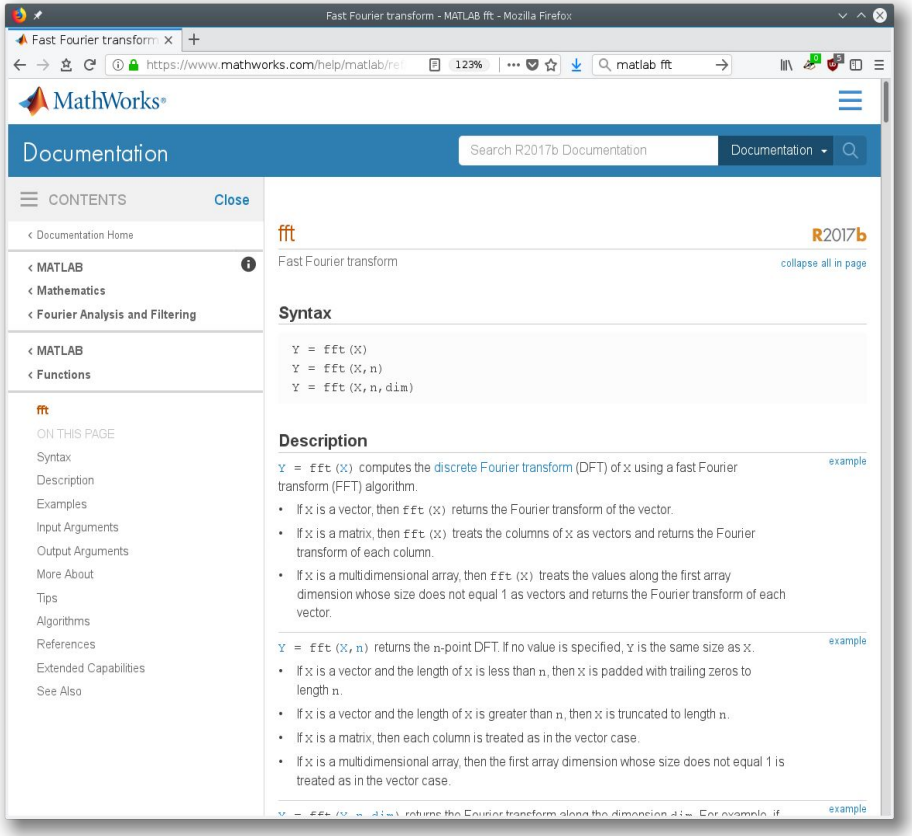
- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu
  - przykłady
  - referencje

References

[1] FFTW (http://www.fftw.org)

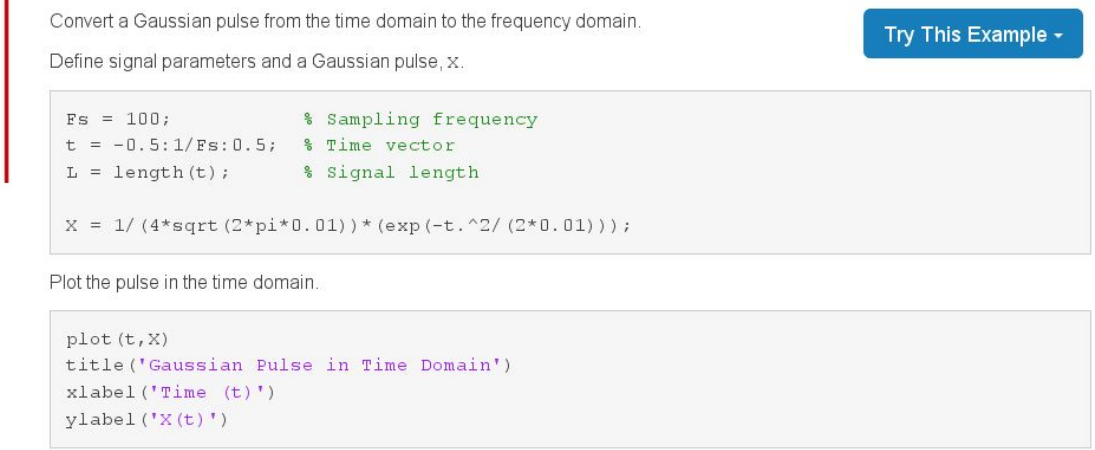
[2] Frigo, M., and S. G. Johnson. "FFTW: An Adaptive Software Architecture for the FFT." *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Vol. 3, 1998, pp. 1381-1384.

Matlab - dokumentacja



- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu
  - przykłady
  - referencje
- dokumentacja **online**

Matlab - dokumentacja



Convert a Gaussian pulse from the time domain to the frequency domain.

Try This Example

Define signal parameters and a Gaussian pulse, X.

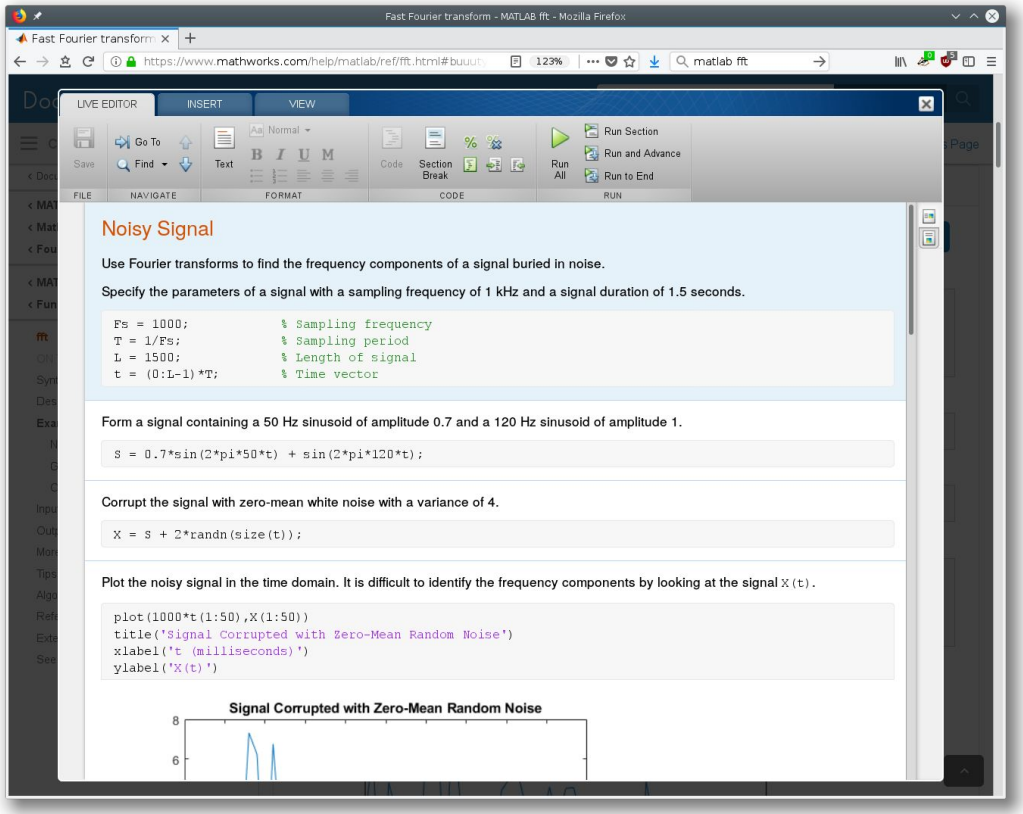
```
Fs = 100;            % Sampling frequency
t = -0.5:1/Fs:0.5;   % Time vector
L = length(t);       % Signal length

X = 1/(4*sqrt(2*pi*0.01))*(exp(-t.^2/(2*0.01)));
```

