Laboratorium z Metod Numerycznych I

Kurs MATLABa - zajęcia w semestrze letnim 2005/2006

Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

opracował:

Łukasz Wojciechowski

Zajęcia nr 1:

Organizacja i regulamin zaliczania zajęć:

I. ZALICZENIE LABORATORIUM

Pierwszych siedem zajęć poświęconych jest pakietowi Matlab (Octave). Zajęcia obejmują zapoznanie się ze środowiskiem, poznanie podstawowych funkcji i naukę programowania.
Obecność na tych zajęciach jest **obowiązkowa**.

Każdy student (grupa studentów) otrzymuje w ciągu semestru 2 tematy projektów.
Każdy projekt składa się z dwóch części: teoretycznej i praktycznej.
Część teoretyczna obejmuje: opis użytych metod numerycznych i opis algorytmu oraz dobór przykładów.
Część praktyczna polega na napisaniu i uruchomieniu programu obliczeniowego na maszynie cyfrowej oraz na analizie wyników.

Podstawą do zaliczenia projektu są: sprawozdanie, program obliczeniowy i ustna rozmowa (obejmująca również teorię).

Sprawozdanie powinno zawierać następujące punkty:
1. Imię i nazwisko studenta (studentów)
2. Numer tematu i treść zadania
3. Opis metody (krótki)
4. Opis programu obliczeniowego (sposobu obsługi programu)
5. Przykłady obliczeniowe (2 . 4 przykłady)
6. Analiza wyników.
7. Kody źródłowe wszystkich procedur.

Programy oceniane są przede wszystkim z punktu widzenia metod numerycznych, jednak sposób zaprogramowania oraz walory użytkowe programu mają również wpływ na ocenę.

Programy obliczeniowe należy wykonać w pakiecie Matlab (Octave). W szczególnych przypadkach na prośbę studenta może być dopuszczone przygotowanie programu w innym języku (jednak dostępnym na sprzęcie w Laboratorium Informatyki Wydziału MiNI).

Każdy projekt powinien być zaliczony w okresie ustalonym w harmonogramie zajęć. Po przekroczeniu tego terminu maksymalna liczba punktów jest obniżana za pierwszy oraz drugi tydzień opóźnienia.
W przypadku wcześniejszego zaliczenia zadania prowadzący zajęcia może dać zadania następne.
W szczególnych przypadkach (choroba, awaria sprzętu) prowadzący może przesunąć termin zaliczenia (indywidualnie lub dla całej grupy).

Z obydwu tematów projektów **można zdobyć po 20 punktów**. W tym:

- 9 p. za program obliczeniowy
- 4 p. za sprawozdanie
- 4 p. za znajomość teorii z zakresu zadania
- 3 p. za przykłady obliczeniowe

Po tygodniu opóźnienia obowiązuje następująca punktacja:
8p. + 3p. + 3p. + 2p. = 16p.

Po dwóch tygodniach opóźnienia obowiązuje punktacja:
7p. + 2p. + 2p. + 1p. = 12p.

UWAGA: Nieprawidłowo działający program, brak sprawozdania lub nieznajomość teorii powodują
przesunięcie terminu zaliczenia o 1 tydzień.

Poza punktami za projekty student może otrzymywać punkty za aktywność na zajęciach laboratoryjnych (5 zadań za 1 punkt i jeden test za 5 punktów).
Maksymalnie można za to zdobyć do 10 punktów.

II. ZAKRES TEMATÓW PROJEKTÓW OBLICZENIOWYCH

TEMAT I

1. Układy równań liniowych.

TEMAT II

1. Interpolacja.
2. Aproksymacja średniokwadratowa.
3. Wyznaczanie zer funkcji.
4. Całkowanie numeryczne.

III. ZALICZENIE PRZEDMIOTU

Zaliczenie przedmiotu (wykład + ćwiczenia + laboratorium) polega na zdobyciu w sumie co najmniej 51 punktów. Nie ma obowiązku uzyskania oceny pozytywnej z każdego zadania laboratoryjnego.

Maksymalnie można zdobyć 100 punktów, w tym 50 punktów z ćwiczeń (2 kolokwia po 25 p.) oraz 50 punktów z laboratorium (2 projekty po 20p. + 10p. za pracę na zajęciach).

Ocena ostateczna wystawiana będzie zgodnie z poniższą regułą:

51-60 p. . 3.0
61-70 p. . 3.5
71-80 p. . 4.0
81-90 p. . 4.5
91-100 p. . 5.0

Obecność na ćwiczeniach i zajęciach laboratoryjnych poświęconych pakietowi Matlab jest obowiązkowa. Dopuszczalna liczba nieobecności na powyższych zajęciach (laboratoria + ćwiczenia) **w sumie wynosi 4.**

IV. HARMONOGRAM ZALICZANIA LABORATORIUM

ROZDANIE ZADAŃ:	TERMIN ZALICZENIA:
Projekt nr 1	
Projekt nr 2	

Wprowadzenie i pierwsze kroki w MATLAB / Octave

• MATLAB - pakiet obliczeniowy firmy MathWorks przeznaczony do:

wykonywania różnorodnych obliczeń numerycznych

implementowania różnych algorytmów

obróbki i wizualizacji danych

modelowania i symulacji

tworzenia wykresów i grafiki naukowej

• Podstawowym typem danych jest macierz, stąd nazwa **MAT**rix **LAB**oratory.

• Pakiet posiada obszerne biblioteki dodatkowych procedur umożliwiające rozwiązywanie typowych problemów obliczeniowych.

• Prosta budowa okienkowa ułatwia korzystanie z programu.

Dokumentacja:

1. strony:

www.mathworks.com

www.octave.org

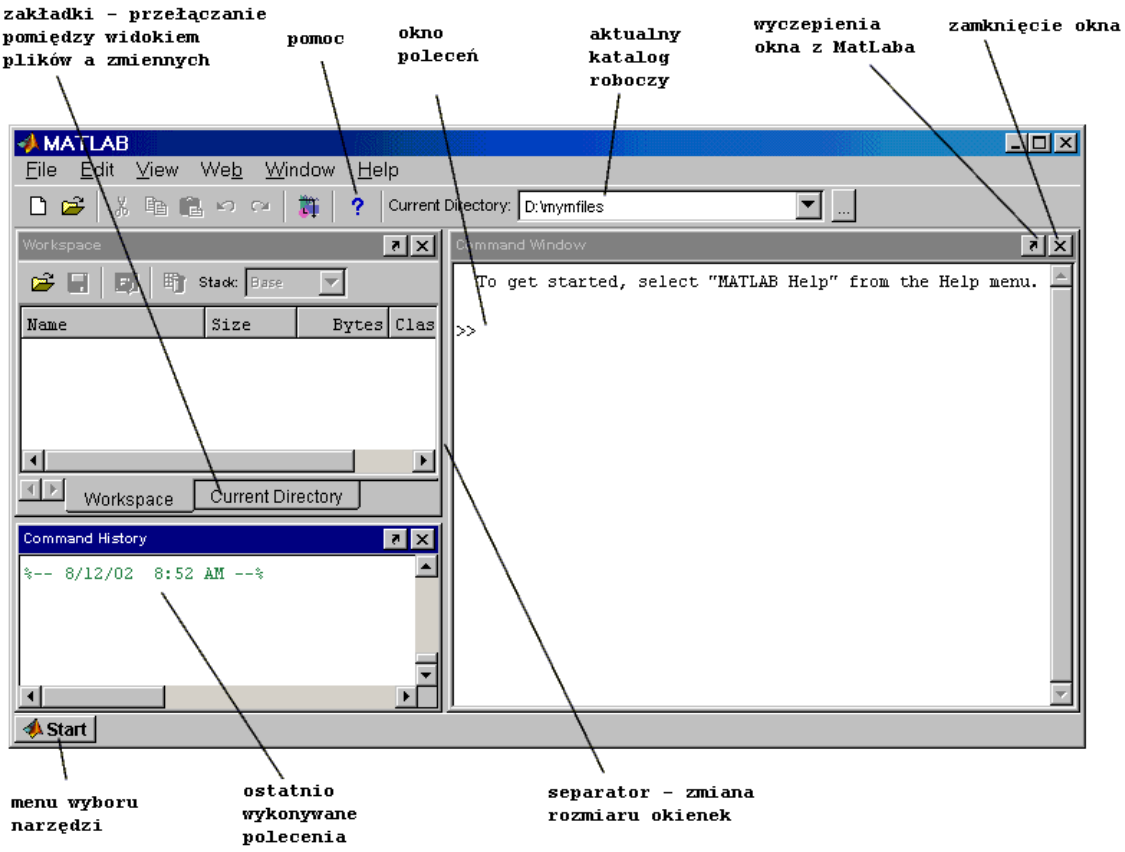
2. help w Matlabie

Uruchamianie:

polecenie `matlab` w konsoli

Zamykanie:

polecenie `exit` w command window



Okna:

okno ‘command window’ - wykonywanie poleceń

okno historii - zawiera ostatnio wykonywane polecenia

Kolejne sesje odzielone są datami. Polecenia te można wykorzystać jeszcze raz, klikając na nie. Zaznaczając jedno lub kilka poleceń można utworzyć z nich plik skryptowy (m-file) lub wykonać kilka z nich.

przycisk start - zawiera szybki dostep do poszczególnych modułów Matlaba

zakładka ‘workspace’ - zawiera aktualnie istniejące w przestrzeni roboczej zmienne wraz z informacją o typie, rozmiarze danych i całkowitej zajmowanej pamięci.

Dane można edytować dwuklikając na nie i zmieniając wartości w specjalnym edytorze.

zakładka ‘current directory’ - zawiera pliki znajdujące się w bieżącym katalogu. Można je stąd otwierać, uruchamiać, kasować, a także zmieniać ich nazwy.

Dodatkowe okna:

Pomoc - omawiany poniżej w dziale pomoc.

Edytor - omawiany na drugich zajęciach przy okazji nauki budowania plików skryptowych (m-file). Warto wspomnieć, że wszystko co można wpisać „ręcznie" do Matlaba da się umieścić w pliku skryptowym i wykonać automatycznie (drugi tryb pracy)

Pomoc:

`help`	- lista tematów pomocy
`help syntax`	- pomoc na temat podstaw składni
`help aaa`	- pomoc na temat procedury lub funkcji aaa
`help katalog`	- pomoc na temat wszystkich m-plików z danego katalogu
`help -`	- pomoc na temat operatorów
`helpwin`	- lista tematów pomocy w interfejsie graficznym
`helpdesk`	- otwiera pomoc dot. Matlaba
`demo`	- otwiera zakładkę demos w pomocy Matlaba
`intro`	- przykładowy plik wyliczający kształt logo Matlaba, rysujący je oraz pokazujący kilka przykładowych obliczeń.

Pomoc można też otworzyć z menu Help.

Okno pomocy ma po lewej stronie 5 zakładek:

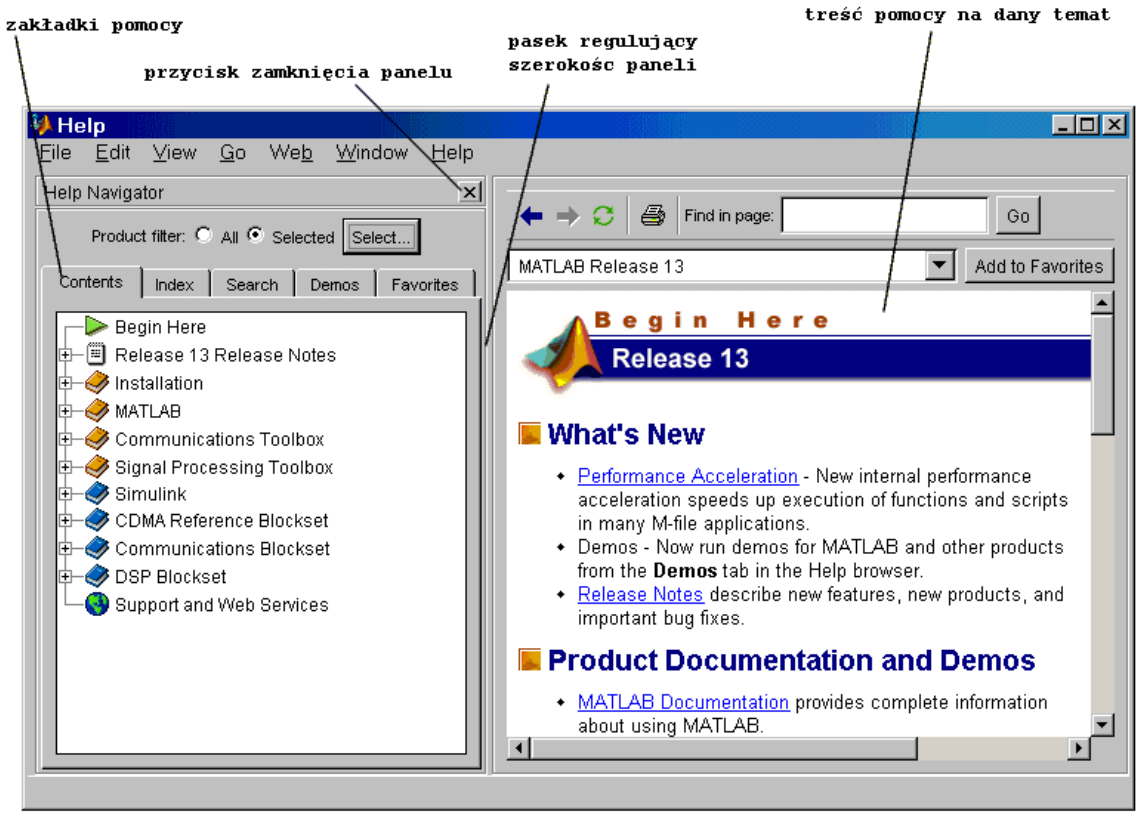
contents - tematy pomocy ułożone tematycznie

index - indeks alfabetyczny funkcji i procedur wbudowanych

search - wyszukiwarka

demos - wbudowane dema Matlaba

favorites - ulubione tematy


zakładki pomocy
przycisk zamknięcia panelu
pasek regulujący szerokośc paneli
treść pomocy na dany temat
Help
File Edit View Go Web Window Help
Help Navigator
Product filter: All Selected Select...
Contents Index Search Demos Favorites
Begin Here
Release 13 Release Notes
Installation
MATLAB
Communications Toolbox
Signal Processing Toolbox
Simulink
CDMA Reference Blockset
Communications Blockset
DSP Blockset
Support and Web Services
Find in page:
Go
MATLAB Release 13
Add to Favorites
Begin Here
Release 13
What's New
Performance Acceleration - New internal performance acceleration speeds up execution of functions and scripts in many M-file applications.
Demos - Now run demos for MATLAB and other products from the Demos tab in the Help browser.
Release Notes describe new features, new products, and important bug fixes.
Product Documentation and Demos
MATLAB Documentation provides complete information about using MATLAB.

Wyrażenia w MATLAB / Octave:

Działania opierają się na operowaniu liczbami, zmiennymi, funkcjami i łączącymi je operatorami.

Liczby:

```
3                   -99                 0.0001
9.6397238           1.60210e-20         6.02252e23
1i                  -3.14159j           3e5i
```

wewnętrznie przechowywane jako long float zgodnie ze specyfikacją IEEE
(16 cyfr znaczących zakres wykładnika e-308, e+308).

Aby zmienić format w jakim wyświetlania należy posłużyć się poleceniem

`format`

`format` - domyślnie, to samo co `format short`

`format short` - przeskalowana ze stałym punktem 5 cyfr znaczących

`format long` - przeskalowana ze stałym punktem 15 cyfr znaczących

`format short e` - notacja naukowa 5 cyfr znaczących

`format long e` - notacja naukowa 15 cyfr znaczących

`format short g` - notacja naukowa lub zwykła (lepiej pasująca) 5 cyfr znaczących

`format long g` - notacja naukowa lub zwykła (lepiej pasująca) 15 cyfr znaczących

`format hex` - reprezentacja heksadecymalna

`format +` - format dający:

- `+` gdy liczba dodatnia
- `-` gdy liczba ujemna
- nic gdy liczba równa zero

`format bank` - 2 miejsca po przecinku

`format rat` - próba aproksymacji ilorazem małych liczb całkowitych

Na potrzeby pierwszych zajęć - proste budowanie macierzy:

```
A = [ 1, 2.4, 4, 6; 2, 5, 0, 3; 2i, 3, 2, 0]
```

Przecinki oddzielają elementy, a średniki przechodzą do następnej linii.

