Ćwiczenia z ...

Matlab

Przykłady i zadania

Anna Kamińska
Beata Pańczyk

MIKOM®

... z każdym bitem serca

Matlab Przykłady i zadania

Ćwiczenia z ... Matlab Przykłady i zadania

Anna Kamińska, Beata Pańczyk

Książka stanowi opis standardowego pakietu Matlab. W prosty i obrazowy sposób, poparty licznymi przykładami, autorki wyjaśniają podstawy pracy w środowisku pakietu. Dużo miejsca poświęcono prezentacji graficznych możliwości programu. Książka przeznaczona jest dla tych wszystkich, którzy w swojej pracy naukowej czy dydaktycznej, potrzebują łatwego narzędzia do obliczeń numerycznych i do efektownej prezentacji wyników.

Czytelnik znajdzie w książce m.in.:

- ✔ wprowadzenie w środowisko pracy pakietu
- ✔ omówienie podstawowych typów danych pakietu, ze szczególnym uwzględnieniem operacji tablicowych i macierzowych
- ✔ opis instrukcji języka Matlab
- ✔ podstawy tworzenia skryptów i funkcji
- ✔ operacje na plikach
- ✔ tworzenie wykresów dwu i trójwymiarowych
- ✔ tworzenie prostych animacji
- ✔ budowanie graficznego interfejsu użytkownika
- ✔ przykłady obliczeń numerycznych

Książka powstała jako kompendium wykładów prowadzonych przez autorki na Wydziale Elektrycznym Politechniki Lubelskiej.

... z każdym bitem serca

ISBN 83-7279-272-0

Wydawnictwo MIKOM

ul. Andrzejowska 3, 02-312 Warszawa

Dział Handlowy (hurt/detal):
tel./fax: (0 22) 823-70-77, (022) 823-94-61

tel. kom. 0 606 29 78 66

e-mail: mikom@mikom.com.pl

Księgarnia wysyłkowa: Skrytka Pocztowa 145, 00-973 Warszawa 22

Księgarnia internetowa: http://www.mikom.com.pl

Redakcja: ul. Kaliska 17 lok. 38, 02-316 Warszawa

tel./fax: (0 22) 659-28-63, tel. 822-00-12

Zapraszamy do współpracy autorów

- Wasze materiały dydaktyczne mogą stać się książką
- Wasze doświadczenie może się przydać innym

Czekamy na uwagi i opinie

- które serie są najbardziej interesujące
- jakich książek brakuje w naszej ofercie
- każda Państwa opinia jest dla nas cenna

Piszcie na adres: redakcja@mikom.com.pl

ANNA KAMIŃSKA
BEATA PIŃCZYK

ĆWICZENIA Z MATLAB

PRZYKŁADY I ZADANIA

... z każdym bitem serca

Projekt okładki: **MacGraf s.c.**

Redakcja: **Joanna Cierkońska**

Skład komputerowy: **Dorota Świstak**

Opiniodawcy: **prof. Andrzej Wac-Włodarczyk**
prof. Stanisław Grzegórski

Matlab jest pakietem przeznaczonym do wykonywania obliczeń numerycznych oraz graficznej prezentacji wyników, opracowanym w firmie MathWorks.

Dostępny jest na różnych platformach sprzętowych oraz systemowych (np. Windows, UNIX, Macintosh). Pełen opis pakietu wraz z jego specjalistycznymi bibliotekami składałby się z setek tysięcy stron. Celem książki nie jest zaprezentowanie wszystkich elementów Matlaba, a jedynie przybliżenie czytelnikowi możliwości jakie on oferuje oraz ułatwienie mu swobodnego poruszania się w jego środowisku. Podstawową konwencją przyjętą w książce jest minimum teorii poparte odpowiednio dobranymi przykładami, które pomogą zrozumieć omawiany temat. Na zakończenie każdego rozdziału zaproponowano szereg zadań przeznaczonych do samodzielnego wykonania.

Wydawca:
Wydawnictwo MIKOM, ul. Andrzejowska 3, 02-312 Warszawa, tel. 823-70-77
Druk:
ZWP „HEL", ul. Grenadierów 77, 04-007 Warszawa, tel. 810-12-71

ISBN 83-7279-272-0

Warszawa, październik 2002

Spis treści

Autorki pragną serdecznie podziękować prof. dr hab. Janowi Sikorze z Politechniki Warszawskiej za wprowadzenie w świat Matlaba oraz wskazanie ogromnych możliwości jego zastosowania w nauce i dydaktyce.

Wstęp

Matlab [1, 4, 5, 7] to pakiet przeznaczony do wykonywania obliczeń numerycznych oraz graficznej prezentacji wyników, opracowany w firmie MathWorks (`www.mathworks.com`). Dostępny jest na różnych platformach sprzętowych oraz systemowych (np. Windows, UNIX, Macintosh). Pełny opis pakietu wraz z jego specjalistycznymi bibliotekami składałby się z tysięcy stron. Celem niniejszej książki nie jest zaprezentowanie wszystkich elementów Matlaba, a jedynie przybliżenie czytelnikowi możliwości, które oferuje, oraz ułatwienie swobodnego poruszania się w jego środowisku. Chcemy to osiągnąć poprzez:

- wprowadzenie w podstawowe tajniki pakietu,
- zapoznanie z językiem programowania,
- zaprezentowanie wybranych funkcji do obliczeń numerycznych, przetwarzania i analizy danych,
- przedstawienie możliwości grafiki dwu- i trójwymiarowej wraz z graficznym interfejsem użytkownika.

Podstawową konwencją przyjętą w książce jest minimum teorii, poparte odpowiednio dobranymi ćwiczeniami, które pomogą zrozumieć omawiany temat. Na zakończenie każdego rozdziału zaproponowano zadania przeznaczone do samodzielnego wykonania.

Zaczynamy od niezbędnego omówienia podstawowych elementów pakietu (zmienne, liczby, funkcje matematyczne). Ponieważ podstawową strukturą danych w Matlabie jest macierz (Matlab – skrót od ang. *matrix laboratory*), szczególną uwagę zwrócono na operacje macierzowe.

Po informacjach wprowadzających kolejny rozdział koncentruje się na instrukcjach, funkcjach oraz tworzeniu programów (skryptów) w Matlabie.

Rozdział trzeci omawia operacje na plikach, a czwarty (najobszerniejszy) opisuje możliwości graficzne pakietu.

W ostatnim rozdziale przeanalizowano funkcje do obliczeń numerycznych (zagadnienie interpolacji i aproksymacji danych, całkowanie numeryczne, rozwiązywanie układów równań liniowych i nieliniowych oraz układów równań różniczkowych zwyczajnych).

Pakiet Matlab omówiono na przykładzie wersji 5.3 do systemu Windows.

W skrypcie przyjęto następujące oznaczenia:

diary	polecenia i funkcje Matlaba, pliki, katalogi, ścieżki dostępu
`» a=[1 2 3]`	zapis polecenia w trybie bezpośrednim
`a=[1 2 3]`	zapis polecenia w m-pliku
`a =` `1 2 3`	wynik działania programu
interfejs API	nowe pojęcia, sprawy istotne dla użytkownika Matlaba itp.

Książka powstała jako kompendium wykładów prowadzonych przez autorki na Wydziale Elektrycznym Politechniki Lubelskiej. Dedykowana jest przede wszystkim studentom i pracownikom uczelni technicznych, ale również wszystkim, którzy poszukują narzędzia mającego bogatą bibliotekę funkcji do obliczeń numerycznych oraz efektowne możliwości graficznej wizualizacji wyników pomiarów i obliczeń.

1. Wprowadzenie do pracy w środowisku pakietu Matlab

Rozdział ten, po krótkim wstępie opisującym system Matlab, ma na celu szybkie przybliżenie sposobu pracy z pakietem poprzez zaprezentowanie jego najbardziej podstawowych poleceń.

1.1. Czym jest Matlab?

Matlab stanowi proste w użyciu środowisko łączące obliczenia, wizualizację i programowanie. Typowe zastosowania pakietu to:

- obliczenia matematyczne,
- algorytmy numeryczne,
- modelowanie i symulacja,
- analiza danych i wizualizacja,
- grafika inżynierska,
- aplikacje z wykorzystaniem graficznych interfejsów użytkownika.

Matlab jest systemem interaktywnym, którego podstawową strukturę danych stanowi dwuwymiarowa tablica dynamiczna o nieokreślonej z góry liczbie elementów. Takie podejście pozwala rozwiązywać wiele technicznych problemów, w szczególności opisanych za pomocą macierzy i wektorów, bez definiowania takich struktur (co nie jest możliwe w językach programowania typu C czy Fortran).

W środowiskach uniwersyteckich Matlab jest standardowym narzędziem do obliczeń numerycznych, w przemyśle wygodnym środowiskiem do prowadzenia wyspecjalizowanych analiz i badań.

Dodatkowe biblioteki Matlaba (ang. *toolboxes*) umożliwiają stosowanie specjalistycznych technologii. Biblioteki składają się z funkcji Matlaba (tzw. m-plików), które poszerzają standardowy zestaw funkcji pakietu o możliwości rozwiązywania specyficznych problemów (np. Control System Toolbox – projektowanie układów sterowania, Optimization Toolbox – metody optymalizacji, Neural Network Toolbox – sieci neuronowe itp.).

W języku Matlab został napisany program Simulink – interaktywny system, umożliwiający graficzne modelowanie i symulację układów dynamicznych.

Pakiet Matlab składa się z pięciu podstawowych elementów. Są to:

- ***język*** Matlab – język wysokiego poziomu, umożliwiający tworzenie zarówno małych programów, jak i kompletnych aplikacji, udostępniający funkcje, obsługę wejścia/wyjścia i elementy programowania obiektowego;
- ***środowisko robocze*** Matlaba – zestaw narzędzi do zarządzania zmiennymi w przestrzeni roboczej, m-plikami, aplikacjami Matlaba oraz do importowania i eksportowania danych;
- ***system graficzny*** – zawierający funkcje wysokiego poziomu do tworzenia dwu- i trójwymiarowych wykresów, funkcje przetwarzania obrazów i tworzenia animacji oraz wiele niskopoziomowych poleceń, umożliwiających pełną kontrolę wyglądu tworzonych grafik i budowę graficznego interfejsu użytkownika;
- ***biblioteka funkcji matematycznych*** – obejmuje zarówno funkcje podstawowe (np. sumowanie, funkcje trygonometryczne, funkcje liczb zespolonych), funkcje macierzowe (np. obliczanie macierzy odwrotnych, wyznaczanie wartości własnych), jak i wiele specjalistycznych funkcji matematycznych, np. funkcje Bessela, FFT (ang. *fast Fourier transform*, szybka transformata Fouriera);
- ***interfejs API*** (skrót od ang. *application program interface*) – biblioteka umożliwiająca tworzenie programów w językach C i Fortran, współpracujących z programami napisanymi w Matlabie.

Praca z pakietem Matlab odbywać się może na dwa sposoby:

- w ***trybie bezpośrednim*** (typowy tryb roboczy, umożliwiający prowadzenie dialogu pomiędzy użytkownikiem a pakietem na zasadzie: pytanie – odpowiedź);
- w ***trybie pośrednim*** (umożliwiającym szybkie i efektywne wykonanie obliczeń i prezentację wyników za pomocą uruchomienia programu napisanego w języku pakietu Matlab, czyli tzw. ***skryptu***) – patrz rozdział 2.

1.2. Podstawy

Po uruchomieniu pakietu polecenia można wydawać bezpośrednio w oknie Matlaba. O gotowości systemu świadczy widoczny w wierszu poleceń znak zgłoszenia (»).

1.2.1. Zmienne

Nazwa zmiennej Matlaba musi się rozpoczynać literą i może się składać z dowolnej liczby liter, cyfr i znaków podkreślenia, przy czym Matlab uwzględnia tylko 31 pierwszych znaków nazwy.

Matlab nie wymaga deklarowania zmiennych ani określania ich rozmiaru. Jeśli w środowisku pojawi się nowa nazwa zmiennej, Matlab automatycznie tworzy ją

i przydziela jej odpowiednią wartość. Jeśli zmienna była już wcześniej, jej poprzednia wartość zostanie zastąpiona nową.

Aby sprawdzić, jaką wartość ma obecnie dana zmienna, wystarczy w wierszu poleceń wpisać jej nazwę.

Czasem użytkownik wyda polecenie Matlaba, nie określając nazwy zmiennej wynikowej. W takiej sytuacji wynik operacji przechowywany jest w specjalnej zmiennej pakietu o nazwie *ans*.

Należy podkreślić, że ***Matlab rozróżnia duże i małe litery***. *X* i *x* to dla języka zupełnie różne zmienne. Standardowe polecenia pakietu pisane są zawsze małymi literami.

Ćwiczenie 1.1

Utwórz zmienną x=2 i sprawdź jej wartość, a następnie oblicz $\sqrt{x}$.

Wydanie polecenia utworzenia zmiennej *x*:

```
» x=2
```

W bezpośrednim trybie pracy z pakietem otrzymamy odpowiedź:

```
x =
     2
```

Kontrola wartości zmiennej:

```
» x
```

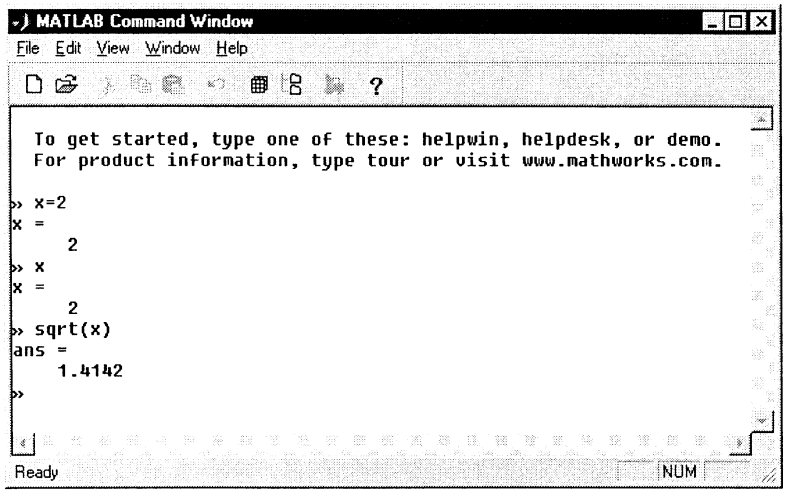
Odpowiedź będzie taka sama jak poprzednio.

Obliczenie $\sqrt{x}$

```
» sqrt(x)
ans =
     1.4142
```

W tym wypadku dla zmiennej *x* obliczona została wartość funkcji Matlaba **sqrt**. Ponieważ nie określono nazwy zmiennej wynikowej, wartość operacji została przyporządkowana zmiennej specjalnej *ans*.

Na rysunku 1.1 widoczne jest okno Matlaba po uruchomieniu pakietu i wydaniu powyższych poleceń.



