dr Tomasz Ścięžor
Wydział Inżynierii Środowiska
Politechnika Krakowska

Podstawy programowania w języku MatLab

wg:

- R. Jankowski, I. Lubowiecka, W. Witkowski, Politechnika Gdańska, Wydział Inżynierii Lądowej, Gdańsk 2003
- M.Czajka „Ćwiczenia. MATLAB", wyd. Helion 2005
- W. Regel „Obliczenia symboliczne i numeryczne w programie MATLAB", wyd. MIKOM, Warszawa 2004

Wydanie III

Kraków 2015

Środowisko i programowanie w języku MATLAB

- MATLAB - pakiet obliczeniowy firmy *MathWorks* jest przeznaczony do wykonywania różnorodnych obliczeń numerycznych.
- Serce pakietu stanowi interpreter języka umożliwiający implementację algorytmów numerycznych oraz biblioteki podstawowych działań na macierzach (odwracanie, dodawanie/odejmowanie, wartości własne itp.).
- Podstawowym typem danych jest macierz, stąd nazwa MATrix LABoratory.
- Pakiet posiada obszerne biblioteki dodatkowych procedur umożliwiające rozwiązywanie typowych problemów obliczeniowych.
- Prosta budowa okienkowa ułatwia korzystanie z programu.
- Łatwa i estetyczna jest wizualizacja wyników w postaci dwu- i trójwymiarowych wykresów.
- Dodatkową zaletą pakietu MATLAB jest możliwość przeprowadzenia obliczeń symbolicznych (na wzorach).

Wprowadzenie do pracy w środowisku języka MATLAB

- Praca w środowisku języka MATLAB polega na wydawaniu poleceń, które po zatwierdzeniu wykonywane są przez interpreter.
- Większą liczbę instrukcji można zapisać w zbiorze tekstowym zwanym skryptem (pliki z rozszerzeniem *.m*).

Przykłady poleceń

- Podstawienie:

```
» a=3;
```

powoduje utworzenie zmiennej `a` o wartości 3.
UWAGA: Średnik po poleceniu powoduje, że wartość będąca wynikiem nie będzie wyświetlana na ekranie.

```
» b=sin(a)
b =
    0.1411
```

oblicza wartość funkcji sinus dla zmiennej *a*, wynik zapisuje do zmiennej *b* i wyświetla na ekranie.

- Jeżeli nie podano nazwy zmiennej to wynik działania jest umieszczany w standardowej zmiennej *ans*, np.:

```
 » cos(pi/3)
ans =
    0.5000
```

- Utworzona (zdefiniowana) zmienna jest pamiętana od momentu utworzenia, aż do chwili jej usunięcia. Możliwa jest przy tym nie tylko zmiana wartości, ale również rozmiaru zmiennej.

Nazwy zmiennych i informacje o nich można uzyskać wywołując funkcje `who` (wylicza zmienne) i `whos` (podaje nazwy, rozmiary, ilość zajmowanej pamięci i klasę zmiennych).

- Usunięcie zmiennej z pamięci:
  `clear a` - usuwa zmienną *a*;
  `clear` - usuwa wszystkie zmienne znajdujące się w pamięci.

- Zapisanie zmiennych na dysku:
  `save nazwa_pliku` (domyślnie przyjmowane jest rozszerzenie *.mat*).

- Wczytanie danych z pliku dyskowego:
  `load nazwa_pliku`

- Korzystanie z podręcznej pomocy podającej opis funkcji:
  `help nazwa_funkcji`

- Zawartość aktualnego katalogu można wyświetlić używając funkcji `dir` lub `ls`.

- Do zmiany katalogu służy polecenie:
  `cd nazwa_katalogu`

Liczby rzeczywiste i ich formaty

- Podstawowym typem dla elementów macierzy wykorzystywanym przez MATLAB są liczby rzeczywiste.
- Maksymalną i minimalną wartość liczby rzeczywistej dodatniej można poznać za pomocą funkcji `realmax` i `realmin`.

- Do określenia sposobu, w jaki liczby rzeczywiste są przedstawione na ekranie służy polecenie `format postać_liczby`, gdzie *postać_liczby* określa postać, w jakiej liczby rzeczywiste będą wyświetlane na ekranie:

`format short` – do 4 miejsca po przecinku
`format long` – do 14 miejsca po przecinku
`format short e` – do 4 miejsca po przecinku w zapisie cecha-mantysa
`format long e` – do 14 miejsca po przecinku w zapisie cecha-mantysa
`format short g` – do 4 miejsca po przecinku
`format long g` – wszystkie miejsca znaczące
`format hex` – w zapisie szesnastkowym
`format bank` – do 2 miejsc po przecinku
`format rat` – jako ułamek zwykły

Przykład:
Przedstaw liczbę 2,5 w różnej postaci używając funkcji format.

```
» format short
» 2.5
ans =
      2.5000
» format short e
» 2.5
ans =
      2.5000e+000
» format long
» 2.5
ans =
      2.500000000000000
```

Pomocne zmienne Matlaba

`pi` – wartość liczby π
`date, clock` – aktualna data i czas
`NaN` – wartość nieokreślona
`Inf` – nieskończoność

Liczby zespolone

Matlab bez problemu rozpoznaje liczby zespolone, i „wie”, jakiego typu jest nowa zmienna:

Przykład:

```
» x=2+4i
x =
   2.0000 + 4.0000i
» y=-3-12i
y =
  -3.0000 -12.0000i
» z=x+y
z =
  -1.0000 - 8.0000i
```

Macierze

Wszelkie wprowadzane i deklarowane dane (liczby, tekst) Matlab traktuje jako macierz – pojedyncza liczba jest traktowana jako macierz o wymiarze 1×1. Matlab wyróżnia następujące typy danych (począwszy od wersji 5.x):

- double – macierz pełna. Liczby są reprezentowane w formacie zmiennoprzecinkowym z podwójną precyzją. W Matlabie możemy wykonywać operacje arytmetyczne tylko na liczbach typu double.

  Przykład wywołania: a=1; A=[1,2,3]

- char – typ tekstowy jakim jest dowolny napis.

  Przykład wywołania: tekst='napis'

- sparse – macierz rzadka. Elementy zerowe macierzy nie są zapamiętywane w pamięci komputera. W przypadku macierzy o dużej liczbie zer (macierze diagonalne, macierze rzadkie etc.) oszczędzamy pamięć oraz uzyskujemy skrócenie czasu obliczeń.