Część urojoną liczb zespolonych oznacza się przez `i` lub `j`.

Zmienne:

- nie wymagają deklaracji ani określania rozmiaru
- tworzone w momencie użycia - Matlab sam określa wielkość potrzebnej pamięci i w razie potrzeby doallokowuje odpowiedni obszar (ważne dla zajęć 4.)
- zmienne zaczynają się od litery po której następuje dowolny ciąg liter, cyfr i znaków podkreślenia _
- Matlab rozróżnia małe i wielkie litery
- Matlab rozróżnia jedynie 31 pierwszych znaków w nazwie zmiennej
- operatorem przypisania do zmiennej jest =

Zmienne specjalne:

`ans` - domyślna nazwa zmiennej

`pi` - stosunek obwodu do średnicy okręgu, $\pi = 3.1415926...$

`eps` - najmniejsza liczba o jaka mogą różnić się dwie liczby

`inf` - nieskończoność np. 1/0

`Inf`

`nan` - not a number, czyli nieliczba np 0/0

`NaN`

`date` - aktualna data

`i` - jednostka urojona

`j`

`realmin` - najmniejsza dodatnia liczba 2^{-1022}

`realmax` - największa liczba $(2-eps)*2^{1023}$

Uwagi na temat wprowadzania danych:

- postawienie średnika na końcu linijka spowoduje brak wypisania wyniku (ale operacje się wykonają)
- do łamania za długich linii służy operator `...`
- w jednej linijce można pisać wiele poleceń rozdzielając je przecinkami lub średnikami (pierwsze wypisują wynik, drugie nie)

Polecenia związane ze zmiennymi i ekranem:

- `who` - wyświetla listę zdefiniowanych zmiennych
- `whos` - wyświetla listę zdefiniowanych zmiennych, ich rozmiar i typ
- `clear` - usuwa wszystkie zmienne
- `clear aaa` - usuwa zmienna aaa
- `clc` - czyści ekran
- `clf` - czyści ekran graficzny (figure) (przyda się zajęcia 5,6)
- `more on` - włącza tryb more (wynik podzielony na ekrany, enter przejście o linijkę niżej, spacja przejście ekran niżej
- `more off` - wyłącza tryb more

Pamiętnik:

- `diary aaa` - zaczyna rejestrować pamiętnik w pliku aaa w bieżącym katalogu
- `diary off` - zamyka pamietnik (kończy rejestracje)
- `diary on` - otwiera ostatnio używany pamiętnik - jeśli takiego brak otwiera pamiętnik o nazwie diary w katalogu lokalnym
- `diary` - zmienia stan pamiętnika na przeciwny

Liczby zespolone:

- `abs(z)` - wartość bezwzględna liczby z
- `angle(z)` - kąt liczby zespolonej z (z = r*cos(kąt) + r*i*sin(kąt))
- `conj(z)` - liczba zespolona sprzężona do z
- `imag(z)` - część urojona liczby z
- `real(z)` - część rzeczywista liczby z

Operatory (w kolejności priorytetów, wszystkie lewo-łączne):

- `(  )`
- `.'  .^  '  ^`
- `+1  -1  ~`
- `.*  ./  .\  *  /  \`
- `+  -`
- `:`
- `<  <=  >  >=  ==  ~=`
- `&`
- `|`
- `&&`
- `||`

Uwagi:

`\`	to operator odwrotnego dzielenia np. `2 \ 8 = 4`
`'`	sprzężenie zespolone
`~`	negacja logiczna
`+1,-1`	czyli + i - unarne
`:`	specjalny operator wyliczeniowy (patrz zajęcia nr 2)
`&,|`	operatory logiczne (liczą oba wyrażenia)
`&&,||`	szybkie operatory logiczne (liczą aż poznają wynik - jak w C)
`.*,./.\.^`	operatory macierzowe działające na pojedynczych elementach

Funkcje - są to wywołania m-plików funkcyjnych. Mogą pobierać wiele argumentów, zwracać wiele wartości (jako wektor) i dokonywać złożonych obliczeń, tworzyć wykresy, wypisywać komunikaty, etc.
Więcej o m-plikach funkcyjnych na zajęciach 3.

Funkcje trygonometryczne:

`sin(x)`	
`cos(x)`	
`tan(x)`	
`cot(x)`	
`asin(x)`	x z przedziału [-1,1] zwraca wartość z zakresu [-pi/2,pi/2]
`acos(x)`	x z przedziału [-1,1] zwraca wartość z zakresu [0, pi]
`atan(x)`	zwraca wartość z zakresu [-pi/2, pi/2]
`atan2(y,x)`	zwraca wartość z przedziału [-pi,pi]
`acot(x)`	

Funkcje trygonometryczne hiperboliczne:

`sinh(x)`	(e^x - e^-x) / 2
`cosh(x)`	(e^x + e^-x) / 2
`tanh(x)`	(e^x - e^-x) / (e^x + e^-x)
`asinh(x)`	ln(x + sqrt(x^2 + 1))
`acosh(x)`	ln(x + sqrt(x^2 - 1))
`atanh(x)`	ln((1 + x) / (1 - x)) / 2

Funkcje wykładnicze i logarytmiczne:

`exp(x)`	e^x
`log(x)`	logarytm naturalny z x dla zespolonych: log(abs(x)) + i * atan2(image(x), real(x)
`log2(x)`	logarytm przy podstawie 2 z x
`[m,w] = log2(x)`	m.*2.^w=x, 0<=m<1
`log10(x)`	logarytm dziesiętny
`pow2(x)`	2^x
`reallog(x)`	logarytm naturalny (dla zespolonych i ujemnych generuje błąd)
`realpow(x,y)`	x.^y dla argumentó rzeczywistych
`realsqrt(x)`	sqrt(x), generuje błąd dla ujemnych i zespolonych wartości

`sqrt(x)`	wyciąga pierwiastek z liczby

Zaokrąglenia:

`fix(x)`	zaokrągla w stronę 0
`rem(x,y)`	reszta z dzielenia x przez y (x = y * fix(x / y) + rem(x , y))
`floor`	zaokrągla w stronę -Inf
`mod`	moduł po dzieleniu x przez y (x = y * floor(x / y) + mod(x , y))
`ceil`	zaokrągla w stronę Inf
`round`	zaokrągla do najbliższej liczby całkowitej
`sign`	znak (-1, 0, 1)

Matematyka dyskretna:

`factor(n)`	rozkład na czynniki pierwsze (zwraca wektor)
`factorial(n)`	n!
`gcd(a,b)`	greatest common divisor
`[c,d,e] = gcd(a,b)`	a .* d + b .* e = c, gdzie c największy wspólny dzielnik
`isprime(a)`	czy liczba pierwsza
`lcm(a,b)`	least common multiple
`nchoosek(n,k)`	symbol Newtona
`nchoosek(v,k)`	gdzie v forma o długości n, zwraca macierz z wszystkimi kombinacjami elementów
`perms(v)`	gdzie v forma, zwraca wszystkie permutacje
`primes(n)`	generuje wszystkie liczby pierwsze mniejsze od n
`[a,b] = rat(x)`	a ./ b daje dobrą ilorazową aproksymację x
`rats(x)`	wywołuje rat i zwraca string (o stringach więcej na zajęciach 4.)

Zajęcia nr 2:

Praca z macierzami i elementy algebry liniowej w Matlab / Octave:

Definiowanie macierzy:

1. wpisywanie

```
>>A=[1,2,4,8,5],B=[3;5;8],C=[1,2,3,7;2,7,1,6;4,8,2,0]
A =
     1     2     4     8     5
B =
     3
     5
     8
C =
     1     2     3     7
     2     7     1     6
     4     8     2     0
```

Uwagi:

, oddziela elementy w wierszu

; oddziela wiersze

2. łączenie

```
>>D=[A;[B,C]]
D =
```

1	2	4	8	5
3	1	2	3	7
5	2	7	1	6
8	4	8	2	0

3. zmiana

```
>> C(1,2)=-5
C =
     1    -5     3     7
     2     7     1     6
     4     8     2     0
```

Uwagi:

indeksowanie od 1

najpierw rząd potem kolumna

4. dopisanie

```
>> A(8)=3
A =
     1     2     4     8     5     0     0     3
```

Uwagi:

Miejsca puste wypełniane są zerami.

Sposób mało efektywny - powoduje przydzielenie nowej pamięci i przekopiowanie starego obiektu.

5. operator :

```
>> E=0:0.1:1
E =
    0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.0
```

Uwagi:

o:d tworzy wektor (poziomy) zawierający wartości:
o, o+1, o+2,... , o+n, gdzie $n = \max_a(o + a) <= d$

o:k:d tworzy wektor (poziomy) zawierający wartości:
o, o+k, o+2*k,... , o+n*k, gdzie $n = \max_a(o + a * k) <= d$

6. specjalne funkcje:
    a. `linspace(od,do,ile)` - tworzy poziomy wektor zawierający `ile` liczb rozłożonych równomiernie liniowo od wartości `od` do wartości `do` Można używać w wersji dwuparametrowej, domyślna wartość ile to 100.
    b. `logspace(od,do,ile)` - tworzy poziomy wektor zawierający `ile` liczb rozłożonych równomiernie w skali logarytmicznej od wartości 10^{od} do wartości 10^{do}. Można używać w wersji dwuparametrowej, domyślna wartość ile to 50.
    c. `eye(n)` - tworzy macierz jednostkową o rozmiarze n na n
       `eye(m,n)` - tworzy macierz m na n z 1 na głównej przekątnej
       `eye(size(A))` - -//- o takich samych rozmiarach jak A
    d. `ones(n)` - tworzy macierz n na n złożoną z samych 1
       `ones(m,n)` - tworzy macierz m na n złożoną z samych 1
       `ones([m,n])` - tworzy macierz m na n złożoną z samych 1
       `ones(d1,d2,d3,...)` - tworzy macierz d1 na d2 na d3 na ... złożoną z samych 1
       `ones([d1,d2,d3,...])` - tworzy macierz d1 na d2 na d3 na ... złożoną z samych 1
       `ones(size(A))` - tworzy macierz o rozmiarach takich samych jak A, złożoną z samych 1
    e. `zeros(n)` - tworzy macierz n na n złożoną z samych 0
       `zeros(m,n)` - tworzy macierz m na n złożoną z samych 0
       `zeros([m,n])` - tworzy macierz m na n złożoną z samych 0
       `zeros(d1,d2,d3,...)` - tworzy macierz d1 na d2 na d3 na ... złożoną z samych 0
       `zeros([d1,d2,d3,...])` - tworzy macierz d1 na d2 na d3 na ... złożoną z samych 0
       `zeros(size(A))` - tworzy macierz o rozmiarach takich samych jak A,

złożoną z samych 0

f. `diag(v)` - tworzy macierz diagonalną o elementach na głównej przekątnej odpowiadającym elementom wektora v

`diag(v,k)` - tworzy macierz o elementach na k-tej przekątnej odpowiadającym elementom wektora v. k-ta przekątna liczona jest od głównej przekątnej macierzy. Dla 0 jest to główna przekątna dla k>0 k-ta przekątna nad główną, a dla k<0 -k-ta przekątna pod główną.

Uwaga:

Polecenie `diag` służy też do wyłuskiwania przekątnych z macierzy do wektora pionowego:

`diag(M)` - zwraca pionowy wektor o elementach przekątnej głównej macierzy M

`diag(M,k)` - zwraca pionowy wektor o elementach k-tej przekątnej macierzy M

g. `rand(n)` - tworzy macierz n na n wypełnioną elementami losowymi o rozkładzie jednostajnym z przedziału [0,1]

poniższe listy argumentów analogicznie

```
rand(m,n)
rand(d1,d2,d3,..)
rand([d1,d2,d3,...])
rand(size(A))
```

h. `randn(n)` - tworzy macierz n na n wypełnioną elemetami losowymi o standardowym rozkładzie normalnym (mediana 0, odchylenie standardowe 1)

poniższe listy argumentów analogicznie

```
randn(m,n)
randn(d1,d2,d3,...)
```

```
randn([d1,d2,d3,...])
randn(size(A))
```

i. `repmat(A,m,n)` - tworzy macierz złożoną z m na n zduplikowanych macierzy A

poniższe listy argumentów analogicznie

```
repmat(A,[m,n])
repmat(A,[d1,d2,d3,...])
```

Uwaga:

`repmat(3,4,5)` jest dużo szybsze w działaniu niż `3*ones(4,5)`

j. `blkdiag(a,b,c,d,...)` - tworzy macierz blokowo diagonalną, kolejne klatki diagonali stanowią macierze a,b,c,d,...

Jest wiele funkcji tworzących specjalne macierze. Studenci powinni przejrzeć te funkcje, korzystając z pomocy Matlaba w celu dobrania odpowiednich przykładów do projektów. Na zajęciach warto wspomnieć o `magic(n)` i `hilb(n)` tworzących odpowiednio macierze magiczne i macierze Hilberta.

Bardzo użyteczną możliwością Matlaba jest zapis i odczyt danych z i do plików.
Możliwa jest obsługa plików binarnych i tekstowych (ASCII).
Domyślnym rozszerzeniem plików binarnych w Matlabie jest *.mat. Pliki te zdolne są do przechowywania wielu zmiennych. Zajmują one mniej miejsca na dysku niż pliki tekstowe, ale mogą nie być rozumiane przez inne aplikacje.
Pliki tekstowe mają również wiele ograniczeń:

- mogą przechowywać tylko dwuwymiarowe macierze
- część urojona liczb zespolonych nie jest zapisywana do pliku
- zapis wielu zmiennych do tego samego pliku spowoduje zapisanie ich bezpośrednio po sobie bez symboli rozdzielających
- nie przechowują nazw zmiennych

- muszą posiadać taką samą liczbę kolumn w każdym rzędzie

Zapis następuje przez funkcję `save`:

```
save nazwapliku zm1 zm2 ... opcje
```

gdzie:

`nazwapliku` - nazwa zapisywanego pliku (w przypadku plików tekstowych należy nadać jej rozszerzenie, w przypadku plików binarnych matlab sam dołącza rozszerzenie '.mat'). Niepodanie nazwy pliku powoduje zapis do pliku 'matlab.mat'

`zm1, zm2` - kolejne zmienne, które mają być zapisane do pliku, w przypadku braku deklaracji zmiennych następuje zapis wszystkich zmiennych znajdujących sie w aktualnej przestrzeni roboczej

`opcje` - decydują o właściwościach zapisu:

- `-append` dopisuje zmienne na koniec istniejącego pliku binarnego
- `-ascii` zapis do pliku tekstowego (8 cyfr znaczących na zmienną)
- `-ascii -double` zapis do pliku tekstowego (16 cyfr znaczących na zmienną)
- `-ascii -tabs` zapis do pliku tekstowego, elementy rozdzielone znakiem tabulacji (8 cyfr znaczących na zmienną)
- `-ascii -double -tabs` zapis do pliku tekstowego, elementy rozdzielone znakiem tabulacji (16 cyfr znaczących na zmienną)
- `-mat` domyślna opcja - zapis do pliku binarnego

Uwaga można użyć tylko jednej z powyższych opcji.

Do odczytu danych służy funkcja `load`:

`load` - wczytuje do przestrzeni roboczej zmienne zapisane w pliku matlab.mat

`load nazwapliku` - jeśli nazwapliku nie ma rozszerzenia to polecenie wczytuje do przestrzeni roboczej zmienne zapisane w pliku binarnym ‘nazwapliku.mat’,
jeśli nazwapliku ma rozszerzenie to plik traktowany jest jako tekstowy, dane z pliku umieszczane są w zmiennej nazwapliku (bez rozszerzenia - jest pomijane) jako dwuwymiarowa tablica, gdzie każdy rząd stanowią dane wczytane z jednej linii pliku

`load nazwapliku zm1 zm2 ...` - wczytuje z pliku binarnego ‘nazwapliku.mat’ tylko zmienne zm1,zm2,...