Rysunek 1.1. Praca w trybie bezpośrednim w systemie Matlab (wersja 5.3 do Windows)

1.2.2. Liczby

Liczby można w Matlabie zapisywać w jednej z poniższych postaci:

- stałopozycyjnej – z opcjonalnym użyciem znaku + lub – oraz kropki dziesiętnej,
- zmiennopozycyjnej – z użyciem znaku **e** lub **E** poprzedzającego wykładnik potęgi 10 (np. **1e2**=10^2).

Obie te formy mogą też być używane jednocześnie.

Do zapisu części urojonej liczb zespolonych używa się w Matlabie stałej **i** lub **j**.

Ćwiczenie 1.2

Utwórz w Matlabie zmienne:

a) a=-0.002

```
» a=-0.0002
```

b) b=20.0341 10^{-12}

```
» b=20.0341E-12
```

lub

```
» b=20.0341e-12
```

c) c=1+3i

```
» c=1+3i
```

lub

```
» c=1+3j
```

lub

```
» c=1+3*i
```

1.2.3. Polecenia

Po wydaniu polecenia i naciśnięciu klawisza **Enter** Matlab natychmiast wyświetla jego wynik. Umieszczenie po poleceniu średnika spowoduje wykonanie obliczeń, ale bez zwracania wyniku. Jest to szczególnie przydatne w wielokrotnych obliczeniach w pętlach czy podczas generowania dużych macierzy.

Polecenie powinno się mieścić w jednym wierszu. Jeśli jest dłuższe, można zakończyć wiersz trzema kropkami i kontynuować je w następnym.

Jeśli chcemy napisać kilka poleceń w jednym wierszu, możemy je oddzielić od siebie średnikami (jeśli nie chcemy oglądać ich wyniku) lub przecinkami (w przeciwnym wypadku).

Ćwiczenie 1.3

a) Przypisz zmiennej x wartość 4 bez zwracania wyniku obliczeń.

```
» x=4;
```

b) Przypisz zmiennej *obecny_wynik* wartość *poprzedni_wynik* + $\alpha[1 - \beta + \varphi(3\gamma - 1)]$.

```
» obecny_wynik = poprzedni_wynik ...
»                + alfa*(1 - beta + fi*(3*gamma - 1))
```

Ćwiczenie 1.4

Zapisz w jednym wierszu trzy polecenia: x=1, y=2, z=3.

Polecenie

```
» x=1; y=2; z=3
```

zwróci wynik

```
z =
     3
```

ponieważ po ostatniej instrukcji nie użyto średnika. Natomiast polecenie

```
» x=1, y=2, z=3
```

wyświetli

```
x =
     1
y =
     2
z =
     3
```

1.2.4. Pomoc systemowa

Najprostszą metodą uzyskania informacji o funkcjach Matlaba podczas pracy z pakietem jest użycie polecenia:

```
» help nazwa_funkcji
```

Korzystając z opcji Help w menu okna Matlaba mamy do dyspozycji wiele opcji oferujących pomoc systemu, włącznie z tzw. Help Desk – podręcznikiem opracowanym w postaci stron HTML.

Ćwiczenie 1.5

Wyświetl informację na temat funkcji **sqrt**.

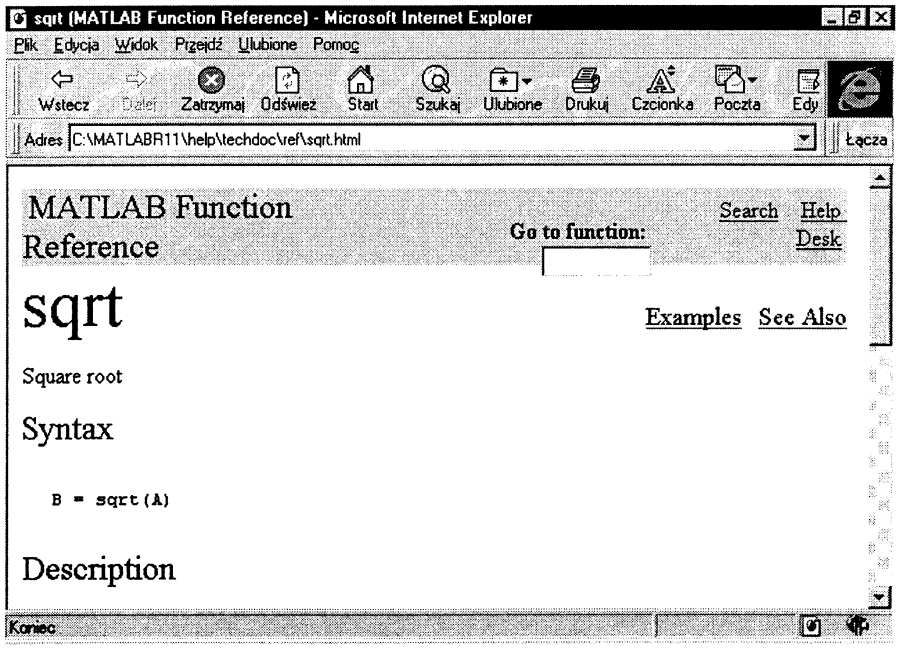
```
» help sqrt
```

Wynik działania polecenia:

```
SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.
```

W celu odróżnienia od tekstu opisu (i tylko dlatego) nazwa szukanej funkcji wyświetlana jest w pomocy systemowej ***zawsze dużymi literami.***

Podręcznik Matlaba wyświetli natomiast informacje widoczne na rysunku 1.2.



Rysunek 1.2. Strona podręcznika Matlaba z fragmentem opisu funkcji **sqrt**

1.3. Macierze

Jak już wspomniano, podstawowym typem zmiennej Matlaba jest ***macierz dwuwymiarowa***. Szczególnymi odmianami macierzy są:

- ***skalar*** – macierz o rozmiarach 1x1,
- ***wektor wierszowy*** – macierz o jednym wierszu,
- ***wektor kolumnowy*** – macierz o jednej kolumnie.

1.3.1. Definiowanie macierzy

Definiowanie macierzy wymaga uwzględnienia następujących reguł:

- elementy w wierszu macierzy muszą być oddzielane spacją lub przecinkiem,
- średnik lub znak nowego wiersza kończy wiersz macierzy i powoduje przejście do następnego,
- cała lista elementów musi być ujęta w nawiasy kwadratowe.

Poniżej przedstawiono podstawowe sposoby tworzenia macierzy. Wszystkie te metody można, w razie potrzeby, łączyć i mieszać ze sobą.

Określenie listy elementów macierzy

Najprostszą metodą definiowania macierzy jest wymienienie jej wszystkich elementów wiersz po wierszu. W definicji liczba elementów w każdym wierszu musi być jednakowa, w przeciwnym wypadku zostanie wyświetlony komunikat:

```
All rows in the bracketed expression must have the same
number of columns.
```

Ćwiczenie 1.6

Zdefiniuj:

a) macierz $\mathbf{A} = \begin{bmatrix} 0 & 2 & -10 \\ 7 & 6 & 1 \end{bmatrix}$

```
» A=[0 2 -10; 7 6 1]
```

lub

```
» A=[0 2 -10
»    7 6 1]
```

b) wektor wierszowy **B** = [1 0 -2 3]

```
» B=[1 0 -2 3]
```

c) wektor kolumnowy $\mathbf{C}=\begin{bmatrix}3\\2\\5\end{bmatrix}$

```
» C=[3; 2; 5]
```

lub

```
» C=[3
»    2
»    5]
```

d) macierz o wartościach zespolonych: $\mathbf{D}=\begin{bmatrix}2+3i & -1+1.5i\\3-7i & 2i\end{bmatrix}$

```
» D=[2  -1; 3  0]+i*[3  1.5; -7  2]
```

lub

```
» D=[2+3i  -1+1.5i; 3-7i  2i]
```

Budowanie macierzy z podmacierzy

Macierz można także zdefiniować, dzieląc ją na ***podmacierze*** i wyliczając je po kolei wierszami. Podmacierze należy zestawiać tak, aby wynikowa macierz miała jednakową liczbę elementów w każdym wierszu.

Ćwiczenie 1.7

Z danej macierzy $\mathbf{A}=\begin{bmatrix}3 & -1\\5 & 0\end{bmatrix}$ oraz wektorów: $\mathbf{B}=[2\ \ 5\ \ 8]$ i $\mathbf{C}=\begin{bmatrix}7\\6\end{bmatrix}$ utwórz macierz **D** postaci:

$$\mathbf{D}=\begin{bmatrix}\mathbf{B}\\\mathbf{C}\ \ \mathbf{A}\end{bmatrix}=\begin{bmatrix}2 & 5 & 8\\7 & 3 & -1\\6 & 5 & 0\end{bmatrix}$$

```
» A=[3 -1; 5 0]
» B=[2 5 8]
» C=[7;6]
» D=[B; C A]
```

Użycie dwukropka

Dwukropek może być w Matlabie wykorzystywany na kilka sposobów. Wyrażenie **minimum:maksimum** generuje wektor wierszowy zawierający liczby całkowite od *minimum* do *maksimum*, według wzoru: [*minimum minimum*+1 *minimum*+2 ... *maksimum*].

Z kolei wyrażenie **minimum:krok:maksimum** pozwala wygenerować wektor wierszowy o wartościach zmieniających się z dowolnym krokiem (także niecałkowitym bądź ujemnym).

Za pomocą tak zdefiniowanych wektorów wierszowych można tworzyć całe macierze.

Ćwiczenie 1.8

Wygeneruj wektory:

a) $\mathbf{x} = [1 \;\; 2 \;\; 3 \;\; 4]$

```
» x=1:4
```

b) $\mathbf{y} = [-1 \;\; -0.8 \;\; -0.6 \;\; -0.4 \;\; -0.2 \;\; 0 \;\; 0.2 \;\; 0.4 \;\; 0.6 \;\; 0.8 \;\; 1]$

```
» y=-1:0.2:1
```

Ćwiczenie 1.9

Wygeneruj macierz $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \end{bmatrix}$.

```
» A=[1:10; 1:2:20]
```

1.3.2. Dostęp do elementów macierzy

Odwołania do elementów macierzy

Element macierzy znajdujący się w wierszu o indeksie *i* oraz kolumnie o indeksie *j* jest określony jako **A(i,j)**. Elementem takim można się posługiwać jak każdą zmienną – zwracać jego wartość, przypisywać nową itp.

Do elementów macierzy można się też odwoływać przy użyciu tylko jednego indeksu, np. **A(k)**. Jeśli zmienna **A** byłaby wektorem wierszowym lub kolumnowym, odwołanie takie oznaczałoby oczywiście *k*-ty element wektora. Natomiast gdy **A** będzie macierzą dwuwymiarową, odwołanie takie zostanie zinterpretowane jako odwołanie do wektora ***kolumnowego***, uformowanego z kolejnych ***kolumn*** oryginalnej macierzy, umieszczonych jedna pod drugą.

Ćwiczenie 1.10

Zdefiniuj macierz $\mathbf{C} = \begin{bmatrix} 2 & 0 & 1 & 4 & 7 \\ 3 & 9 & 8 & 5 & 6 \\ 1 & 5 & 0 & 4 & 3 \end{bmatrix}$

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

Podstawienie postaci:

```
» y=C(3,2)
```

przypisze zmiennej *y* wartość 5.

Natomiast (co już nie jest tak oczywiste) podstawienie:

```
» y=C(6)
```

również przypisze zmiennej *y* wartość 5. W tym wypadku jednak macierz **C** zostanie zinterpretowana jako wektor kolumnowy, składający się z kolejnych kolumn oryginalnej macierzy **C**. Wektor ten, zapisany w wierszu, miałby postać [2 3 1 0 9 5 1 8 0 4 5 4 7 6 3].

Wykonaj ponadto następujące polecenia:

a) przypisz zmiennej *x* wartość $C_{3,1}$

```
» x=C(3,1)
x =
     1
```

b) nadaj elementowi $C_{2,3}$ wartość 7.

```
» C(2,3)=7
```

Odwołania do fragmentów macierzy

Za pomocą dwukropka można się odwołać do wybranych fragmentów wektorów i macierzy, jak pokazano w tabeli 1.1.

Tabela 1.1. Użycie dwukropka w odwołaniach do podmacierzy

Odwołanie	Opis
x(j:k)	elementy wektora wierszowego **x** o numerach od *j* do *k*
A(i, :)	wszystkie elementy w wierszu *i* macierzy **A**
A(i, j:l)	wszystkie elementy w wierszu *i* macierzy **A** o numerach od *j* do *l*
A(i:k, j:l)	wszystkie elementy w kolumnach od *j* do *l* wierszy od *i* do *k* macierzy **A**
A(x, j:l)	wszystkie elementy w kolumnach od *j* do *l* w wierszach macierzy **A** o numerach określonych przez elementy wektora **x**
A(:, :)	cała dwuwymiarowa macierz **A**
A(:)	cała macierz **A** w postaci wektora kolumnowego

Ćwiczenie 1.11

W wypadku macierzy **C** z ćwiczenia 1.10 wykonaj następujące polecenia:

a) wyświetl jej trzeci wiersz

```
» C(3,1:5)
```

lub

```
» C(3,:)
ans =
     1     5     0     4     3
```

b) wyświetl jej drugą i trzecią kolumnę i przypisz wynik zmiennej **D** (będzie to macierz o rozmiarach 3x2)

```
» D=C(:,2:3)
D =
     0     1
     9     8
     5     0
```

c) wyświetl jej drugą i czwartą kolumnę

```
» C(:,[2 4])
ans =
     0     4
     9     5
     5     4
```

d) wyświetl jej fragment znajdujący się między wierszami 2 i 3 oraz między kolumnami 2 i 4.

```
» C(2:3, 2:4)
ans =
     9     8     5
     5     0     4
```

Usuwanie fragmentów macierzy

Wybrany element macierzy usuwa się, przypisując mu wartość w postaci macierzy ***pustej***. Macierz taka nie ma zawartości i składa się tylko z dwóch nawiasów kwadratowych ([]).

Ćwiczenie 1.12

Usuń trzeci wiersz macierzy **C** z ćwiczenia 1.10.

```
» C(3,:)=[]
C =
     2     0     1     4     7
     3     9     8     5     6
```

1.3.3. Wyświetlanie macierzy i ich rozmiarów

W tabeli 1.2 zestawione zostały funkcje wyświetlające zawartość i rozmiar macierzy.

Tabela 1.2. Funkcje wyświetlające macierze i ich rozmiary

Funkcja	Opis
disp(A)	wyświetla zawartość macierzy **A** w oknie poleceń
size(A)	wyświetla rozmiar dwuwymiarowej macierzy **A** (liczbę wierszy i kolumn) w postaci dwuelementowego wektora wierszowego [*liczba_wierszy liczba_kolumn*]
[n m]=size(A)	przypisuje zmiennej *n* liczbę wierszy, a zmiennej *m* – liczbę kolumn macierzy **A**
n=size(A,1)	przypisuje zmiennej *n* liczbę wierszy macierzy **A**
m=size(A,2)	przypisuje zmiennej *m* liczbę kolumn macierzy **A**
length(x)	zwraca długość wektora **x** lub dłuższy z wymiarów macierzy

Podobny efekt do **disp(A)** można uzyskać, wpisując po prostu nazwę macierzy

```
» A
```

W tym wypadku wyświetlona zawartość macierzy zostanie poprzedzona nagłówkiem "A = ".