  Przykład wywołania: A=sparce(1)

- struct – struktura. Tak jak w każdym języku programowania struktura jest typem danych zawierającym w swoich polach dane różnych typów.

  Przykład wywołania: a.skladnik=1

- cell – macierz komórkowa (blokowa) – pojedynczymi elementami takiej macierzy mogą być nie tylko liczby ale i dowolne dane z powyższych

typów. W macierzy komórkowej możemy przechować kilka macierzy o różnym wymiarze, tekst i strukturę jednocześnie.

Przykład wywołania: a={1}

- uint8 – 8-bitowy typ całkowity (zakres 0–255) przeznaczony do zapisywania w pamięci obrazów graficznych. Na tym typie danych nie można wykonywać żadnych operacji arytmetycznych.

Poniższe przykłady podane są dla macierzy pełnej:

- Definicja macierzy przez wyliczenie elementów:

Przykład:

```
» A=[2 2 2 1; 1 2 3 1];
```

lub:

```
» A=[2 2 2 1
1 2 3 1]
A =
    2 2 2 1 1 2 3 1
```

Poszczególne elementy macierzy oddziela się spacjami, a wiersze średnikami lub umieszcza się je w oddzielnych liniach.

- Definicja macierzy przez wygenerowanie elementów:
  `A=[min:krok:max]`

Polecenie generuje wektor poczynając od elementu o wartości *min*, kończąc na elemencie o wartości *max* z krokiem *krok*. Jeżeli parametr *krok* zostanie pominięty, przyjmuje się, iż *krok*=1.

Przykład:

Wygeneruj macierz dwuwierszową o wyrazach od 1 do 10 w pierwszym wierszu i o wyrazach od 2 do 20 (co 2) w wierszu drugim.

```
» A=[1:10; 2:2:20]
A =
    1  2  3  4  5  6  7  8  9 10
    2  4  6  8 10 12 14 16 18 20
```

- Definicja macierzy wykorzystując elementy innych macierzy:

Przykład:
Utwórz macierz **D** budując ją ze zdefiniowanych macierzy **A**, **B** i **C**.

```
» A=[1 4 1; 2 0 1];
» B=[3 1; 4 1];
» C=[1 2 2 0 1; 2 4 7 1 0];
» D=[A B; C]
D =
    1 4 1 3 1
    2 0 1 4 1
    1 2 2 0 1
    2 4 7 1 0
```

UWAGA:
Przy takim budowaniu macierzy należy pamiętać o zgodności wymiarów.

Wymiar i wyświetlanie macierzy

- `[n,m]=size(A)` – zwraca liczbę kolumn *n* i wierszy *m* macierzy **A**;
- `n=length(B)` – zwraca wymiar wektora **B** (lub większy z wymiarów macierzy **B**);
- `A lub disp(A)` – pokazuje macierz **A** na ekranie;

Funkcje wspomagające konstruowanie macierzy

- Definicja macierzy jednostkowej:

Przykład:
Utwórz kwadratową macierz jednostkową **A** o wymiarze 3×3.

```
» A=eye(3)
A =
    1 0 0
    0 1 0
    0 0 1
```

- Definicja macierzy wypełnionej jedynkami:

Przykład:
Utwórz macierz **A** o wymiarze 2×3 wypełnionej jedynkami.

```
» A=ones(2,3)
A =
    1 1 1
    1 1 1
```

- Definicja macierzy wypełnionej zerami:

Przykład:
Utwórz macierz **A** o wymiarze 3×2 wypełnionej zerami.

```
» A=zeros(3,2)
A =
    0 0 0
    0 0 0
```

Dostęp do elementów macierzy

- Odwołanie do elementów:

Przykład:

```
» A=[1 2 3; 0 9 8; 1 1 0]
A =
    1 2 3
    0 9 8
    1 1 0
```

» A(2,3) – odwołanie do elementu w wierszu 2 i kolumnie 3;

```
ans =
    8
```

» A(3,2)– odwołanie do elementu w wierszu 3 i kolumnie 2

```
ans =
    1
```

» A(:,2)– odwołanie do kolumny 2

```
ans =
2
9
1
```

» A(3,:)– odwołanie do wiersza 3

```
ans =
1 1 0
```

» A(:)– odwołanie do wszystkich danych w formie wektora

```
ans =
     1
     0
     1
     2
     9
     1
     3
     8
     0
```

- Wybór największego elementu

max(A) – zwraca największy element wektora **A**. W przypadku gdy **A** jest macierzą, zwraca wektor wierszowy, którego elementami są maksymalne elementy z każdej kolumny **A**

Przykład:

```
» max(A)
ans =
    1 9 8
```

- Wybór najmniejszego elementu

min(A) – zwraca najmniejszy element wektora **A**. W przypadku gdy **A** jest macierzą, zwraca wektor wierszowy, którego elementami są maksymalne elementy z każdej kolumny **A**

Przykład:

```
» min(A)
ans =
    0 1 0
```

- Obliczanie wartości średniej elementów

`mean(A)` – zwraca średnią arytmetyczną elementów wektora **A**. W przypadku gdy **A** jest macierzą, zwraca wektor wierszowy, którego elementami są średnie arytmetyczne elementów z każdej kolumny **A**

Przykład:

```
» mean(A)
ans =
    0.6667 4.0000 3.6667
```

- Odwołanie do podmacierzy

Przykład:

```
» A=[1 2 3 4 5 6; 0 9 8 7 6 5; 1 1 0 0 2 2]
A =
    1 2 3 4 5 6
    0 9 8 7 6 5
    1 1 0 0 2 2
```

`» B=A(:,[1:3 5])` – utworzenie macierzy **B** poprzez pobranie z macierzy **A** kolumn: 1-3 oraz 5

```
B=
    1 2 3 5
    0 9 8 6
    1 1 0 2
```

`» B=A([1 3],1:2:5)` – utworzenie macierzy **B** z elementów macierzy **A** leżących na przecięciu wierszy 1 i 3 z kolumnami 1, 3 i 5

```
B=
    1 3 5
    1 0 2
```

- Usuwanie wektora z macierzy:

Przykład:

```
» A=[1 2 3 4; 4 5 6 7]
A =
    1 2 3 4
    4 5 6 7
```

» `A(2,:)=[ ]` – usuwa drugi wiersz z macierzy **A**

```
A =
    1 2 3 4
```

» `A(:,1:2)=[ ]` - usuwa dwie pierwsze kolumny z macierzy **A**

```
A =
    3 4
```

Działania na macierzach

- Suma i różnica macierzy

Przykład:

Zdefiniuj dwie macierze **A** i **B**, a następnie oblicz ich sumę, różnicę oraz dodaj do elementów macierzy **A** liczbę 2.

