

Instrukcje sterujące

MATLAB jest wyposażony w instrukcje sterujące o składni zapożyczonych z języka C.

Lista instrukcji MATLABa jest krótka:

- instrukcja warunkowa **if**
- instrukcja wyboru **switch**
- instrukcje iteracyjne: **for, while**
- instrukcje przerywania: **break i return**

```
▪ if wyrażenie1
    polecenia
elseif wyrażenie2
    polecenia
else
    polecenia
end
```

Wyrażenia logiczne mają najczęściej postać:

wyrażenie operator_logiczny wyrażenie

Jako operatory logiczne stosowane są: **< <= > >= == ~=**

Wynik wyrażen używanych w instrukcji warunkowej, tj. pisanych po **if** oraz **elseif**, musi być liczbą (zero lub $\neq 0$). Wyrażenia te są zazwyczaj relacją lub wyrażeniem logicznym. W MATLABie wynikiem relacji bądź wyrażenia logicznego może też być wektor lub macierz. W celu zredukowania takiego wyniku do wartości skalarnej stosuje się funkcje logiczne:

all	wszystkie elementy wektora niezerowe
any	jakikolwiek element wektora jest niezerowy
isfinite	element argumentu jest określony
isinf	element argumentu jest równy +Inf lub -Inf
isnan	element argumentu jest nieokreślony (NaN)
isempty	argument jest macierzą pustą
i inne	

```
▪ switch wyrażenie
    case wartość1
        polecenia
    case wartość2
        polecenia
    ...
    otherwise
        polecenia
end
```

Wynik wyrażenia zapisanego po **switch** może być wartością skalarną lub łańcuchem. Po **case** umieszcza się wyrażenie, pojedynczą stałą lub listę stałych wyboru.

```
▪ for zmienna-iterowana = macierz-wartości
    polecenia
end
```

Macierz wartości stosowane w instrukcji **for** jest najczęściej zapisywana w postaci wyrażenia: *wartość_początkowa* : *krok* : *wartość_końcowa*

Pętle **for** mogą być wzajemnie zagnieżdżane. Opuszczenia pętli **for** przed zakończeniem jej wykonywania można dokonać za pomocą **break**. W przypadku pętli zagnieżdżonych **break** powoduje przejście do pętli nadrzędnej.

```
▪ while wyrażenie
    polecenia
end
```

Używając operatora równości == w instrukcjach iteracyjnych i warunkowych należy zachować szczególną ostrożność pamiętając, że nie wszystkie liczby rzeczywiste są w zapisie zmiennoprzecinkowym reprezentowane dokładnie.

przykład: nie kończąca się pętla:

```
s=0;
while s~=100,
    s=s+0.01;
end
```

Skrypty i funkcje

MATLAB oferuje dwa narzędzia: skrypty i funkcje. Zapewniają one wygodną i interaktywną współpracę programu z użytkownikiem.

Pliki zawierające polecenia MATLABa nazywa się m-plikami. M-pliki mogą być skryptami lub funkcjami.

Większe lub częściej wykorzystywane sekwencje poleceń najlepiej jest umieścić w tzw. pliku skryptowym. Pliki skryptowe zawierają ciągi poleceń MATLABa przeznaczonych do wykonania przez interpreter. Operują one na zmiennych dostępnych w przestrzeni roboczej.

Dla wszystkich skryptów przyjęto standardowe rozszerzenie **.m**. Skrypt nie musi spełniać żadnych dodatkowych wymogów poza poprawnością składniową i semantyczną znajdujących się w nim poleceń.

W m-plikach skryptowych używa się poleceń, które umożliwiają interaktywną współpracę programu z użytkownikiem:

input	wczytanie wartości z wyświetleniem tekstu zachęty
keyboard	przerywa wykonywanie skryptu i przekazuje kontrolę użytkownikowi
menu	generowanie okna menu
pause	wstrzymanie wykonywania skryptu i oczekiwanie na dowolny klawisz

Wadą m-pliku skryptowego jest możliwość powstania konfliktów nazw zmiennych w przestrzeni roboczej. M-pliki skryptowe należy traktować jako etap wstępny w przygotowaniu m-pliku funkcyjnego.