Plot the pulse in the time domain.

```
plot(t,X)
title('Gaussian Pulse in Time Domain')
xlabel('Time (t)')
ylabel('X(t)')
```

- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu
  - przykłady
  - referencje
- dokumentacja **online**
  - dla każdej wersji
  - dostępna free
- Matlab online

Matlab - dokumentacja



- **help xxx:** dokumentacja inline
- **doc xxx**: szczegółowa dokum.
  - różne przypadki
  - opis algorytmu
  - przykłady
  - referencje
- dokumentacja **online**
  - dla każdej wersji
  - dostępna free
- Matlab online

Funkcje

funkcje vs skrypty

Matlab - funkcje

```
function y = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

    y = 2*x;

end
```

- 1 funkcja == 1 plik
- nazwa_funkcji == nazwa_pliku

Matlab - funkcje

```
function y = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y = 2*x;

end
```

- 1 funkcja == 1 plik
- nazwa_funkcji == nazwa_pliku
- Słowo kluczowe function
  wynik = nazwa(lista, arg)
- Koniec funkcji: end

Matlab - funkcje

```
function y = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y = 2*x;

end
```

- 1 funkcja == 1 plik
- nazwa_funkcji == nazwa_pliku
- Słowo kluczowe function
  wynik = nazwa(lista, arg)
- Koniec funkcji: end
- Zasięg zmiennych (jak w C++)
- Argumenty przekazywane przez kopię
- Wynik przekazywany przez kopię

Matlab - funkcje

```
function y = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y = 2*x;

end
```

```
clear;

x = 1:10;
y = myFun(x)

% y =
%   Columns 1 through 8
%     2     4     ....
```

- 1 funkcja == 1 plik
- nazwa_funkcji == nazwa_pliku
- Słowo kluczowe function
  wynik = nazwa(lista, arg)
- Koniec funkcji: end
- Zasięg zmiennych (jak w C++)
- Argumenty przekazywane przez kopię
- Wynik przekazywany przez kopię
- Użycie funkcji: można nie odbierać wartości zwracanej

Matlab - funkcje

```
function [y1, y2] = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

    y1 = 2*x;

end
```

```
clear;

x = 1:10;
[a, b] = myFun(x)

% y =
%   Columns 1 through 8
%      2     4     ....
```

» Funkcja może zwrócić wiele zmiennych (liczby, wektory, macierze)

Matlab - funkcje

```
function [y1, y2] = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y1 = 2*x;

end
```

```
clear;

x = 1:10;
[a, b] = myFun(x)

% y =
%   Columns 1 through 8
%     2     4     ....
```

- Funkcja może zwrócić wiele zmiennych (liczby, wektory, macierze)
- Jaki błąd zrobiłem?

Matlab - funkcje

```
function [y1, y2] = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y1 = 2*x;
   y2 = 2*y1;
end
```

```
clear;

x = 1:10;
[a, b] = myFun(x)

% y =
%   Columns 1 through 8
%     2     4     ....
```

- Funkcja może zwrócić wiele zmiennych (liczby, wektory, macierze)
- Plik myFun.m w bieżącej ścieżce dostępu lub w “**path**”
- **help myFun** wydrukuje komentarz

Matlab - funkcje

```
function [y1, y2] = myFun( x )
% myFun multiple arg by 2
%  y = myFun(x); multiply by 2

   y1 = 2*x;
   y2 = 2*y1;
end
```

```
clear;

x = 1:10;
[a, b] = myFun(x)

% y =
%   Columns 1 through 8
%     2     4     ....
```

- Funkcja może zwrócić wiele zmiennych (liczby, wektory, macierze)
- Plik myFun.m w bieżącej ścieżce dostępu lub w “**path**”
- **help myFun** wydrukuje komentarz
- plik: ~/git/lab/myfun.m
- plik: ~/git/lab/ex01.m

Jak coś zapisać na dysk

I/O

Matlab - I/O

```
clear;

x = 20;
save ex03.mat
```

- Zapisz cały “Workspace”
  - do pliku ex03.mat
  - wszystkie zmienne

Matlab - I/O

```
clear;

x = 20;
save ex03.mat

clear;
load ex03.mat

disp(x)
```

- Zapisz cały “Workspace”
  - do pliku ex03.mat
  - wszystkie zmienne
- Wczytaj wszystkie zmienne z pliku ex03.mat i umieść je w workspace:
  przywróć zapisany stan

Matlab - I/O

```
clear;
y = 0;
x = 20;
save ex03.mat x y

clear x;
load ex03.mat x

disp(x)
```

- Zapisz cały “Workspace”
  - do pliku ex03.mat
  - wszystkie zmienne
- Wczytaj wszystkie zmienne z pliku ex03.mat i umieść je w workspace:
  przywróć zapisany stan
- Bardziej specyficznie

Matlab - I/O

```
clear;
y = 0;
x = 20;
save( 'ex03.mat', 'x', 'y' );

clear x;
load( 'ex03.mat', 'x' );

disp(x)
```