Definicja macierzy:

```
» A=[1 -1 2; -2 3 1]
A =
    1 -1 2 -2 3 1
» B=[1 1 1; 0 -2 2]
B =
    1  1  1
    0 -2  2
```

Suma:

```
» A+B
ans =
     2  0  3
    -2  1  3
```

Różnica:

```
» A-B
ans =
     0 -2  1
    -2  5 -1
```

Dodanie do elementów macierzy **A** liczby 2:

```
» A+2
ans =
    3   1   4
    0   5   3
```

- Mnożenie macierzy

Przykład:

Zdefiniuj dwie macierze **A** i **B**, a następnie oblicz ich iloczyn oraz pomnóż elementy macierzy **A** przez 2.

Definicja macierzy:

```
» A=[1 1 0; 2 1 1]
A =
    1   1   0
    2   1   1
» B=[2; 2; 2]
B=
    2
    2
    2
```

Iloczyn macierzowy:

```
» A*B
ans =
    4
    8
```

Iloczyn macierzy przez liczbę:

```
» A*2
ans =
    2   2   0
    4   2   2
```

- Odwracanie i transpozycja

Przykład:

Zdefiniuj macierz **A**, a następnie wyznacz macierz odwrotną do niej i dokonaj transpozycji.

```
» A=[1 2 3; 0 9 8; 3 4 7]
A =
    1   2   3
    0   9   8
    3   4   7
```

»`inv(A)` – zwraca macierz odwrotną do **A**

```
ans =
   -15.5000 1.0000  5.5000
   -12.0000 1.0000  4.0000
   13.5000 -1.0000 -4.5000
```

» `A'` – transponuje macierz **A**

```
ans =
    1   0   3
    2   9   4
    3   8   7
```

Przykład

Zdefiniuj wektor kolumnowy **A**, a następnie oblicz sumę kwadratów elementów tego wektora.

```
» A=[1 2 3]'
A =
    1
    2
    3
» A'*A
ans =
    14
```

Działania tablicowe

Działanie tablicowe jest działaniem, które przekształca poszczególne elementy macierzy oddzielnie.

Przykład:
Zdefiniuj dwie macierze **A** i **B**, a następnie wykonaj działania mnożenia, dzielenia i potęgowania tablicowego.

```
» A=[5 -6 2; -2 4 1]
A =
     5  -6   2
    -2   4   1
» B=[5 2 2; -1 -2 1]
B =
     5  2  2
    -1 -2  1
```

Mnożenie tablicowe:

```
» A.*B
ans =
    25 -12  4
     2  -8  1
```

Dzielenie tablicowe:

```
» A./B ans =
    1 -3  1
    2 -2  1
```

Potęgowanie tablicowe (podniesienie elementów macierzy **A** do drugiej potęgi):

```
» A.^2
ans =
    25 36  4
     4 16  1
```

Algebra liniowa

- `det(A)` – obliczanie wyznacznika macierzy **A**
- `eig(A)` – obliczanie wartości własnych macierzy **A**
- `poly(A)` – obliczanie współczynników wielomianu charakterystycznego macierzy **A**
- `rank(A)` – obliczanie rzędu macierzy **A**
- `diag(A)` – wyznaczanie elementów leżących na głównej przekątnej macierzy **A**

Przykład: Zdefiniuj macierz **A** o wymiarze 4x4, a następnie wyznacz jej wyznacznik, wartości własne, współczynniki wielomianu charakterystycznego oraz zbadaj rząd macierzy.

```
» A=[1 3 0 -2; 2 0 3 -1; 0 5 0 0; 1 0 2 0];
» det(A)
ans =
    0
» eig(A)
ans =
   -4.5414
    4.0000
    1.5414
    0.0000
» poly(A)
ans =
    1.0000 -1.0000 -19.0000 28.0000 0.0000
» rank(A)
ans =
    3
```

Przykład:

Rozwiąż układ równań liniowych:

$$\begin{cases} x+2y-z=3 \\ 3x-4y+2z=-5 \\ 5x-2y+3z=2 \end{cases}$$

UWAGA: Układ ten można zapisać w postaci macierzowej: **A**·**X**=**B**, gdzie:

$$A=\begin{bmatrix} 1 & 2 & -1 \\ 3 & -4 & 2 \\ 5 & -2 & 3 \end{bmatrix}, \quad X=\begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad B=\begin{bmatrix} 3 \\ -5 \\ 2 \end{bmatrix}$$

dla której rozwiązanie ma postać: $\mathbf{X}=\mathbf{A}^{-1}\cdot\mathbf{B}$

```
» A=[1 2 -1; 3 -4 2; 5 -2 3];
» B=[3 -5 2]';
» X=inv(A)*B
X =
    0.2000
    2.3500
    1.9000
```

Operacje na łańcuchach

- Uzupełniającym typem danych w języku MATLAB jest typ łańcuchowy (tekstowy). Do definiowania zmiennej tego typu stosuje się apostrofy, np.:

```
» s='MATLAB'
s =
MATLAB
```

- Na zmiennych typu łańcuchowego można dokonywać niektórych działań macierzowych, na przykład transpozycji:

```
» s'
ans =
M
A
T
L
A
B
```

- Zmienna typu łańcuchowego może zawierać nazwę instrukcji, którą można wykonać używając funkcji `eval`.

  Przykład:

  ```
  » t=[0:0.2:1];
  » s='sin(t)';
  » eval(s)
  ans =
    0 0.1987 0.3894 0.5646 0.7174 0.8415
  ```

- Można wysyłać na ekran wywołanie zachęty oczekujące na wprowadzenie przez użytkownika danej wartości lub łańcucha znaków, np.:

  ```
  » a=input('Podaj wartość a: ')
  Podaj wartość a:
  ```

  lub:

  ```
  » wzor=input('Podaj wzór funkcji f(x): ','s')
  Podaj wzór funkcji f(x):
  ```

  UWAGA:

  Użycie parametru *'s'* w funkcji `input` powoduje, iż wprowadzona dana jest traktowana jako łańcuch znaków.