`load nazwapliku -ascii` - wymusza odczyt w trybie tekstowym

`load nazwapliku -mat` - wymusza odczyt w trybie binarnym

Uwagi:

- w przypadku odczytu pliku tekstowego jeśli nazwa pliku zawiera znaki niedozwolone w nazwie zmiennej lub nie zaczyna się od litery - znak początkowy może być zastąpiony literą `X` a znaki niedozwolone znakami _
- wynik polecenia `load` można przypisać do zmiennej, np.
  `A=load('plik.dat')`
  W przypadku pliku tekstowego do zmiennej zostanie przypisana tablica dwuwymiarowa z danymi. W przypadku pliku binarnego zmienna stanie się strukturą o polach, będących nazwami zmiennych przechowywanych w pliku i odpowiednich wartościach. Do pól struktury można odwoływać się poprzez operator `.`

Struktury można tworzyć poprzez zdefiniowanie jakiegokolwiek pola zmiennej, np.

```
osoba.wiek=21
```

Pliki skryptowe m-file.

Plik skryptowy to sekwencja prostych poleceń zapisanych w pliku. Dzięki temu można ją wykonywać wielokrotnie przy użyciu pojedynczego polecenia.

Budowa pliku skryptowego nazwa_pliku.m:

```
%pierwsza linia komentarza - H1
%druga linie komentarza - H2
%...
%ostatnia linia komentarza - Hk
treść
```

Taka budowa powoduje, że:

- przy wywołaniu `nazwa_pliku` wykonana zostanie treść skryptu
- przy wywołaniu opcji `help nazwa_pliku` pokazany zostanie opis pliku, czyli linie komentarza H1-Hk.
- przy wywołaniu `help nazwa_katalogu_z_plikem` pokazana zostanie linia nagłówkowa H1.

przykład moj_pierwszy_skrypt.m:

```
%MOJ_PIERWSZY_SKRYPT - skrypt tworzący kilka macierzy
%skrypt generuje macierz magiczna o rozmiarach 4 na 4
%oraz macierz diagonalna
A=magic(4);
B=diag(5:9)
```

Warto zwrócić uwagę na:

- nie stosowanie polskich znaków
- podanie w pierwszej linii nazwy pliku i ogólnego działania
- szczegółowy opis w kolejnych liniach

- zastosowanie średnika spowoduje wyświetlenie wyniku tylko wyniku dla macierzy B.

Warto wykonać następujące wywołania:

```
moj_pierwszy_skrypt
help moj_pierwszy_skrypt
help katalog_bieżący
```

Uwagi:

- wszystkie zmienne utworzone w czasie działania skryptu istnieją po jego zakończeniu i mają określone przez skrypt wartości
- skrypt nie pobiera żadnych argumentów i nie zwraca wartości
- inne własności będą miały funkcje (m-pliki funkcyjne) - patrz zajęcia 3.

Adresowanie w macierzach:

- Stworzenie macierzy magicznej 4 na 4

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

- Odwołanie bezpośrednie do komórki macierzy - 2. rząd 3. kolumna

```
>> A(2,3)
ans =
    10
```

- Adresowanie do bloku wybranych kolumn i rzędów - rzędy od 1 do 3 kolumna 4.

```
>> A(1:3,4)
ans =
```

```
    13
     8
    12
```

- Wybór 2. i 4. rzędu, oraz wszystkich kolumn. Dwukropek oznacza wszystkie kolumny lub wierze w zależności od pozycji na której stoi.

```
>> A([2,4],:)
ans =
     5    11    10     8
     4    14    15     1
```

- Rzędy 3.,3. i pierwszy - w ten sposób można powtarzać dane lub zmieniać ich kolejność. Słowo kluczowe `end` oznacza przy adresowaniu ostatni wiersz lub ostatnią kolumnę.

```
>> A([3,3,1],[end,2])
ans =
    12     7
    12     7
    13     2
```

- Funkcje logiczne - zwracają elementy dla których wartość jest prawdą. Przy użyciu adresowania możemy nie tylko wypisywać ale też zamieniać, czy podstawiać.

```
>> A(A<10)=0
A =
    16     0     0    13
     0    11    10     0
     0     0     0    12
     0    14    15     0
>> A(1,:)=diag(A).'
A =
    16    11     0     0
     0    11    10     0
```

```
     0     0     0    12
     0    14    15     0
```

Poniższy przykład buduje macierz B jako sumę siedmiu macierzy jednostkowych minus macierz samych jedynek, następnie zamienia 2 i 3 wiersz miejscami wpisując wynik do macierzy C by w końcu usunąć z macierzy C drugą i ostatnią kolumnę:

```
>> B=7*eye(4)-ones(4)
B =
     6    -1    -1    -1
    -1     6    -1    -1
    -1    -1     6    -1
    -1    -1    -1     6
>> C=B([1,3,2,4],:)
C =
     6    -1    -1    -1
    -1    -1     6    -1
    -1     6    -1    -1
    -1    -1    -1     6
>> C(:,[2,end])=[]
C =
     6    -1
    -1     6
    -1    -1
    -1    -1
```

Funkcje przydatne przy adresowaniu:

- `isfinite` - zwraca skończone elementy macierzy
- `isinf` - zwraca nieskończone elementy macierzy
- `isnan` - zwraca nie liczbowe elementy macierzy
- `isprime` - zwraca liczby pierwsze

```
>> C=[0/0,1/0,2;2/3,-3/0,log(-1)]
```

```
C =
      NaN                 Inf             2.0000
   0.6667                -Inf             0 + 3.1416i
>> D=C(isfinite(C))
D =
     0.6667
     2.0000
     0 + 3.1416i
```

Przykład złożony:

```
>> [repmat(1:3,3,1)+linspace(0,6,3).'*ones(1,3),...
fliplr(4*eye(3));repmat(5*diag([1,1]),1,3)]
ans =
     1     2     3     0     0     4
     4     5     6     0     4     0
     7     8     9     4     0     0
     5     0     5     0     5     0
     0     5     0     5     0     5
```

Występuje tu nieznana jeszcze funkcja `fliplr`, która daje odbicie lustrzane macierzy.

Operatory działań po współrzędnych `.*,./.\.^` :

```
>> A=eye(4)
A =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> B=hilb(4)
B =
    1.0000    0.5000    0.3333    0.2500
```

```
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
>> A/B
ans =
  1.0e+003 *
    0.0160   -0.1200    0.2400   -0.1400
   -0.1200    1.2000   -2.7000    1.6800
    0.2400   -2.7000    6.4800   -4.2000
   -0.1400    1.6800   -4.2000    2.8000
>> ans*B
ans =
    1.0000    0.0000         0   -0.0000
         0    1.0000   -0.0000   -0.0000
         0         0    1.0000    0.0000
         0   -0.0000   -0.0000    1.0000
>> A./B
ans =
     1     0     0     0
     0     3     0     0
     0     0     5     0
     0     0     0     7
```

Funkcje dotyczące macierzy, wektorów i algebry liniowej:

`dot(A,B)` - zwraca iloczyn skalarny wektorów A, B. Wektory A, B muszą być tej samej długości. Jeśli A i B są wielowymiarowymi macierzami, iloczyn skalarny liczony jest w pierwszym nietrywialnym wymiarze

`dot(A,B,n)` - jak wyżej, iloczyn w przypadku macierzy liczony w n-tym wymiarze

`length(A)` - zwraca długość wektora A. W przypadku macierzy zwraca maksimum po wszystkich wymiarach wielkości macierzy (`max(size(A))`)

`ndims(A)` - zwraca ilość wymiarów macierzy (pomijając trywialne wymiary, czyli takie dla których `size(A,dim)==1`). Jest odpowiednikiem polecenia `length(size(A))`

`d = size(X)` - zwraca wektor zawierający wielkość macierzy X w kolejnych wymiarach

`[m,n] = size(X)` - dla macierzy dwuwymiarowej - zwraca wielkość w oddzielnych zmiennych m i n

`m = size(X,dim)` - zwraca wielkość macierzy X w dim-tym wymiarze

`[d1,d2,d3,...,dn] = size(X)` - zwraca wielkość macierzy w kolejnych wymiarach do kolejnych zmiennych. Jeśli liczba zmiennych większa niż wymiarów - pozostałe końcowe zmienne zawierają 1. Jeśli liczba zmiennych mniejsza niż liczba wymiarów, zmienna dn zawiera iloczyn wielkości pozostałych wymiarów.

`n = norm(A)` - norma macierzy A - największa wartość szczególna macierzy

`n = norm(A,p)` - p-ta norma macierzy A:

- p=1 `max(sum(abs(A))` - maksimum z sumy kolumn
- p=2 największa wartość szczególna macierzy (opcja domyślna)
- p=inf `max(sum(abs(A')))` - maksimum sumy wierszy
- p='fro' norma Frobeniusa `sqrt(sum(diag(A'*A)))`

Uwaga w przypadku wektorów:

`norm(V,p)` - zwraca: `sum(abs(V).^p)^(1/p)`, dla 1 <= p <= inf

`norm(V)` - zwraca `norm(V,2)`

`norm(V,inf)` - zwraca `max(abs(V))`

`norm(V,-inf)` - zwraca `min(abs(V))`

`rank(A)` - zwraca rząd macierzy A (w praktyce ilość wartości szczególnych macierzy A większych od eps)

`rank(A,tol)` - zwraca ilość wartości szczególnych macierzy A większych od tol

`det(A)` - wyznacznik macierzy A

`trace(A)` - ślad macierzy A (`sum(diag(A))`)

`null(A)` - baza jądra przekształcenia A

`orth(A)` - ortonormalna baza przekształcenia A

`inv(A)` - macierz odwrotna do A

`cond(A)` - współczynnik uwarunkowania macierzy

`cond(A,p)` - współczynnik uwarunkowania macierzy oparty na odpowiedniej normie (`1, 2, inf, 'fro'`)

`R = chol(X)` - rozkład Cholskiego macierzy hermitowskiej X. Jeśli X nie jest macierzą hermitowską, rozważana jest tylko macierz trójkątna górnaz przekątna X, a pozostała część jest dopełniana do spełnienia warunku bycia hermitowską. Zwracana macierz R jest macierzą trójkątna górną taką że: R'*R=X

`[L,U] = lu(X)` - zwraca macierze L - permutację wierszową macierzy trójkątnej dolną (z jedynkami na głównej przekątnej) i U - macierz trójkątną górną, takie że X=L*U

`[L,U,P] = lu(X)` - zwraca macierze L - trójkątną dolną (z 1 na macierzy), U- trójkątna górną oraz P macierz permutacji wierszy, takie że: L*U=P*X

`Y = lu(X)` - zwraca macierz Y zawierającą rozkład LU zapisany w jednej macierzy (przekątna macierzy L zawierająca same jedynki jest pomijana)

`tril(U,k)` - zwraca część macierzy U pod k-tą przekątna wraz z tą przekątną. Główna przekątna to zerowa przekątna, przekątne nad nią mają numery dodatnie a pod nią ujemne. Domyślną wartością k przy wywołaniu tril(U) jest 0

`triu(U,k)` - analogicznie tylko zwraca wszystko nad przekątną wraz z przekątną

`[Q,R] = qr(A)` - zwraca rozkład QR macierzy A, czyli rozkład na macierz trójkątną górną i unitarną Q ($Q^H*Q=I$)

`[Q,R,E] = qr(A)` - zwraca macierze: Q - unitarną, R - trójkątną górną taką że elementy na przekątnej są ułożone w porządku malejącym co do modułu oraz macierz permutacji kolumn E, takie że: `A*E=Q*R`

`X = qr(A)` - zwraca rozkład Cholesky'ego `triu(X)=R,` `R'*R=A'*A`

`B = pinv(A)` - zwraca pseudo odwrotność macierzy A - macierz B taką że:

`A*B*A=A`

`B*A*B=B`

`A*B` jest macierzą hermitowską

`B*A` jest macierzą hermitowską

`d = eig(A)` - zwraca wektor wartości własnych macierzy A (`Ax=λx`)

`d = eig(A,B)` - zwraca wektor uogólnionych wartości własnych kwadratowych macierzy A, B (`Ax=λBx`)

`[V,D] = eig(A)` - zwraca wektory i wartości własne macierzy A w postaci macierzy V, której kolumny są wektorami własnymi i w postaci macierzy D, która posiada wartości własne na głównej przekątnej, A*V=V*D

`[V,D] = eig(A,B)` - analogicznie

`s = svd(X)` - zwraca wektor wartości szczególnych macierzy

`[U,S,V] = svd(X)` - zwraca macierz S - diagonalną zawierającą na przekątnej wartości szczególne macierzy X w porządku malejącym oraz macierze U i V unitarne, takie że: X=U*S*V'

`poly(A)` - zwraca współczynniki wielomianu charakterystycznego macierzy A od najwyższej potęgi do najniższej

`poly(r)` - zwraca współczynniki wielomianu o pierwiastkach określonych przez wektor r

`H = hess(A)` - zwraca macierz Hessenberga macierzy A

`[P,H] = hess(A)` - zwraca macierz Hessenberga macierzy A i macierz unitarną P taką że: A=P*H*P' Macierz Hessenberga ma 0 pod subdiagonalą, w przypadku macierzy A symetrycznej lub hermitowskiej macierz H jest trójdiagonalna

`T = schur(A)` - zwraca postać Schura macierzy A

`[U,T] = schur(A)` - dokonuje rozkładu Schura macierzy A na macierz unitarną U i macierz Schura T takie że:`A=U*T*U'` Macierz Schura to macierz blokowo diagonalna, zawierająca na przekątnej dla kolejnych wartości własnych rzeczywistych - te wartości, a dla zespolonych bloki 2 na 2 opisujące te wartości

Sumy, iloczyny, maksima, minima:

`B = sum(A)` - zwraca sumę elementów wektora, w przypadku macierzy zwraca wektor poziomy zawierający sumy elementów w kolumnach

`B = sum(A,k)` - zwraca sumę elementów w k-tym wymiarze

`B = cumsum(A)` - zwraca wektor skumulowanych sum elementów wektora, w przypadku macierzy zwraca macierz zawierającą skumulowane sumy elementów w kolumnach

`B = cumsum(A,k)` - zwraca skumulowane sumy elementów w k-tym wymiarze

`B = prod(A)` - zwraca iloczyn elementów wektora, w przypadku macierzy zwraca wektor poziomy zawierający iloczyny elementów w kolumnach

`B = prod(A,k)` - zwraca iloczyn elementów w k-tym wymiarze

`B =cumprod(A)` - zwraca wektor skumulowanych iloczynów elementów wektora, w przypadku macierzy zwraca macierz zawierającą skumulowane iloczyny elementów w kolumnach

`B = cumprod(A,k)` - zwraca skumulowane iloczyny elementów w k-tym wymiarze

`C = min(A)` - zwraca minimum elementów wektora, w przypadku macierzy zwraca wektor poziomy zawierający minima elementów w kolumnach

`C = min(A,B)` - dla macierzy A i B o takich samych rozmiarach zwraca macierz zawierającą minima odpowiadających elementów z A i B

`C = max(A)` - zwraca maksimum elementów wektora, w przypadku macierzy zwraca wektor poziomy zawierający maksima elementów w kolumnach

`C = max(A,B)` - dla macierzy A i B o takich samych rozmiarach zwraca macierz zawierającą maksima odpowiadających elementów z A i B

Zajęcia nr 3:

Programowanie w środowisku Matlab:

Wyrażenia logiczne.

Wyrażeniem logicznym może być dowolna formułą dająca w wyniku liczbę. Każda niezerowa wartość interpretowana jest jako prawda, a 0 jako fałsz. Takie wartości mogą być poddane działaniu operatorów logicznych (`&, |, &&, ||, ~`) czy funkcji takich jak choćby `xor(a,b)`.

Wynikiem działania operatorów i funkcji logicznych jest 1 lub 0. 1 oznacza prawdę, 0 oznacza fałsz.

Instrukcje warunkowe.

```
if wyrazenie_logiczne
     polecenia
     polecenia
     ...
end
```

Jeśli wyrażenie logiczne jest prawdziwe - wykonywane są polecenia z wnętrza wyrażenia `if...end`, jeśli nie wykonywane są kolejne polecenia po instrukcji `end`.