- Zapisz cały “Workspace”
  - do pliku ex03.mat
  - wszystkie zmienne
- Wczytaj wszystkie zmienne z pliku ex03.mat i umieść je w workspace:
  przywróć zapisany stan
- Bardziej specyficznie
- Jako funkcja, argumenty to:
  - ścieżka dostępu
  - nazwa zmiennej a nie zmienna !!!!

general purpose I/O

clear;

» Konwencja jak w języku **C**:

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );
```

- Konwencja jak w języku **C**:
  - otworzyć plik

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );

if fileID ~= -1


end
```

- Konwencja jak w języku **C**:
  - otworzyć plik
  - sprawdzić czy się udało

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );

if fileID ~= -1
    x = fread( fileID, 'double' );

end
```

- Konwencja jak w języku **C**:
  - otworzyć plik
  - sprawdzić czy się udało
  - przeczytać (format danych)

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );

if fileID ~= -1
    x = fread( fileID, 'double' );
    fclose( fileID );
end
```

- Konwencja jak w języku **C**:
  - otworzyć plik
  - sprawdzić czy się udało
  - przeczytać (format danych)
  - zamknąć

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );

if fileID ~= -1
    x = fread( fileID, 'double' );
    fclose( fileID );
end
```

```
clear;
x = 0:pi/10:2*pi;

fileID = fopen( 'nazwa_pliku', 'wb' );
if fileID ~= -1
    x = fwrite( fileID, x, 'double' );
    fclose( fileID );
end
```

- Konwencja jak w języku **C**:
  - otworzyć plik
  - sprawdzić czy się udało
  - przeczytać (format danych)
  - zamknąć
- Zapis do pliku analogicznie:
  - dane binarne, nie .mat !!!

general purpose I/O

```
clear;

fileID = fopen( 'nazwa_pliku', 'rb' );

if fileID ~= -1
    x = fread( fileID, 'double' );
    fclose( fileID );
end
```

```
clear;
x = 0:pi/10:2*pi;

fileID = fopen( 'nazwa_pliku', 'wb' );
if fileID ~= -1
    x = fwrite( fileID, x, 'double' );
    fclose( fileID );
end
```

- Konwencja jak w języku **C**:
  - otworzyć plik
  - sprawdzić czy się udało
  - przeczytać (format danych)
  - zamknąć
- Zapis do pliku analogicznie:
  - dane binarne, nie .mat !!!
  - cały wektor na raz

dedykowane I/O

```
clear;

X1 = textread( 'nme.txt', '%d %f' );
```

- Czyta i konwertuje ascii -> bin

dedykowane I/O

```
clear;

X1 = textread( 'nme.txt', '%d %f' );

var = textscan( fileID, '%d %f' );
str = '2 3.14';
var = textscan( str, '%d %f' );
```

- Czyta i konwertuje ascii -> bin
- Obecnie rekomendowany sposób:
  - z pliku (fileID)
  - z dowolnego stringu

dedykowane I/O

```
clear;

X1 = textread( 'nme.txt', '%d %f' );

var = textscan( fileID, '%d %f' );
str = '2 3.14';
var = textscan( str, '%d %f' );

A = csvread( 'filename', R1, C1 );
A = dlmread( 'filename', delimiter );
A = xlsread( 'filename', sheet );
```

- Czyta i konwertuje ascii -> bin
- Obecnie rekomendowany sposób:
  - z pliku (fileID)
  - z dowolnego stringu
- Czytanie csv
- Z pliku txt - dowolny delimiter
- Z arkusza kalkulacyjnego (MS)

dedykowane I/O

```
clear;

X1 = textread( 'nme.txt', '%d %f' );

var = textscan( fileID, '%d %f' );
str = '2 3.14';
var = textscan( str, '%d %f' );

A = csvread( 'filename', R1, C1 );
A = dlmread( 'filename', delimiter );
A = xlsread( 'filename', sheet );

[y, FS] = audioread( 'fname.mp3' );
A = imread( 'len_full.jpg' );
```

- Czyta i konwertuje ascii -> bin
- Obecnie rekomendowany sposób:
  - z pliku (fileID)
  - z dowolnego stringu
- Czytanie csv
- Z pliku txt - dowolny delimiter
- Z arkusza kalkulacyjnego (MS)
- Plik dźwiękowy
  - wav, flac, mp3, …
- Plik graficzny (jpg, png, tiff)
  - A to macierz **W**x**H**x**3**

dedykowane I/O

```
clear;

X1 = textread( 'nme.txt', '%d %f' );

var = textscan( fileID, '%d %f' );
str = '2 3.14';
var = textscan( str, '%d %f' );

A = csvread( 'filename', R1, C1 );
A = dlmread( 'filename', delimiter );
A = xlsread( 'filename', sheet );

[y, FS] = audioread( 'fname.mp3' );
A = imread( 'len_full.jpg' );

imwrite( A, 'out.jpg');
```

- Czyta i konwertuje ascii -> bin
- Obecnie rekomendowany sposób:
  - z pliku (fileID)
  - z dowolnego stringu
- Czytanie csv
- Z pliku txt - dowolny delimiter
- Z arkusza kalkulacyjnego (MS)
- Plik dźwiękowy
  - wav, flac, mp3, …
- Plik graficzny (jpg, png, tiff)
  - A to macierz **W**x**H**x**3**
- Zapis (z parametrami)

AGH

Jak wygenerować sygnał

proste DSP

generowanie sygnałów

```
clear;
```

» Cel: wygenerowanie wykresu funkcji y = 1/x

generowanie sygnałów

```
clear;

x = 0:100;
```

» Cel: wygenerowanie wykresu funkcji y = 1/x
  – dziedzina (argumenty)

generowanie sygnałów

```
clear;

x = 0:100;
y = 1/x;
```

» Cel: wygenerowanie wykresu funkcji y = 1/x
  – dziedzina (argumenty)
  – wartości

generowanie sygnałów

```
clear;

x = 0:100;
y = 1/x;
```

- Cel: wygenerowanie wykresu funkcji y = 1/x
  - dziedzina (argumenty)
  - wartości
- Błąd:

```
Error using  /
Matrix dimensions must agree.