Skrypty

- ***Przykład:***

Napisz skrypt (otwierając z menu *File* z opcji *New* plik *M-file*), który kreśli wykres wybranej przez użytkownika funkcji jednej zmiennej w przedziale od 0 do 4π.

```
% skrypt rysuje wykres wybranej funkcji
x=[0:0.1:4*pi];
wzor=input('Podaj wzór funkcji f(x): ','s')
y=eval(wzor);
plot(x,y); % kreślenie wykresu funkcji y=f(x)
```

Zapisz go pod nazwą *wykres.m*, a następnie uruchom wpisując w oknie komend jego nazwę:

```
» wykres
```

WSKAZÓWKA:
Podaj na przykład funkcję: `sin(x)+2*cos(2*x)`

Operatory logiczne

- Operatory logiczne w języku MATLAB:

==	równe
~=	różne
<	mniejsze
>	większe
<=	mniejsze równe
>=	większe równe
&	i
|	lub

Funkcje matematyczne

`sin(x)`	sinus
`cos(x)`	cosinus
`tan(x)`	tangens
`cot(x)`	cotangens
`asin(x)`	arcus sinus
`acos(x)`	arcus cosinus
`atan(x)`	arcus tangens
`sinh(x)`	sinus hiperboliczny
`cosh(x)`	cosinus hiperboliczny
`tanh(x)`	tangens hiperboliczny
`asinh(x)`	arcus sinus hiperboliczny
`acosh(x)`	arcus cosinus hiperboliczny
`atanh(x)`	arcus tangens hiperboliczny
`sqrt(x)`	pierwiastek kwadratowy
`exp(x)`	e^x
`log(x)`	logarytm naturalny
`log2(x)`	logarytm przy podstawie 2
`log10(x)`	logarytm przy podstawie 10

Funkcje związane z obliczeniami w dziedzinie liczb zespolonych

`abs(x)`	macierz modułów elementów macierzy x
`angle(x)`	macierz argumentów elementów macierzy x
`real(x)`	macierz części rzeczywistych elementów macierzy x
`imag(x)`	macierz części urojonych elementów macierzy x
`conj(x)`	macierz o elementach sprzężonych z elementami macierzy x

Funkcje dodatkowe

`abs(x)`	podaje wartość bezwzględną liczby
`ceil(x)`	zaokrąglanie w górę
`floor(x)`	zaokrąglanie w dół
`fix(x)`	zaokrąglanie zbliżające do zera
`round(x)`	zaokrągla elementy macierzy x do najbliższej liczby całkowitej
`rand(x)`	tworzy macierz o wymiarze x wypełnioną liczbami losowymi od 0 do 1

`rem(x,y)`	oblicza resztę z dzielenia odpowiadających sobie elementów macierzy x i y
`sum(x,1)`	sumuje elementy w kolumnach macierzy
`sum(x,2)`	sumuje elementy w wierszach macierzy
`gcd(a,b)`	oblicza największy wspólny dzielnik liczb a i b
`lcm(a,b)`	oblicza najmniejszą wspólną wielokrotną liczb a i b

Instrukcje sterujące

- Pętla FOR („dla”):

for `zmienna_iterowana = macierz_wartości`
`ciąg_instrukcji`
end

Działanie pętli polega na wykonaniu *ciągu_instrukcji* dla kolejnych wartości *zmiennej_iterowanej*. Wartościami tymi są kolejne wektory kolumnowe pobrane z *macierzy_wartości* (jeżeli jest to wektor, to kolejno zostaną wykonane instrukcje dla danych elementów tego wektora).

Przykład:
Napisz skrypt, który generuje wektor **A** o wymiarze 1×5, którego elementy spełniają zależność:

$$A_i = \sqrt{1+i}$$

```
%Próba realizacji pętli FOR
for i=1:5
A(i)=sqrt(1+i);    % pierwiastek kwadratowy
end
A
```

Rozbuduj powyższy skrypt, aby generował macierz **A** o wymiarze 10×5, którego elementy spełniają zależność:

$$A_i = \sqrt{1+\frac{i}{j}}$$

```
%Próba realizacji pętli FOR
for i=1:10
    for j=1:5
        A(i,j)=sqrt(1+i/j); %pierwiastek kwadratowy
    end
end
A
```

- Pętla WHILE („dopóki”):

```
while wyrażenie_warunkowe
  ciąg_instrukcji
end
```

Działanie pętli polega na wykonaniu *ciągu_instrukcji* dopóki *wyrażenie_warunkowe* jest spełnione.

Przykład:

```
% Próba realizacji pętli WHILE
i=0;
while i<100
    i=i+1
end
```

Instrukcja warunkowa IF („jeżeli”):

```
if wyrażenie_warunkowe1
    ciąg_instrukcji1
elseif wyrażenie_warunkowe2
    ciąg_instrukcji2
else
    ciąg_instrukcji3
end
```

Działanie instrukcji jest następujące: Jeżeli *wyrażenie_warunkowe1* jest spełnione, to wykonywany jest *ciąg_instrukcji1,* w przeciwnym razie sprawdzane jest *wyrażenie_warunkowe2,* jeżeli jest ono spełnione wykonywany jest *ciąg_instrukcji2,* jeżeli nie, wykonywany jest *ciąg_instrukcji3.* Instrukcję warunkową IF można rozbudować dla większej liczby *wyrażeń_warunkowych* i odpowiadających im *ciągów_instrukcji.*

Przykład:
Napisz skrypt używając instrukcji warunkowej IF do zrealizowania wyboru jednej z dostępnych opcji (polecenie menu):

```
% Próba realizacji instrukcji IF
o=menu('Przykładowe menu', 'Opcja 1', 'Opcja 2',
'Opcja 3');
if (o==1)
    disp('Opcja 1')
elseif (o==2)
    disp('Opcja 2')
elseif (o==3)
    disp('Opcja 3')
end
```

Instrukcja wyboru `switch`:

```
switch wyrażenie
case wyrażenie 1
    blok_poleceń 1
case (wyrażenie 2, wyrażenie 3, wyrażenie 4)
    blok_poleceń 2
otherwise
    blok_poleceń 3
end
```

Przykład:

```
» x=3;
» switch x
    case 1
        disp('Odp A')
    case 2
        disp('Odp B')
    case 3
        disp('Odp C')
    otherwise
        disp('Brak odpowiedzi')
    end
Odp C
```

Funkcje

W języku MATLAB istnieje możliwość definiowania własnych funkcji, jako elementów strukturalnych programu. Definicja funkcji ma następującą postać:

```
function[wartość_funkcji]=nazwa_funkcji(argument1,..,argumentN)
ciąg instrukcji
```