przykłady: skrypt `sinus.m` obliczający w 101 punktach wartość funkcji $x = \sin 2t + \sin t$ i rysujący jej wykres na przedziale $\langle 0, 2\pi \rangle$

```
% skrypt
t=[0:0.01:2*pi];
A=[1 1];
x=A*[sin(2*t); sin(t)];
plot(t,x)
```

skrypt `cosinus.m` – algorytm obliczania wartości funkcji $\cos(x)$ z żadaną dokładnością e jako obciętej sumy szeregu nieskończonego:

$$\cos(x) = 1 - \frac{x^2}{1 \cdot 2} + \frac{x^4}{1 \cdot 2 \cdot 3 \cdot 4} - \frac{x^6}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} + \dots$$

```
% obliczanie cosinusa
% w przestrzeni powinny być: x i e
y=1; k=0; s=1; x2=x*x;
while abs(y)>e
    k=k+2;
    y=-y*x2/(k*(k-1));
    s=s+y;
end
wywołanie: >> x=0.5; e=0.001; cosinus; s

plik szereg.m
t=0:0.1:2*pi;
plot(t,cos(t))
title('Funkcja cos jako suma szeregu')
pause
e=0.0001; n=10; % n-max liczba wykonywanych zmian
x=input('Podaj argument funkcji x: ');
cosinus;
x1=num2str(x); m1=int2str(k/2); e1=sprintf('%1.1e',e);
s1=sprintf('%1.9f',s); s2=sprintf('%1.9f',cos(x));
disp(['Wynik funkcji standardowej: cos(',x1,')= ',s2])
disp(['Wynik sumowania szeregu:      cos(',x1,')= ',s1])

wybor=menu('Wybierz piosenke:', ...
    'Prawy do lewego', ...
    'We will rock U', ...
    'Mydelko Fa', ...
    'Spadowa z imprezy');
polecenie=['play (rightleft.mp3)'; ...
    'play (rock.mp3)'; ...
    'play (fa.mp3)'; ...
    'return'];
eval(polecenie(wybor,:));
```

W języku MATLABa komentarze poprzedza się znakiem `%`. Powoduje on, że teksty występujące po nim w wierszu nie są analizowane. Komentarze pełnią istotną rolę dokumentacyjną w przypadku funkcji i skryptów – pierwszy blok komentarzy jest wysyłany na ekran jako pomoc związana z danym skryptem.

przykład: skrypt sincos.m
 % przykładowy skrypt: sincos
 % generuje wektor liczb rzeczywistych t
 % i oblicza dla kolejnych wartości z tego wektora
 % wartości funkcji $x=\sin(t)+\cos(t)$

 % tego komentarza nie zobaczysz w helpie
 t=[1:0.05;4*pi];
 x=sin(t)+cos(t);
 %koniec
polecenie: >> help sincos.m

Funkcje są m-plikami, które przyjmują argumenty wejściowe oraz zwracają argumenty wyjściowe., operują one na zmiennych lokalnych, niedostępnych w przestrzeni roboczej MATLABa.

Definicję funkcji umieszcza się w skrypcie o nazwie identycznej z nazwą definiowanej funkcji i z rozszerzeniem .m. M-funkcja posiada następującą strukturę:

```
function wartości-funkcji=nazwa-funkcji(parametry)  
linia pomocy H1  
tekst pomocy  
ciało funkcji
```

Parametry i ciało funkcji są opcjonalne.

przykład: function [x,y,z]=funkcja(a,b,c,d)
 % funkcja coś licząca: [x,y,z]=funkcja(a,b,c,d)
 % obszerny komentarz

 for i=1:5,
 ...
 end

Zasady poprawnego programowania w MATLABie

- ♦ MATLAB jest językiem macierzowym, co oznacza że został zoptymalizowany pod kątem operacji na macierzach. Język MATLABa umożliwia efektywną realizację operacji macierzowych i tablicowych. Dobry program w języku MATLAB można poznać po dużej ilości operacji wektorowych oraz prawie zupełnym braku instrukcji for.

przykład: `for i=1:size(A,:); C(i)=A(i)*B(i); end`

znacznie szybciej: `C=A.*B`

przykład: porównanie efektywności pętli i obliczeń macierzowych
obliczanie wartości funkcji cos w punktach z przedziału <-10,10> co 0.001:

1. w pętli ze zwiększaniem macierzy wyników w każdym przebiegu
2. jw. z wcześniejszą rezerwacją macierzy
3. macierzowo

```
% test prędkości obliczeń
clear
t=-10:0.001:10;
[n,m]=size(t);
y(1,m)=0;
clear y;
% w pętli z powiększaniem wektora wyników
i=0;
t0=clock;
for x=t,
    i=i+1; y(i)=cos(x);
end
t1=etime(clock,t0);
% z utworzonym najsmwier wektorem wyników
i=0;
clear y;
y(1,m)=0;
t0=clock;
for x=t,
    i=i+1; y(i)=cos(x);
end
t2=etime(clock,t0);
% macierzowo
clear y;
t0=clock;
y=cos(t);
t3=etime(clock,t0);
[t1 t2 t3]
```

wyniki:	280	9.72	0.06	<- 486
	40.04	0.99	0.05	<- Pentium 266
	21.18	0.39	0.016	<- 2x Celeron 400