Error in ex09 (line 4)
y = 1/x;
```

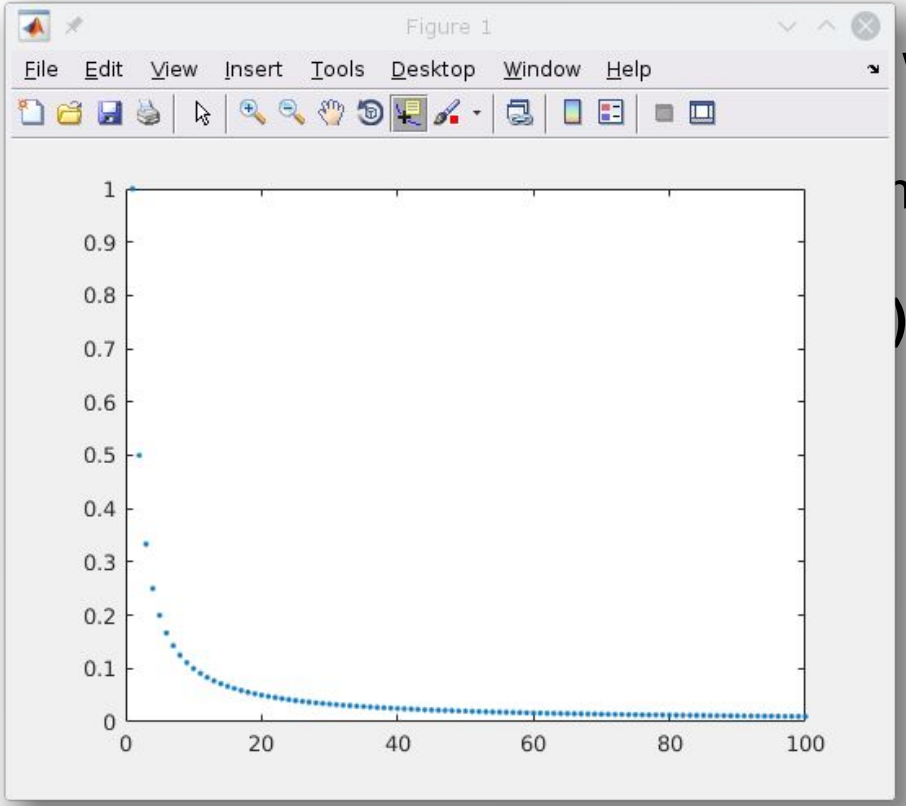
generowanie sygnałów

```
clear;

x = 0:100;
y = 1./x;

plot(x, y, '.');
```

- Cel: wygenerowanie wykresu funkcji y = 1/x
  - dziedzina (argumenty)
  - wartości
- Plot **size(x)==size(y)**

generowanie sygnałów

```
clear;

x = 0:100;
y = 1./x;

plot(x, y, '.');
```



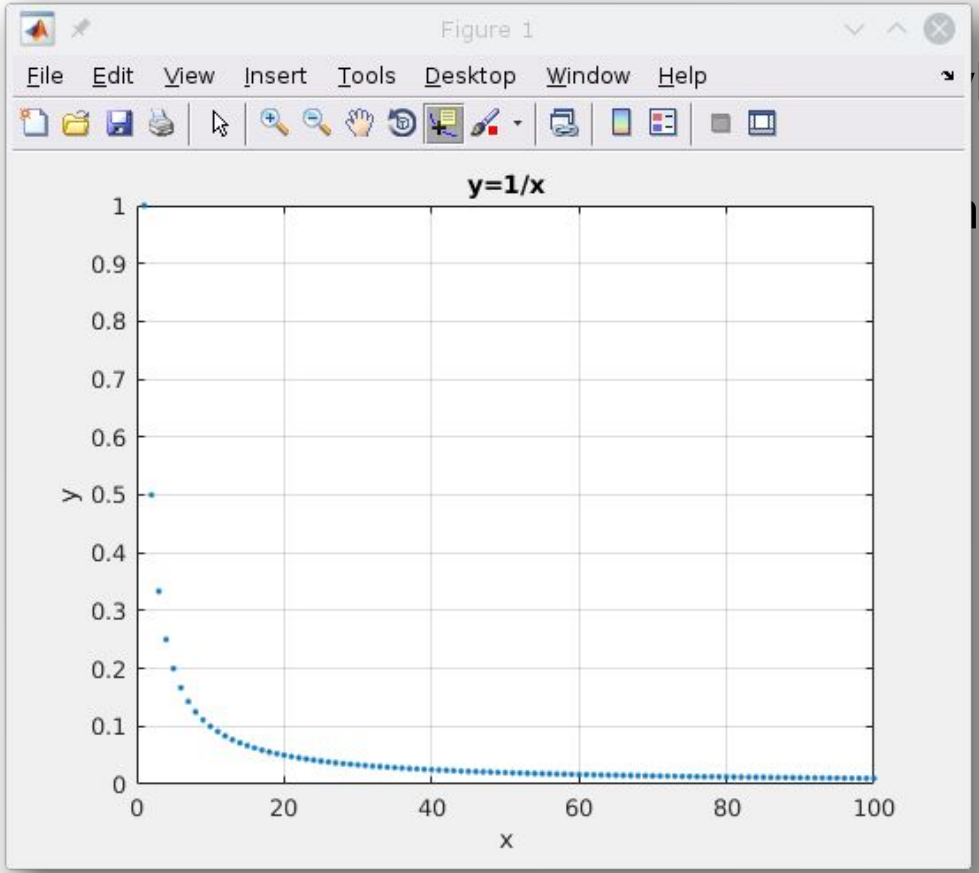
wykresu

nenty)

)

generowanie sygnałów

```
clear;

x = 0:100;
y = 1./x;

plot(x, y, '.');

grid;
xlabel('x');
ylabel('y');
title('y=1/x');
```



kresu

ty)

generowanie sygnałów

```
clear;
```

» Cel: wygenerowanie wykresu funkcji y = sin(x)

generowanie sygnałów

```
clear;

f = 50;
```

- Cel: wygenerowanie wykresu funkcji y = sin(x)
  - częstotliwość 50 Hz

generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
```

- Cel: wygenerowanie wykresu funkcji y = sin(x)
  - częstotliwość 50 Hz
  - fs 1000 Hz

generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 1000;
```

- Cel: wygenerowanie wykresu funkcji y = sin(x)
  - częstotliwość 50 Hz
  - fs 1000 Hz
  - długość sygnału 1000 próbek

generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 1000;
t = 0:N-1;
```

- Cel: wygenerowanie wykresu funkcji y = sin(x)
  - częstotliwość 50 Hz
  - fs 1000 Hz
  - długość sygnału 1000 próbek
  - wektor czasu