Przykład:

Napisz funkcję (otwierając z menu *File* z opcji *New* plik *M-file*) wyznaczającą wartość silni *n!*, gdzie *n* jest liczbą naturalną.

```
% Funkcja wyznacza wartość n!
function[wynik]=silnia(n)
wynik=1;
for i=1:n
    wynik=wynik*i;
end
```

Wywoływanie, np. 5!:

```
» silnia(5)
ans =
    120
```

Obliczenia symboliczne

Rozpoczynając pracę na zmiennych symbolicznych zawsze należy pamiętać o zadeklarowaniu zmiennych. Służy do tego polecenie:

```
syms arg1 arg2 arg3
```

- **Granice ciągów i funkcji**

Do obliczania granic na podstawie wyrażenia symbolicznego służy instrukcja `limit`, której składnia może być następująca:

`limit(F,zmienna,a)` – pozwala wyznaczyć granicę dla wyrażenia symbolicznego F, względem wskazanej zmiennej, w punkcie x = a.

`limit(F)` – pozwala wyznaczyć granicę dla wyrażenia symbolicznego F, względem wskazanej zmiennej, w punkcie x = 0

`limit(F,zmienna,a,'left')` – pozwala wyznaczyć granicę lewostronną dla wyrażenia symbolicznego F

`limit(F,zmienna,a,'right')` – pozwala wyznaczyć granicę prawostronną dla wyrażenia symbolicznego F

Przykład:

$$\lim_{n\to\infty}\frac{1-3n}{1+n}$$

```
» syms n
» limit((1-3*n)/(1+n),inf)
 ans =
 -3
```

- **Obliczanie pochodnych funkcji**

Aby obliczyć pochodne funkcji, posłużymy się poleceniem `diff`. Jego parametrami powinna być funkcja, której pochodna będzie liczona, oraz – opcjonalnie – zmienna, po której owa pochodna będzie liczona.

Przykład:

– Policzyć pochodną funkcji $f(x)=x^2$:

```
» syms x
» diff(x^2)
 ans =
       2*x
```

– Policzyć pochodną funkcji $f(x,y,z)=(xyz)^x+\left(\frac{1}{xy}\right)^2$:

```
» syms x y z
» diff((x*y*z)^x+(1/(x*y))^2)
ans =
    (x*y*z)^x*(log(x*y*z)+1)-2/x^3/y^2
» diff((x*y*z)^x+(1/(x*y))^2,x)
ans =
    (x*y*z)^x*(log(x*y*z)+1)-2/x^3/y^2
» diff((x*y*z)^x+(1/(x*y))^2,y)
ans =
    (x*y*z)^x*x/y-2/x^2/y^3
» diff((x*y*z)^x+(1/(x*y))^2,z)
ans =
    (x*y*z)^x*x/z
```

- **Całkowanie funkcji**

W Matlabie można obliczać całki za pomocą polecenia `int`. Jego argumentem powinna być funkcja, której całkę chcemy obliczyć, oraz opcjonalnie zmienna całkowania oraz granice całkowania.

Przykład:
– Oblicz całkę funkcji f(a,b)=a+b

```
» syms a b
» int(a+b)
ans =
    a*b+1/2*b^2
» int(a+b,a)
ans =
    1/2*a^2+a*b
» int(a+b,b)
ans =
    a*b+1/2*b^2
```

– Policz całkę $\int_1^3 x^2 dx$

```
» syms x
» int(x^2,1,3)
ans =
    26/3
```

– Policz całkę f(x,y)=x^2+y+1 w przedziale całkowania od -3 do 3.

```
» syms x y
» int(x^2+y+1,x,-3,3)
ans =
    24+6*y
» int(x^2+y+1,y,-3,3)
ans =
    6*x^2+6
```

Budowa strukturalna programu

- Skrypty, które stanowią większą całość nazywamy programami.
- W skrypcie możemy wywołać istniejące już (wcześniej zdefiniowane) inne skrypty lub funkcje.
- Polecenie help nazwa_skryptu wyświetla na ekranie tekst umieszczony w pierwszych liniach komentarza.

Przykład:
Napisz program, który wypisuje na ekranie informację o jego działaniu oraz imię i nazwisko autora, a następnie wyznacza wartość *n!* dla podanej przez użytkownika wartości *n*. (Uwaga: użyta w poniższym przykładzie funkcja round(n) zaokrągla liczbę rzeczywistą *n* do liczby całkowitej)

```
% Program oblicza wartość silni n! dla wprowadzonej
przez
% użytkownika wartości n
disp('Program oblicza wartość silni n! dla
wprowadzonej przez') disp('użytkownika wartości n')
disp(' ')
disp('Autor:')
disp('Imię i Nazwisko')
disp(' ')
n=input('Podaj wartość n: ');

%sprawdzenie czy n jest liczbą naturalną
while n<0 | n~=round(n)
    disp('Proszę podać liczbę naturalną')
    n=input('Podaj wartość n: ');
end

disp('Wartość n! wynosi:')
silnia(n)
```

Grafika dwuwymiarowa

- Najczęściej spotykanym sposobem graficznej prezentacji danych w języku MATLAB jest wykres funkcji jednej zmiennej. Służy do tego funkcja `plot(x,y)`, gdzie y=f(x);
- Okno graficzne można wyczyścić wywołując funkcję `clf;`
- Zamknięcie okna graficznego odbywa się poprzez wywołanie funkcji `close;`
- Dodatkowe okna można otworzyć przy pomocy funkcji `figure;`
- Otworzyć jak i zamknąć można dowolne okno podając jego numer jako argument;
- W celu uzyskania kilku wykresów w jednym oknie należy wykorzystać funkcję `subplot(m,n,p)`, gdzie:
  m – liczba wykresów w pionie;
  n – liczba wykresów w poziomie;
  p – kolejny numer wykresu.
- Skala wykresu dobierana jest automatycznie. Chcąc ją zmienić, trzeba wywołać funkcję `axis([xmin xmax ymin ymax])` i jako argument podać wektor określający nowe parametry osi.

Wykres można opisać podając nazwy zmiennych, tytuł, itp.