Ćwiczenie 1.13

Zdefiniuj macierz $\mathbf{D} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 5 & 0 \end{bmatrix}$.

```
» D=[2  0  1;  1  5  0]
```

Następnie wykonaj następujące polecenia:

a) wyświetl rozmiar macierzy

```
» size(D)
ans =
     2     3
```

b) przypisz zmiennej *ile_wierszy* liczbę wierszy, a zmiennej *ile_kolumn* liczbę kolumn macierzy **D**

```
» [ile_wierszy ile_kolumn]=size(D)
ile_wierszy =
                2
```

```
ile_kolumn  =
                3
```

lub (wynik będzie taki sam)

```
»  ile_wierszy=size(D,1), ile_kolumn=size(D,2)
```

c) utwórz macierz **D1** o rozmiarze macierzy **D**, składającą się z samych jedynek (zostanie użyta funkcja **ones**, opisana w tabeli 1.4).

```
»  D1=ones(size(D))
D1 =
     1     1     1
     1     1     1
```

1.3.4. Arytmetyka macierzowa i tablicowa

W szczególnym przypadku macierz można potraktować jako zwykłą prostokątną tablicę liczb. Operacje na macierzach dokonywane są zgodnie z algebrą macierzy, natomiast operacje na tablicach wykonywane są ***element po elemencie***. Matlab dostarcza w tym celu odrębnych arytmetycznych operatorów macierzowych i tablicowych.

Niech $\mathbf{A}=\begin{bmatrix}1 & -1\\ -2 & 3\end{bmatrix}$, $\mathbf{B}=\begin{bmatrix}1 & 1\\ 0 & -2\end{bmatrix}$.

W tabeli 1.3 zestawiono wyniki operacji dla danych **A** i **B**, wykorzystując operatory macierzowe i tablicowe. Operatory tablicowe poprzedzone są ***zawsze*** kropką, np.:

- mnożenie macierzowe – gwiazdka (*)
- mnożenie tablicowe – kropka i gwiazdka (.*)

Tabela 1.3. Efekty działania operatorów macierzowych i tablicowych

Operator arytmetyczny	Operator macierzowy	Operator tablicowy
dodawanie	$\mathbf{A+B}=\begin{bmatrix}2 & 0\\ -2 & 1\end{bmatrix}$	tak jak macierzowy
odejmowanie	$\mathbf{A-B}=\begin{bmatrix}0 & -2\\ -2 & 5\end{bmatrix}$	tak jak macierzowy
mnożenie	$\mathbf{A*B}=\begin{bmatrix}1 & 3\\ -2 & -8\end{bmatrix}$ $\mathbf{B*A}=\begin{bmatrix}-1 & 2\\ 4 & -6\end{bmatrix}$ (brak przemienności mnożenia macierzy)	$\mathbf{A.*B=B.*A}=\begin{bmatrix}1 & -1\\ 0 & -6\end{bmatrix}$

Operator arytmetyczny	Operator macierzowy	Operator tablicowy
dzielenie	**C1=B/A=**$\begin{bmatrix} 1 & 3 \\ -2 & -8 \end{bmatrix}$ – dzielenie prawostronne (**B=C1*A**) **C2=A\B=**$\begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$ – dzielenie lewostronne (**B=A*C2**)	**B./A=A.\B=**$\begin{bmatrix} 1 & -1 \\ 0 & -2/3 \end{bmatrix}$
potęgowanie	**A^2=A*A=**$\begin{bmatrix} 3 & -4 \\ -8 & 11 \end{bmatrix}$	**A.^ 2=**$\begin{bmatrix} 1 & 1 \\ 4 & 9 \end{bmatrix}$
transpozycja (zamiana wierszy na kolumny)	**A'=**$\begin{bmatrix} 1 & -2 \\ -1 & 3 \end{bmatrix}$	tak jak macierzowy

Podczas mnożenia macierzowego **A*B** należy dopilnować, aby liczba wierszy macierzy **A** była równa liczbie kolumn macierzy **B**. Ponadto w rachunku macierzowym nie jest spełnione prawo przemienności mnożenia. Na komentarz zasługuje również występowanie zarówno prawostronnego (/), jak i lewostronnego (\) operatora dzielenia macierzowego. Z operatorów tych należy korzystać bardzo ostrożnie, gdyż jak demonstruje przykład z tabeli 1.3, otrzymane wyniki są zupełnie różne. Problemu tego nie ma w przypadku operacji na tablicach.

Transpozycja macierzy o składnikach rzeczywistych daje takie same wyniki niezależnie od tego, czy wykorzystywany jest operator macierzowy, czy tablicowy. Różnice pojawiają się, kiedy składniki macierzy są zespolone. W takim przypadku transpozycja tablicowa powoduje tylko zamianę wierszy i kolumn macierzy, natomiast transpozycja macierzowa zwraca dodatkowo macierz o elementach sprzężonych z odpowiednimi elementami macierzy zespolonej.

Jeśli $\mathbf{z}=\begin{bmatrix} 1+i & 1+2i \\ -2-3i & 3-i \end{bmatrix}$, to $\mathbf{z'}=\begin{bmatrix} 1-i & -2+3i \\ 1-2i & 3+i \end{bmatrix}$, $\mathbf{z.'}=\begin{bmatrix} 1+i & -2-3i \\ 1+2i & 3-i \end{bmatrix}$.

Ćwiczenie 1.14

a) Zdefiniuj macierze **A** i **B**: $\mathbf{A}=\begin{bmatrix} 1 & 0 \\ 3 & 2 \end{bmatrix}, \mathbf{B}=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

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

b) Oblicz iloczyny macierzy **A·B** i **B·A** macierzowo i tablicowo.

```
» A*B,  B*A      (mnożenie macierzowe)
» A.*B, B.*A     (mnożenie tablicowe)
```

c) Oblicz $\mathbf{A}^3$ macierzowo i tablicowo.

```
» A^3          (mnożenie macierzy: A*A*A)
» A.^3         (potęgowanie pojedynczych elementów A_ij)
```

d) Oblicz iloczyn $(\mathbf{A}\cdot\mathbf{B})^{-1}\cdot(\mathbf{A}\cdot\mathbf{B})$.

```
» G=(A*B).^(-1)*(A*B)
```

e) Wyznacz macierz $\mathbf{C}=(\mathbf{A}+\mathbf{B}^T)/2$, gdzie $\mathbf{B}^T$ – macierz transponowana.

```
» C=(A+B')./2
```

Ćwiczenie 1.15

Zdefiniuj wektor wierszowy **x**=[1 4] i macierz $\mathbf{A}=\begin{bmatrix}4 & 1\\ 7 & 2\end{bmatrix}$. Wykonaj mnożenie **A**·**x**.

```
» x=[1 4], A=[4 1; 7 2]
```

W Matlabie nie można zapisać iloczynu tych zmiennych jako

```
» A*x
```

ponieważ liczba kolumn macierzy **A** nie jest równa liczbie wierszy wektora **x**.
W takim przypadku pojawi się komunikat o błędzie:

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Prawidłowym zapisem będzie natomiast

```
» A*x'
```

gdzie **x'** – transpozycja wektora **x**, czyli wektor kolumnowy $\begin{bmatrix}1\\4\end{bmatrix}$. W wyniku otrzymamy wektor:

```
ans =
     8
    15
```

Ćwiczenie 1.16

Przyjmując wektor $\mathbf{x}=\left[0\ \frac{\pi}{2}\ \pi\ \frac{3\pi}{2}\ 2\pi\right]$, oblicz wartości funkcji $y=2x\sin(1+x^2)$.

```
» x=0:pi/2:2*pi;
» y=2*x.*sin(1+x.^2)
y =
    0   -1.0055   -6.2334   -8.8360    4.4537
```

Operacje matematyczne zostały wykonane kolejno dla każdego elementu wektora **x**. Otrzymany wektor **y** ma tyle samo elementów, co wektor **x**.

Mnożenie i dzielenie ***prawostronne*** wektora przez liczbę (lub liczby przez liczbę) daje takie same wyniki niezależnie od tego, czy użyto operatora macierzowego, czy tablicowego (w ćwiczeniu: **pi/2**, **2*x**). Różnice pojawiają się przy mnożeniu i dzieleniu prawostronnym wektora przez wektor – np. wyrażenie x^2 może być poprawnie zapisane tylko jako **x.*x** (w przypadku zapisu **x*x** zostanie zwrócony błąd).

1.3.5. Funkcje generujące i przekształcające macierze

W tabeli 1.4 przedstawiono funkcje Matlaba ułatwiające definiowanie niektórych macierzy specjalnych.

Tabela 1.4. Funkcje generujące macierze

Funkcja	Opis
eye(n)	tworzy macierz jednostkową o rozmiarze *nxn* (jedynki na głównej przekątnej, reszta elementów równa zeru), np. **X=eye(3)**
ones(n)	tworzy macierz o rozmiarze *nxn* o wszystkich elementach równych 1
zeros(n)	tworzy macierz o rozmiarze *nxn* o wszystkich elementach równych 0
rand(n)	tworzy macierz o rozmiarze *nxn* wypełnioną liczbami pseudolosowymi z przedziału <0,1> o rozkładzie jednostajnym
randn(n)	tworzy macierz o rozmiarze *nxn* wypełnioną liczbami pseudolosowymi o rozkładzie normalnym ze średnią 0 i wariancją równą 1

Wszystkie funkcje przedstawione w tabeli 1.4 mogą generować macierze prostokątne o rozmiarze *mxn*. Odpowiednią funkcję należy wtedy wywołać z dwoma argumentami, np. **rand(m,n)**.

W operacjach na macierzach mogą też pomóc funkcje przedstawione w tabeli 1.5.

Tabela 1.5. Funkcje przekształcające macierze

Funkcja	Opis
A=diag(x)	utworzenie macierzy przekątniowej **A** ze składnikami wektora **x** na głównej przekątnej
x=diag(A)	utworzenie wektora **x** z elementów znajdujących się na głównej przekątnej macierzy **A**
inv(A)	utworzenie macierzy odwrotnej do **A**; **inv(A)=A^(-1)**

Funkcja	Opis
repmat(A,n,m)	utworzenie macierzy przez powielenie podmacierzy **A** *m* razy w poziomie i *n* razy w pionie
reshape(A,n,m)	utworzenie macierzy o *n* wierszach i *m* kolumnach z elementów branych kolejno ***kolumnami*** z macierzy **A**; jeśli **A** nie zawiera *mxn* elementów, to pojawi się komunikat o błędzie
rot90(A)	obrócenie macierzy **A** o 90° w kierunku przeciwnym do ruchu wskazówek zegara
tril(A)	utworzenie z macierzy **A** macierzy trójkątnej dolnej (wyzerowanie elementów leżących powyżej głównej przekątnej)
triu(A)	utworzenie z macierzy **A** macierzy trójkątnej górnej (wyzerowanie elementów leżących poniżej głównej przekątnej)

Ćwiczenie 1.17

Dla danych macierzy **A** i **B** z ćwiczenia 1.14 wyznacz macierz $(\mathbf{A}\cdot\mathbf{B})^{-1}$.

```
» D=inv(A*B)      (macierz odwrotna do iloczynu macierzy A·B)
```

lub

```
» D=(A*B)^(-1)
```

Ćwiczenie 1.18

Zdefiniuj macierz $\mathbf{A}=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$.

Następnie wykonaj następujące polecenia:

a) powiel macierz **A** dwa razy w pionie i trzy razy w poziomie

```
» B=repmat(A,2,3)
B =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

b) zmień rozmiar macierzy **B** utworzonej w punkcie poprzednim na 2 wiersze i 12 kolumn.

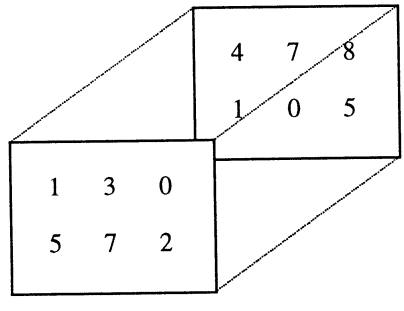
```
» C=reshape(B,2,12)
C =
     1     1     2     2     1     1     2     2     1     1     2     2
     3     3     4     4     3     3     4     4     3     3     4     4
```

1.3.6. Macierze wielowymiarowe

Matlab dopuszcza definiowanie macierzy wielowymiarowych. Odwoływanie się do elementów takich macierzy wymaga liczby indeksów większej niż 2. Przyjmuje się, że indeksy oznaczają:

- pierwszy indeks – wiersz macierzy (wymiar 1),
- drugi indeks – kolumnę macierzy (wymiar 2),
- trzeci indeks – ***stronę*** macierzy (wymiar 3 i następne).

Na rysunku 1.3 widoczna jest macierz trójwymiarowa o rozmiarze 2x3x2 (2 wiersze i 3 kolumny na każdej stronie, 2 strony).



Rysunek 1.3. Macierz trójwymiarowa

Macierz taką definiuje się ***stronami***

```
» D(:,:,1)=[1 3 0; 5 7 2]        (strona 1)
» D(:,:,2)=[4 7 8; 1 0 5]        (strona 2)
```

Odwołanie np. do elementu w pierwszym wierszu, drugiej kolumnie, na stronie drugiej ma postać

```
» D(1,2,2)
ans =
     7
```

Macierze wielowymiarowe można także tworzyć za pomocą zwykłych funkcji generujących macierze, np.