- ◆ Jeśli znany jest rozmiar macierzy wygenerowanej jako wynik, zaleca się wcześniejsze zarezerwowanie pamięci dla niej przez utworzenie tej macierzy przed wykonaniem operacji np. poleceniem **zeros** czy **ones**.

W ten sposób MATLAB nie potrzebuje zwiększać rozmiaru macierzy wynikowej po obliczeniu wartości kolejnego elementu,

- ◆ MATLAB umożliwia prace w dwóch trybach: interaktywnym (interpreter) wsadowym

Długie lub często wykonywane sekwencje poleceń można zapisać w tzw. m-pliku, czyli skrypcie. Jest to niesformatowany plik tekstowy zawierający instrukcje do wykonania. Jego wywołanie polega na wpisaniu nazwy pliku w linii poleceń.

Wśród m-plików wyróżnia się m-pliki funkcyjne (które zawierają funkcje). pliki funkcyjne operują na zmiennych lokalnych, zaś skryptowe – na globalnych.

- ◆ Lepiej jest pisać m-pliki funkcyjne niż skryptowe, gdyż te pierwsze operują na zmiennych lokalnych (nie przechowywanych w przestrzeni roboczej MATLABa), co chroni przed błędami spowodowanymi wprowadzaniem tych samych nazw dla różnych zmiennych.
- ◆ Własne m-pliki należy tak pisać, aby działały zarówno dla wielkości skalarnych, jak i macierzowych.
- ◆ We własnych m-plikach należy stosować komentarze.

przykład m-pliku:

```
% przykład m-pliku
% wykreslenie linii prostej i paraboli

echo off % wylaczenie echa
clc      % kasowanie ekranu
% dane dla linii prostej
xpr=[-1 1];      ypr=2*xpr;
% dane dla paraboli
xpar=[-1:0.1:1]; ypar=xpar.*xpar-1;
%wykres prostej i paraboli
plot(xpr,ypr,xpar,ypar)
%wlaczenie siatki
grid on
echo on % wlaczenie echa
```

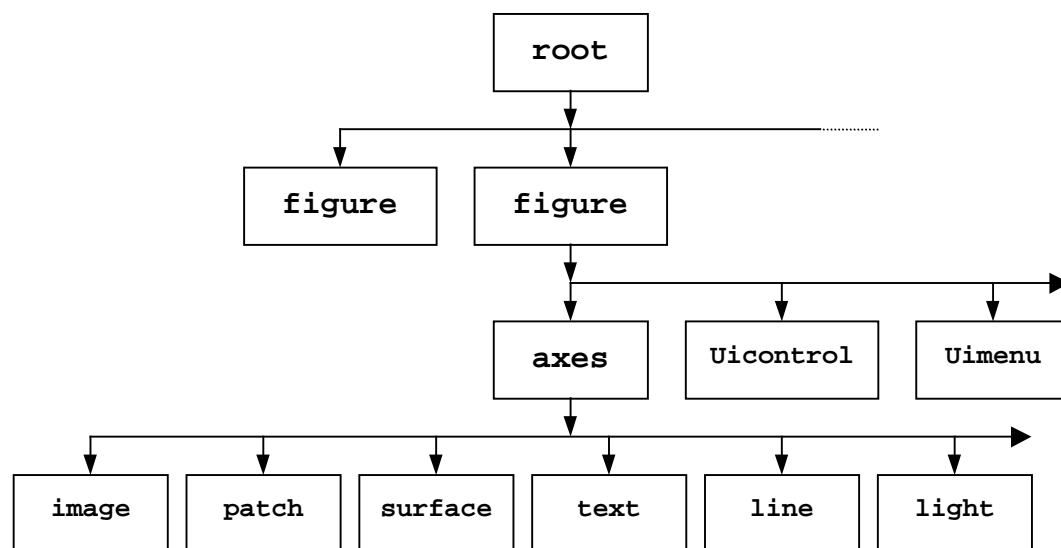
Grafika obiektowa

Poznane do tej pory funkcje pozwalają na tworzenie różnego rodzaju wykresów. Są to tak zwane funkcje wysokiego poziomu. Większość z nich tworzy gotowy rysunek ustalając automatycznie wiele parametrów takich jak kolor, rozmieszczenie elementów itp. Wszystkie te funkcje składają tworzone przez siebie rysunki z kilku podstawowych elementów: linii, wielokątów, układów współrzędnych.