`title('tekst')` – tytuł rysunku;

`xlabel('tekst')` – opis osi x;

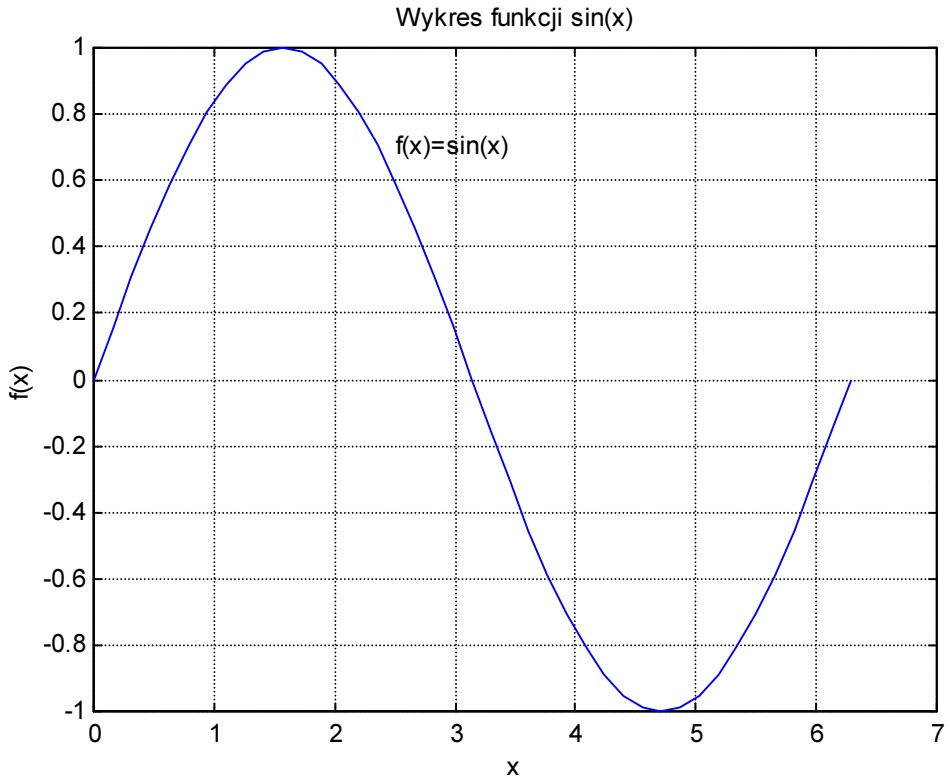
`ylabel('tekst')` – opis osi y;

`text(x,y,'tekst')` - umieszcza *'tekst'* w dowolnym punkcie o współrzędnych (x,y);

`grid` - włącza lub wyłącza siatkę;

Przykład:
Napisz skrypt kreślący przykładowy wykres wraz z opisem.

```
% Skrypt kreśli przykładowy wykres wraz z opisem
x=[0:pi/20:2*pi];
y=sin(x);
plot(x,y)
title('Wykres funkcji sin(x)')
xlabel('x')
ylabel('f(x)')
text(2.5,0.7,'f(x)=sin(x)')
grid
```



Rysowanie

- Istnieją funkcje pozwalające na tworzenie dowolnych rysunków z linii i wielokątów.
  `line(x,y)` – rysuje linię łamaną łącząc wierzchołki punktów wyznaczonych przez elementy wektorów **x** i **y**;
  `fill(x,y,'c')` – rysuje wielokąt o wierzchołkach w punktach wyznaczonych przez elementy wektorów **x** i **y** wypełniony kolorem określonym przez argument *c* według poniższego opisu kolorów:
  y – żółty
  m – fioletowy
  c – turkusowy
  r – czerwony
  g – zielony
  b – niebieski
  w – biały
  k – czarny

  Przykład:
  Narysuj trójkąt o wierzchołkach w punktach (0,1), (3,4), (4,2) używając funkcji `line` oraz `fill` z wypełnieniem w kolorze niebieskim.

```
» line([0 3 4 0],[1 4 2 1])
» fill([0 3 4],[1 4 2],'b')
```

Grafika trójwymiarowa

Większość funkcji języka MATLAB generujących rysunki trójwymiarowe służy do kreślenia powierzchni. W praktyce definiując powierzchnię trzeba się ograniczyć do skończonego zbioru punktów należących do obszaru.

`[x,y]=meshgrid(X,Y)` – tworzy macierze **x** i **y** opisujące położenie węzłów prostokątnej siatki pobierając wartości z wektorów **X** i **Y**.

`mesh(x,y,z)` – rysuje siatkę powierzchni opisanej przez macierze **x**, **y** i **z**.

`surf(x,y,z)` – rysuje kolorową powierzchnię opisaną przez macierze **x**, **y** i **z**.

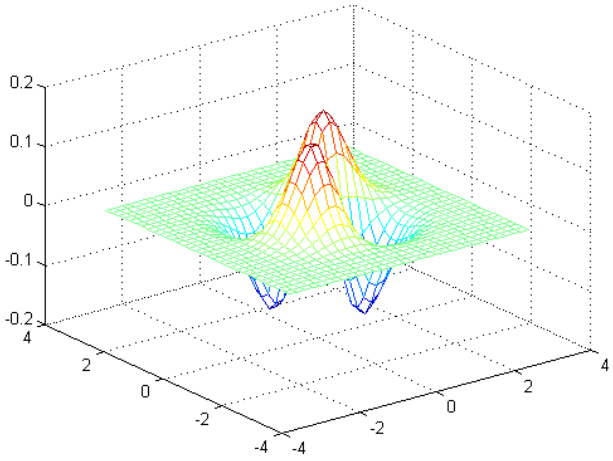
`surfl(x,y,z)` – rysuje kolorową powierzchnię opisaną przez macierze **x**, **y** i **z** uwzględniając na niej odbicie światła.

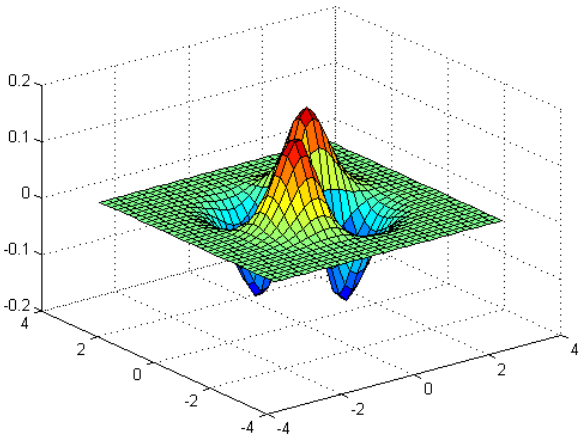
`plot3(x,y,z)` – rysuje krzywą w przestrzeni opisaną przez wektory **x**, **y** i **z**.