```
» A=randn(3,5,4)
```

W praktyce pracy z Matlabem macierze o wymiarze większym niż 2 wykorzystywane są rzadko.

1.4. Podstawowe funkcje i stałe matematyczne

Matlab udostępnia wiele standardowych funkcji matematycznych. Najważniejsze z nich zostały przedstawione w tabeli 1.6. Większość funkcji dopuszcza użycie argumentów zespolonych. Ponieważ wszystkie zmienne Matlaba są traktowane jako macierze, ***argumentem każdej funkcji może też być macierz***. W takim wypadku odpowiednia operacja wykonywana jest na każdym elemencie macierzy z osobna.

Tabela 1.6. Funkcje wykonujące podstawowe operacje matematyczne

Funkcja	Opis
sin(z), cos(z), tan(z), cot(z)	funkcje trygonometryczne: sinus, cosinus, tangens, cotangens; argument funkcji podawany jest w radianach
asin(z), acos(z), atan(z), acot(z)	funkcje cyklometryczne; wynik podawany jest w radianach
sinh(z), cosh(z), tanh(z), coth(z)	funkcje hiperboliczne; argument funkcji podawany jest w radianach
asinh(z), acosh(z), atanh(z)	funkcje odwrotne do hiperbolicznych; wynik podawany jest w radianach
sqrt(z)	$\sqrt{z}$ (jeśli $z<0$, wynik jest zespolony)
exp(z)	e^z
log(z)	$\ln z$ (jeśli $z<0$, wynik jest zespolony)
log2(z)	$\log_2 z$ (jeśli $z<0$, wynik jest zespolony)
log10(z)	$\log_{10} z$ (jeśli $z<0$, wynik jest zespolony)
abs(z)	$\lvert z \rvert$ lub moduł liczby zespolonej
angle(z)	argument liczby zespolonej
real(z), imag(z)	część rzeczywista i urojona liczby zespolonej
conj(z)	liczba zespolona sprzężona
complex(x,y)	utworzenie liczby zespolonej: **complex(x,y)=x + y*i**
ceil(z)	zaokrąglenie liczby w górę
floor(z)	zaokrąglenie liczby w dół
fix(z)	zaokrąglenie liczby dodatniej w dół, ujemnej w górę
round(z)	zaokrąglenie liczby do najbliższej liczby całkowitej
rem(x,y)	reszta z dzielenia x przez y obliczona według wzoru: **rem(x,y)=x-n*y**, gdzie **n=fix(x/y)**

Funkcja	Opis
mod(x,y)	reszta z dzielenia *x* przez *y* obliczona według wzoru: **mod(x,y)=x-n*y**, gdzie **n=floor(x/y)**; **mod(x,y)=rem(x,y)**, jeśli *x* i *y* mają ten sam znak
sign(x)	funkcja signum (1 dla *x*>0, 0 dla *x*=0, -1 dla *x*<0)

Jedna z grup funkcji matematycznych Matlaba (tabela 1.7) operuje tylko na wektorach. Jeśli argumentami tych funkcji są macierze, zwracają one wynik w postaci wektora, obliczony oddzielnie dla każdej ***kolumny*** macierzy.

Tabela 1.7. Funkcje operujące na wektorach

Funkcja	Opis
max(x)	zwraca największy element wektora **x**
min(x)	zwraca najmniejszy element wektora **x**
sum(x)	zwraca sumę elementów wektora **x**
prod(x)	zwraca iloczyn elementów wektora **x**
mean(x)	zwraca średnią arytmetyczną elementów wektora **x**

W Matlabie zostały też zdefiniowane niektóre stałe matematyczne (tabela 1.8). Posługując się nimi, należy jednak pamiętać, że ich nazwy nie są w żaden sposób zastrzeżone. Przez nieuwagę ich wartość może ławo zostać zastąpiona inną. Ustawienia domyślne wartości stałych przywraca funkcja **clear** (patrz podrozdział 1.6.2).

Tabela 1.8. Stałe matematyczne

Stała	Opis
pi	przybliżenie wartości π
i lub **j**	$\sqrt{-1}$
eps	względna dokładność zmiennoprzecinkowa
Inf lub **inf**	nieskończoność (ang. *infinity*); jest rezultatem operacji, która przekracza zakres arytmetyki komputera – np. dzielenie przez zero
NaN lub **nan**	nie liczba (ang. *Not-a-Number*); jest wynikiem matematycznie niezdefiniowanych operacji – np. **Inf/Inf** lub **0/0**; dzięki takiemu podejściu dzielenie przez zero nie prowadzi do przerwania operacji – wyświetlany jest odpowiedni komunikat, a zmiennej nadawana jest wartość specjalna

Ćwiczenie 1.19

Oblicz wartości funkcji sin(x), przyjmując, że x zmienia się od wartości -π do π z krokiem 0,1.

```
» x=-pi:0.1:pi;
» sin(x)
```

Można zaobserwować, że wartość **sin(pi)** nie jest dokładnie równa zeru. Wynika to z błędów reprezentacji zmiennopozycyjnej wartości π.

1.5. Typy danych

Matlab dopuszcza użycie sześciu typów danych: **double**, **char**, **sparse**, **storage**, **cell** i **struct**. W praktyce najczęściej stosowane są dwa:

- **double** – liczby podwójnej precyzji,
- **char** – znaki i łańcuchy znaków.

1.5.1. Liczby podwójnej precyzji

Wszystkie obliczenia Matlaba wykonywane są na liczbach podwójnej precyzji. Tablica liczb podwójnej precyzji jest podstawową zmienną programu i wszystkie dotąd omówione formaty liczb w rzeczywistości przechowywane są w pamięci jako właśnie taki typ danych. Zakres typu jest zależny od maszyny i określony przez stałe **realmin** i **realmax**.

1.5.2. Łańcuchy znakowe

Łańcuchy są wektorami składającymi się ze znaków. Łańcuch znakowy definiuje się za pomocą apostrofów, np.

```
» s='Matlab'
s =
Matlab
```

Łańcuch przechowywany jest w pamięci w postaci wektora liczb całkowitych reprezentujących kody ASCII poszczególnych znaków. Konwersji łańcucha na wektor kodów ASCII można dokonać za pomocą polecenia

```
» a = double(s)
a =
    77   97   116   108   97   98
```

Konwersję odwrotną wykonuje polecenie **char(a)**.

Z łańcuchów można budować macierze znakowe. Polecenie

```
» w=['Jezyk ' s]
```

połączy łańcuchy w poziomie w 12-elementowy wektor znakowy **w**

```
w =
Jezyk Matlab
```

Z kolei polecenie

```
» M=['Jezyk '; s]
```

połączy łańcuchy w pionie i zwróci macierz znakową **M** o rozmiarze 2x6

```
M =
Jezyk
Matlab
```

Oba wiersze macierzy **M** muszą mieć jednakową długość.

Ponieważ łańcuch jest wektorem, można na nim wykonywać operacje, jak na zwykłych wektorach. Przykładowo polecenie

```
» length(s)
```

zwróci liczbę 6 (liczba znaków w łańcuchu *s*). Z tego samego względu do wyświetlania łańcuchów można użyć funkcji **disp**, np.

```
» disp(w)
```

Oprócz tego Matlab zawiera liczne funkcje operujące na łańcuchach. Niektóre z nich zostały przedstawione w tabeli 1.9.

Tabela 1.9. Funkcje przetwarzające łańcuchy

Funkcja	Opis
deblank(s)	usuwa spacje z końca łańcucha, np. **s=deblank(s)**
findstr(s1,s2)	szuka krótszego z łańcuchów *s1* i *s2* w dłuższym; zwraca wektor indeksów, od których zaczyna się występowanie krótszego łańcucha
lower(s)	zmienia wszystkie litery w łańcuchu na małe
strcat(s1,s2,s3,...)	łączy łańcuchy w poziomie z pominięciem spacji na końcu każdego z nich; polecenie **strcat('Język ', s)** zwróci łańcuch '*JęzykMatlab*'
strcmp(s1,s2)	porównuje dwa łańcuchy; jeśli są identyczne, zwraca 1, jeśli nie – 0; funkcja rozróżnia wielkość liter
strcmpi(s1,s2)	porównuje dwa łańcuchy bez rozróżniania wielkości liter
strncmp(s1,s2,n)	porównuje *n* pierwszych znaków w dwu łańcuchach

Funkcja	Opis
strvcat(s1,s2,s3)	łączy łańcuchy w pionie, dodając na końcu każdego z nich odpowiednią liczbę spacji; zwraca macierz znakową
upper(s)	zmienia wszystkie litery w łańcuchu na duże

Czasem zachodzi konieczność przekształcenia danej liczbowej na tekst (np. podczas wyświetlania wartości liczbowych w oknie graficznym) lub odwrotnie. Służą do tego m.in. funkcje zestawione w tabeli 1.10.

Tabela 1.10. Funkcje konwertujące łańcuchy

Funkcja	Opis
int2str(n)	konwertuje liczbę całkowitą *n* na łańcuch (liczba niecałkowita przed konwersją zostanie zaokrąglona) ; argumentem funkcji może być macierz
num2str(x)	konwertuje wyrażenie Matlaba (liczbę, macierz lub polecenie) na łańcuch
str2double(s)	konwertuje łańcuch *s* na liczbę (rzeczywistą lub zespoloną); liczba w łańcuchu musi mieć prawidłowy format

Ćwiczenie 1.20

Zdefiniuj zmienne *x*=12.34, *y*=12.34e-5, *s*='*program*' oraz *s1*='*56.78*'.

```
» x=12.34,y=12.34e-5, s='program', s1='56.78'
x =
   12.3400
y =
   1.2340e-004
s =
program
s1 =
56.78
```

Następnie wykonaj poniższe polecenia:

a) dokonaj konwersji wartości liczbowych *x* i *y* na łańcuchy znaków,

```
» s2=num2str(x), s3=int2str(x), s4=num2str(y)
s2 =
12.34
s3 =
12                          (liczba rzeczywista została zaokrąglona)
s4 =
0.0001234
```

b) dokonaj konwersji łańcuchów *s* i *s1* na liczby rzeczywiste,

```
» x1=str2double(s), x2=str2double(s1)
x1 =
     NaN          (nie ma możliwości konwersji na liczbę)
x2 =
56.7800
```

c) dokonaj konwersji wyrażenia $\sin(2\pi x)$ na łańcuch znaków.

```
» num2str(sin(2*pi*x))
ans =
0.84433
```

1.6. Środowisko Matlaba

Niniejszy rozdział poświęcono zagadnieniom związanym z obsługą środowiska programu, odczytem i zapisem danych w plikach, a także zarządzaniem pamięcią.

1.6.1. Okno poleceń

Przed przystąpieniem do pracy z pakietem można, ustawiając odpowiednie opcje, przygotować środowisko pracy. Służą do tego specjalne funkcje obsługujące okno poleceń (tabela 1.11).

Tabela 1.11. Funkcje obsługujące okno poleceń

Funkcja	Opis
clc	wyczyszczenie okna poleceń i umieszczenie kursora w jego lewym górnym rogu
home	umieszczenie wiersza poleceń i kursora w lewym górnym rogu okna poleceń
echo on/echo off	włącza/wyłącza wysyłanie na ekran treści wykonywanych poleceń
more on/more off	włącza/wyłącza stronicowanie tekstów wysyłanych na ekran
diary plik	polecenia i teksty (bez grafiki) wysyłane na ekran będą zapisywane w pliku o podanej nazwie
diary off/on	przełącznik funkcji **diary**

Do określenia sposobu wyświetlania liczb rzeczywistych w oknie służy funkcja **format**. Użycie funkcji nie ma wpływu na dokładność wykonywanych obliczeń, a tyl-

ko na widok liczby na ekranie. W tabeli 1.12 przedstawiono reprezentację liczby 1/23 w różnych formatach.

Tabela 1.12. Formaty liczb

Format	Opis	Wynik dla liczby 1/23
short	5-cyfrowa liczba stałopozycyjna (format domyślny)	0,0435
short e	5-cyfrowa liczba zmiennopozycyjna	4,3478e-002
long	15-cyfrowa liczba stałopozycyjna	0,04347826086957
long e	15-cyfrowa liczba zmiennopozycyjna	4,347826086956522e-002
short g	5 znaczących cyfr liczby stało- lub zmiennopozycyjnej	0,043478
long g	15 znaczących cyfr liczby stało- lub zmiennopozycyjnej	0,0434782608695652
hex	liczba szestnastkowa	3fa642c8590b2164
+	drukuje znak + dla liczb dodatnich, – dla liczb ujemnych, spację dla zera	+
bank	format walutowy (pełna część całkowita i do dwóch miejsc po przecinku)	0,04
rat	przybliża liczbę ułamkami małych liczb całkowitych	1/23
compact	wyłącza dodawanie dodatkowych pustych wierszy	
loose	włącza dodawanie dodatkowych pustych wierszy	

Odpowiedni format ustawia się poleceniem np.

```
» format bank
```

Ćwiczenie 1.21

Przygotuj środowisko pracy Matlaba. W tym celu:

a) rozpocznij zapisywanie wydawanych poleceń w pliku **Matlab.txt**,

```
» diary Matlab.txt
```

b) wyczyść okno poleceń, wyłącz opcję wysyłania na ekran treści wykonywanych poleceń oraz ustaw format wyświetlania danych na **long e**,

```
» clc, echo off, format long e
```

c) zdefiniuj x=12.34 oraz zakończ zapisywanie w pliku **Matlab.txt**.

```
» x=12.34, diary off
```

Po wykonaniu powyższych poleceń plik **Matlab.txt** zawiera tekst:

```
clc, echo off, format long e
x=12.34, diary off
x =
    1.234000000000000e+001
diary off
```

1.6.2. Przestrzeń robocza

Przestrzeń robocza Matlaba jest to obszar pamięci, w którym przechowywane są zmienne utworzone w oknie poleceń. W Matlabie są funkcje umożliwiające operacje na tych zmiennych (tabela 1.13).

Tabela 1.13. Funkcje obsługujące zarządzanie pamięcią

Funkcja	Opis
who	wyświetla listę wszystkich zmiennych znajdujących się aktualnie w pamięci
whos	wyświetla listę wszystkich zmiennych wraz z informacją na temat ich rozmiaru i rodzaju
who global **whos global**	wyświetlają informacje o zmiennych globalnych (patrz rozdział 2.3.2)
clear	usuwa z pamięci wszystkie zmienne
clear z	usuwa z pamięci zmienną o nazwie *z*
clear z1 z2 z3	usuwa z pamięci wymienione zmienne
clear global z	usuwa z pamięci zmienną globalną o nazwie *z*
clear all	usuwa z pamięci wszystkie zmienne i funkcje; pozostawia przestrzeń roboczą zupełnie pustą

Ponieważ zmienne przechowywane są w pamięci tylko podczas danej sesji pracy z pakietem (lub dopóki nie zostaną usunięte), można zapisać całą aktualną zawartość przestrzeni roboczej w pliku, a później ją odtworzyć (tabela 1.14).

Tabela 1.14. Funkcje obsługujące zapis i odczyt danych

Funkcja	Opis
save	zapisuje binarnie wszystkie zmienne w pliku **matlab.mat**
save plik	zapisuje binarnie wszystkie zmienne w pliku o nazwie **plik.mat**

Funkcja	Opis
save plik lista	zapisuje binarnie w pliku o nazwie **plik.mat** tylko zmienne wymienione jako *lista*
load	wczytuje zmienne zapisane w pliku **matlab.mat**
load plik	wczytuje zmienne zapisane w pliku **plik.mat**
load plik.rozsz	wczytuje zmienne zapisane w pliku tekstowym o podanej nazwie i dowolnym rozszerzeniu; dane muszą tworzyć tablicę prostokątną; wczytane dane zostaną zapisane w macierzy o nazwie **plik**

Ćwiczenie 1.22

Wykonaj następujące polecenia:

a) zdefiniuj zmienne k=2 i pi=5,

```
» k=2; pi=5;
```

b) oblicz wyrażenia $x = k\pi$,

```
» x=k*pi
x =
    10
```

jak widać, domyślna wartość stałej **pi** została zastąpiona nową

c) zapisz zmienne k i pi w pliku o nazwie **wynik.mat**,

```
» save wynik k pi
```

d) przywróć domyślną wartość stałej **pi**,

```
» clear pi
```

obliczając ponownie $x = k\pi$, otrzymamy wynik

```
x =
    6.2832
```

e) odczytaj dane zapisane w pliku **wynik.mat**.