Grafika w MATLABie jest obiektowo zorientowana i umożliwia modyfikowanie elementów już wykonanego rysunku.

Każdy fragment rysunku stanowi pewien obiekt graficzny. Obiekt ma przypisany swój unikalny identyfikator (*handle*), który jest po prostu liczbą rzeczywistą.

Każdy obiekt zawiera strukturę danych – rekord, w którym przechowywane są jego parametry i związane z nim dane. Rekord ten ma pola, w których można umieścić identyfikatory innych obiektów – przodków i potomków danego obiektu. Obiekt ma zawsze jednego przodka i może mieć dowolną liczbę potomków.



Korzeń – **root** odpowiada całemu ekranowi. Korzeń jest tylko jeden, nie ma przodka (pole zawierające listę przodków jest zawsze puste), a jego identyfikatorem jest liczba 0.

Potomkami korzenia są okna graficzne – rysunki **figures**. Przodkiem dla obiektu **Figure** jest **Root**.

Rysunki mają z kolei potomków w postaci układów współrzędnych **axes**, a także **uicontrol** (elementy interfejsu graficznego użytkownika – przyciski, okienka dialogowe, suwaki...) oraz **uimenu** (klasyczne menu, potomkami uimenu mogą być dalsze obiekty uimenu, których funkcje tworzą menu rozwijalne).

Potomkami układów współrzędnych Axes mogą być:

- linie `lines`
- powierzchnie `surfaces`
- teksty `text`
- obrazy `images`
- wycinki `patches`
- oświetlenie `light`

- tworzenie obiektów

Poleceniem utworzenia obiektu graficznego może być każde polecenie MATLABa, które otwiera okno graficzne bądź wykonuje dowolny fragment rysunku. Wymienione nazwy klas obiektów są identyczne z nazwami poleceń utworzenia danego obiektu. Rezultatem takiego polecenia, oprócz utworzenia nowego obiektu graficznego, jest przypisanie mu identyfikatora.

Każdy obiekt jest identyfikowany przez liczbę (*handle*). Jest ona przypisywana do obiektu w momencie jego utworzenia. Wtedy też identyfikator tego obiektu można przypisać dowolnej zmiennej.

zmienna = polecenie-utworzenia-obiektu-graficznego

funkcje:

- **`id=figure`** – tworzy nowy rysunek – okno graficzne i zwraca jego identyfikator, okno, którego wnętrze zajmuje nowo utworzony rysunek staje się oknem aktywnym, na które będą działały wszystkie następne operacje, jeżeli w ich argumentach nie zostanie podany identyfikator innego okna
- `id1=figure(id2)`** – jeżeli istnieje rysunek o identyfikatorze równym argumentowi, to stanie się on aktywnym rysunkiem, else zostanie utworzony
- **`id=axes('position',[lewy,dolny,szerokość,wysokość])`** – *position* określa pozycję w rysunku
- `axes(id)`**

- **id=findobj(*nazwa-własności1,wartość1,...*)**
 - znajduje obiekty o podanych własnościach,
- id=findobj(*lista-id,nazwa-własności1,wartość1,...*)**
 - szuka tylko w obrębie obiektów z listy identyfikatorów i ich potomków,
- id=findobj(*lista-id*)**
 - podaje wektor zawierający identyfikatory obiektu i jego potomków,
- id=findobj()**
 - podaje wektor zawierający identyfikator korzenia i jego potomków, czyli wszystkich istniejących obiektów.

- usuwanie obiektów

Każdy, oprócz korzenia, obiekt graficzny może zostać usunięty. Do usuwania obiektów (wraz z ich potomkami) służą funkcje **delete** oraz **close**. **delete** służy do usuwania obiektów dowolnego typu, **close** jest przeznaczona przede wszystkim do usuwania rysunków (zamykania okien), ale pozwala także usuwać obiekty innych typów.

- **delete(id)**
- **close**
close(id)

Parametry *id* mogą być wektorami.