Przykład:
Napisz skrypt kreślący siatkę wartości funkcji $f(x,y)=\sin(x)\cdot\sin(y)\cdot\exp(-x^2-y^2)$ w przedziale od $-\pi$ do π.

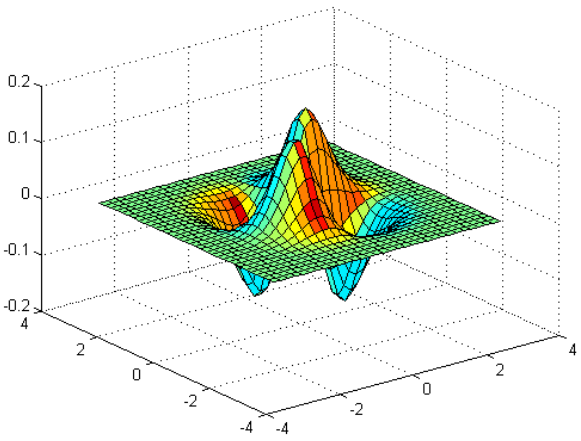
```
% Skrypt rysuje siatkę wartości funkcji
clf
[x,y]=meshgrid(-pi:0.2:pi,-pi:0.2:pi)
z=sin(x).*sin(y).*exp(-x.^2-y.^2)
mesh(x,y,z)
```

Rozbuduj powyższy skrypt o rysowanie kolorowej powierzchni poprzez dodanie na końcu polecenia: `surf(x,y,z)` lub: `surfl(x,y,z)`



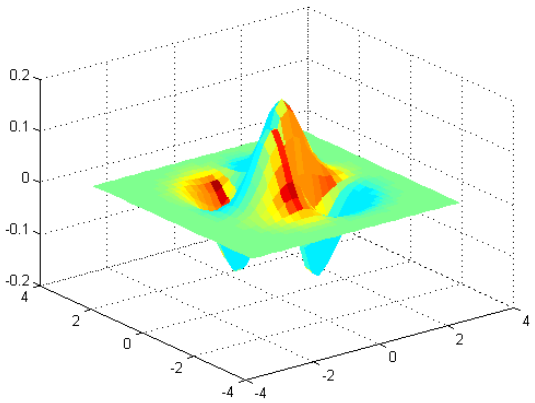


surf(x,y,z)

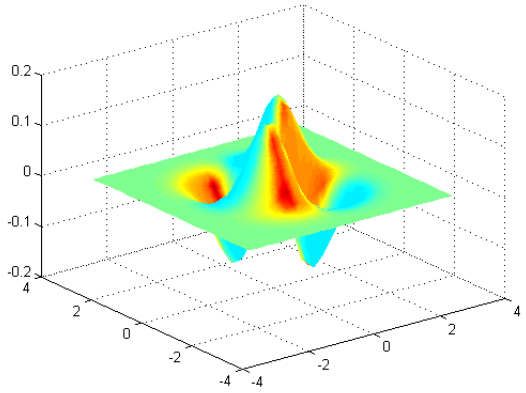


surfl(x,y,z)

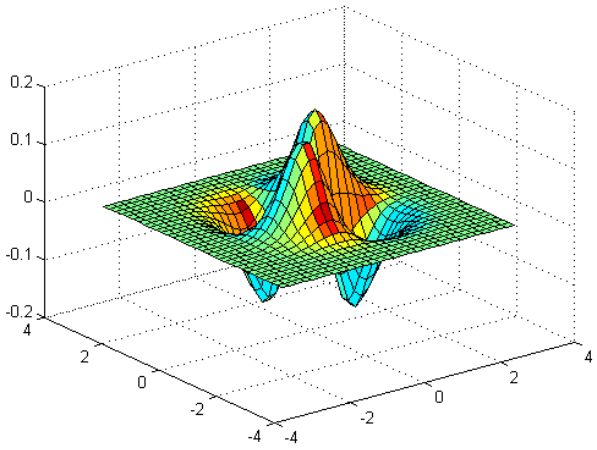
Wykreślone powierzchnie można poddać cieniowaniu używając funkcji:



`shading flat`



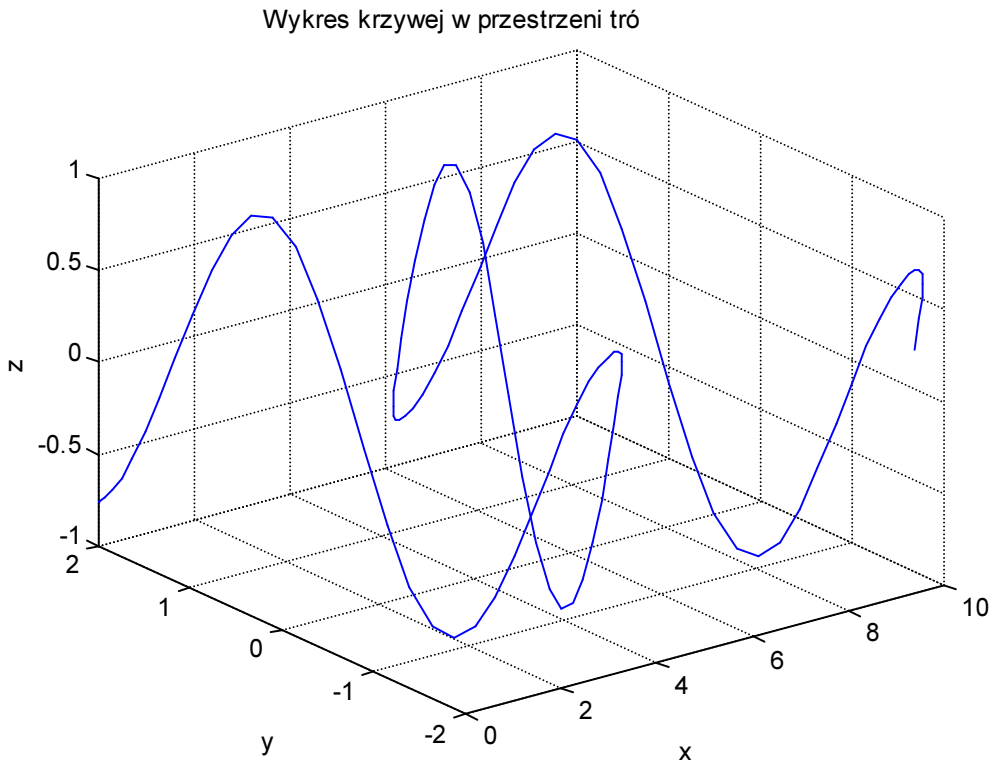
`shading interp`



`shading faceted`

Przykład:
Napisz skrypt kreślący krzywą w przestrzeni trójwymiarowej:

```
% Skrypt kreśli krzywą w przestrzeni trójwymiarowej
x=[0:0.1:10];
y=2*cos(x);
z=sin(2*y);
plot3(x,y,z)
grid
title('Wykres krzywej w przestrzeni trójwymiarowej')
xlabel('x')
ylabel('y')
zlabel('z')
```