```
» load wynik
```

Ponieważ w punkcie c) w pliku **wynik.mat** zostały zapisane zmienne k=2 i pi=10, ostatnie polecenie odczyta te wartości i znów zmieni wartość stałej **pi**.

1.6.3. Polecenia systemowe

Matlab zawiera bardzo wygodne polecenia, umożliwiające wykonanie typowych poleceń systemu operacyjnego, takich jak: przeglądanie zawartości pliku i katalogu, czy usunięcie pliku. W opisie tych poleceń często występuje pojęcie ***katalogu bieżącego***. Jest to katalog, w którym zapisywane są pliki tworzone podczas pracy z pakietem.

Tabela 1.15. Funkcje obsługujące polecenia systemowe

Funkcja	Opis
dir lub **ls**	wyświetla pliki w bieżącym lub podanym katalogu (dozwolone jest użycie znaków masek: *, ?)
cd katalog	zmienia katalog bieżący na podany
delete plik	usuwa plik o podanej nazwie
pwd	wyświetla pełną ścieżkę określającą katalog bieżący
!polecenie	wykonuje dowolne polecenie systemu operacyjnego (np. **!type matlab.txt** wyświetli zawartość pliku **matlab.txt**)

1.6.4. Funkcje Matlaba, lista ścieżek

Funkcje Matlaba dzielą się ogólnie na:

- ***wbudowane*** – będące częścią jądra pakietu, do których użytkownicy pakietu nie mają dostępu (np. **sqrt**);
- ***implementowane w m-plikach*** – przechowywane w ogólnie dostępnych plikach (np. **sinh**) takie m-pliki użytkownicy mogą również tworzyć sami (więcej o m-plikach w rozdziale 2).

Kiedy interpreter Matlaba natrafia w wierszu poleceń np. na nazwę **kot**, to:

- szuka zmiennej **kot**;
- sprawdza, czy **kot** jest funkcją wbudowaną;
- szuka w bieżącym katalogu pliku **kot.m**;
- szuka pliku **kot.m**, sprawdzając listę ścieżek.

Lista ścieżek (ang. *search path*) to lista katalogów, do których Matlab ma dostęp. Jest zdefiniowana w pliku **pathdef.m**, znajdującym się w podkatalogu **toolbox\local** katalogu z Matlabem. Można ją wyświetlać lub zmieniać, jak to pokazano w tabeli 1.16.

Tabela 1.16. Funkcje zarządzające listą ścieżek

Funkcja	Opis
path	wyświetla listę ścieżek
path(path,s1)	dodaje do listy ścieżek katalog określony łańcuchem *s1*
path(s1)	zmienia listę ścieżek na składającą się tylko z katalogu określonego łańcuchem *s1*

Ćwiczenie 1.23

Po uruchomieniu pakietu Matlab wykonaj następujące polecenia:

a) wyświetl katalog bieżący, jego zawartość i listę dostępnych ścieżek,

```
» pwd, dir, path
```

b) zmień katalog bieżący na **A:** ,

```
» cd A:\
```

c) utworz podkatalog **TESTY** i uczyń go katalogiem bieżącym,

```
» !md testy, cd testy
```

d) utwórz zmienne x=12, y=14 i zapisz je w pliku binarnym **dane.bin**,

```
» x=12, y=14, save dane.bin x y
```

e) wyświetl katalog bieżący, jego zawartość i zawartość pliku **dane.bin**,

```
» pwd, dir, type dane.bin
```

f) usuń plik **dane.bin** i katalog **TESTY**.

```
» delete dane.bin, cd .. , !rd testy
```

1.6.5. Pomiar czasu

Do odmierzania upływu czasu (np. podczas wykonywania obliczeń) i określania aktualnej daty służą funkcje zestawione w tabeli 1.17.

Tabela 1.17. Funkcje obsługujące pomiar czasu

Funkcja	Opis
clock	podaje aktualną datę i czas w postaci sześcioelementowego wektora [*rok miesiąc dzień godzina minuta sekunda*]
date	podaje aktualną datę w postaci łańcucha o formacie: '*dd-mmm-rrrr*'
etime(t2, t1)	podaje różnicę czasu, który upłynął między chwilami **t1** i **t2** (**t1**, **t2** – wektory o formacie, jak w poleceniu **clock**)
tic	zeruje odmierzanie czasu przed użyciem polecenia **toc**
toc	podaje czas (w sekundach), który upłynął od momentu użycia polecenia **tic**

Ćwiczenie 1.24

Wygeneruj wektor **x**=1:0.0001:1000 i sprawdź, ile czasu zajęło jego utworzenie.

```
» tic, x=1:0.0001:1000; toc
elapsed_time =
    1.7530
```

Próba wygenerowania wektora **x**=1:0.0001:10000 zakończy się już niepowodzeniem i komunikatem o przekroczeniu dostępnej pamięci:

```
??? Error using ==> :
Out of memory. Type HELP MEMORY for your options.
```

1.7. Zadania do samodzielnego wykonania

Zadanie 1.1

Utwórz macierze **A** i **B** oraz wektor **f** określone jako:

$$\mathbf{A}=\begin{bmatrix}2 & -7\\ 5 & 4\end{bmatrix},\ \mathbf{B}=\begin{bmatrix}6 & 1\\ 4 & -3\end{bmatrix},\ \mathbf{f}=\begin{bmatrix}4 & 1\end{bmatrix}.$$

Zadanie 1.2

Wykonaj polecenia:

a) wyświetl rozmiar macierzy **A** oraz wektora **f** z zadania 1.1,
b) oblicz transpozycję macierzy **B**,
c) oblicz wyrażenia: $(\mathbf{A}+\mathbf{B})^2+2(\mathbf{A}-\mathbf{B})$,
d) utwórz macierz $\mathbf{C}=[[\mathbf{A}]\ \ [\mathbf{B}]]$ i wektor $\mathbf{h}=[\mathbf{f}\ \ \mathbf{f}]$,
e) oblicz iloczyn $\mathbf{C}\cdot\mathbf{h}$.

Zadanie 1.3

Wykonaj polecenia:

a) wyświetl wszystkie zmienne lokalne,
b) zapisz zmienne w pliku **dane**,
c) usuń wszystkie zmienne,
d) wyświetl zawartość katalogu roboczego.

Zadanie 1.4

Wykonaj polecenia:

a) utwórz 24-elementowy wektor **x** = [1 2 ... 24],
b) za pomocą funkcji **reshape** utwórz z wektora **x** macierz **Y** o postaci:

$$\mathbf{Y}=\begin{bmatrix}1 & 7 & 13 & 19\\ 2 & 8 & 14 & 20\\ \dots & \dots & \dots & \dots\\ 6 & 12 & 18 & 24\end{bmatrix}$$

Zadanie 1.5

Oblicz:

a) $e^{2\sin(2\pi)}$,

b) $\cos\left(\frac{\pi}{3}\right)^4$,

c) $\ln(\sqrt{5})$.

Zadanie 1.6

Oblicz moduł liczby zespolonej **z** = 3-2 *j*, jeje argument oraz liczbę zespoloną sprzężoną.

Zadanie 1.7

Utwórz macierze o rozmiarze 3x4:

a) o wszystkich elementach równych 1,
b) o wszystkich elementach równych 0,
c) wypełnioną liczbami pseudolosowymi.

2. Programowanie w Matlabie

Jak już wspomniano na wstępie, praca w Matlabie może się odbywać zarówno w trybie konwersacyjnym, jak też w trybie pośrednim, po uruchomieniu specjalnego pliku tekstowego. Taki plik (tzw. ***m-plik***) powinien mieć rozszerzenie **.m** i zawierać kod napisany w języku Matlab. Nazwa m-pliku musi się zaczynać literą.

Są dwa rodzaje m-plików – ***skrypty*** i ***funkcje***. Zarówno w skryptach, jak i w funkcjach można używać instrukcji języka Matlab, których składnia odpowiada składni znanej z klasycznych języków programowania

2.1. Instrukcje

2.1.1. Wyrażenia warunkowe

Wyrażenia warunkowe służą do porównywania zmiennych ***o tych samych rozmiarach***. W szczególnym przypadku porównywane zmienne mogą być skalarami. Jeśli wyrażenie warunkowe jest spełnione, porównanie zwraca wtedy wartość 1 (wartość logiczną „prawda"). W przeciwnym przypadku zwracana jest wartość 0 (wartość logiczna „fałsz").

W wyrażeniach warunkowych mogą występować operatory porównania i operatory logiczne, zestawione w tabelach 2.1 i 2.2.

Tabela 2.1. Operatory porównania

Operator	Relacja
a == b	$a = b$
a ~= b	$a \neq b$
a < b	$a < b$
a > b	$a > b$
a <= b	$a \leq b$
a >= b	$a \geq b$

Tabela 2.2. Operatory logiczne

Operator	Opis	Relacja
a | b	alternatywa	*a* lub *b*
a & b	koniunkcja	*a* i *b*
~a	negacja	nie *a*

Jeśli porównywane zmienne są macierzami lub wektorami o tych samych rozmiarach, to porównanie wykonywane jest element po elemencie (tablicowo) i zwraca macierz, której elementem o indeksie *ij* jest 1 (jeśli warunek jest spełniony) lub 0 (w przeciwnym wypadku).

Funkcje logiczne Matlaba ułatwiają operacje porównywania macierzy. Niektóre z nich zostały przedstawione w tabeli 2.3.

Tabela 2.3. Funkcje logiczne

Funkcja	Opis
all(A)	sprawdza, czy wszystkie elementy wektora **A** są różne od 0 i zwraca wartość 1 („prawda") lub 0 („fałsz"); jeśli **A** jest macierzą, każda kolumna jest traktowana jak wektor – zwracany jest wektor wierszowy zer i jedynek
any(A)	sprawdza, czy którykolwiek z elementów wektora **A** jest różny od 0 i zwraca wartość 1 („prawda") lub 0 („fałsz"); jeśli **A** jest macierzą, każda kolumna jest traktowana jak wektor – zwracany jest wektor wierszowy zer i jedynek
isequal(A,B,...)	zwraca 1, jeśli argumenty funkcji są macierzami o jednakowym rozmiarze i zawartości – w przeciwnym wypadku zwraca 0
isempty(A)	zwraca 1, jeśli macierz **A** nie ma zawartości – w przeciwnym wypadku zwraca 0

Oprócz wymienionej w tabeli funkcji **isempty** w Matlabie można znaleźć cały zestaw funkcji logicznych, określanych ogólnie jako **is***. Są to np. **ischar**, **isreal**, **isnan** i wiele innych.

Ćwiczenie 2.1

Dla danych *a*=1, *b*=2 sprawdź wyrażenia Matlaba **a==b** i **a<b**.

```
» a=1;b=2;
» a==b
ans =
     0
```

```
» a<b
ans =
     1
```

2.1.2. Instrukcja if

Instrukcja warunkowa **if** może mieć jedną z dwóch postaci:

if *wyrażenie*
  instrukcje
end

if *wyrażenie*
  instrukcje
elseif *wyrażenie*
  instrukcje
else
  instrukcje
end

Wyrażenie w instrukcji **if** jest wyrażeniem warunkowym i powinno zwracać wartość logiczną – *instrukcje* zostaną wykonane, jeśli *wyrażenie* będzie spełnione.

Ćwiczenie 2.2

Oblicz wartość funkcji $f(x)$, określoną wzorem:

$$f(x)=\begin{cases} x^2-6 & \text{dla } x>3 \\ x & \text{dla } x\in<-1,3> \\ x^2-2 & \text{dla } x<-1 \end{cases}$$

```
» if x>3
»    f=x.*x-6;
» elseif x>=-1 & x<=3
»    f=x;
» else
»    f=x.*x-2;
» end
```

Należy zwrócić uwagę na zastosowany tu operator mnożenia tablicowego (.*).

Ćwiczenie 2.3

Za pomocą funkcji **reshape** utwórz z elementów wektora **y** macierz **Z** o rozmiarze *w*x*k*. Sprawdź, czy wektor **y** zawiera *w***k* elementów.

```
» if length(y) == w*k
»      Z=reshape(y,w,k)
» else
»      'Nie można wykonać operacji!'
» end
```

2.1.3. Instrukcja switch

Do wyboru jednego z kilku wariantów służy instrukcja **switch** postaci:

switch *wyrażenie*
  case *wartość1*
    instrukcje
  case *wartość2*
    instrukcje
  ...
  otherwise
    instrukcje
end

Wyrażenie w instrukcji **switch** może być liczbą lub łańcuchem znakowym. Wartość *wyrażenia* jest porównywana z wartościami kontrolnymi kolejnych przypadków **case** (*wartość1*, *wartość2* ...). Jeśli jedna z wartości kontrolnych jest równa *wyrażeniu*, wykonywane są odpowiednie *instrukcje*, po czym następuje opuszczenie bloku **switch**. Jeśli żadna z wartości kontrolnych nie odpowiada poszukiwanemu *wyrażeniu*, wykonywane są *instrukcje* po opcjonalnym słowie **otherwise**.

Ćwiczenie 2.4

Utwórz macierz **A** o rozmiarze *nxn*. W zależności od wartości zmiennej *c* ma to być:

a) gdy *c*=0 – macierz o wszystkich elementach równych 0,
b) gdy *c*=1 – macierz o wszystkich elementach równych 1,
c) gdy *c*=2 – macierz jednostkowa,
d) dla innych wartości *c* – macierz wypełniona liczbami pseudolosowymi.