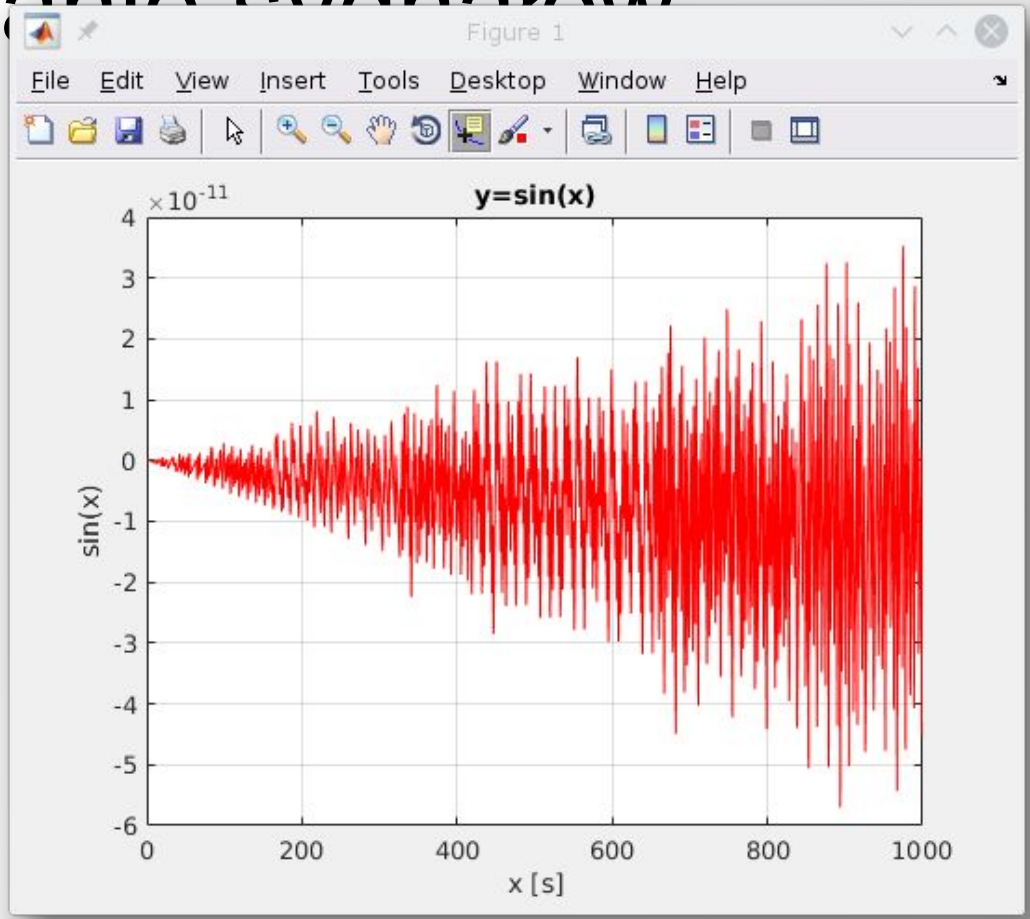
generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 1000;
t = 0:N-1;
plot(t, sin(2*pi*t*f), 'r');

grid;
xlabel('x [s]');
ylabel('sin(x)');
title('y=sin(x)');
```

- Cel: wygenerowanie wykresu funkcji y = sin(x)
  - częstotliwość 50 Hz
  - fs 1000 Hz
  - długość sygnału 1000 próbek
  - wektor czasu
  - **wykres** + opis osi

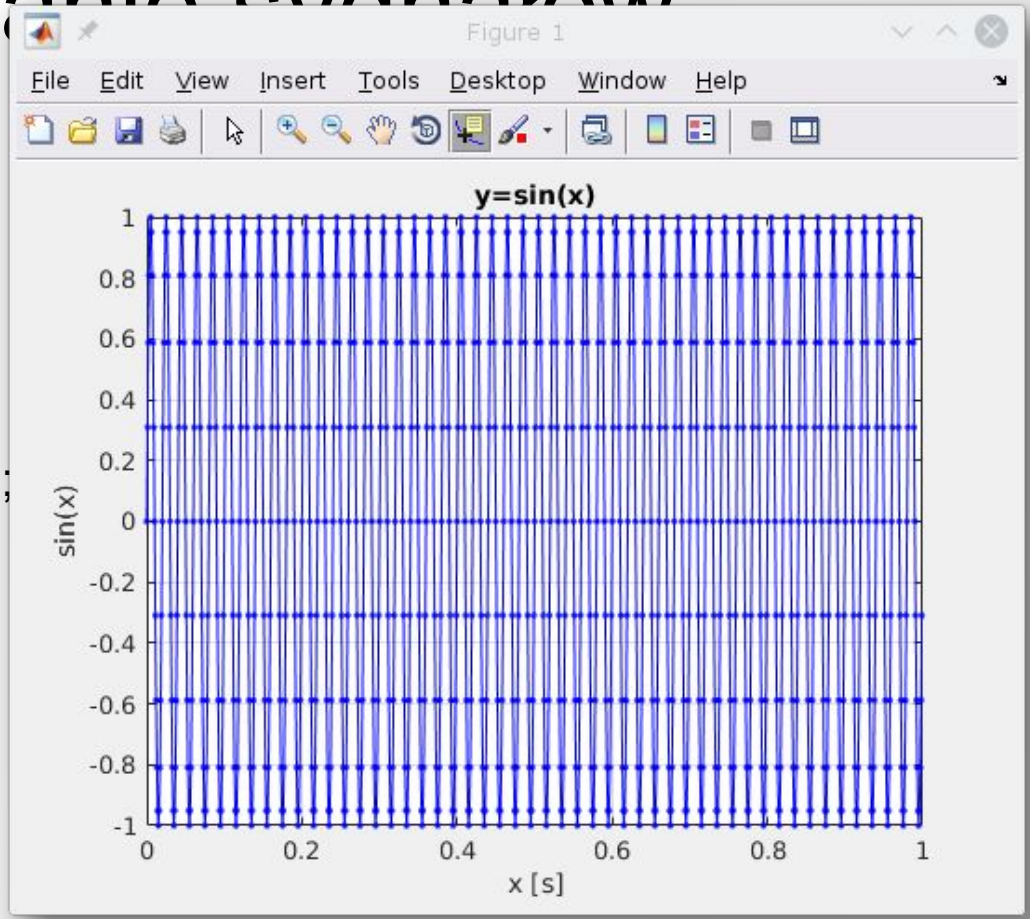
generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 1000;
t = 0:N-1;
plot(t, sin(2*pi*t*f), 'r');

grid;
xlabel('x [s]');
ylabel('sin(x)');
title('y=sin(x)');
```