- **clf** – czyści aktywny rysunek usuwając wszystkie leżące na nim, czyli będące jego potomkami, obiekty graficzne,
- clf reset** – dodatkowo przywraca domyślne wartości wszystkim, oprócz położenia, własnościom rysunku,
- **cla** – jw. w odniesieniu do układu współrzędnych,
- cla reset**

- odczytywanie i zmiana własności obiektów

Do odczytywania i zmiany własności obiektów służą funkcje **get** i **set**.

- **get(handle)** – odczytanie wszystkich własności
- get(handle,'nazwa-pola')** – jednej
- get(0)**
- get(0,'Screansize')**
- get(0,'Units')**
- get(0,'un')**

- **set(handle, 'nazwa-pola', wartość)**
`set(obiekt1, 'Color', 'g')`
`set(obiekt2, 'Color', 'g', 'LineWidth', 2)`
`set(obiekt3, 'LineStyle', '-', 'LineWidth', 2.0)`

Jedyna wartość, której nie można zmienić to *Parent*.

Wszystkim własnościom danego obiektu można przywrócić domyślne wartości używając funkcji **reset**. Ma ona tylko jeden argument – identyfikator obiektu.

-
- własności wspólne dla wszystkich typów obiektów:
 - **Type**
 - zawiera łańcuch będący nazwą typu obiektu,
 - wartość ustalana w momencie tworzenia obiektu,
 - możliwy tylko odczyt,
 - wartości: *root, figure, axes, line, patch, surface, text, image, uicontrol, uimenu*
 - **ButtonDownFcn** – może zawierać dowolny ciąg znaków stanowiący sekwencję poleceń systemu MATLAB,
 - polecenia zawarte we własności zostaną wykonane, jeśli użytkownik kliknie w czasie, gdy kursor jest nad danym obiektem,
 - polecenia zostaną wykonane szybciej, jeśli własność będzie zawierała nazwę pliku (skryptu)
 - **Clipping** – decyduje o tym, czy części obiektu znajdujące się poza obiektem-przodkiem będą wyświetlane, czy nie,
 - on* – wystające części będą obcinane
 - off* – będą wyświetlane
 - **Interruptible** – określa, czy wykonanie poleceń zawartych we własności *ButtonDownFcn* może zostać przerwane na rzecz wykonania poleceń innego obiektu,
 - yes*
 - no* – domyślnie
 - **Parent** – identyfikator przodka
 - **UserData** – dowolna macierz
 - **Visible** – określa, czy dany obiekt jest widoczny, tzn. czy ma być rysowany,
 - on* – domyślnie
 - off*

- własności różnych obiektów:

- **Root**

- `CurrentFigure` – identyfikator aktywnego rysunku
- `Diary` – czy prowadzić dziennik
(zapisywać wszystkie polecenia do pliku)
on / off
polecenie: `diary`
- `DiaryFile` – nazwa pliku dziennika
- `Echo` – czy wykonywane polecenia zawarte w plikach
mają być wyświetlane na ekranie
on / off
polecenie: `echo`
- `Format` – sposób wyświetlania liczb
short, long, shortE, longE, hex, bank, +, rat
polecenie: `format`
- `FormatSpacing` – format wyświetlania macierzy:
compact – wiersze jeden po drugim
loose – puste wiersze między wierszami
polecenie: `format`
- `PointerWindow` – identyfikator okna, w obrębie którego znajduje się kursor,
możliwy tylko odczyt
- `PointerLocation` – dwuelementowy wektor, którego elementami są
aktualne współrzędne ekranowe kursora myszki
(w jednostkach określonych przez własność *Units*)
- `ScreenDepth` – na ilu bitach kolor punktu
- `ScreenSize` – wektor czteroelementowy – wymiary ekranu w
jednostkach określonych przez *Units*:
wsp. *x* lewej krawędzi ekranu,
wsp. *y* dolnej krawędzi,
szerokość, wysokość
- `Units` – jednostki: *inches, centimeters*
normalized (ułamki wymiarów ekranu)
points (1/72 cala),
pixels (domyślnie)

- **Figure**

- `Color` – kolor tła
wektor RGB, można podać nazwę koloru
- `ColorMap` – macierz mapy kolorów
(*surface, image, patch*)