```
» switch c
»    case 0
»       A=zeros(n)
»    case 1
»       A=ones(n)
»    case 2
»       A=eye(n)
»    otherwise
»       A=rand(n)
» end
```

2.1.4. Pętla for

Ogólna postać instrukcji **for** jest następująca:

for *zmienna* = *macierz_wartości*,
  instrukcje
end

W czasie wykonywania instrukcji **for** kolumny *macierzy_wartości* przyporządkowywane są kolejno iterowanej *zmiennej*.

W praktyce wyrażenie *macierz_wartości* ma najczęściej jedną z postaci:

- **minimum : maksimum**
- **minimum : krok : maksimum**

przy czym *krok* może być zarówno dodatni, jak i ujemny, a *zmienna* nie musi przyjmować wartości całkowitych.

Ćwiczenie 2.5

Utwórz taką macierz **A** o rozmiarze 5x4, że $A_{ij} = \frac{i+j}{i+j+1}$

```
» for i = 1:5,
»     for j = 1:4,
»         A(i,j) = (i+j)/(i+j+1);
»     end
» end
```

2.1.5. Pętla while

Instrukcja **while** ma postać:

while *wyrażenie,*
  instrukcje
end

Instrukcje w bloku **while** są powtarzane dopóty, dopóki część rzeczywista *wyrażenia* ma wszystkie elementy różne od zera (warunek przyjmuje wartość „prawda"). Postać *wyrażenia* jest taka sama, jak w instrukcji **if**.

Ćwiczenie 2.6

Oblicz wartość setnego elementu ciągu Fibonacciego danego wzorem:

$$\begin{cases} u_1 = 1 \\ u_2 = 1 \\ \dots \\ u_n = u_{n-1} + u_{n-2} \quad \text{dla } n > 2 \end{cases}$$

```
» u1=1; u2=1; i=2;n=100;
» while i<n
»       u3=u1+u2;
»       u1=u2;
»       u2=u3;
```

```
»          i=i+1;
»    end
»    u3
```

lub to samo zapisane w bardziej zwartej (ale mniej czytelnej) postaci:

```
»    u1=1; u2=1; i=2;n=100;
»    while i<n,u3=u1+u2;u1=u2;u2=u3;i=i+1;end
»    u3
```

W wyniku działania powyższych instrukcji otrzymamy:

```
u3 =
     3.5422e+020
```

Warto przypomnieć, że umieszczenie średnika po instrukcji powoduje niewyświetlenie wyników jej działania na ekranie. Natomiast przecinek służy jako separator i jest konieczny wówczas, gdy w jednym wierszu poleceń chcemy umieścić więcej instrukcji, które dodatkowo nie są zakończone średnikiem (jeśli jest średnik – przecinek nie jest konieczny).

2.1.6. Instrukcja break

Instrukcja **break** zatrzymuje wykonywanie pętli **for** lub **while** i powoduje przejście do następnej instrukcji za pętlą. W Matlabie instrukcja **break** ***nie jest wykorzystywana*** w bloku **switch**. Inaczej niż w C i jego językach pochodnych, w bloku tym po wykonaniu instrukcji związanej z jednym przypadkiem ***nie następuje przejście do następnego przypadku*** i użycie **break** nie jest potrzebne.

Ćwiczenie 2.7

Poniższa pętla nie jest, wbrew pozorom, nieskończona (co sugerowałby zapis **while 1**). Zakończenie działania pętli następuje bowiem w wyniku wykonania instrukcji **break** dla wartości n=6.

```
»    n=0;
»    while 1
»          n=n+1
»          if n>5, break, end
»    end
```

lub w jednym wierszu:

```
»    n=0; while 1, n=n+1, if n>5, break, end, end
```

2.2. Skrypty

Skrypt jest plikiem tekstowym o rozszerzeniu **.m** (m-plikiem), zawierającym polecenia i instrukcje Matlaba. Skrypty nie pobierają żadnych argumentów wejściowych ani nie zwracają argumentów wyjściowych – mogą operować tylko na zmiennych dostępnych w przestrzeni roboczej Matlaba. Tam też są zapisywane wszystkie zmienne utworzone w skryptach.

2.2.1. Tworzenie skryptu

Skrypt można utworzyć za pomocą dowolnego edytora plików tekstowych lub bezpośrednio w pakiecie Matlab. Należy pamiętać, że (podobnie jak w oknie poleceń) skrypt po uruchomieniu wyświetli wynik działania każdego polecenia, które nie zostało zakończone średnikiem.

Komentarze w skrypcie poprzedza się znakiem **%**. Interpreter zignoruje tekst znajdujący się między znakiem **%** a końcem wiersza.

Można tworzyć objaśnienia („helpy") do własnych skryptów, umieszczając odpowiedni komentarz w pierwszych wierszach skryptu (nie jest istotna liczba wierszy, tylko ich lokalizacja). Objaśnienie zostanie wyświetlone na ekranie po wywołaniu polecenia:

```
» help nazwa_skryptu
```

gdzie **nazwa_skryptu** oznacza nazwę pliku tekstowego bez rozszerzenia **.m**.

Ćwiczenie 2.8

Napisz skrypt, który generuje macierz 3x3 wypełnioną liczbami pseudolosowymi, a następnie tworzy z niej macierz trójkątną dolną oraz macierz trójkątną górną. Skrypt zachowaj w pliku **p1.m**.

```
% Pierwszy przykładowy skrypt
% Generuje macierz 3x3 wypełnioną liczbami pseudolosowymi
% i tworzy z niej macierz trójkątną dolną oraz górną
A=rand(3)      % tworzenie macierzy A
Ad=tril(A)     % tworzenie macierzy Ad
Ag=triu(A)     % tworzenie macierzy Ag
% koniec pierwszego przykładowego skryptu
```

2.2.2. Uruchamianie skryptu

Skrypt uruchamia się, podając jego nazwę (bez rozszerzenia) w wierszu poleceń Matlaba lub w innym skrypcie. W wypadku skryptu z ćwiczenia 2.8 o nazwie **p1.m** poleceniem takim będzie po prostu:

```
» p1
```

jeśli tylko skrypt znajduje się w katalogu bieżącym lub w katalogu udostępnionym poleceniem **path**.

Uruchomienie skryptu spowoduje wykonanie kolejno wszystkich poleceń w nim zawartych i wyświetlenie wyniku:

```
A =
    0.6038    0.0153    0.9318
    0.2722    0.7468    0.4660
    0.1988    0.4451    0.4186
Ad =
    0.6038         0         0
    0.2722    0.7468         0
    0.1988    0.4451    0.4186
Ag =
    0.6038    0.0153    0.9318
         0    0.7468    0.4660
         0         0    0.4186
```

Gdyby w skrypcie po instrukcjach tworzenia macierzy **A**, **Ad** i **Ag** umieszczono średniki, zmienne zostałyby utworzone, ale na ekranie nie pojawiłby się żaden efekt uruchomienia skryptu. Po wykonaniu skryptu zmienne **A**, **Ad** i **Ag** zostaną zapisane w przestrzeni roboczej Matlaba.

Natomiast polecenie:

```
» help p1
```

wyświetli informację:

```
Pierwszy przykladowy skrypt
Generuje macierz 3x3 wypelniona liczbami pseudolosowymi
i tworzy z niej macierz trójkatna dolna oraz górna
```

2.2.3. Obsługa wejścia/wyjścia skryptu

Podczas wykonywania skryptu można wpisywać dane z klawiatury oraz wyświetlać wyniki w oknie poleceń Matlaba. Jest to najprostsza forma obsługi w tym języku wejścia/wyjścia (ang. *input/output*, I/O).

W tabeli 2.4 zestawiono polecenia wykorzystywane do obsługi wejścia skryptów. Natomiast aby wyświetlić wartość zmiennej, wystarczy wpisać w wierszu jej nazwę lub (bardziej elegancko) użyć znanej z rozdziału 1 funkcji **disp**, umożliwiającej także wyświetlanie tekstów.

Tabela 2.4. Funkcje obsługi wejścia skryptu

Funkcja	Opis
x=input(tekst)	wyświetla łańcuch *tekst*, oczekuje na wpisanie przez użytkownika ***danej liczbowej*** i przypisuje ją zmiennej liczbowej *x*; zamiast danej liczbowej można wpisać wyrażenie Matlaba, które funkcja obliczy
x=input(tekst,'s')	wyświetla łańcuch *tekst*, oczekuje na wpisanie przez użytkownika ***łańcucha znakowego*** i przypisuje go zmiennej ***x***
pause	zatrzymuje wykonywanie skryptu do momentu naciśnięcia przez użytkownika dowolnego klawisza
pause(n)	zatrzymuje wykonywania skryptu na *n* sekund

Ćwiczenie 2.9

Napisz skrypt **kwadrat**, obliczający pierwiastki równania kwadratowego. W skrypcie wykonaj następujące polecenia:

a) wczytaj współczynniki równania kwadratowego *a*, *b*, *c*,
b) wyświetl pierwiastki równania *x1* i *x2*.

Poniższy skrypt umieść w pliku **kwadrat.m**.

```
%Skrypt oblicza rzeczywiste pierwiastki
%równania kwadratowego y=a*x^2+b*x+c
a=input('a='); b=input('b='); c=input('c=');
delta=b.*b-4*a.*c;
if delta > 0
   x1=(-b-sqrt(delta))/(2*a); x2=(-b+sqrt(delta))/(2*a);
   disp('x1='),disp(x1)
   disp('x2='),disp(x2)
elseif delta == 0
   x1=-b/(4*a); disp('x1=x2='), disp(x1)
else
   disp('Równanie nie ma pierwiastków rzeczywistych')
end
```

Po wydaniu polecenia

```
» kwadrat
```

interpreter Matlaba wyświetli tekst

```
a=
```

i będzie oczekiwał wpisania liczby. Procedura zostanie powtórzona dla danych *b* i *c*.

Przyjmując a=1, b=1, c=-2, otrzymamy wynik

```
x1=
     -2
x2=
      1
```

Ponieważ po wykonaniu każdego polecenia Matlaba następuje przejście do nowego wiersza, aby uzyskać bardziej elegancki efekt działania skryptu, należy połączyć wyświetlanie tekstu i liczby w jedno polecenie. W tym celu trzeba przekształcić zmienne liczbowe *x1*, *x2* na łańcuchy znakowe. Ponieważ jednak argumentem funkcji **disp** może być tylko jedna zmienna, łańcuchy należy następnie połączyć. Odpowiednie wiersze skryptu przyjmą wtedy postać:

```
disp(strcat('x1=', num2str(x1)))
disp(strcat('x2=', num2str(x2)))
```

a uruchomienie skryptu da wynik

```
x1=-2
x2=1
```

Chcąc wyświetlić wyniki w jednym wierszu, należy dokonać kolejnej modyfikacji:

```
disp(strcat('x1=', num2str(x1),', x2=', num2str(x2) ))
```

co da wyniki w postaci:

```
x1=-2, x2=1
```

2.3. Funkcje

2.3.1. Definicja funkcji

Jak już wspomniano w rozdziale 1.6.4, funkcje Matlaba dzielą się na wbudowane oraz przechowywane w m-plikach. Takie specjalne m-pliki z definicjami własnych funkcji może tworzyć również sam użytkownik.

Pierwszy wiersz m-pliku musi zawierać definicję nowej funkcji:

- słowo kluczowe **function**;
- nazwę funkcji – musi być taka sama, jak nazwa pliku (bez rozszerzenia **.m**), w którym znajduje się funkcja;
- wartości funkcji (lista argumentów wyjściowych);
- parametry funkcji (lista argumentów wejściowych).

Definicja funkcji wygląda następująco:

function [*wart_fun1*, *wart_fun2*,....] = *nazwa_funkcji*(*parametr1*, *parametr2*,...)
% ewentualny opis funkcji w formie komentarza – w drugim wierszu pliku
instrukcje

W ciele funkcji, wśród instrukcji, powinno się znaleźć przypisanie:

wart_fun1 = wynik1;
wart_fun2 = wynik2;
...

Tak zdefiniowana funkcja może być używana na równi z innymi funkcjami Matlaba.

Podczas wykonywania funkcji (podobnie jak podczas wykonywaniu skryptu) zostanie wyświetlony wynik działania każdego polecenia niezakończonego średnikiem.

Ćwiczenie 2.10

Zdefiniuj funkcję **esin(x)** obliczającą wyrażenie $e^{\sin(x)}$.

```
function y = esin(x)
y = exp(sin(x));
```

Funkcja **esin** musi zostać zapisana w pliku **esin.m**. Użyć jej można np. do obliczenia wyrażenia $d = e^{\sin(2)}$:

```
» d=esin(2)
d =
    2.4826
```

W funkcji nie umieszczono jej opisu.

Ćwiczenie 2.11

Napisz funkcję przeliczającą radiany na stopnie. Sprawdź działanie funkcji, przeliczając na stopnie argument liczby zespolonej z=2-3j:

```
function [y]=radst(x)
%% przelicza radiany na stopnie
%% postać ogólna: radst(x), gdzie x-ilość radianów
y=(x.*180)/pi;
%% koniec
```

Funkcję należy zapisać w pliku **radst.m** – polecenie

```
» help radst
```

wyświetli wynik

```
przelicza radiany na stopnie
postać ogólna: radst(x), gdzie x-ilość radianów
```

Aby przeliczyć na stopnie argument liczby zespolonej, należy wydać polecenie:

```
» radst(angle(2-3j))
ans =
      -56.3099
```

Ćwiczenie 2.12

Zdefiniuj funkcję **prosta**, która oblicza współczynniki *A*, *B*, *C* prostej $Ax+By+C=0$ przechodzącej przez dwa dane punkty: P1($x1$,$y1$) i P2($x2$,$y2$).

```
function [A, B, C]=prosta(x1,y1,x2,y2)
% Wynikiem działania funkcji prosta o 4 parametrach
% wejściowych będących współrzędnymi dwóch punktów
% P1(x1,y1)i P2(x2,y2) są współczynniki A, B, C
% prostej Ax+By+C=0 przechodzącej przez te punkty
A=y2-y1;
B=x1-x2;
C=y1.*x2-y2.*x1;
```

Powyższa funkcja musi być zapisana w pliku **prosta.m**

W celu policzenia współczynników prostej przechodzącej przez punkty P(1,1) i Q(2,2) funkcję **prosta** może wywołać:

```
» [A1,B1,C1]=prosta(1,1,2,2)
```

a wynikiem wywołania będzie:

```
A1 =
     1
B1 =
    -1
C1 =
     0
```

Natomiast polecenie **help prosta** wyświetli informację:

```
Wynikiem działania funkcji prosta o 4 parametrach
wejściowych będących współrzędnymi dwóch punktów
P1(x1,y1)i P2(x2,y2) są współczynniki A, B, C
prostej Ax+By+C=0 przechodzącej przez te punkty
```

2.3.2. Zmienne lokalne i globalne

Zmienne występujące w funkcjach (również argumenty wejściowe) są ***lokalne*** w ciele funkcji i nieprzechowywane w przestrzeni roboczej Matlaba. Nie są też widziane przez inne funkcje czy skrypty. Zmienne utworzone w przestrzeni roboczej Matlaba (np. w skryptach) nie są z kolei dostępne w ciele funkcji. Można jednak uczynić zmienną ***globalną***. Służy do tego polecenie **global**, postaci np.