Polecenia `if...end` można zagnieżdżać:

```
if a<0
     b=b+1;
     disp(a);
     if b>10
          c=a/b;
     end
end
```

```
if wyrazenie_logiczne
     polecenia1
else
     polecenia2
end
```

Jeśli wyrażenie logiczne jest prawdziwe - wykonywane są polecenia z wnętrza wyrażenia `if...else`, a następnie te za instrukcją `end`. Jeśli wyrażenie jest fałszywe wykonywane są kolejne polecenia po instrukcji `else` a następnie te po instrukcji `end`.

```
if wyrazenie_logiczne
     polecenia1
elseif wyrazenie_logiczne2
     polecenia2
end
```

Konstrukcja umożliwiająca badanie wielu warunków: jeśli wyrażenie logiczne jest prawdziwe wykonywane są polecenia1 a następnie te za instrukcją `end`, jeśli wyrażenie jest fałszywe sprawdzany jest warunek 2. Jeśli ten jest prawdziwy wykonywane są polecenia2, jeśli nie to program wykonuje instrukcje za słowem `end`.

Oba tryby można łączyć ze sobą:

```
if temperatura > 100
     disp('Uwaga grozba uszkodzenia urzadzenia')
elseif temperatura > 90
     disp('Normalna temperatura pracy')
elseif temperature > 50
     disp('Ponizej optymalnej temperatury pracy')
else
     disp('Zbyt zimno - wylacz urzadzenie')
end
```

```
switch wyrazenie
     case wyr_testowe_1
          polecenia_1
     case {wyr_testowe_2,wyr_testowe_3,wyrtestowe_4}
          polecenia_2
     ...
     otherwise
          polecenia_inne
end
```

Instrukcja `switch` pozwala na wybór jednej ze ścieżek wykonywania programu w zależności od tego, ile wynosi wartość wyrażenia. W zależności od przypadku wykonywane są polecenia z `1, 2, ...` grupy. Kilka wartości dla których mają być wykonywane te same polecenia można połączyć w listę w nawiasach `{}`. Jeśli wyrażenie nie pasuje do żadnego wzorca wtedy wykonywane są `polecenia_inne`.

Pętle.

```
for licznik=wyrazenie
     polecenia
end
```

Pętla for wykonuje polecenia tyle razy ile kolumn ma wyrażenie przypisane do zmiennej licznik. W każdym przebiegu pętli licznik przyjmuje wartość kolejnej kolumny wyrażenia.
Jeśli wyrażenie jest pustą macierzą - pętla nie wykona się ani razu.
Jeśli wyrażenie jest skalarem - pętla wykona się raz.
W trakcie wykonywania pętli nie można zmienić ilości iteracji zmieniając licznik.
Po zakończeniu pętli for licznik zawiera wartość z ostatniego przebiegu pętli.
Często używaną konstrukcją jest:

```
for k=od:przyrost:do
```

Przykład:

```
[m n] = size(x);
xmax = x(1,1);
r=1; c=1;
for k=1:m
     for l=1:n
          if x(k,l)> xmax
               xmax = x(k,l);
               r=k;
               c=l;
          end
     end
end
```

Zakładając że w zmiennej x znajduje się pewna macierz, powyższy przykład przeszukuje ją ustawiając r i c na indeksy rzędu i kolumny wskazującej największy element tej macierzy.

```
while wyrazenie
     polecenia
end
```

Pętla `while` wykonuje `polecenia` tak długo jak `wyrażenie` jest prawdziwe (wszystkie jego elementy są prawdziwe). Jeśli wyrażenie jest stałe i nie zmienia się w czasie działania pętli i jest prawdziwe, to jest to pętla nieskończona.

Do natychmiastowego przejścia do kolejnej iteracji w pętlach `for...end` oraz `while...end` służy polecenie `continue`.

Do natychmiastowego przerwania jednej z tych dwóch pętli służy polecenie `break`.

Przykład:

```
% Skrypt obliczajacy wartosc trojmianu ax^2 +bx + c
disp('Obliczam trojmian ax^2+bx+c')
disp('dla podanych przez uzytkownika: a, b, c, x')
a=1; b=1; c=1; x=0;
while a~=0 | b~=0 | c~=0 | x~=0
     disp('Wprowadz a=b=c=x=0 aby zakonczyc')
     a = input('Wprowadz a: ');
     b = input('Wprowadz b: ');
     c = input('Wprowadz c: ');
     x = input('Wprowadz x: ');
     if a==0 & b==0 & c==0 & x==0
          break
     end
     quadratic = a*x^2 + b*x + c;
     disp('Wartosc trojmianu:')
     disp(quadratic)
end
```

Powyższy przykład oblicza wartości kolejnych trójmianów kwadratowych podanych przez użytkownika. Aby zakończyć program należy wprowadzić same 0.
Pojawia się tu nowe polecenie `input`, które zwraca wartość wprowadzoną przez użytkownika w odpowiedzi na tekst wyświetlony jako argument funkcji.

Poniższe przykłady prezentują jak można ulepszyć kod zawierający wykonania pętli:

Przykład 1.

```
clear,tic,for t=1:20000 y(t)=sin(2*pi*t/10); end,toc
```

Przykład 2.

```
clear,tic,y=zeros(1,20000);for t=1:20000 ...
y(t)=sin(2*pi*t/10); end,toc
```

Przykład 3.

```
clear,tic,t=1:20000;y=sin(2*pi*t/10); toc
```

Funkcje `tic toc` służą do pomiaru czasu - `tic` włącza licznik, `toc` wyłącza i zwraca wartość czasu jaki upłynął od ostatniego `tic` wyrażony w sekundach.

Obsługa błędów.

```
try
     polecenia_1
catch
     polecenia_2
end
```

Taki blok służy do łapania błędów mogących powstać w skutek działania programu. Jeżeli w trakcie wykonywania poleceń z grupy pierwszej wystąpi błąd, nastąpi przeskok programu do poleceń z grupy drugiej i ich wykonanie.

Istnieje grupa poleceń które rzucają błędy podczas próby wykonania z niewłaściwymi argumentami, np. `realsqrt`, `realpow`. Błędy występują też często w przypadku niezgodności wymiarów macierzy.

Aby we własnej procedurze lub funkcji zgłosić własny błąd - należy użyć funkcji

```
error('tekst w postaci stringu, czyli w apostrofach')
error('string formatujacy',arg1,arg2,...)     - konwencja jak w C
error('identyfikator błędu','treść błędu')
error('identyfikator błędu','string formatujący',arg1,...)
```

gdzie identyfikator błędu podaje się w następującej postaci: 'autor:typbłędu'

Błąd w rzeczywistości jest strukturą złożona z 2 pól: `message` oraz `identifier`. W pierwszym polu przechowywana jest właściwa treść błędu, w drugim identyfikator.

Do pobrania ostatniego błędu służą dwie funkcje:

`lasterr` - zwracająca ostatnią wiadomość błędu

`lasterror` - zwracająca cały błąd (strukturę)

Przykład:

```
try
     a=-1
     error('MojBlad:PoProstuBlad','tresc bledu')
     b=1
catch
     e1=lasterr;
     e2=lasterror;
end
e1
e2
```

Uwagi:

- linijka `b=1` nigdy nie zostanie wykonana
- identyfikator błędu musi zaczynać się od litery i może być zbudowany jedynie z liter, cyfr znaku _ oraz pojedynczego znaku oddzielającego :
- zmienna `e1` zawiera po wykonaniu jedynie treść błędu
- zmienna `e2` zawiera całą strukturę błędu

Po złapaniu błędu można zmodyfikować zawartą w nim informację i błąd rzucić dalej poleceniem:

```
     rethrow(blad)
```

gdzie błąd jest strukturą z informacją o błędzie i identyfikatorem błędu.

Podobnie do błędów można generować ostrzeżenia za pomocą funkcji `warning`. Różnica jest taka, że ostrzeżenia nie powodują zatrzymania pracy programu - wyświetlają jedynie informację. Zainteresowani obsługą ostrzeżeń znajdą potrzebne informacje we wbudowanej pomocy MATLABa.

Sterowanie użytkownika w czasie działania programu.

`x=input('wyswietlany tekst')` - pobiera wartość numeryczną

`s=input('wyswietlany tekst','s')` - pobiera łańcuch znaków (string)

`keyboard` - przekazuje kontrolę użytkownikowi, który może wykonywać dowolne polecenia, powrót do programu następuje po wywołaniu polecenia:
`return`

`pause` - zatrzymuje działanie aż do wciśnięcia przycisku przez użytkownika

`pause(n)` - zatrzymuje działanie programu na n sekund

`pause on` - zezwala na działanie powyższych poleceń

`pause off` - ignoruje efekt `pause` i `pause(n)`

```
k = menu('naglowek','opcja1','opcja2','opcja3')
```

zwraca numer opcji wybrany przez użytkownika (1,2,3,...).

Więcej na temat funkcji graficznych można znaleźć przy opisie funkcji `uicontrol` lub w dziale Graphics Guide w pomocy MATLABa.

Wyświetlanie wyników.

`disp(A)` - powoduje wyświetlenie zawartości zmiennej A. W odróżnieniu od zwykłego polecenia A nie powoduje wyświetlenia początkowej nazwy zmiennej i znaku =.

`disp('tekst')` - wyświetla tekst

`sprintf('string formatujacy',arg1,arg2,...)` - zwraca sformatowany zgodnie ze składnią języka C string

Wywołania funkcji.

Istnieją dwa sposoby wywoływania funkcji:

- składnia funkcyjna, np.
  `[a,b,c]=funkcja(d,e)`
  Ten sposób pozwala na zwracanie wartości (jednej lub wielu).
  Przekazuje argumenty przez wartość.
- składnia poleceniowa, np.
  `save plik x y z`
  Nie pozwala na zwracanie wartości.
  Przekazuje argumenty przez ciąg znaków.

Różnica między dwoma sposobami widoczna jest na przykładzie następujących wywołań:

```
A=eye(3);
disp(A)
disp A
```

Funkcje.

Funkcje definiuje się w m-plikach funkcyjnych w następujący sposób:

```
function y=nazwa(argumenty)
%pierwsza linia pomocy
%pomoc
%pomoc
tresc funkcji
```

Uwagi na temat funkcji:

- Ograniczenia nazwy funkcji takie same jak w przypadku zmiennych.
- Nazwa funkcji powinna zgadzać się z nazwą pliku. Jeśli tak nie jest nazwa funkcji jest ignorowana a za nazwę funkcji przyjmuje się nazwę pliku.
- Jeśli funkcja ma zwracać wiele wartości to definiujemy ją następująco:
  `function [x, y, z] = sfera(theta, phi, rho)`
  Jeśli ma nie zwracać żadnego argumentu, to:
  `function sphere(theta, phi, rho)`
  lub
  `function [] = sphere(theta, phi, rho)`
- Funkcjonalność komentarzy dotyczących pomocy jest taka sama jak w przypadku plików skryptowych.
- W momencie pierwszego użycia funkcji, jest ona tłumaczona przez MATLAB i przechowywana w pamięci dla potrzeb potencjalnych późniejszych wywołań. Można ją stamtąd usunąć poleceniem
  `clear nazwa_funkcji` lub `clear functions`, lub `clear all`
- Każda funkcja ma własną przestrzeń roboczą ze zmiennymi. W związku z tym funkcja może pobierać i zmieniać wartości jedynie tych zmiennych, które znajdują się w jej przestrzeni roboczej. Mechanizmem pozwalającym na wykorzystywanie zmiennej przez wiele funkcji są zmienne globalne.
- Wartości z funkcji zwraca się przypisując po prostu odpowiednim zmiennym (zapisanym w deklaracji funkcji) odpowiednie wartości.
- W dowolnym momencie działania funkcji można zakończyć jej działanie poleceniem `return`.

Kolejność przeszukiwania zasobów przez MATLAB w przypadku napotkania nowej nazwy:

- sprawdza czy nie jest to zmienna
- sprawdza czy nie jest to subfunkcja funkcji w której pojawiła się nowa nazwa
- sprawdza czy nie jest to funkcja prywatna
- sprawdza ścieżkę przeszukiwań MATLABa w poszukiwaniu skryptu lub funkcji

Zmienne globalne.

Zmienne globalne to zmienne dostępne w dowolnej przestrzeni nazw. Aby zadeklarować zmienną globalną należy użyć deklaracji:

```
global nazwa_zmiennej
```

Taka deklaracja wymagana jest we wszystkich funkcjach, skryptach, w których ta zmienna ma być używana. Aby wyświetlić aktualnie istniejące zmienne globalne wystarczy użyć polecenia

```
who global
whos global
```

Zmienne trwałe (persistent).

Zmienne trwałe to zmienne:

- dozwolone jedynie w funkcjach
- zachowujące wartość pomiędzy kolejnymi wywołaniami funkcji
- wymagające wcześniejszej deklaracji:
  `persistant nazwa_zmiennej`
- usuwane razem z funkcją z pamięci poleceniem `clear`

Subfunkcje.

M-plik funkcyjny może zawierać więcej niż jedną funkcję. Pierwsza funkcja w m-pliku związana jest z nazwą pliku i jest funkcją główną. Pozostałe funkcje w m-pliku to subfunkcje. Widoczne one są jedynie dla innych subfunkcji i dla funkcji głównej z tego samego pliku. Deklaruje się je tak samo jak zwykłe funkcje, np.

```
function [avg,med] = newstats(u) % Funkcja podstawowa
% NEWSTATS Znajduje wartosc srednia i mediane dla wektora u
n = length(u);
avg = mean(u,n);
```

```
med = median(u,n);

function a = mean(v,n)              % Subfunkcja
% Liczy wartosc srednia.
a = sum(v)/n;

function m = median(v,n)            % Subfunkcja
% Liczy mediane.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

Uwagi:

- W m-pliku można zapisać jedynie informacje pomocy na temat pierwszej funkcji.
- Subfunkcje nie mają wspólnej przestrzeni nazw, dlatego dane pomiędzy nimi muszą być wymieniane przez zmienne globalne lub przez argumentu funkcji.

Funkcje prywatne.

Funkcje prywatne tworzy się umieszczając m-pliki w katalogu o nazwie private. Dostępne są one jedynie dla funkcji umieszczonych w katalogu nadrzędnym względem danego katalogu private, np.
Jeśli funkcję funk.m umieści się w katalogu /home/mn1/private, to będzie ona funkcją prywatną, widoczną jedynie dla funkcji z katalogu /home/mn1
Funkcje prywatne pozwalają „nadpisać" inne funkcje systemowe MATLABa, gdyż są wyszukiwane wcześniej niż funkcje wbudowane w MATLABa.

Zmienna ilość argumentów i wartości zwracanych w funkcjach.

Funkcje mogą zwracać lub pobierać mniejszą liczbę argumentów niż przewiduje składnia. Do zbadania z jaką ilością argumentów wywołał funkcję użytkownik służy zmienna `nargin`. Przechowuje ona rzeczywistą ilość argumentów w wywołaniu funkcji. Oczekiwania ilość wartości zwracanych zapamiętana jest w zmiennej `nargout`. MATLAB posiada dodatkowo wbudowane dwie funkcje ułatwiające sprawdzenie, czy użytkownik podał dopuszczalną przez autora funkcji ilość argumentów i zmiennych, do których mają być zapisane wyniki. Są to:

`nargchk(min,max,liczba)` - zraca komunikat `'Not enough input parameters'` w przypadku gdy liczba<min lub liczba >max. W przeciwnym wypadku zwraca macierz pustą. Warto stosować w następujący sposób:

```
error(nargchk(min,max,nargin))
```

`nargoutchk(min, max,liczba)` - analogicznie dotyczy zwracanych wartości

```
function y=nazwa1(arg1,varargin)
function [s,varargout]=nazwa2(k)
```

Powyższa składnia zapewnia, że w przypadku pierwszej funkcji pierwszym argumentem jest `arg1`, kolejne argumenty będą umieszczone w liście `varargin`. W przypadku drugiej funkcji pierwszą zwracaną wartością jest s, kolejne zwracane wartości to wartości z listy `varargout`.

Listy tworzy się następująco:

```
list1={1,eye(3),1:9,'aaa'}
list1(5)={7}
```

Elementy z listy pobiera się następująco:

```
A=list{1}
```

Ewaluacja funkcji

`eval(string)`	- wykonuje instrukcję podaną w parametrze
`eval(string,catch_string)`	- dodatkowo w przypadku wystąpienia błędu, jest on łapany i wykonywana jest instrukcja `catch_string`
`[a1,a2,a3,...]=eval(string)`	- funkcja `eval` zwraca wyniki instrukcji wywołanej za jej pomocą

Przykład 1.