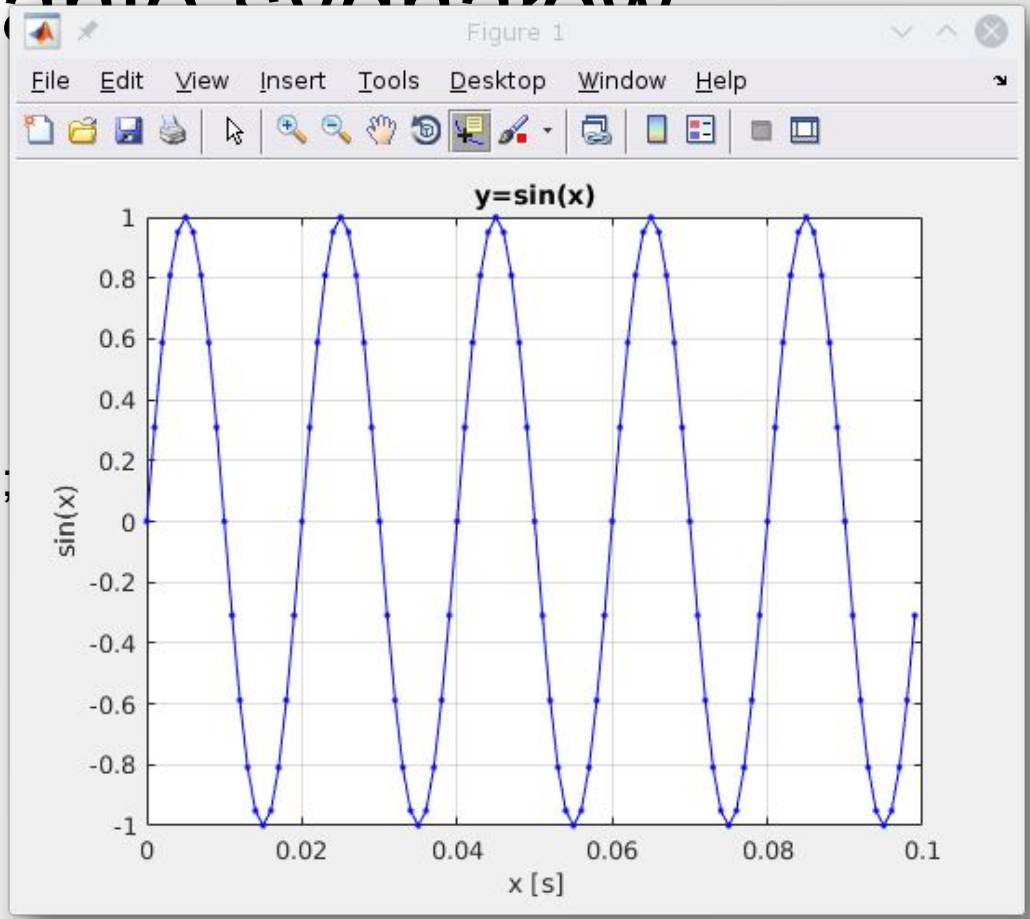
generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 1000;
t = (0:N-1)/fs;
plot(t, sin(2*pi*t*f), 'b.-');

grid;
xlabel('x [s]');
ylabel('sin(x)');
title('y=sin(x)');
```



generowanie sygnałów

```
clear;

f = 50;
fs = 1000;
N = 100;
t = (0:N-1)/fs;
plot(t, sin(2*pi*t*f), 'b.-');

grid;
xlabel('x [s]');
ylabel('sin(x)');
title('y=sin(x)');
```



Wizualizacja danych

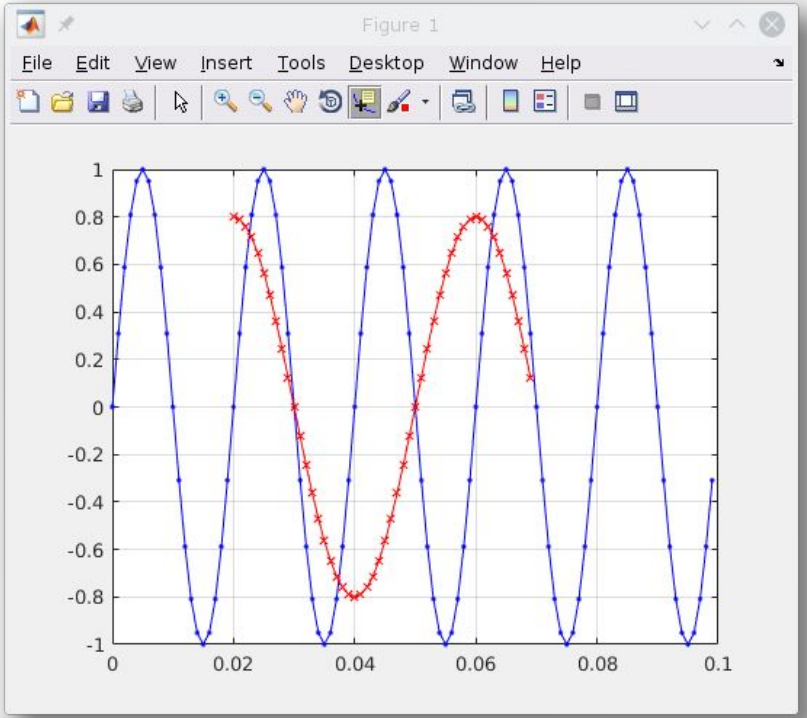
plot() i okolice

Wizualizacja danych