```
» global W
```

Polecenie takie należy umieścić ***wszędzie tam, gdzie zmienna ma być widoczna*** – w ciele funkcji, skrypcie lub bezpośrednio w wierszu poleceń.

Ćwiczenie 2.13

Napisz funkcję **funkw**, która dla danego x obliczać będzie wartość $y(x)=ax^2+bx+c$. Funkcję tę wykorzystaj w skrypcie **kwadrat2.m** do obliczenia $y(x)$ w przedziale <-10,10>. Wartości parametrów a, b, c określ w skrypcie **kwadrat2.m**.

Poniżej przedstawiono przykładową zawartość obu plików.

Plik **funkw.m**

```
function y=funkw(x)
global a b c
y=a*x.*x+b*x+c
```

Plik **skrypt2.m**

```
global a b c
a=1; b=-2; c=1;
x=[-10:10];
y=funkw(x)
```

Po zadeklarowaniu zmiennych a, b, c jako globalne można ich używać wewnątrz funkcji **funkw**, jak gdyby były zadeklarowane lokalnie lub przekazane do funkcji jako argument.

2.3.3. Instrukcja return

Instrukcja **return** jest wykorzystywana do wymuszenia natychmiastowego zakończenia funkcji i powrotu do miejsca jej wywołania.

Ćwiczenie 2.14

Napisz funkcję obliczającą wartość $n!$ (silnia). Dla każdej wartości $n<1$ funkcja powinna zwracać wartość 0.

```
function s=silnia(n)
% funkcja oblicza n!, dla n<1 zwraca wartość 0
s=0;
if (n<1)return
else
   s=1;
```

```
   for i=2:n, s=s*i;end
end
```

Przykładowe wywołania funkcji:

```
» k=silnia(4)
k =
    24

» k=silnia(-4)
k =
     0
```

2.3.4. Funkcje inline

Poza dotąd omówionymi funkcjami w Matlabie można również stosować tzw. funkcje **inline** („w wierszu"). Funkcję taką można zdefiniować za pomocą polecenia:

funkcja=inline(wyrażenie)

gdzie:

- **funkcja** – nazwa definiowanej funkcji,
- *wyrażenie* – wyrażenie Matlaba w postaci łańcucha znaków.

Funkcje **inline** należy stosować jedynie do prostych obliczeń i nie nadużywać ich.

Ćwiczenie 2.15

Zdefiniuj jako funkcję **inline**:

a) funkcję jednej zmiennej $f1(x)=\cos(2x-\pi)$ oraz oblicz jej wartość dla $x=3$,

```
» f1=inline('cos(2*x-pi)')
» y=f1(3)
f1 =
     Inline function:
     f1(x) = cos(2*x-pi)
y =
   -0.9602
```

b) funkcję dwóch zmiennych $f2(x,y)=x^2+y^2$ oraz oblicz jej wartość dla $x=3$, $y=-1$.

```
» f2=inline('x.^2+y.^2');
» z=f2(3,-1)
f2 =
     Inline function:
     f2(x,y) = x.^2+y.^2
z =
    10
```

Taki sam efekt osiągniemy, definiując funkcję w postaci

```
» f2a=inline('x(1).^2+x(2).^2');
f2a =
     Inline function:
     f2a(x) = x(1).^2+x(2).^2
```

Tym razem jednak podczas wywołania funkcji należy pamiętać, że jej argumentem będzie wektor **x** o dwóch składowych $x(1)$ i $x(2)$, czyli:

```
» z=f2a([3,-1])
z =
    10
```

2.4. Zadania do samodzielnego wykonania

Zadanie 2.1

Utwórz taką macierz **A** o rozmiarze 6x6, że: $a_{ij} = \begin{cases} \dfrac{1}{i-j} & dla\ \ i \neq j \\ 0 & dla\ \ i = j \end{cases}$

Zadanie 2.2

Wygeneruj macierz o rozmiarze 10x10, wypełnioną liczbami pseudolosowymi. Wyświetl wszystkie elementy macierzy z przedziału (0.2,0.5).

Zadanie 2.3

Napisz następujący skrypt:

a) w pierwszych trzech wierszach tekstu zamieść komentarz:

```
Skrypt  ten jest skryptem testowym
Obliczane jest wyrazenie sin(a)*cos(b),
gdzie a, b - dwie wczytane liczby
```

b) zamieść polecenie wczytywania dwóch danych: a i b,

c) oblicz wyrażenie $\sin(a)\cdot\cos(b)$.

Uruchom skrypt, a następnie za pomocą funkcji **help** wyświetl objaśnienia do niego.

Zadanie 2.4

Napisz skrypt, którego zadaniem będzie prezentacja operacji na macierzach. W skrypcie wykonaj następujące polecenia:

a) wczytaj:
   - rozmiar macierzy (n),

- dwie macierze o rozmiarze *nxn*: **A** i **B**,

b) oblicz:
- macierz odwrotną do **A**,
- transpozycję macierzy **B**,
- sumę macierzy – **A+B**,
- różnicę macierzy – **A-B**,
- iloczyn macierzowy – **A·B**,
- iloczyn macierzy **A** przez dowolną liczbę.

Zadanie 2.5

Napisz skrypt porównujący czas operacji obliczania wartości funkcji $y=\cos(x)$ w punktach z przedziału <-10,10> z krokiem 0,001 w przypadku:

a) obliczeń $y(i)$ realizowanych w pętli (element po elemencie),
b) tablicowo (jednorazowo).

Zadanie 2.6

Napisz funkcję **metrs**, dokonującą przeliczenia prędkości wyrażonej w [km/h] na prędkość w [m/s].

3. Operacje na plikach

Oprócz opisanej w rozdziale 2.2.3 obsługi wejścia/wyjścia skryptów Matlab umożliwia przeadresowywanie wejścia i wyjścia programu (*input/output*, I/O) do lub z plików. Tak zwane ***funkcje I/O niskiego poziomu*** pozwalają zapisywać dane generowane przez pakiet w żądanym formacie lub czytać dane zapisane w innych formatach. Definicje funkcji I/O Matlaba są oparte na analogicznych definicjach funkcji I/O z języka C [3].

3.1. Otwieranie i zamykanie plików

Przed zapisaniem lub odczytaniem danych należy ***otworzyć*** plik za pomocą funkcji **fopen**:

id_pliku = fopen(nazwa_pliku, rodzaj_dostępu)

gdzie

- *nazwa_pliku* – łańcuch znaków z nazwą otwieranego pliku,
- *rodzaj_dostępu* – łańcuch znaków o dopuszczalnych wartościach jak w tabeli 3.1.

Tabela 3.1. Wartości argumentu *rodzaj_dostępu* funkcji **fopen**

Wartość argumentu	Opis
'r'	otwarcie pliku do odczytu
'w'	usunięcie zawartości istniejącego pliku lub utworzenie nowego i otwarcie go do zapisu
'a'	otwarcie pliku w celu dopisywania elementów na jego końcu
'r+'	otwarcie pliku do odczytu i zapisu
'w+'	usunięcie zawartości istniejącego pliku lub utworzenie nowego i otwarcie go do odczytu i zapisu
'a+'	otwarcie pliku w celu czytania lub dopisywania elementów na jego końcu

Funkcja **fopen** otwiera plik wskazany łańcuchem *nazwa_pliku* i zwraca unikatowy ***identyfikator pliku*** (zmienną *id_pliku*). Identyfikator ten powinien być używany

we wszystkich operacjach wejścia i wyjścia wykonywanych na danym pliku. Jeśli operacja otwarcia pliku zakończy się sukcesem, zmienna *id_pliku* będzie nieujemną liczbą całkowitą, w przeciwnym przypadku przyjmie wartość -1.

Druga postać wywołania funkcji **fopen** jest następująca:

[id_pliku, informacja] = fopen(nazwa_pliku, rodzaj_dostępu)

Informacja jest łańcuchem znakowym, który może być pomocny w ustaleniu przyczyny błędu. Jest on zwracany tylko w przypadku, kiedy operacja otwarcia pliku zakończy się niepowodzeniem.

Po zakończeniu operacji na pliku należy go ***zamknąć***. Plik o podanym identyfikatorze zamyka funkcja **fclose** postaci:

status=fclose(id_pliku)

Natomiast wszystkie otwarte pliki zamyka polecenie:

status=fclose('all')

Jeśli operacja zamknięcia pliku się powiedzie, zmienna *status* przyjmie wartość 0, w przeciwnym razie *status* = -1.

Ćwiczenie 3.1

Otwórz do odczytu plik **dane.dat**.

```
» p0 = fopen('dane.dat', 'r');
```

Identyfikatorem pliku **dane.dat** jest teraz zmienna *p0*. Plik ten zamknie polecenie:

```
» st = fclose(p0);
```

3.2. Pliki binarne

3.2.1. Zapis danych w pliku binarnym

Funkcja **fwrite** zapisuje elementy macierzy **A** w pliku binarnym określonym identyfikatorem *id_pliku*. Funkcja zwraca ilość zapisanych danych. Poniżej przedstawiono sposób wywołania funkcji.

liczba = fwrite(id_pliku, A, typ)

Argument funkcji *typ* pozwala określić, na ilu bitach mają być zapisane dane i jak powinny być zinterpretowane (jako znaki, jako liczby całkowite czy jako liczby rzeczywiste). Wartością domyślną argumentu jest *'uchar'*.

Tabela 3.2. Wartości argumentu *typ* funkcji **fwrite** (niezależne od platformy systemowej)

Wartość argumentu	Interpretacja
'uchar'	pojedynczy znak zapisany na 8 bitach bez znaku +/- (ang. *unsigned char*)
'schar'	pojedynczy znak zapisany na 8 bitach, w tym jeden bit przeznaczony na znak +/- (ang. *signed char*)
'int8', 'int16', 'int32','int64'	liczba całkowita ze znakiem zapisana odpowiednio na 8,16,32 lub 64 bitach
'uint8', 'uint16', 'uint32','uint64'	liczba całkowita bez znaku (ang. *unsigned integer*) zapisana odpowiednio na 8,16,32 lub 64 bitach
'single'	liczba zapisana w formacie zmiennopozycyjnym na 32 bitach
'float32'	liczba zapisana w formacie zmiennopozycyjnym na 32 bitach
'double'	liczba zapisana w formacie zmiennopozycyjnym na 64 bitach
'float64'	liczba zapisana w formacie zmiennopozycyjnym na 64 bitach

Dopuszczalne są również wartości argumentu *typ* zestawione w tabeli 3.3. Są to jednak formaty zależne od platformy systemowej i mogą wystąpić różnice w liczbie bitów przeznaczonych na zapis danych.

Tabela 3.3. Wartości argumentu *typ* funkcji **fwrite** (zależne od platformy systemowej)

Wartość argumentu	Interpretacja
'char'	znak zapisany na 8 bitach (bez lub ze znakiem)
'short'	liczba całkowita zapisana na 16 bitach
'int'	liczba całkowita zapisana na 16 bitach
'long'	liczba całkowita zapisana na 32 lub 64 bitach
'ushort'	liczba całkowita bez znaku zapisana na 16 bitach
'uint'	liczba całkowita bez znaku zapisana na 32 bitach
'ulong'	liczba całkowita bez znaku zapisana na 32 lub 64 bitach
'float'	liczba zapisana w formacie zmiennopozycyjnym na 32 bitach

Ćwiczenie 3.2

Sformuj macierz **A** o rozmiarze 10x10, wypełnioną liczbami pseudolosowymi. Utwórz plik binarny **macierz.bin** i zapisz w nim wszystkie elementy macierzy **A** w postaci 32-bitowych liczb rzeczywistych.

```
» pl2 = fopen('macierz.bin','w');
» n = fwrite(pl2, rand(10), 'float');
» status = fclose(pl2)
```

3.2.2. Odczyt danych z pliku binarnego

Do odczytywania plików binarnych służy funkcja **fread**. Jej wywołanie może mieć postać:

A = fread(id_pliku, rozmiar, typ)

lub

[A, liczba] = fread(id_pliku, rozmiar, typ)

Funkcja **fread** wczytuje dane z pliku binarnego określonego przez identyfikator *id_pliku* i zapisuje je w macierzy **A**. Opcjonalnie funkcja może zwrócić zmienną *liczba*, określającą liczbę wczytanych elementów. Z kolei liczbę elementów, które powinny zostać wczytane z pliku, można określić za pomocą opcjonalnego argumentu *rozmiar*. Dopuszczalne wartości argumentu zestawiono w tabeli 3.4. Jeśli argument *rozmiar* nie zostanie określony, funkcja wczyta wszystkie dane aż do końca pliku i zapisze je w wektorze kolumnowym.

Tabela 3.4. Wartości argumentu *rozmiar* funkcji **fread**

Wartość argumentu	Opis
n	odczytuje *n* elementów i zapisuje je w wektorze kolumnowym
[*m*, *n*]	odczytuje tyle elementów, aby wypełniły kolumnami macierz o rozmiarze *mxn*; brakujące elementy są zastępowane zerami

Trzeci argument funkcji **fread** – *typ* wczytywanych danych – określany jest podobnie jak w przypadku funkcji **fwrite** (tabele 3.2, 3.3).

Ćwiczenie 3.3

Otwórz plik binarny **macierz.bin** utworzony z ćwiczeniu3.2. Wczytaj z pliku 25 liczb rzeczywistych do:

a) kolumnowego wektora **W**

```
» plik = fopen('macierz.bin', 'r');
```

```
» W = fread(plik, 25, 'float');
» fclose(plik);
```

b) macierzy **T** o rozmiarze 5x5

```
» plik = fopen('macierz.bin', 'r');
» T = fread(plik, [5 5], 'float');
» fclose(plik);
```

3.3. Sformatowane pliki tekstowe

3.3.1. Zapis danych w pliku tekstowym

Funkcja **fprintf** o wywołaniu:

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

zwraca liczbę zapisanych bajtów. Funkcja umożliwia konwersję danych przechowywanych w części rzeczywistej macierzy **A** kolumnowo na łańcuchy znakowe i zapisanie ich w pliku tekstowym o podanym identyfikatorze *id_pliku*.

Argument *format* jest łańcuchem znakowym określającym m.in. rodzaj konwersji (tabela 3.5), szerokość pola i liczbę cyfr znaczących każdej wymienionej w funkcji **fprintf** macierzy. *Format* może także zawierać zwykłe znaki alfanumeryczne oraz znaki specjalne (tabela 3.6).

Funkcja **fprintf** może też wyświetlać sformatowane dane na ekranie. W wywołaniu pomija się wtedy argument *id_pliku*:

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

Wywołując funkcję **fprintf**, można wymieniać po przecinku wiele macierzy, które zostaną przekształcone na łańcuchy znakowe, a następnie zapisane w pliku ***kolumnami*** (kolejno: wszystkie kolumny pierwszej macierzy, wszystkie kolumny drugiej macierzy itd.)

Tabela 3.5. Rodzaje konwersji używane w argumencie *format* funkcji **fprintf**

Rodzaj konwersji	Opis
%d	do zapisu liczb całkowitych
%f	do zapisu liczb rzeczywistych w formacie stałoprzecinkowym
%e	do zapisu liczb rzeczywistych w formacie zmiennoprzecinkowym
%g	automatyczny dobór krótszego formatu (*%e* lub *%f*)
%c	do zapisu pojedynczych znaków
%s	do zapisu łańcuchów znakowych

Tabela 3.6. Znaki specjalne używane w argumencie *format* funkcji **fprintf**

Znak specjalny	Opis
\b	cofnięcie o jeden znak (ang. *backspace*)
\f	nowa strona (ang. *form feed*)
\n	nowy wiersz
\r	powrót karetki (przesunięcie kursora do początku wiersza)
\t	znak tabulatora
\"	znak apostrofu
\\	znak lewego ukośnika
%%	znak procentu

Ćwiczenie 3.4

Utwórz wektor **x** o wartościach zmieniających się od 0 do π z krokiem 0.1. Wykonaj następujące polecenia:

a) zapisz w pliku **x.txt** elementy wektora **x**,

```
» x=0:0.1:pi;
» plik1=fopen('x.txt','w')
» fprintf(plik1, '%4.2f\n',x);
» close(plik1)
```

Elementy wektora **x** zostaną zapisane w formacie stałoprzecinkowym, w polu o szerokości co najmniej czterech znaków, z dwoma znakami dziesiętnymi.

b) zapisz w pliku **sin.txt** tablicę wartości funkcji sin(x) dla poszczególnych elementów wektora **x**

```
» plik2=fopen('sin.txt','w');
» fprintf(plik2, '%4.2f   %6.5f\n',[x; sin(x)]);
» fclose(plik2);
```

W każdym wierszu pliku zostaną zapisane kolejno: element wektora **x**: x_i i odpowiadająca mu wartość sin(x_i). Aby osiągnąć taki efekt, należało utworzyć macierz, której pierwszą kolumnę tworzy wektor **x**, a drugą – wektor obliczonych wartości sinus.

Ćwiczenie 3.5

Napisz skrypt wykorzystujący funkcję **radst** z ćwiczenia 2.11 do przeliczenia radianów na stopnie. Wyniki obliczeń zapisz w pliku **radiany.txt**.