- `CurrentAxes` – identyfikator aktywnego układu współrzędnych
- `CurrentCharacter` – symbol ostatnio naciśniętego klawisza za panowania danego okna
- `CurrentMenu` – identyfikator elementu menu należącego do tego rysunku, który został uaktywniony jako ostatni
- `CurrentObject` – identyfikator obiektu w obrębie którego znajduje się *CurrentPoint*
- `CurrentPoint` – współrzędne aktualnego punktu rysunku (aktywny = z ostatniego kliknięcia)
własność można wykorzystać do znalezienia punktu, w którym ostatnio kliknięto myszką
- `InvertHardCopy` – *on* – wydruk na białym tle, lub *off*
- `KeyPressFcn` – akcja do wykonania, jeśli użytkownik naciśnie klawisz w momencie, gdy aktywnym oknem będzie okno danego rysunku
- `MenuBar` – sposób wyświetlania menu
- `MinColorMap` – liczba kolorów palety (domyślnie 64)
- `Name` – nazwa rysunku w pasku tytułowym
- `NumberTitle` – czy w nazwie ma się pojawić nr rysunku (*on/off*)
- `PaperUnits` – jednostki papieru: *normalized, points, inches, centimeters,*
- `PaperOrientation` – ułożenie wydruku rysunku na kartce: *portrait, landscape*
- `PaperSize` – szerokość i wysokość papieru
- `PaperType` – format papieru
- `Pointer` – kształt kursora myszki po wjechaniu na teren okna: *crosshair, arrow* (domyślnie), *topl, topr, botl, botr, circle, cross, fleur*
- `Position` – pozycja i wymiary danego okna
- `Resize` – czy można zmieniać wymiary okna myszką (*on/off*)
- `Units` – jednostki
- `WindowButtonDownFcn` – akcje do wykonania:
`WindowButtonMotionFcn` gdy naciśnięto / zwolniono przycisk myszki nad rysunkiem
`WindowButtonUpFcn` lub co jakiś czas w trakcie ruchu myszki nad rysunkiem

➤ **Axes**

- `AxesRatio` – proporcje osi układu współrzędnych
- `Box` – czy osie na czterech (*on*), czy dwóch krawędziach (*off*)
- `FontName` – czcionki tytułów, opisów osi, itp.
- `FontSize`
- `FontAngle` – *normal, oblique, italic*
- `FontStrikeThrough` – *on/off*
- `FontUnderline` – *on/off*
- `FontWeight` – *light, normal, demi, bold*
- `GridLineStyle` – - - : -.
- `LineWidth` – grubość osi
- `Nextplot` – gdzie umieścić następny wykres utworzony przy użyciu funkcji graficznych wysokiego poziomu:
new – w nowym układzie
add – na tle obecnego wykresu
replace – w miejsce tegoż
- `Position` – położenie i wymiary układu współrzędnych wewnątrz rysunku
- `Units` – jednostki
- `TickLength` – długość kresek na osiach
- `TickDir` – *in* – wewnątrz, *out* – na zewnątrz
- `Title`
- `View` – dwuelementowy wektor, kąt pod jakim obserwator widzi układ współrzędnych
funkcja: `view`
- `Xcolor, Ycolor, ZColor` – kolory osi
- `XDir, YDir, ZDir` – zwroty osi: *normal/reverse*
- `XGrid, YGrid, ZGrid` – linie siatki: *on/off*
- `XLim, YLim, ZLim` – granice przedziałów przedstawianych na osiach
- `XScale, YScale, ZScale` – skale osi: *linear/log*

➤ **Line**

- `Color`
- `LineStyle` – - + -- o : * -. x
- `LineWidth`

przykład:

```
t=2/3:1/6:.7*pi;
y=sin(t);
[a,b,c]=cylinder(y)      % dane do narysowania bryły
subplot(1,2,1)
h1=surf(a,b,c)
title('zwykly flakon')
set(h1,'facecolor','y')  % żółty
subplot(1,2,2)
h2=surf(a,b,c)
set(h2,'facelighting','phong','backfacelight','lit') % odblaski
set(h2,'facecolor','y')
light('position',[-2,-3,.6]) % obiekt: źródło światła
title('flakon odbijajacy swiatlo')
```


Graficzny system komunikacji z użytkownikiem

Zaprojektowanie graficznego systemu komunikacji z użytkownikiem GUI nie jest trudne. Wymaga poznania kilku obiektów GUI oraz najważniejszych reguł programowania obiektowego.

Elementy komunikacyjne dostępne w MATLABie:

♦ przycisk	<i>button</i>
♦ wyłącznik	<i>check box</i>
♦ przełącznik	<i>radio button</i>
♦ lista rozwijalna	<i>pop-up menu</i>
♦ pole edycyjne	<i>editable text</i>
♦ suwak	<i>scroll bar</i>
♦ ramka	<i>frame</i>
♦ pole tekstowe	<i>static text</i>

Odrębnymi prawami rządzą się elementy systemu zwane menu.