Operacje na plikach

Każdy plik, który chcemy wczytać bądź zapisać (dowolną metodą), musi zostać otwarty, a po użyciu zamknięty:

1. Otwieranie i zamykanie plików

id_pliku=fopen(nazwa_pliku,rodzaj_dostępu)

id_pliku – *identyfikator pliku, tzw. indeks*
nazwa_pliku – *łańcuch znaków z nazwą otwieranego pliku*
rodzaj_dostępu – *łańcuch znaków jak niżej:*

- 'r'– *tylko odczyt*
- 'w'– *tylko zapis*
- 'a'– *dopisywanie elementów na końcu pliku*
- 'r+' – *odczyt i zapis*
- 'w+'– *nadpisanie zawartości istniejącego pliku i otwarcie do zapisu i odczytu*
- 'a+'– *otwarcie w celu czytania lub pisania na jego końcu*

Po zakończeniu operacji na pliku należy go zamknąć funkcją fclose()

status=fclose(id_pliku)
status=fclose('all') – *zamyka wszystkie otwarte pliki*

np.

```
plik=fopen('dane.dat','r');
st=fclose(plik);
```

2. Zapis do pliku

liczba=fwrite(id_pliku,A,typ)
A – *elementy macierzy A*
typ – *format danych*

3. Odczyt danych z pliku

A=fread(id_pliku,rozmiar,typ)
odczytuje n elementów i zapisuje w wektorze kolumnowym
lub
[A,liczba]=fread(id_pliku,rozmiar,typ)
odczytuje tyle elementów, aby wypełniły kolumnami macierz o rozmiarze mxn, brakujące
elementy są zastępowane zerami

4. Zapis w sformatowanym pliku tekstowym

liczba=fprintf(id_pliku, format, A, ...)

%d – *liczby całkowite*
%f – *liczby rzeczywiste w formacie stałoprzecinkowym*
%e – *liczby rzeczywiste w formacie zmiennoprzecinkowym*
%c – *do zapisu pojedynczych znaków*
%s – *do zapisu łańcuchów znakowych*
znaki specjalne
\b – *backspace*
\f – *nowa strona*
\n – *nowy wiersz*
\r – *powrót karetki*
\t – *tabulator*
\" – *apostrof*
\\ – *lewy ukośnik*
%% – *procent*

5. Odczyt danych z pliku tekstowego

A=fscanf(id_pliku, format, rozmiar)

Odczyt kolumnami (pierwsza kolumna, druga, itd.).
id_pliku – *wskaźnik pliku,*
format – *format odczytu*
rozmiar — *[m,n], podaje liczbę m wierszy i n kolumn, które mają być odczytane (wartość n może być inf). Pojedyncze N podaje liczbę danych do odczytu.*

Przykład:

```
macierz1=fopen('mac_1.dat','r')
A=fscanf(macierz1,'%d%d%d%d%d',[5 5])
fclose(macierz1)
```

6. Odczyt danych z pliku tekstowego z dowolnymi separatorami

A=dlmread(plik, separator)

7. Zapis i odczyt danych z pliku tekstowego z separatorami przecinkami (pliki CSV – Comma Separated Value)

Zapis:

csvwrite('nazwa_pliku',zapisywana_macierz)

Odczyt:

A = csvread('nazwa_pliku')

8. Odczyt plików tekstowych ogólnego formatu (w starszych wersjach MatLaba)

Użycie funkcji textread nie wymaga otwarcia i zamknięcia pliku.

[A]=textread('nazwa','format',N,'headerlines',l_h)

- 'nazwa' – nazwa pliku z ewentualną ścieżką dostępu –
- 'format' – format danych w wierszu pliku tekstowego, jak w p. 4
- N – liczba powtórzeń łańcucha formatu; jeśli jej brak, to plik czytany jest do końca
- 'headerlines' – parametr mówiący, że należy pominąć podaną dalej liczbę wierszy (l_h), także inne parametry

9. Odczyt plików tekstowych ogólnego formatu (w nowszych wersjach MatLaba)

Pamiętajmy!
Plik musi być otwarty poleceniem fopen,
a potem zamknięty poleceniem fclose!

Wczytanie wszystkich wierszy z pliku o stałej szerokości kolumn:

```
K=textscan(id_pliku,'format')
```

Wczytanie N wierszy z pliku o stałej szerokości kolumn:

```
K=textscan(id_pliku,'format',N)
```

Wczytanie wszystkich wierszy z pliku, w którym separatorami są średniki:
(jeżeli jako separator nic nie wpiszemy, wtedy wczytany będzie cały wiersz jako jedna kolumna)

```
K=textscan(id_pliku,'format', 'delimiter', ';')
```

Wczytanie wszystkich wierszy z pliku, z pominięciem l_h wierszy początkowych:

```
K=textscan(id_pliku,'format', 'headerlines', l_h)
```

Uwagi takie, jak w p.8.

Zmienna K jest macierzą komórkową, czyli dane z niej pobieramy poprzez użycie nawiasów klamrowych!

Przykład:
Proszę wczytać dane z pliku wykres.dat (plik zawiera liczby całkowite zapisane w dwu kolumnach), a następnie przypisać dane z kolumny 1 do zmiennej x, a dane z kolumny 2 do zmiennej y.

```
wskaznik=fopen('wykres.dat','r');
tekst=textscan(wskaznik,'%d %d');
    x=tekst{1};
    y=tekst{2};
fclose(wskaznik);
```

Należy pamiętać, że jeżeli plik nie zostanie zamknięty (**fclose**), kolejne polecenie **textscan** czyta plik źródłowy począwszy od miejsca, gdzie zakończyło się poprzednie czytanie!