```
clear;

f = 50;
fs = 1000;
N = 100;
t1 = (0:N-1)/fs;
y1 = sin(2*pi*t1*f);

t2 = (0:N/2-1)/fs;
y2 = 0.8*cos(2*pi*t2*f/2);

plot(t1, y1, 'b.-', t2+0.02, y2, 'rx-');
grid;

figure,
subplot(2,1,1); plot(t1, y1, 'b.-')
subplot(2,1,2); plot(t2+0.02, y2, 'rx-');
```

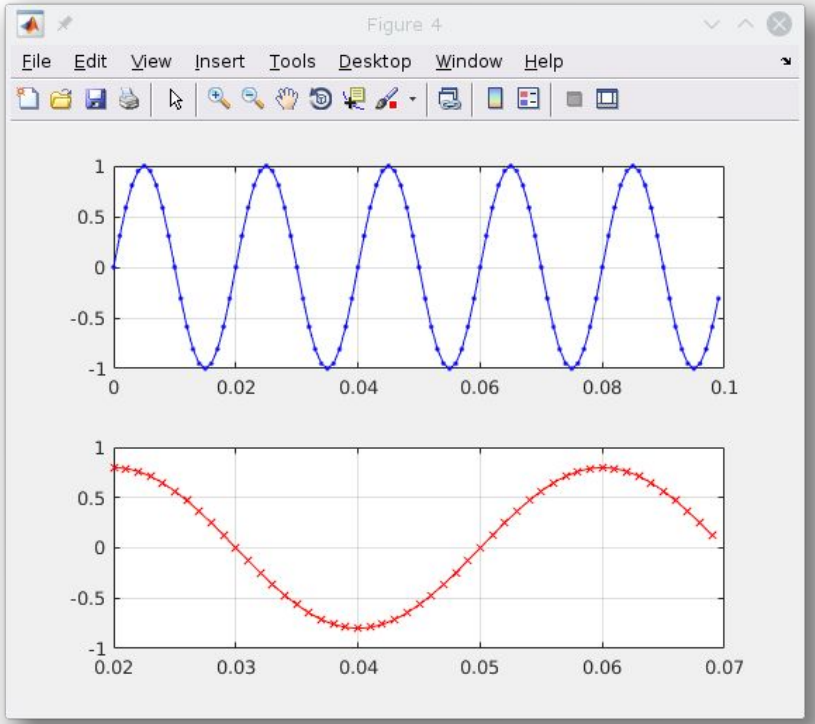
Wizualizacja danych

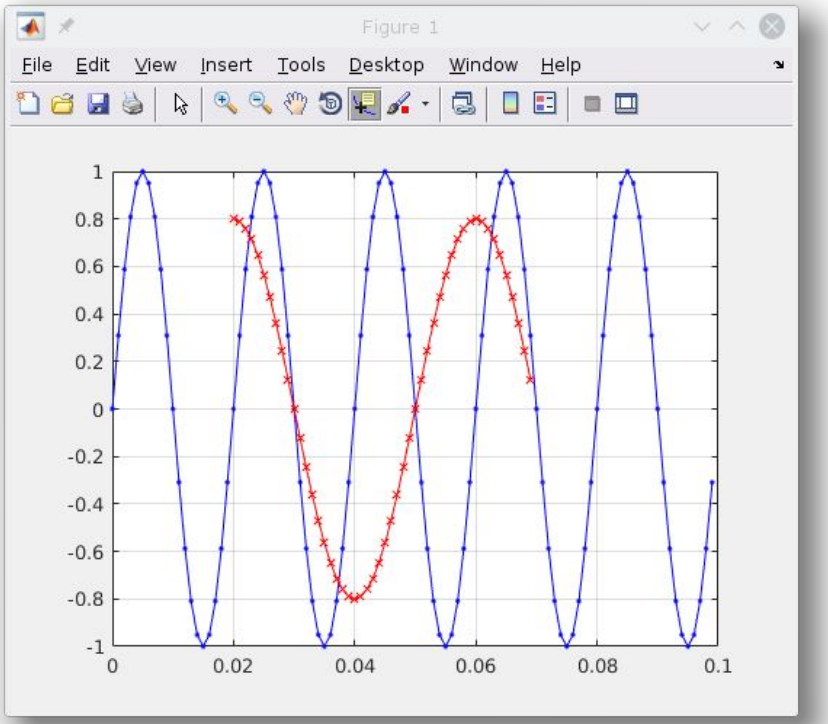
```
clear;

f = 50;
fs = 1000;
N = 100;
t1 = (0:N-1)/fs;
y1 = sin(2*pi*t1*f);

t2 = (0:N/2-1)/fs;
y2 = 0.8*cos(2*pi*t2*f/2);

plot(t1, y1, 'b.-', t2+0.02, y2, 'rx-');
grid;

figure,
subplot(2,1,1); plot(t1, y1, 'b.-')
subplot(2,1,2); plot(t2+0.02, y2, 'rx-');
```



Wizualizacja danych

```
clear;
```





Układ równań liniowych

rozwiązanie w Matlabie

Układ równań liniowych

» Równanie:

$$2X = 3Y + 1$$

$$X + Y = 4$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

» Równanie:

$$\begin{aligned} 2X &= 3Y + 1 \\ X + Y &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2X - 3Y &= 1 \\ X + Y &= 4 \end{aligned}$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

» Równanie:

$$2X = 3Y + 1 \qquad 2X - 3Y = 1$$
$$X + Y = 4 \qquad X + Y = 4$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

» Równanie:

$$2X = 3Y + 1 \qquad 2X - 3Y = 1$$
$$X + Y = 4 \qquad X + Y = 4$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$A*x=B$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

```
A = [2 -3; 1 1]
% A =
%      2    -3
%      1     1

B = [1; 4]
% B =
%      1
%      4
```

» Równanie:

$$\begin{aligned} 2X &= 3Y + 1 \\ X + Y &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2X - 3Y &= 1 \\ X + Y &= 4 \end{aligned}$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$A*x=B$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

```
A = [2 -3; 1 1]
% A =
%      2    -3
%      1     1

B = [1; 4]
% B =
%      1
%      4

x = A\B
A^-1*B
inv(A)*B
pinv(A)*B
mldivide(A, B)
```

» Równanie:

$$\begin{aligned} 2X &= 3Y + 1 \\ X + Y &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2X - 3Y &= 1 \\ X + Y &= 4 \end{aligned}$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$\boldsymbol{A*x=B}$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

```
A = [2 -3; 1 1]
% A =
%      2    -3
%      1     1

B = [1; 4]
% B =
%      1
%      4

x = A\B
A^-1*B
inv(A)*B
pinv(A)*B
mldivide(A, B)
```

```
% ans =
%     2.6000
%     1.4000
```

» Równanie:

$$\begin{aligned} 2X &= 3Y + 1 \\ X + Y &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2X - 3Y &= 1 \\ X + Y &= 4 \end{aligned}$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$A*x=B$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Układ równań liniowych