- tworzenie obiektów

Lista funkcji służących do posługiwania się graficznym systemem komunikacji z użytkownikiem jest bardzo skromna – ogranicza się do dwóch funkcji, ale wystarcza całkowicie do tworzenia elementów i łączenia ich z odpowiednimi procedurami.

Do modyfikacji parametrów utworzonych już elementów służą funkcje `set` i `get`.

- **`id=uicontrol(nazwa-własności1,wartość1,...)`**
`id=uicontrol(idf,nazwa-własności1,wartość1,...)`
 - służy do tworzenia przycisków, przełączników, wyłączników, suwaków, pól edycyjnych, ramek i pól tekstowych,
 - pierwszym elementem może być identyfikator rysunku (okna), wewnątrz którego ma się pojawić obiekt, jeżeli nie jest podany, obiekt zostanie utworzony w aktywnym rysunku,
 - wszystkie elementy, które można utworzyć, mają taką samą listę własności, jednak ich znaczenie może być różne.
 - przykład: `uicontrol('style','pushbutton',...
'position',[10 10 100 20],...
'string','Czyść',...
'callback','cla')`
tworzy w lewym dolnym rogu aktywnego okna przycisk z napisem *Czyść*, którego naciśnięcie powoduje wyczyszczenie aktywnego wykresu

▪ **`id=uimenu(nazwa-własności1,wartość1,...)`**

`id=uimenu(idf,nazwa-własności1,wartość1,...)`

- służy do tworzenia elementów menu głównego oraz elementów podmenu, każde wywołanie funkcji powoduje utworzenie pojedynczego wiersza podmenu lub jednej opcji menu
- pierwszym elementem może być identyfikator rysunku (okna) lub innego elementu menu bądź podmenu, jeżeli parametrem tym będzie identyfikator rysunku, to tworzony element zostanie włączony do menu głównego tego rysunku, jeżeli pierwszym parametrem będzie identyfikator istniejącego elementu menu to element tworzony zostanie umieszczony w podmenu związanym z elementem, którego identyfikator został podany jako pierwszy parametr, pominięcie pierwszego elementu (identyfikatora *idf*) spowoduje umieszczenie tworzonego elementu w głównym menu aktywnego rysunku,
- przykład:

```
idm1=uimenu('Label','E&xtra'); % opcja w menu głównym
uimenu(idm1,'Label','&1',... % pierwsza opcja menu rozwijanego
        'Callback','subplot(1,1,1)')
idm2=uimenu(idm1,'Label','&2'); % druga opcja rozwijanego
idm3=uimenu(idm1,'Label','&4'); % trzecia opcja rozwijanego
uimenu(idm2,'Label','&1/2','Callback','subplot(2,1,1)');
uimenu(idm2,'Label','&2/2','Callback','subplot(2,1,2)');
uimenu(idm3,'Label','&1/4','Callback','subplot(2,2,1)');
uimenu(idm3,'Label','&2/4','Callback','subplot(2,2,2)');
uimenu(idm3,'Label','&3/4','Callback','subplot(2,2,3)');
uimenu(idm3,'Label','&4/4','Callback','subplot(2,2,4)');
uimenu(idm1,'Label','&Czyść','Callback','cla','separator','on');
```

 tworzy w aktywnym oknie dodatkową opcję menu pozwalającą podzielić okno na 2 lub 4 układy współrzędnych oraz czyścić aktywny układ

▪ **`menu(tytuł,opcja1,opcja2,opcja3,...)`**

- pozwala na błyskawiczne tworzenie samodzielnych menu,
- wywołanie funkcji powoduje utworzenie nowego rysunku, na którym jeden pod drugim umieszczone są przyciski z nazwami kolejnych opcji, naciśnięcie dowolnego przycisku powoduje zamknięcie okna i zakończenie działania funkcji,
- funkcja zwraca numer wybranej funkcji,
- argumentami funkcji są łańcuchy znaków zawierające tytuł menu oraz nazwy kolejnych opcji.