```
for n = 1:12
      magic_str = ['M',int2str(n),' = magic(n)'];
      eval(magic_str)
end
```

Przykład ten tworzy dwanaście macierzy magicznych M1, M2, ...M12.
Użyte jest tu nowe polecenie - `int2str(n)`, które zamienia liczbę na ciąg znaków (string) opisujący tę liczbę. Ponieważ ciąg znaków jest tak w MATLABie wektorem znaków, więc kolejne łańcuchy znaków można łączyć za pomocą operacji znanych dla wektorów.

Przykład 2.

```
% ----- plik showdemo.m -----
     function showdemo(demos)
     errstring = 'Blad uruchamiania dema: ';
     n = input('Podaj numer dema: ');
     eval(demos(n,:),'[errstring demos(n,:)]')
% ----- koniec pliku showdemo.m -----
D = ['odedemo'; 'quademo'; 'fitdemo'];
showdemo(D)
Podaj numer dema: 2
```

```
ans =
Blad uruchamiania dema: quademo
```

Uwaga:

Możliwe są następujące dwa sposoby pobrania wyników z funkcji wykonywanej:

```
[a,b,c]=eval('funkcja(d,e)')
eval('[a,b,c]=funkcja(d,e)')
```

Dokumentacja MATLABa zaleca użycie sposobu pierwszego, gdy tylko jest to możliwe. Sposób ten umożliwia wykrycie większej liczby błędów składniowych dotyczących niezgodności liczby wyników.

```
T=evalc(string)
T=evalc(string,catch_string)
[T,a1,a2,a3,...]=evalc(string)
```

Funkcje `evalc` i `eval` są identyczne w działaniu i składni. Różnica polega na tym, że funkcja `eval` wyświetla wyniki swojego działania, natomiast funkcja `evalc` przekazuje wyniki w postaci łańcucha znaków do zmiennej `T`. Kolejne linie oddzielone są znakiem `'\n'`.

```
evalin(pr,string)
evalin(pr,string,catch_string)
[a1,a2,a3,...]=evalin(pr,string)
```

Funkcja `evalin` wykonuje obliczenia w przestrzeni roboczej `pr`. `pr` może przyjmować jedną z dwóch wartości: `'base'` lub `'caller'`. Obliczenia wykonywane są wtedy odpowiednio w bazowej przestrzeni roboczej (tej która jest przestrzenią roboczą dal poleceń wykonywanych bezpośrednio w MATLABie) lub w przestrzeni nadrzędnej do aktualnej (czyli w przestrzeni funkcji lub procedury, która wywołała aktualnie wykonywaną funkcje). Uwaga nie jest możliwe rekurencyjne przechodzenie do coraz wyższych przestrzeni:

```
evalin('caller','evalin(''caller'',''x'')')
```

Można tylko przejść o jedną przestrzeń wyżej.

```
builtin(funkcja,x1,...,xn)
[y1,..,yn] = builtin(funkcja,x1,...,xn)
```

Funkcja `builtin` wywołuje wbudowaną funkcję MATLABa o nazwie funkcja z argumentami `x1,x2,...xn` zwracającą wartości `y1,y2,...yn`. Ignorowane są przeciążone nazwy funkcji.

`assignin(pr,'zmienna',wartosc)` - przypisuje w przestrzeni roboczej `pr` (`'base'`, `'caller'`) do zmiennej `zmienna` wartość `wartosc`

Przykład

```
nazwy_pol = {'Wprowadz wartosc dla zmiennej A:',...
'Wprowadz wartosc dla zmiennej B:'};
tytul = 'Przykladowy tytul';
ilosc_linii = 1;
domyslne = {'Ala ma kota','a kot ma ale'};
odpowiedz = inputdlg(nazwy_pol,tytul,ilosc_linii,domyslne);
assignin('base','A',odpowiedz{1});
assignin('base','B',odpowiedz{2});
```

`run nazwa_skryptu` - uruchamia skrypt o podanej nazwie i ścieżce

Funkcja ta powinna być używana wtedy gdy chcemy wykonać plik znajdujący się w katalogu nie znajdującym się na ścieżce przeszukiwań.

`[y1,y2,...] = feval(fhandle,x1,...,xn)` - wykonuje funkcję do której uchwyt jest podany w pierwszym parametrze z parametrami `x1,x2,...xn`, zwracając `y1,y2...`

`[y1,y2,...] = feval(function,x1,...,xn)` - wykonuje funkcję, której nazwa jest podana w pierwszym parametrze z parametrami `x1,x2,...xn`, zwracając `y1,y2...`

Uchwyty do funkcji tworzy się w następujący sposób:

```
uchwyt_do_funkcji_sin=@sin
```

Uchwyty mogą wskazywać na: funkcje biblioteczne MATLABa, na funkcje zawarte w M-plikach funkcyjnych.

Przykład 1.

```
[V,D] = eig(A)
[V,D] = feval(@eig,A)
```

Powyższe wywołania mają takie same znacznie.

Przykład 2.

```
funkcja=[@sin,@cos,@log];
k=input('Podaj numer funkcji: ');
x=input('Podaj argument funkcji: ');
feval(funkcja(k),x)
```

Zajęcia nr 4:

Programowanie w środowisku Matlab - ciąg dalszy:

Zwiększanie efektywności programów:

1. Wektoryzacja kodu- gdy tylko jest to możliwe należy korzystać z gotowych struktur MATLABa i posługiwać się wbudowanymi macierzami, wektorami i funkcjami na nich działającymi. W przypadku, kiedy można należy usuwać z kodu pętle na rzecz działania na wektorach. Odpowiedni przykład podany był na poprzednich zajęciach.
2. Prealokacja dużych struktur danych. Kiedy zachodzi konieczność prowadzenia obliczeń na dużych macierzach lub też innych strukturach danych, warto wcześniej prealokować taką strukturę poprzez przypisanie od danej zmiennej pustej struktury o wymaganym rozmiarze. Zabieg taki eliminuje konieczność doalokowywania pamięci na dane w trakcie wykonywania programu i sztucznego przenoszenia danych z jednej struktury do drugiej, powodując skrócenie czasu działania programu.

Łańcuchy znaków (stringi)

Łańcuchy znaków to nic innego jak tablice liczb naturalnych, które mogą również być interpretowane jako litery.

Tworzenie:

1. Najprostszym sposobem budowania stringów jest umieszczenie żądanego tekstu w znakach ‘ , np.

```
imie='Alicja';
```

2. Istniejące już łańcuchy znaków można łączyć jak zwykłe tablice przy użyciu `[]` lub specjalnym poleceniem `strcat`:

```
t = strcat(s1,s2,s3,...)
```

3. Do pionowego połączenia dwóch łańcuchów znaków służy zaś polecenie:

```
S = strvcat(t1,t2,t3,...)
```

Ponieważ podobnie jak w przypadku zwykłych macierzy w każdym rzędzie musi być taka sama liczba elementów, polecenie `strvcat` automatycznie dodaje do odpowiednich słów końcowe spacje w takiej ilości aby wyrazy uformowały macierz.

4. W przypadku ręcznego tworzenia macierzy stringów należy zadbać o taką samą ilość elementów w każdym wierszu, np.

```
name = ['Thomas R. Lee   '; 'Senior Developer'];
```

5. Konieczność budowania stringów o jednakowej długości można porzucić wykorzystując funkcję `char`, np.

```
name = char('Thomas R. Lee','Senior Developer')
```

6. Zbędną ilość końcowych spacji można usunąć poleceniem `deblank`,np.

```
trimname = deblank(name(1,:))
```

7. Aby zbudować string złożony z samych spacji należy użyć polecenia `blanks(n)`, gdzie n jest liczbą żądanych spacji, np.

```
blanks(20);
```

Konwersje:

`double(str)`	- zamienia string w tablice liczb (kodów ASCII zapisanych tam znaków)
`char(str)`	- zamienia tablice kodów ASCII na string
`int2str(n)`	- zamienia liczbę całkowitą na string (niecałkowite są zaokrąglane)
`mat2str(A)`	- zamienia macierz A na string w postaci [..,..;..,..] precyzja double
`mat2str(A,n)`	- jak wyżej, ale z dokładnością do n cyfr
`num2str(A)`	- zamienia liczbę na string z dokładnością 4 cyfr
`num2str(A,prec)`	- jak wyżej tylko z żądaną dokładnością

`num2str(A,format)` - jak wyżej tylko o określonym formacie wyświetlenia , np. `'%11.4g'` (jak w funkcji printf w ANSI C)

`str2double('str')` - zamienia string na jego wartość liczbową

Zmiany systemu:

`base2dec('strn',base)` - zamienia string w bazie base na liczbę, np. `base2dec('212',3)`

`dec2base(d,base)` - zamienia liczbę d na string odpowiadający jej reprezentacji w bazie base

`dec2base(d,base,n)` - tak samo tylko z precyzją n cyfr

`bin2dec(binarystr)` - zamienia string z postacią binarną liczby na liczbę

`dec2bin(d)` - konwersja odwrotna

`dec2bin(d,n)` - jak wyżej z dokładnością n cyfr

`hex2dec(str)` - zamienia string z postacią heksadecymalną liczby na liczbę

`dec2hex(d)` - konwersja odwrotna

`dec2hex(d,n)` - jak wyżej z dokładnością n cyfr

Porównania stringów:

`strcmp('str1','str2')` - porównanie stringów (1 jeśli równe, 0 jeśli różne)

`strcmpi(str1,str2)` - jak wyżej, ignoruje wielkość liter

`strncmp('str1','str2',n)` - jak wyżej porównuje pierwsze n znaków

`strncmpi('str1','str2',n)` - jak wyżej porównuje pierwsze n znaków ignorując wielkość liter

Uwaga operatory `==,~=,<,>,<=,>=` działają po współrzędnych - porównują pojedyncze odpowiadające sobie znaki.

`isletter('str')` - zwraca wektor, który dla każdego znaku określa czy znajduje się tam litera (1) lub inny znak (0)

`isspace('str')` - analogicznie tylko bada spacje

Wyszukiwanie i zamiana:

`strrep(str1,str2,str3)` - zamienia w str1 wystąpienia str2 na str3, np:

```
s1 = 'This is a good example.';
str = strrep(s1,'good','great')
```

`findstr(str1,str2)` - przeszukuje dłuższy string w poszukiwaniu krótszego i zwraca indeksy wszystkich wystąpień krótszego w dłuższym

`token = strtok('str',delimiter)` - zwraca początek 'str' aż do napotkania znaku rozdzielającego `delimiter`. Początkowe wystąpienia znaku rozdzielającego są ignorowane

`token = strtok('str')` - jak wyżej tylko znakiem rozdzielającym są wszystkie białe znaki

`[token,rem] = strtok(...)` - jak wyżej, oprócz początku zwraca również pozostały łańcuch znaków wraz z rozpoczynającym znakiem rozdzielającym do drugiej zmiennej

`strmatch('str',STRS)` - przeszukuje tablicę stringów STRS w poszukiwaniu tych, które rozpoczynają się od 'str'. Zwraca indeksy rzędów które pasują.

`strmatch('str',STRS,'exact')` - jak wyżej tylko całe stringi muszą się zgadzać dokładnie

`lower('str')` - zamienia wielkie litery na małe

`upper('str')` - zamienia małe litery na wielkie

Wielomiany.

`k = roots(w)` - zwraca wektor kolumnowy zawierający miejsca zerowe wielomianu podanego jako wektor poziomy współczynników wielomianu od najwyższej do najniższej potęgi

`poly(A)` - zwraca wielomian charakterystyczny macierzy A

`poly(v)` - zwraca wielomian, którego miejsca zerowe określone są w wektorze v

`polyval(p,x)` - oblicza wartość wielomianu p dla wartości x, np. `polyval([1,2,3],[4,5,6])` - oblicza wartość wielomianu `x^2+2*x+3` kolejno dla wartości `x=4,5,6`

`p = polyfit(x,y,n)` - oblicza współczynniki wielomianu p(x) stopnia n, takie że w sensie średniokwadratowym najlepiej pasują do punktów `x(k), y(k)`

`conv(p,q)` - wykonuje iloczyn dwóch wielomianów p i q przez siebie

`[q,r]=deconv(v,u)` - dzieli wielomiany z resztą (`v=u*q+r`)

`yi = interp1(x,Y,xi)` - dla węzłów x i wartości w nich Y buduje interpolację liniową i zwraca wartości tej interpolacji w punktach xi

`yi = interp1(Y,xi)` - dla węzłów `1:N`, gdzie `N=length(Y)` i wartości w nich Y buduje interpolację liniową i zwraca wartości tej interpolacji w punktach xi

`yi = interp1(x,Y,xi,metoda)` - jak wyżej z tą różnicą że interpolacja nie musi być liniowa. Zależne to jest od metody, która może przyjąć jedną z następujących wartości:

- `'nearest'` - interpolacja wartością najbliższą
- `'linear'` - wartość domyślna - interpolacja liniowa
- `'spline'` - spline kubiczny
- `'pchip'` - interpolacja splinem kubicznym Hermite'a

`yy = spline(x,y,xx)` - analogiczne do `interp1(x,y,,xx,'spline')`

Macierze rozrzedzone.

`S = sparse(A)` - tworzy macierz rozrzedzoną z macierzy pełnej

`S = sparse(i,j,s,m,n,nzmax)` - tworzy macierz rozrzedzoną o rozmiarach `m` na `n`, allokując pamięć na maksymalnie `nzmax` elementów

i wypełniając ją wartościami z wektora `s` w miejscach wskazywanych prze indeksy `i` i `j`, czyli `S(i(k),j(k))=s(k)`

`S = sparse(i,j,s,m,n)` - jak wyżej, allokuje pamięć w wielkości: `length(i)= length(j)= length(s)`

`S = sparse(i,j,s)` - jak wyżej, `n=max(i), m=max(j)`

`S = sparse(m,n)` - tworzy pustą macierz rzadką o rozmiarach `m` na `n`

`A=full(S)` - konwertuje macierz rzadką do macierzy pełnej

`S = speye(m,n)` - macierz jednostkowa m na n

`S = speye(n)` - macierz jednostkowa n na n

`R = sprand(S)` - tworzy macierz wielkości innej macierzy rzadkiej `S` o elementach losowych (rozkład jednostajny [0,1]) w tych samych miejscach co w macierzy `S`

`R = sprand(m,n,den)` - tworzy macierz losową (rozkład jednostajny [0,1]) `m` na `n` o gęstości den z przedziału [0,1], to znaczy, że średnio można się spodziewać `m*n*den` niezerowych elementów w macierzy wynikowej

`R = sprandn(S)` - jak wyżej tylko standardowy rozkład normalny

`R = sprandn(m,n,density)`

`v=find(A)` - wyszukuje niezerowe elementy macierzy pełnej A i zwraca ich indeksy w ten sposób, że traktuje macierz A jako wektor kolumnowy `A(:)=[A(:,1};A(:,2);...;A(:,end)]`

`[i,j]=find(A)` - zwraca numery wierszy i kolumn, w których występują niezerowe elementy

`[i,j,k]=find(A)` - zwraca numery wierszy i kolumn oraz elementy niezerowe macierzy A

`...=find(A>6)` - można wywoływać również z warunkami logicznymi dotyczącymi macierzy A

`S=spconvert(T)` - zamienia macierz `T` (w postaci 3 kolumn, z czego 1. i 2. zawierają wartości całkowite, a 3. wartości typu double) na macierz rzadką taką, że `S(T(k,1),T(k,2)=T(k,3)`

`spy(S)` - daje wykres macierzy, np.