```
A = [2 -3; 1 1]
% A =
%      2    -3
%      1     1

B = [1; 4]
% B =
%      1
%      4

x = A\B
A^-1*B
inv(A)*B
pinv(A)*B
mldivide(A, B)
```

```
% ans =
%     2.6000
%     1.4000
```

» Równanie:

$$2X = 3Y + 1 \quad \Rightarrow \quad 2X - 3Y = 1$$
$$X + Y = 4 \qquad\qquad X + Y = 4$$

» W formacie macierzowym:

$$\begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$A*x=B$$

https://www.mathworks.com/videos/solving-linear-equations-97489.html

Jak pisać kod żeby był szybki

jeżeli to możliwe nie używaj pętli

Optymalizacja szybkości

```
clear;

tic;
% code
% code
% code
toc;
```

- Pomiar czasu
- Rezultat to liczba sekund pomiędzy **toc** i **tic**

Optymalizacja szybkości

```
clear;

tic;
% code
% code
% code
elapsed = toc;
```

- Pomiar czasu
- Rezultat to liczba sekund pomiędzy **toc** i **tic**
- Zamiast wypisywać na ekranie można dane o szybkości zgromadzić i użyć (np. w celu optymalizacji)
- Potencjalne niedokładności:
  - mało złożony kod 1e-5s
  - pierwsze iteracje algorytmu (działający JIT)
  - wielozadaniowość
  - hardware dependent

Optymalizacja szybkości

- Profilowanie prostego kodu

```
clear;

f = 50;
fs = 10000;
N = 1e7;

% sin(2*pi*t*f);
```

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end

c = 2*pi/fs*f;
for n=0:N-1
    y1(n+1) = sin(c*n);
end
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy
- **1.79 s**: stałe liczone raz

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end

c = 2*pi/fs*f;
for n=0:N-1
    y1(n+1) = sin(c*n);
end

y2 = zeros(1,N);
for n=0:N-1
    y2(n+1) = sin(c*n);
end
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy
- **1.79 s**: stałe liczone raz
- **0.73 s**: prealokacja

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end

c = 2*pi/fs*f;
for n=0:N-1
    y1(n+1) = sin(c*n);
end

y2 = zeros(1,N);
for n=0:N-1
    y2(n+1) = sin(c*n);
end

t = (0:N-1)/fs;
y3 = sin(2*pi*t*f);
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy
- **1.79 s**: stałe liczone raz
- **0.73 s**: prealokacja
- **0.35 s**: Matlab-way

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end

c = 2*pi/fs*f;
for n=0:N-1
    y1(n+1) = sin(c*n);
end

y2 = zeros(1,N);
for n=0:N-1
    y2(n+1) = sin(c*n);
end

t = (0:N-1)/fs;
y3 = sin(2*pi*t*f);
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy
- **1.79 s**: stałe liczone raz
- **0.73 s**: prealokacja
- **0.35 s**: Matlab-way

5.5x szybciej !!!

Optymalizacja szybkości

```
for n=1:N
    y0(n) = sin(2*pi*(n-1)/fs*f);
end

c = 2*pi/fs*f;
for n=0:N-1
    y1(n+1) = sin(c*n);
end

y2 = zeros(1,N);
for n=0:N-1
    y2(n+1) = sin(c*n);
end

t = (0:N-1)/fs;
y3 = sin(2*pi*t*f);
```

- Profilowanie prostego kodu
- **1.93 s**: najgorzej, C-like, automatyczne rozszerzanie tablicy
- **1.79 s**: stałe liczone raz
- **0.73 s**: prealokacja
- **0.35 s**: Matlab-way

5.5x szybciej !!!

- znowu zrobiłem błąd...

Efektywny kod

sumowanie

Optymalizacja szybkości

y2 = ….
y3 = ….

» Porównanie wyników

Optymalizacja szybkości

```
y2 = ….
y3 = ….

sum = 0;
for n = 1:N
    diff = y2(n) - y3(n);
    sum = sum + abs(diff);
end
```

- Porównanie wyników
- **0.21 s**: C-way

Optymalizacja szybkości

```
y2 = ….
y3 = ….

sum = 0;
for n = 1:N
    diff = y2(n) - y3(n);
    sum = sum + abs(diff);
end

diff = y2-y3;
sum(abs(diff));
```

- Porównanie wyników
- **0.21 s**: C-way
- **0.13 s**: Matlab-way

Optymalizacja szybkości

```
y2 = ….
y3 = ….

sum = 0;
for n = 1:N
    diff = y2(n) - y3(n);
    sum = sum + abs(diff);
end

diff = y2-y3;
sum(abs(diff));

sum(abs(y2-y3));
```

- Porównanie wyników
- **0.21 s**: C-way
- **0.13 s**: Matlab-way
- **0.09 s**: The best

2.3x szybciej

Optymalizacja szybkości

```
y2 = ….
y3 = ….

sum = 0;
for n = 1:N
    diff = y2(n) - y3(n);
    sum = sum + abs(diff);
end

diff = y2-y3;
sum(abs(diff));

sum(abs(y2-y3));
```

- Porównanie wyników
- **0.21 s**: C-way
- **0.13 s**: Matlab-way
- **0.09 s**: The best

2.3x szybciej

- znowu zrobiłem błąd...

Efektywny kod

mnożenie

Optymalizacja szybkości

y2 = ….
y3 = ….

y4(n) = y2(n) * y3(n)

» Mnożenie

Optymalizacja szybkości

```
y2 = ….
y3 = ….

for n = 1:N
    y4(n) = y2(n) * y3(n);
end
```

- Mnożenie
- **1.05 s**: C-way

Optymalizacja szybkości

```
y2 = ….
y3 = ….

for n = 1:N
    y4(n) = y2(n) * y3(n);
end

y5 = zeros(1,N);
for n = 1:N
    y5(n) = y2(n) * y3(n);
end
```

- Mnożenie
- **1.05 s**: C-way
- **0.15 s**: + prealokacja

Optymalizacja szybkości

```
y2 = ….
y3 = ….

for n = 1:N
    y4(n) = y2(n) * y3(n);
end

y5 = zeros(1,N);
for n = 1:N
    y5(n) = y2(n) * y3(n);
end

y6 = y2.*y3;
```

- Mnożenie
- **1.05 s**: C-way
- **0.15 s**: + prealokacja
- **0.028s**: Matlab-way

38x szybciej!!!

AGH

Dziękuję