• własności elementów niezależnych:

- **Style** – określa typ elementu,
 - wartości: *pushbutton* – przycisk
 - radiobutton* – przełącznik
 - checkbox* – wyłącznik
 - edit* – pole edycyjne
 - text* – pole tekstowe
 - slider* – suwak
 - frame* – ramka
 - popupmenu* – lista rozwijana

- **Callback** – określa akcję podejmowaną po uaktywnieniu elementu,
 - wartością własności może być ciąg znaków zawierający dowolne polecenie systemu MATLAB; ciąg poleceń, nazwę funkcji, zmiennej lub skryptu,
 - kiedy element jest uaktywniany, MATLAB traktuje ten ciąg tekstowy jako argument funkcji *eval*,
 - ponieważ wykonanie tych poleceń odbywa się w przestrzeni roboczej, więc mnoga się one odnosić do zmiennych zdefiniowanych w tej przestrzeni,
 - akcja jest wykonywana najszybciej, jeśli łańcuch zawiera tylko nazwę skryptu zamiast sekwencji poleceń,
 - akcja jest wykonywana w przypadku:
 - naciśnięcia przycisku
 - zmiany stanu przełącznika lub wyłącznika
 - wybraniu pozycji z rozwijalnej listy
 - zakończenia edycji tekstu
- **Position**
- **Units**
- **String** – zawiera łańcuch będący etykietą elementu,
 - dla pól edycyjnych zawiera edytowany tekst
 - w liście rozwijalnej zawiera elementy listy (jeden ciąg znaków, przedzielone |),
- **Value** – zawiera liczbę określającą aktualny stan elementu,
 - modyfikowana automatycznie, ale można też zmienić przez *set*,
 - przyciski mają wartość *Value* równą własności *Min*, w chwili naciśnięcia *Max*, po zakończeniu wykonywania poleceń w *Callback* – *Min*,
 - przełączniki i wyłączniki mają wartość równą własności *Min* w stanie wyłączonym i *Max* w stanie włączonym,
 - własność *Value* suwaków przyjmuje wartość z przedziału określonego przez *Min* i *Max*
 - własność *Value* listy rozwijanej zawiera nr wybranej pozycji
- **Min,Max** – domyślnie: 0 i 1
- **BackgroundColor,ForegroundColor**

-
- własności elementów menu:
 - **Label** – etykieta
 - **Callback**
 - **Checked** – decyduje o pojawieniu się ptaszka na lewo od nazwy: *on/off*
 - **Enable** – decyduje o dostępności danego elementu w menu
 - **Separator** – *on/off*
 - **BackgroundColor,ForegroundColor**
 - **Position** – pozycja elementu wewnątrz menu
-

przykłady:

```
%Usuń wszystkie rysunki
close all;

%Narysuj jakiś wykres 3D
[x,y]=meshgrid(-2:0.2:2);
z=exp(-x.^2-y.^2);
id=surf(x,y,z);

%Opisz osie
xlabel('x');
ylabel('y');
zlabel('z');

%Na wszelki wypadek - widoczek trójwymiarowy
view(3);

%Suwak poziomy
uicontrol(gcf,...
    'style','slider',...
    'units','normalized',...
    'position',[0.09 0.97 0.90 0.03],...
    'max',180,'min',-180,...
    'value',-get(gca,'view')*[1 0]',...
    'callback',...
    'set(gca','view',get(gca,'view').*[0 1]-[get(gco,'value') 0])));
%% funkcja callback powoduje zmianę azymutu punktu obserwacji
%% na wartość wybraną suwakiem

%Suwak pionowy
uicontrol('style','slider',...
    'units','normalized',...
    'position',[0 0.02 0.02 0.9],...
    'max',90,'min',-90,...
    'value',-get(gca,'view')*[1 0]',...
    'callback',...
    'set(gca','view',get(gca,'view').*[1 0]-[0 get(gco,'value')])));
%% funkcja callback powoduje zmianę elewacji punktu obserwacji
%% na wartość wybraną suwakiem
```

```
%Przycisk zmniejszania
uicontrol(gcf,...
    'style','push',...
    'units','normalized',...
    'string','- ',...
    'position',[0.03 0.97 0.03 0.03],...
    'callback',...
    ['set(gca, 'xlim', get(gca, 'xlim')*2, ', ...
        'ylim', get(gca, 'ylim')*2, ', ...
        'zlim', get(gca, 'zlim')*2)']);
%% funkcja callback powoduje dwukrotne zmniejszenie zakresów
%% na osiach x,y i z

%Przycisk powiększania
uicontrol(gcf,...
    'style','push',...
    'units','normalized',...
    'string','+ ',...
    'position',[0 0.94 0.02 0.03],...
    'callback',...
    ['set(gca, 'xlim', get(gca, 'xlim')/2, ', ...
        'ylim', get(gca, 'ylim')/2, ', ...
        'zlim', get(gca, 'zlim')/2)']);
%% funkcja callback powoduje dwukrotne zwiększenie zakresów
%% na osiach x,y i z
```