```
spy(wilkinson(12))
```

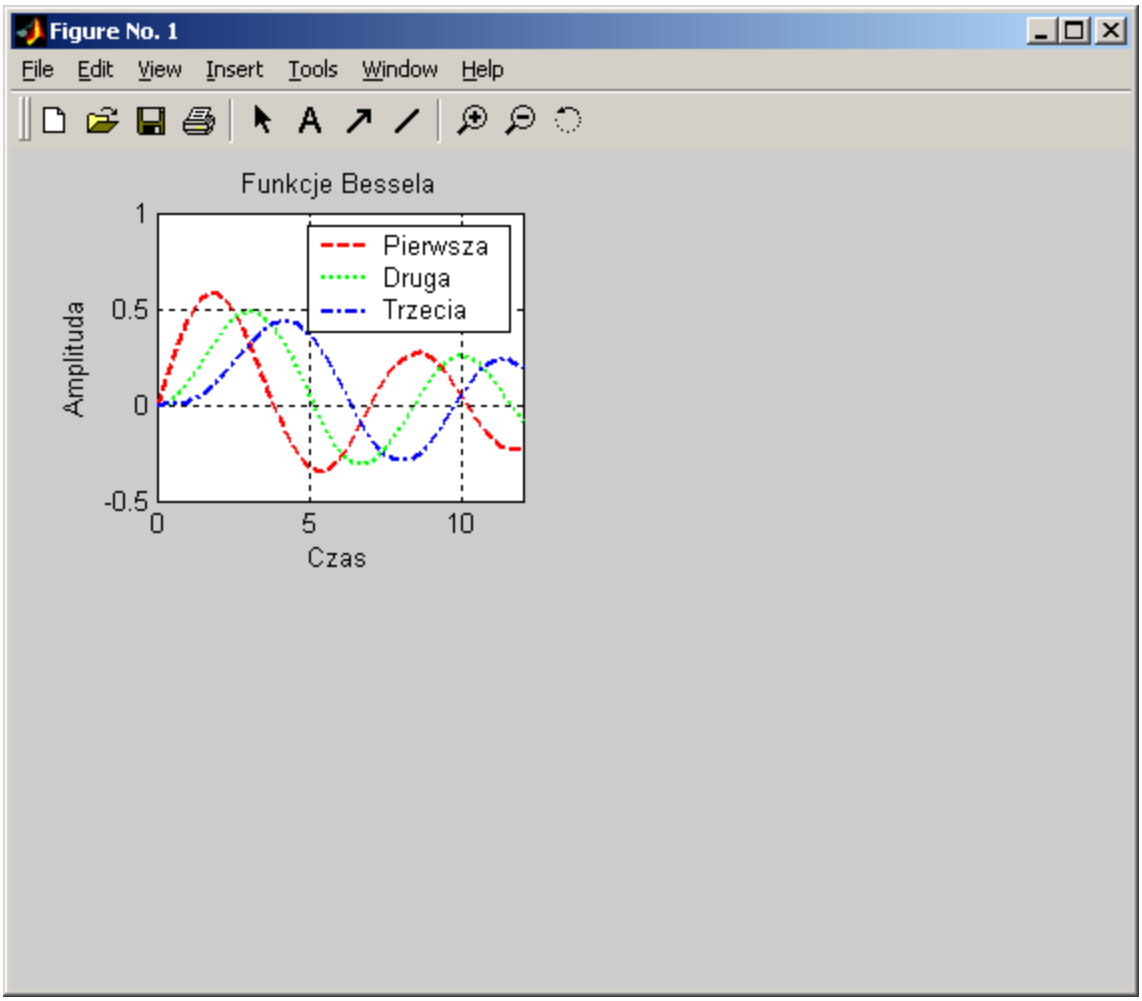
`nnz(S)` - zwraca ilość elementów niezerowych

Zajęcia nr 5:

Wizualizacja graficzna w MATLABie:

Przykład:
http://gagarin.hopto.org/public/mn/wiz01.m

```
%Pierwszy przyklad wizualizacyjny - Funkcje Bessela
x=0:0.2:12;
y1=bessel(1,x);
y2=bessel(2,x);
y3=bessel(3,x);
figure(1)                %otwiera okno graficzne
subplot(2,2,1)           %dzieli okno na 2 kolumny i 2
                         %wiersze i wybiera 1. podokno
h=plot(x,y1,x,y2,x,y3);  %rysuje 3 funkcje na jednym
                         %wykresie
set(h,'LineWidth',2,{'LineStyle'},{'--';':';'-.'})
                         %ustawia grubosc i styl linii
set(h,{'Color'},{'r';'g';'b'})
                         %oraz jej kolor
axis([0,12,-0.5,1])      %ustawia zakres osi wykresu
grid on                  %wlacza siatke
xlabel('Czas')           %nazywa osie
ylabel('Amplituda')
legend(h,'Pierwsza','Druga','Trzecia')
                         %dodaje legende
title('Funkcje Bessela') %oraz tytul
```



`plot(x1,y1,SL,x2,y2,SL,...)` - rysuje dowolną ilość funkcji podanych jako punkty xy na wykresie. Wykres każdej z funkcji może, ale nie musi być opisany przez string SL, stanowiący specyfikację linii. Specyfikacja linii może składać się z:

- określenia typu linii
- określenia typu znacznika punktów
- określenia koloru linii

Nie ma wymogu określania wszystkich trzech parametrów, można określić jedynie jedne lub swa z nich. Kolejne elementy można zdefiniować w dowolnej kolejności w stringu, bez

Typ linii może być jednym z danych typów: `-` `--` `:` `-.`

Kolor linii może być jednym z następujących:

`r`	czerwony
`g`	zielony
`b`	niebieski
`c`	cyjanowy
`m`	fioletowy
`y`	żółty
`k`	czarny
`w`	biały

Typ markera może być jednym z następujących typów:

`+`	krzyżyk
`o`	kólko
`*`	gwiazdka
`.`	punkt
`x`	krzyżyk
`s`	kwadrat
`d`	romb
`^`	trójkąt z wierzchołkiem w górę
`v`	trójkąt z wierzchołkiem w dół
`<`	trójkąt z wierzchołkiem w lewo
`>`	trójkąt z wierzchołkiem w prawo
`p`	gwiazda pięcioramienna
`h`	gwiazda sześcioramienna

np. `h=plot(x,y,'-.or')`

Polecenie axis - służy do zmiany osi na wykresie:

`axis([xmin,xmax,ymin,ymax])` - ustawia zakres osi zgodnie z podanymi parametrami

`axis auto` - domyślne dopasowanie osi do danych

`axis manual` - wyłącza autoprzeskalowanie wykresu

`axis tight` - ściśle dopasowuje do danych

`axis ij` - oś y rośnie w dół ekranu

`axis xy` - oś y rośnie w górę ekranu

`axis equal` - jednakowe jednostki na osiach

`axis image` - jednakowe jednostki na osiach i ścisłe dopasowanie wykresu do danych

`axis square` - obszar wykresu kwadratowy

`axis off` - wyłącza osie

`axis on` - włącza osie

Aby użyć autoskalowania częściowego a częściowo ustawić określony zakres należy użyć polecenia axis następująco:

`axis([-Inf,xmax,ymin,Inf])` - takie użycie zezwoli na autoskalowanie wartości minimalnej na osi x i wartości maksymalnej na osi y, pozostałe dwie wartości zostaną ustawione zgodnie z wartościami podanymi przez użytkownika

Do uzyskania wykresów o skalach nieliniowych służą polecenia:

`loglog` - osie x i y są logarytmiczne

`semilogx` - oś x logarytmiczna, oś y liniowa

`semilogy` - oś x liniowa, oś y logarytmiczna

Polecenia te można stosować wymiennie z poleceniem plot.

Mają one taką samą składnię.

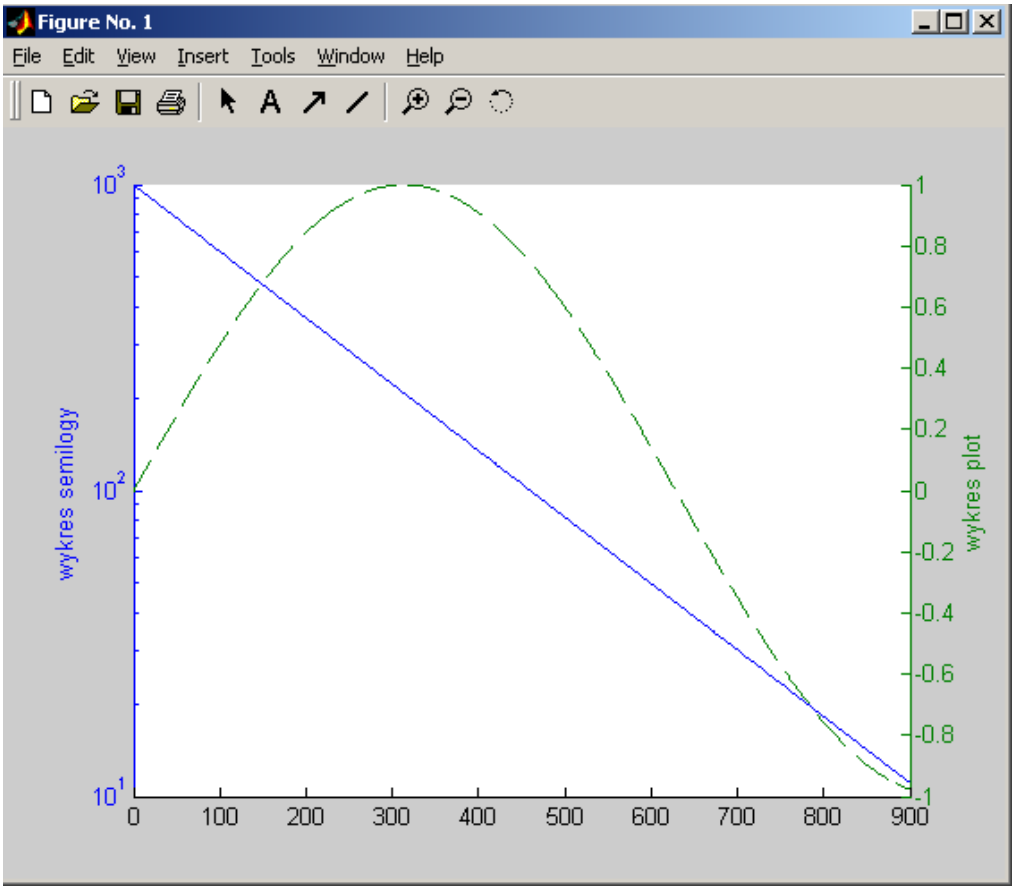
Polecenie `plotyy` - pozwala umieścić dwie osie y na jednym wykresie

```
plotyy(x1,y1,x2,y2)
plotyy(x1,y1,x2,y2,'typ1','typ2')
```

przykład:

http://gagarin.hopto.org/public/mn/wiz02.m

```
%Drugi przyklad wizualizacyjny - Funkcje plotyy, axes
t=0:900;
A=1000;
a=0.005;
b=0.005;
z1=A*exp(-a*t);
z2=sin(b*t);
[haxes,hline1,hline2]=plotyy(t,z1,t,z2,'semilogy','plot');
axes(haxes(1))
ylabel('wykres semilogy');
axes(haxes(2))
ylabel('wykres plot');
set(hline2,'LineStyle','--');
```



Polecenie:

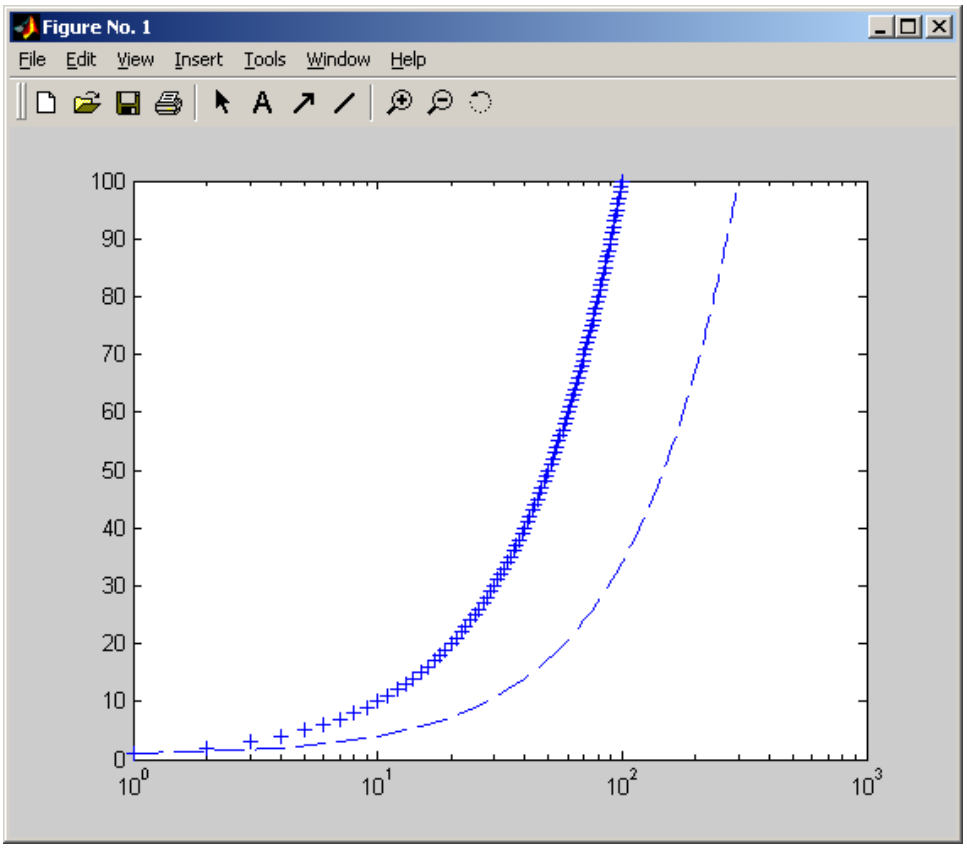
`hold on` - powoduje zachowanie całości wykresu i umożliwia dorysowanie kolejnych wykresów w danym oknie. Nie zmieniają się osie, skalowanie, itp.

`hold off` - przywraca normalny tryb

przykład:

http://gagarin.hopto.org/public/mn/wiz03.m

```
%Trzeci przyklad wizualizacyjny - Funkcja hold
semilogx(1:100,'+')
hold on
plot(1:3:300,1:100,'--')
hold off
```



`set(uchwyt_na_wykres, wlasciwosc, wartosc,...)`

- ustawia dla danego poprzez `uchwyt` wykresu `wlasciwosc` danego wykresu na `wartosc`

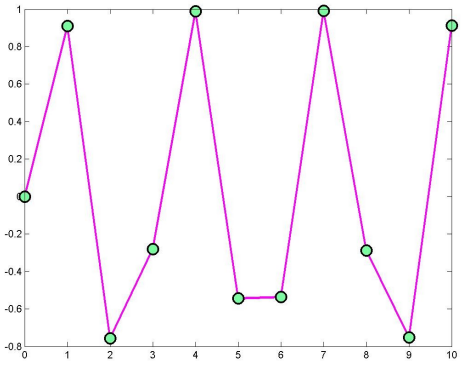
Właściwości mogą być następujące:

`'LineStyle'`	- typ kreski
`'Color'`	- kolor linii
`'Marker'`	- kształt znacznika
`'LineWidth'`	- grubość linii
`'MarkerEdgeColor'`	- kolor krawędzi znacznika
`'MarkerFaceColor'`	- kolor wypełnienia znacznika
`'MarkerSize'`	- wielkość znacznika

przykład:

http://gagarin.hopto.org/public/mn/wiz04.m

```
%Czwarty przyklad wizualizacyjny - Funkcja set
t=0:10;
h=plot(t,sin(2*t),'-mo');
set(h,'LineWidth',2)
set(h,'MarkerEdgeColor','k')
set(h,'MarkerFaceColor',[.49 1 .63])
set(h,'MarkerSize',12)
```



`bar(Y)` - tworzy wykres słupkowy dla danych z macierzy Y. Każda kolumna oznacza kolejną serię. Kolejne wiersze wewnątrz serii tworzą słupki w punktach 1, 2, ...

`bar(x,Y)` - analogicznie, ale słupki tworzone są w punktach wyznaczonych przez wartości z wektora x

`bar(...,width)` - pozwala dodatkowo określić względną szerokość słupków. Domyślnie ta wartość wynosi 0.8, co oznacza w praktyce, że słupki nie dotykają się, ale są między nimi wąskie przerwy. Ustawienie szerokości na 1, spowoduje stykanie się słupków

`bar(...,'style')` - styl: `'grouped'` - słupki ustawione jeden obok drugiego
styl: `'stacked'` - słupki ustawione jedne na drugim

`bar(...,LineSpec)` - pozwalana ustawienie koloru ‘r’, ‘g’, ‘b’,... itp tak jak przy funkcji plot

`h = bar(...)` - wszystkie wywołania funkcji zwracają uchwyt na wykres

`barh(...)` - dostępna jest również identyczna wersja z poziomymi słupkami

`h = barh(...)`

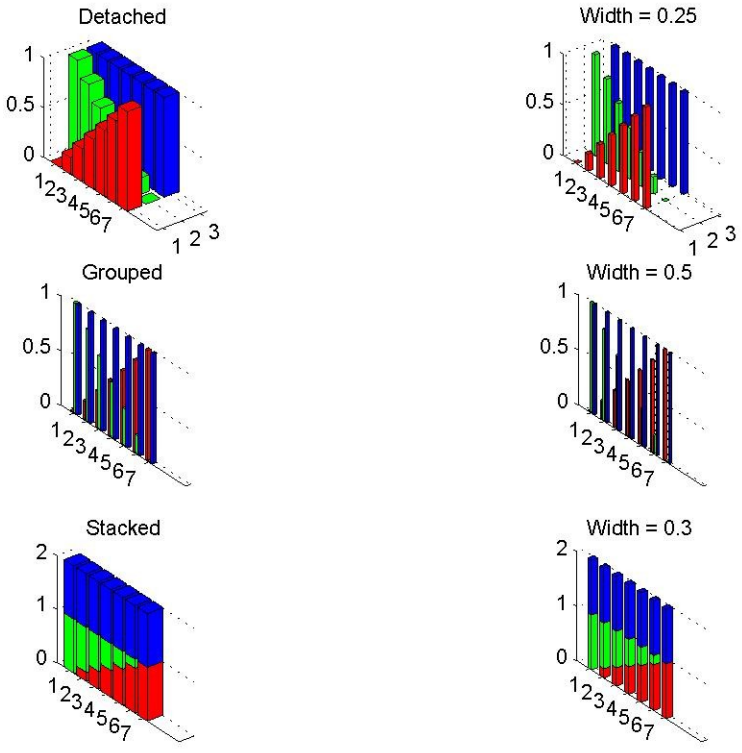
Istnieją również polecenie `bar3` i `bar3h` o identycznej składni, tworzące trójwymiarowe wykresy słupkowe. Jedyną różnicą jest istnienie jeszcze jednego dodatkowego stylu `'detached'` , umieszczającego słupki wzdłuż 3-ciej osi.

przykład:

http://gagarin.hopto.org/public/mn/wiz05.m

```
%Piaty przyklad wizualizacyjny - Funkcja bar3
Y = cool(7);
subplot(3,2,1)
bar3(Y,'detached')
title('Detached')
subplot(3,2,2)
bar3(Y,0.25,'detached')
```

```
title('Width = 0.25')
subplot(3,2,3)
bar3(Y,'grouped')
title('Grouped')
subplot(3,2,4)
bar3(Y,0.5,'grouped')
title('Width = 0.5')
subplot(3,2,5)
bar3(Y,'stacked')
title('Stacked')
subplot(3,2,6)
bar3(Y,0.3,'stacked')
title('Width = 0.3')
colormap([1 0 0;0 1 0;0 0 1])
```



`area(Y)` - tworzy wykres podobny do słupkowego z atrybutem ‘stacked’, z tą różnicą że łączy odcinkami kolejne punkty(słupki) i wypełnia obszary pod utworzonymi w ten sposób łamanymi

`area(x,Y)` - analogicznie jak w bar

`area(...,ymin)` - dolna granica wypełniania - domyślnie 0

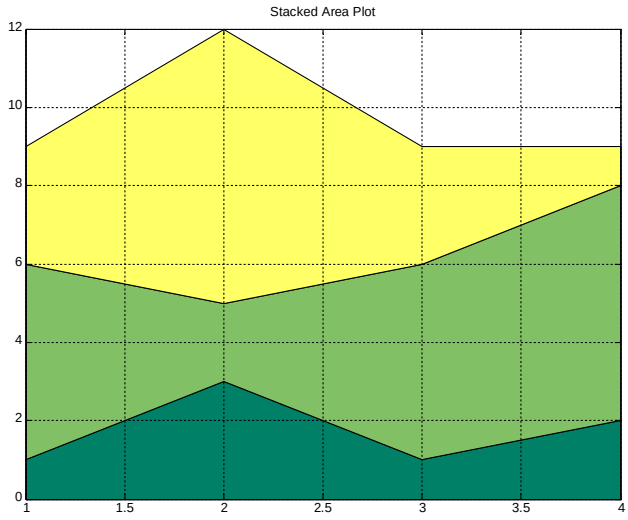
`area(...,'PropertyName',PropertyValue,...)` - możliwość ustawienia wartości jak w przypadku funkcji set

`h = area(...)` - zwraca uchwyt na wykres

przykład:

http://gagarin.hopto.org/public/mn/wiz06.m

```
%Szosty przyklad wizualizacyjny - Funkcja area
Y = [   1, 5, 3;
        3, 2, 7;
        1, 5, 3;
        2, 6, 1];
area(Y)
grid on
colormap summer
set(gca,'Layer','top')
title 'Stacked Area Plot'
```



`pie(X)` - tworzy diagram kołowy o wycinkach kołowych odpowiadających kolejnym wartościom wektora X. O ile `sum(X)>=1` narysowany zostanie pełne koło, w przypadku gdy `sum(X)<1` narysowana zostanie jedynie część koła.

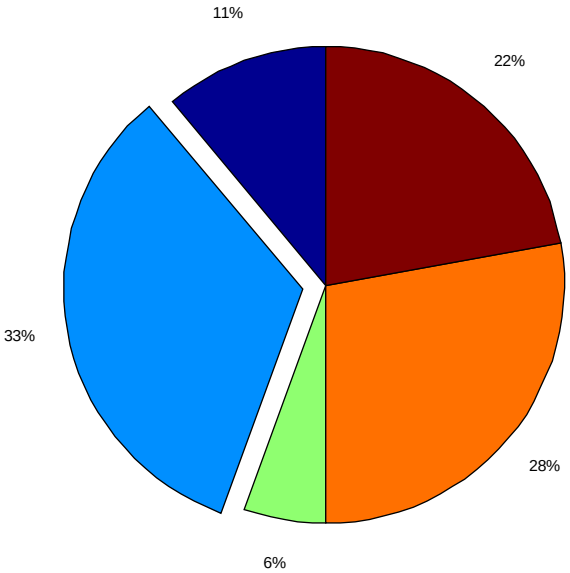
`pie(X,explode)` - jak wyżej, pozwala dodatkowo „wysuwać” wycinki kołowe poprzez zdefiniowanie wektora `explode`. Wartość 0 oznacza że odpowiedni wycinek nie będzie wysunięty, wartość niezerowa, spowoduje natomiast wysunięcie.

`h = pie(...)` - zwraca uchwyt na wykres

przykład:

http://gagarin.hopto.org/public/mn/wiz07.m

```
%Siodmy przyklad wizualizacyjny - Funkcja pie
x = [1 3 0.5 2.5 2];
explode = [0 1 0 0 0];
pie(x,explode)
colormap jet
```



`n = hist(Y)` - tworzy histogram zbioru Y(wektora), czyli zlicza ilości elementów wpadających do „woreczków” Domyślnie jest 10 woreczków - rozmieszczonych równomiernie wzdłuż osi x. Jeśli Y jest macierzą to rozważanych jest kilka serii niezależnie od siebie

`n = hist(Y,x)` - analogicznie tylko że tworzy length(x) woreczków o środkach w wartościach x

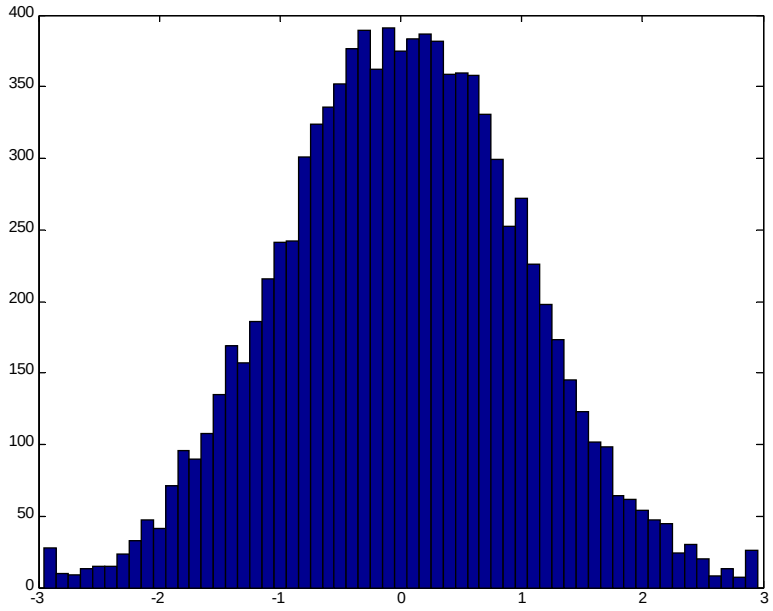
`n = hist(Y,nbins)` - tworzy nbins woreczków

`[n,xout] = hist(...)` - domyślnie zwraca wektor n, który zawiera ilości elementów jakie trafiły do poszczególnych woreczków, opcjonalnie może zwracać jeszcze wektor xout z położeniami środków woreczków.

przykład:

http://gagarin.hopto.org/public/mn/wiz08.m

```
%Osmy przyklad wizualizacyjny - Funkcja hist
x = -2.9:0.1:2.9;
y = randn(10000,1);
hist(y,x)
```



`rose(theta)` - analogiczny histogram kołowy

`rose(theta,x)`

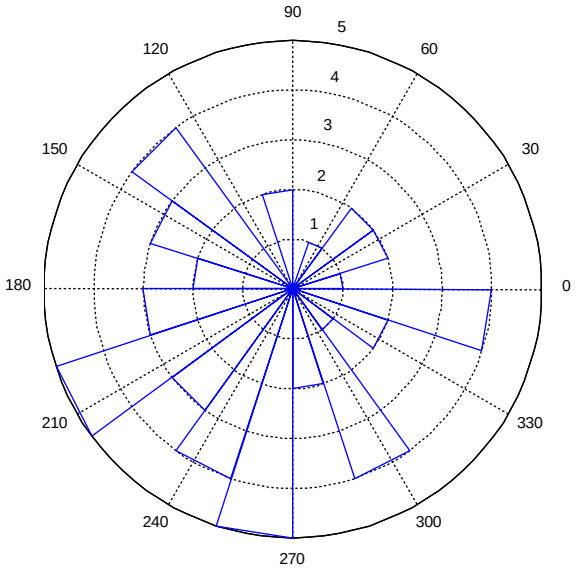
`rose(theta,nbins)`

`[tout,rout] = rose(...)`

przykład:

http://gagarin.hopto.org/public/mn/wiz09.m

```
%Dziewiaty przyklad wizualizacyjny - Funkcja rose
theta = 2*pi*rand(1,50);
rose(theta)
```



`stem(Y)` - rysuje wykres szpilkowy (domyślnie za oś X przyjmuje 1, 2, ...)

`stem(X,Y)`

`stem(...,'fill')` - przy włączonym parametrze główki szpilek są wypełniane

`stem(...,LineSpec)` - specyfikacja analogiczna, jak w przypadku plot

`h = stem(...)` - zwraca wektor uchwytów (od 6.13 są to uchwyty na kolejne serie):

h(1) - uchwyt na znaczniki (główki szpilek) % do wersji 6.13

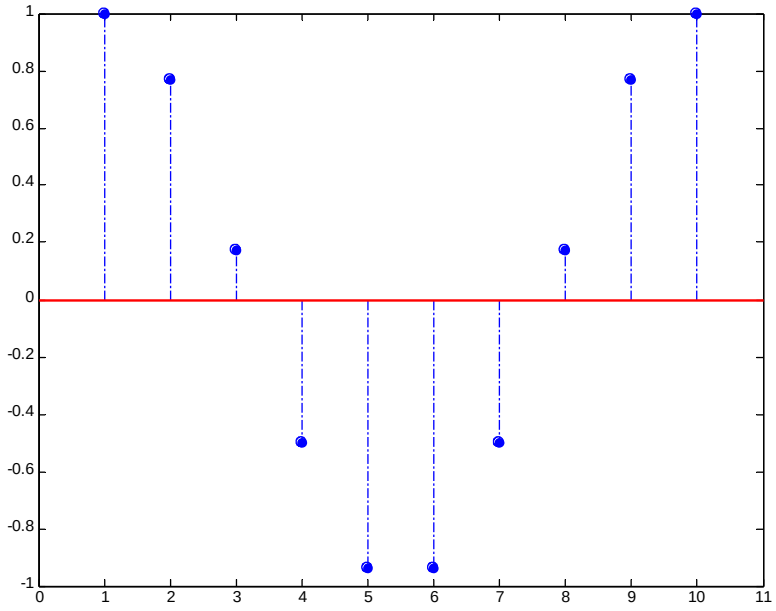
h(2) - uchwyt na nóżki szpilek % do wersji 6.13

h(3) - linia bazowa % do wersji 6.13

przykład:

http://gagarin.hopto.org/public/mn/wiz10.m

```
%Dziesiaty przyklad wizualizacyjny - Funkcja stem
y = linspace(0,2*pi,10);
h = stem(cos(y),'fill','-.');
set(h(3),'Color','r','LineWidth',2) % Zmienia linie bazowa
%uwaga powyzsza linijka dziala tylko do wersji 6.13 Matlaba
axis ([0 11 -1 1])
```



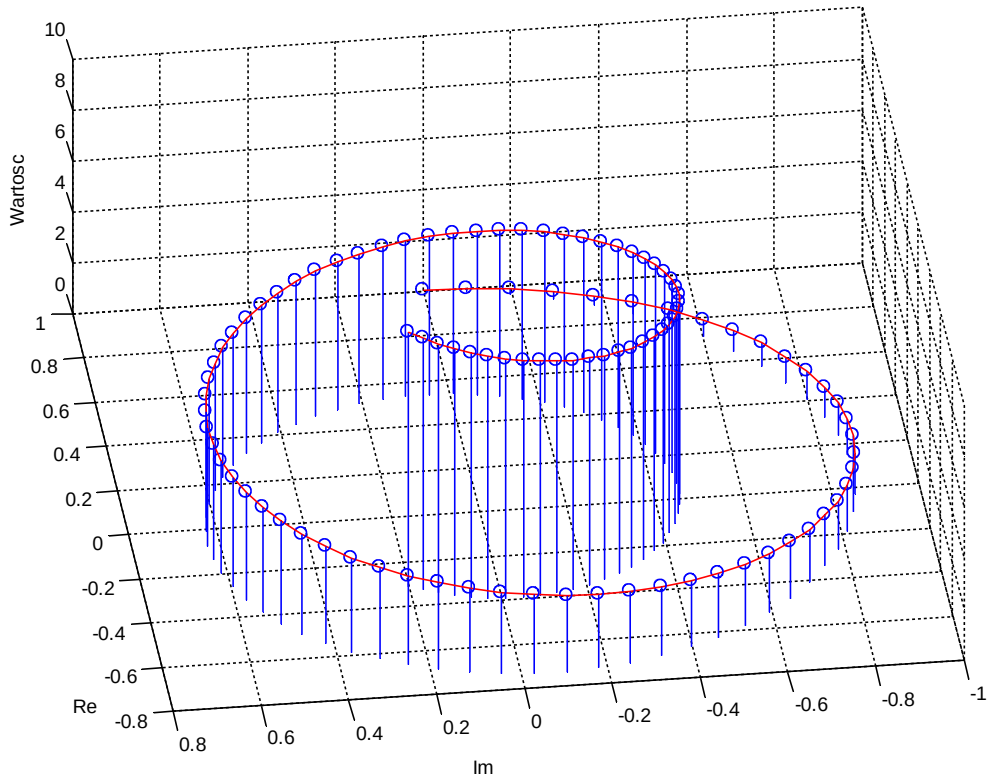
`stem3(Z)` - tworzy trójwymiarowy wykres szpilkowy. Jeśli X Y są niepodane domyślnie przyjmowane są X=1, 2, ..., Y=1, 2, ...w zależności od wymiarów macierzy Z (może być również wektorem)

```
stem3(X,Y,Z)
stem3(...,'fill')
stem3(...,LineSpec)
h = stem3(...)
```

przykład:

http://gagarin.hopto.org/public/mn/wiz11.m

```
%Jedenasty przyklad wizualizacyjny - Funkcja stem3
t=0:0.1:10;
s=0.1+i;
y=exp(-s*t);
stem3(real(y),imag(y),t)
hold on
plot3(real(y),imag(y),t,'r')
hold off
view([-39,5,62])
xlabel('Re')
ylabel('Im')
zlabel('Wartosc')
```



`stairs(Y)` - tworzy wykres schodkowy

`stairs(X,Y)`

`stairs(...,LineSpec)`

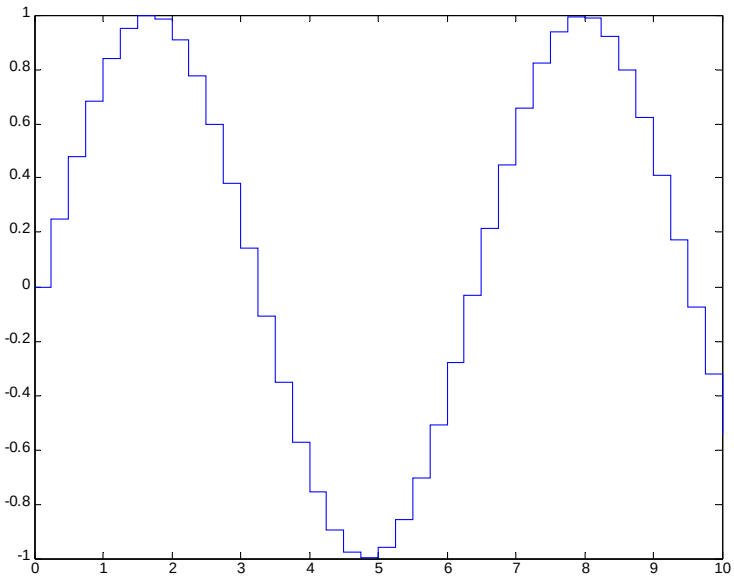
`[xb,yb] = stairs(Y)` - nie tworzy wykresu lecz zwraca wektory punktów xb, yb takie że `plot(xb,yb)` rysuje wykres schodkowy

`[xb,yb] = stairs(X,Y)`

przykład:

http://gagarin.hopto.org/public/mn/wiz12.m

```
%Dwunasty przyklad wizualizacyjny - Funkcja stairs
x = 0:.25:10;
stairs(x,sin(x))
```



Zajęcia nr 6:

Wizualizacja graficzna w MATLABie -cd:

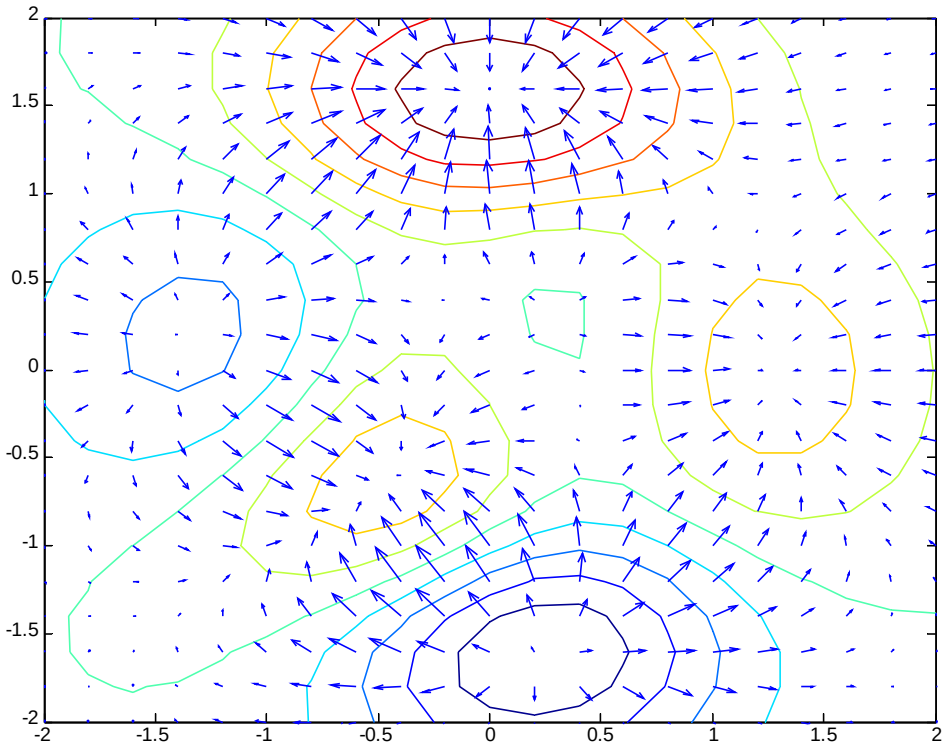
Złożone funkcje graficzne omówione zostaną na konkretnych przykładach, gdyż ilość możliwości wywołań każdej z nich i mnogość parametrów jest przytłaczająca.

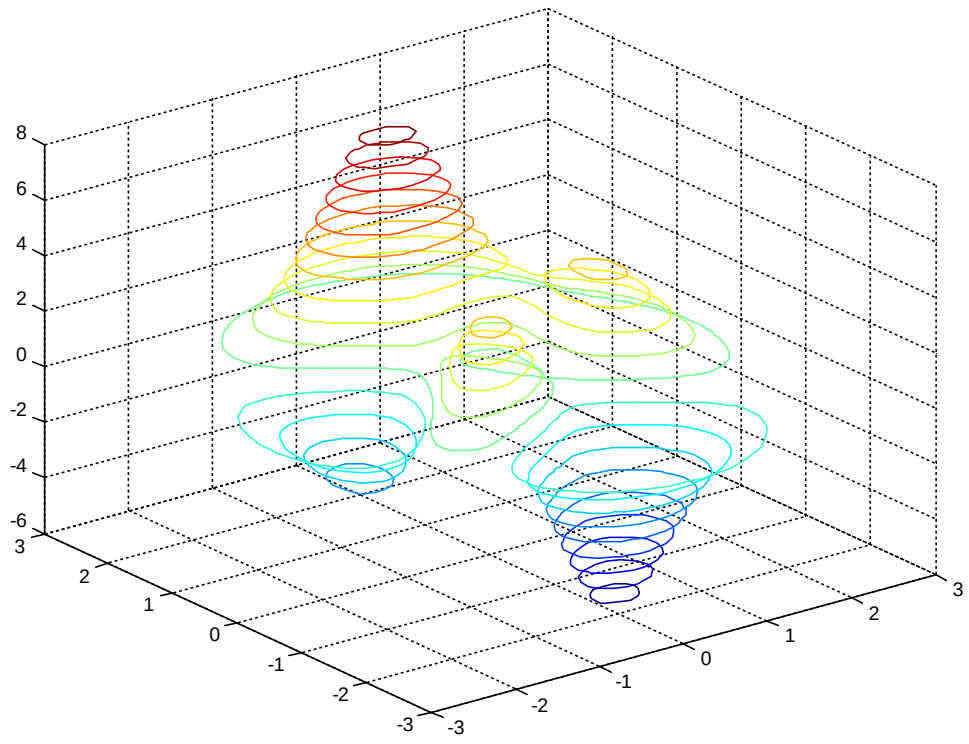
przykład:
http://gagarin.hopto.org/public/mn/wiz13.m

```
%Trzynasty przyklad wizualizacyjny
%Funkcje peaks, contour, gradient,quiver
n=-2.0:0.2:2.0;
[X,Y,Z]=peaks(n);   %oblicza gorki
contour(X,Y,Z,10);  %wykresla poziomice
[U,V]=gradient(Z,0.2);  %liczy gradient
hold on
quiver(X,Y,U,V);    %wykresla pole wektorowe gradientu
hold off
```

przykład:
http://gagarin.hopto.org/public/mn/wiz14.m

```
%Czternasty przyklad wizualizacyjny - Funkcja contour3
[X,Y,Z]=peaks;
contour3(X,Y,Z,20);
```

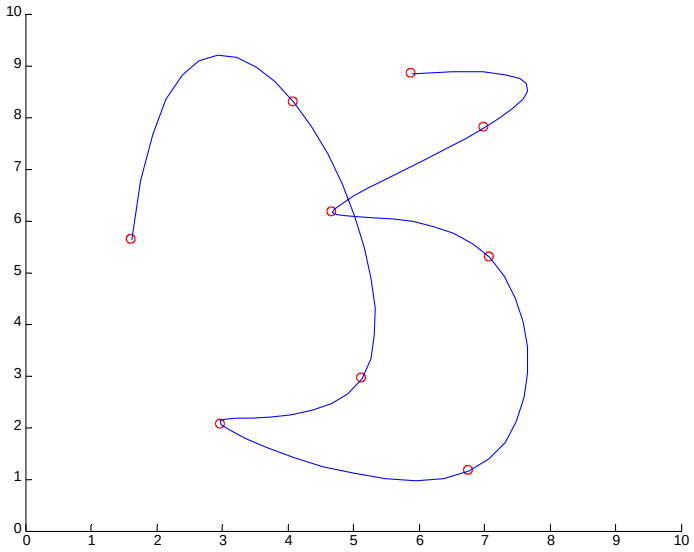

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


8
6
4
2
0
-2
-4
-6
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

przykład:

http://gagarin.hopto.org/public/mn/wiz15.m

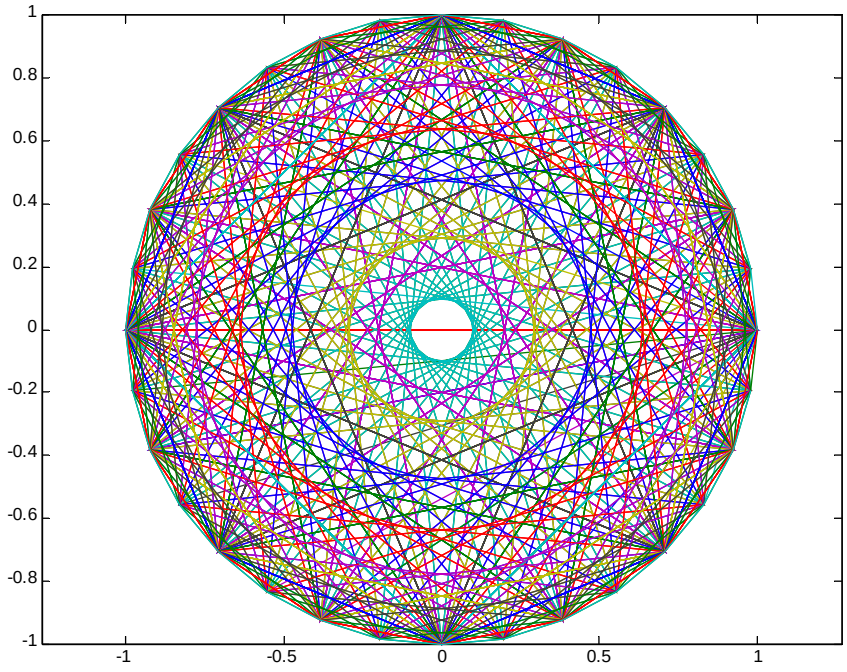
```
%Pietnasty przyklad wizualizacyjny - Funkcja ginput
axis([0 10 0 10])
hold on
xy=[];
n=0;
but=1;
while but==1
    [xi,yi,but]=ginput(1)
    plot(xi,yi,'ro')
    n=n+1;
    xy(:,n)=[xi;yi];
end
t=1:n;
ts=1:0.1:n;
xys=spline(t,xy,ts);
plot(xys(1,:),xys(2,:),'b-');
hold off
```



przykład:

http://gagarin.hopto.org/public/mn/wiz16.m

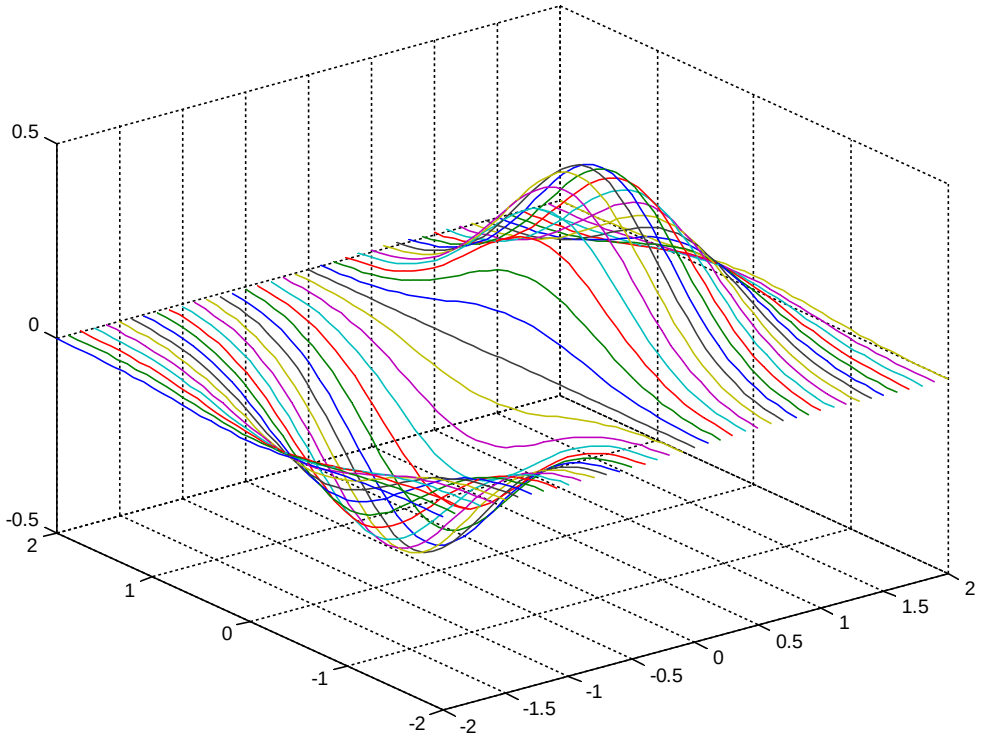
```
%Szesnasty przyklad wizualizacyjny - Funkcje getframe, movie
for k=1:16
    plot(fft(eye(k+16)))
    axis equal
    M(k)=getframe;
end
movie(M,30)
```



przykład:

http://gagarin.hopto.org/public/mn/wiz17.m

```
%Siedemnasty przyklad - Funkcje meshgrid, plot3
[X,Y]=meshgrid(-2:0.1:2);
Z=X.*exp(-X.^2-Y.^2);
plot3(X,Y,Z);
grid on
```



przykład:

http://gagarin.hopto.org/public/mn/wiz18.m

```
%Osiemnasty przyklad wizualizacyjny - Funkcje ...
Z=peaks(20);
figure(1)
subplot(2,1,2)
h=surf(Z);
pause(2)
colormap hsv
pause(2)
colormap cool
pause(2)
colormap summer
pause(2)
colormap gray
pause(2)
```

```
colormap jet
pause(2)
colormap hot
pause(2)
shading flat
pause(2)
shading faceted
pause(2)
shading interp
pause(2)
set(h,'EdgeColor','k')
pause(2)
light('Position',[-2,2,20])
pause(2)
lighting none
pause(2)
lighting flat
pause(2)
lighting gouraud
pause(2)
lighting phong
pause(2)
material([1.0,0.2,0.1,30])
pause(2)
material([0.3,0.9,0.7,30])
pause(2)
material([0.4,0.6,0.5,2])
pause(2)
material([0.4,0.6,0.5,120])
pause(2)
material([0.4,0.6,0.5,30])
pause(2)
```

```
set(h,'FaceColor',[0.7,0.7,0],'BackFaceLighting','lit');
pause(2)
view([30,25])
pause(2)
set(gca,'CameraViewAngleMode','Manual')
pause(2)
axis([5,15,5,15,-8,8])
pause(2)
set(gca,'ZTickLabel','Negative||Positive')
pause(2)
set(gca,'PlotBoxAspectRatio',[2.5,2.5,1])
```