```
n=0:0.25:2;              %wektor pomocniczy
rad=n*pi;                %wektor radianów rad=0:0.25*pi:pi
```

```
st=radst(rad);                          %wektor stopni
wynik=[n;rad;st];                       %macierz pomocnicza
pr=fopen('radiany.txt','w');            %otwarcie pliku do zapisu
%zapis do pliku tytułu w postaci łańcucha znaków
fprintf(pr,'   Radiany        Stopnie\n');
%zapis do pliku łańcucha znaków i macierzy wynik
fprintf(pr,'%4.2f*pi = %4.2f    %5d\n',wynik)
fclose(pr);                             %zamknięcie pliku
```

Poniżej przedstawiono zawartość pliku **radiany.txt**.

```
    Radiany        Stopnie
0.00*pi = 0.00        0
0.25*pi = 0.79       45
0.50*pi = 1.57       90
0.75*pi = 2.36      135
1.00*pi = 3.14      180
1.25*pi = 3.93      225
1.50*pi = 4.71      270
1.75*pi = 5.50      315
2.00*pi = 6.28      360
```

Ćwiczenie 3.6

Napisz skrypt **pisz_dane.m**, który tworzy macierz liczb pseudolosowych **A** o rozmiarze 3x4. Zapisz w pliku **dane.txt** kolejno: liczbę wierszy i liczbę kolumn macierzy **A** oraz samą macierz **A**.

Zawartość skryptu widoczna jest poniżej.

```
n=4; m=3; A=rand(n,m);    %określenie n, m, A
p=fopen('dane.txt','w'); %otwarcie pliku dane.txt do zapisu
fprintf(p,'%2d\n',n);     %zapis do pliku dane.txt liczby n
fprintf(p,'%2d\n',m);     %zapis do pliku dane.txt liczby m
fprintf(p,'%5.2f %5.2f %5.2f\n',A);  %zapis macierzy A
fclose(p);                %zamknięcie pliku dane.txt
disp(A)                   %wyświetlenie zawartości macierzy A
```

Uruchomienie skryptu może spowodować wygenerowanie i wyświetlenie macierzy **A** o zawartości np.

```
0.5828    0.4329    0.5298
0.4235    0.2259    0.6405
0.5155    0.5798    0.2091
0.3340    0.7604    0.3798
```

natomiast plik tekstowy **dane.txt** będzie w takim przypadku miał zawartość:

```
4                                    (zmienna n)
3                                    (zmienna m)
0.58   0.42   0.52
0.33   0.43   0.23                   (macierz A)
0.58   0.76   0.53
0.64   0.21   0.38
```

Jak widać, wiersze macierzy **A** zostały zapisane w pliku ***kolumnami***. Jednak, jak zobaczymy w ćwiczeniu 3.9, macierz zostanie odczytana z pliku prawidłowo.

3.3.2. Odczyt danych z pliku tekstowego

Funkcja **fscanf** odczytuje wszystkie dane z pliku tekstowego o identyfikatorze *id_pliku*, konwertuje je w sposób określony w argumencie *format* i zapisuje w macierzy **A**. Jej wywołanie może mieć postać:

A = fscanf(id_pliku, format, rozmiar)

lub

[A, liczba] = fscanf(id_pliku, format, rozmiar)

W drugim wypadku funkcja może zwracać zmienną *liczba*, określającą liczbę wczytanych elementów.

Opcjonalny argument *rozmiar* określa liczbę elementów, które powinny zostać wczytane z pliku. Może on przyjmować wartości identyczne jak w wypadku funkcji **fread** (tabela 3.4). Jeśli argument *rozmiar* nie zostanie określony, dane będą zapisane w wektorze kolumnowym.

Polecenie **fscanf** dopuszcza takie same rodzaje konwersji, jak funkcja **fprintf** (tabela 3.5).

Ćwiczenie 3.7

Odczytaj informację zapisaną w pliku **sin.txt**, utworzonym w ćwiczeniu 3.4.

```
» pl=fopen('sin.txt', 'r');
» y=fscanf(pl, '%f %f')
» fclose(pl);
```

Ponieważ w poleceniu **fscanf** nie podano argumentu *rozmiar*, dane zostały zapisane w kolumnowym wektorze **y**. Aby uzyskać macierz dwukolumnową, można wydać polecenie:

```
» x=reshape(y,floor(length(y)/2),2)
```

Aby uzyskać taki sam efekt przy użyciu argumentu *rozmiar* w funkcji **fscanf**, należałoby znać rozmiar wczytywanej macierzy:

```
» y=fscanf(pl, '%f %f', [13 2])
```

Ćwiczenie 3.8

Napisz skrypt wykonujący polecenia zamieszczone w ćwiczeniu 3.7. Dodaj polecenie sprawdzania, czy plik istnieje.

```
pl=fopen('sin.txt', 'r');                %otwarcie pliku
if pl>-1                    %kontrola poprawności otwarcia pliku
  y=fscanf(pl, '%f %f');%wczytanie treści pliku do wektora y
  x=reshape(y, floor(length(y)/2),2)  %wyświetlenie macierzy
  fclose(pl);                           %zamknięcie pliku
else                        %wyświetlenie komunikatu o błędzie
  disp('Błąd otwarcia pliku')
end
```

Ćwiczenie 3.9

Napisz skrypt **czytaj_dane** wczytujący zawartość pliku **dane.txt** utworzonego w ćwiczeniu 3.6.

W poniższym skrypcie wczytywane są kolejno: liczba wierszy macierzy (zmienna *k*), liczba kolumn macierzy (zmienna *l*) oraz macierz o rozmiarze *k*x*l* (zmienna **Y**).

```
p=fopen('dane.txt','r');  %otwarcie pliku dane.txt do odczytu
if p>-1                   %kontrola poprawności otwarcia pliku
   k=fscanf(p,'%d',1)     %wczytanie liczby całkowitej k
   l=fscanf(p,'%d',1)     %wczytanie liczby całkowitej l
   Y=fscanf(p,'%f',[k,l]) %wczytanie macierzy liczb
                            %rzeczywistych Y o rozmiarze kxl
   fclose(p);             %zamknięcie pliku dane.txt
end
```

Ponieważ polecenia wczytywania nie są zakończone średnikami, uruchomienie skryptu spowoduje wyświetlenie wyniku:

```
k =
     4
l =
     3
Y =
    0.5800    0.4300    0.5300
    0.4200    0.2300    0.6400
    0.5200    0.5800    0.2100
    0.3300    0.7600    0.3800
```

Wiersze z pliku ponownie (jak w przypadku funkcji **fprintf**) zostały wczytane do macierzy ***kolumnami***. Chociaż macierz widoczna w pliku **dane.txt** wyglądała inaczej niż macierz **A** z ćwiczenia 3.6, uzyskana z tego pliku macierz **Y** jest dokładnie taka sama jak **A**.

3.4. Zadania do samodzielnego wykonania

Zadanie 3.1

Zapisz w binarnym pliku **dane.bin** 50-elementowy wektor liczb pseudolosowych.

Zadanie 3.2

Otwórz plik binarny **dane.bin** utworzony w zadaniu 3.1. Wczytaj pierwsze 20 wartości z pliku do macierzy o rozmiarze 4x5.

Zadanie 3.3

Napisz skrypt, który przelicza temperaturę wyrażoną w stopniach Celsjusza na temperaturę w stopniach Fahrenheita według wzoru $C = \frac{5}{9}(F - 32)$. Należy:

a) wygenerować wektor **Tc** z wartościami temperatury w skali Celsjusza zmieniającymi się od 0 do 300 z krokiem 20,
b) przeliczyć temperaturę na stopnie Kelwina i zapisać ją w wektorze **Tk**,
c) wektory **Tc** i **Tk** zapisać w pliku tekstowym **temperatura.txt**.

Zadanie 3.4

Napisz skrypt wczytujący dane zapisane w pliku tekstowym **temperatura.txt** i wyświetlający je na ekranie.

4. Grafika

Funkcje graficzne Matlaba można podzielić na cztery podstawowe grupy:

- przeznaczone do tworzenia wykresów dwu- i trójwymiarowych,
- prezentujące wykresy ciągłe i dyskretne,
- umożliwiające tworzenie grafiki wektorowej i rastrowej,
- wysokiego i niskiego poziomu.

Grafika ***trójwymiarowa*** (3D) rozumiana jest jako przedstawienie na płaszczyźnie ekranu monitora rzutów figur trójwymiarowych. Grafika ***dwuwymiarowa*** (2D) może być realizowana jako szczególny przypadek grafiki trójwymiarowej poprzez rzutowanie rysunków na płaszczyznę XY.

Większość funkcji graficznych Matlaba wykreśla dane opisane wektorami w sposób ***ciągły***. Punkty na wykresie łączone są liniami (wykresy 2D) lub wycinkami powierzchni (wykresy 3D). Dane mogą być też przedstawiane w sposób ***dyskretny*** – jako słupki, odcinki itp.

Podstawę grafiki systemu Matlab stanowi ***grafika wektorowa*** – tworzone obrazy składają się z linii, punktów i wielokątów o określonych współrzędnych. Czasami jednak przydatne są operacje na pojedynczych punktach rastra. Matlab umożliwia to, oferując zestaw funkcji ***grafiki rastrowej***.

Większość opisanych w tym rozdziale funkcji graficznych Matlaba to tzw. ***funkcje wysokiego poziomu***, które automatycznie ustalają większość parametrów tworzonych rysunków. Natomiast ***funkcje niskiego poziomu***, wykorzystywane do obsługi obiektowego systemu graficznego, umożliwiają prawie dowolne kontrolowanie wyglądu tworzonego rysunku.

4.1. Okna graficzne

4.1.1. Zarządzanie oknami

Grafika jest w pakiecie Matlab wyświetlana w specjalnym ***oknie graficznym***. Funkcje graficzne wyświetlają wyniki swoich działań w otwartym oknie lub jeśli żadne okno nie jest otwarte, automatycznie tworzą nowe.

Jednocześnie może być otwartych wiele okien. Jedno z nich jest zawsze oknem ***aktywnym*** i do niego Matlab kieruje grafikę. Domyślnie aktywne jest okno otwarte ostatnio.

Do tworzenia nowych okien służy funkcja **figure**. Każde okno ma kolejny numer, widoczny w jego nagłówku: „Figure No.1", „Figure No.2" itd.

Funkcje wykonujące operacje na oknach graficznych zostały zebrane w tabeli 4.1.

Tabela 4.1. Funkcje obsługujące okna graficzne

Funkcja	Opis
figure	tworzy nowe okno graficzne i uaktywnia je
figure(n)	uaktywnia okno o numerze *n* (jeśli jest otwarte) lub tworzy nowe okno i przyporządkowuje mu numer *n*
close	zamyka okno aktywne
close(n)	zamyka okno o numerze *n*; argumentem funkcji może też być macierz zawierająca numery okien, które mają zostać zamknięte
close all	zamyka wszystkie okna
clf	usuwa zawartość aktywnego okna

4.1.2. Podział okna

Matlab umożliwia także umieszczanie wielu rysunków w jednym oknie. Służy do tego funkcja **subplot**, która dzieli okno na mniejsze prostokątne okienka. W każdym z nich można narysować odrębny wykres w wybranym układzie współrzędnych. Polecenie:

subplot(m,n,p)

dzieli okno graficzne na *m*x*n* okienek (wykresów), rozmieszczając je w *m* wierszach i *n* kolumnach, oraz uaktywnia okienko o numerze *p*. Okienka są numerowane od lewej do prawej, wierszami od góry do dołu.

Inne wywołanie funkcji **subplot**:

subplot('Position', [lewy dolny szerokość wysokość])

tworzy w obrębie aktywnego rysunku nowe prostokątne okienko o podanym położeniu i wymiarach. Położenie jest podawane względem lewego dolnego rogu rysunku. Parametry *szerokość*=*wysokość*=1 oznaczają układ o rozmiarach rysunku.

Ćwiczenie 4.1

Wykonaj polecenia:

a) utwórz nowe okno graficzne

```
» figure
```

b) podziel okno na dwa okienka (wykresy) w pionie i uaktywnij wykres z lewej strony

```
» subplot(2,1,1)
```

c) uaktywnij wykres z prawej strony

```
» subplot(2,1,2)
```

4.2. Grafika dwuwymiarowa

4.2.1. Wykreślanie wektorów – funkcja plot

Do wykreślania danych przechowywanych w wektorach służy funkcja **plot**. Generuje ona dwuwymiarową krzywą złożoną z punktów (x_i, y_i), których współrzędne zostały określone w wektorach **x** i **y**. Wektory te muszą być równej długości. Wywołanie funkcji **plot** może mieć różną postać w zależności od liczby argumentów (tabela 4.2).

Tabela 4.2. Sposoby wywoływania funkcji **plot**

Polecenie	Opis
plot(x,y)	rysuje wykres elementów wektora **y** względem elementów wektora **x**; jeśli jeden z argumentów jest macierzą, a drugi wektorem, ciąg elementów wektora jest rysowany względem wierszy lub kolumn macierzy
plot(y)	rysuje wykres elementów wektora **y**, przyjmując **x=1:length(y)**
plot(x,y,s)	rysuje wykres $y(x)$ z określeniem dokładnego wyglądu linii; s – łańcuch znaków zawierający kody, jak w tabeli 4.3
plot(x1,y1,x2,y2,...)	rysuje w jednym oknie wiele wykresów: $y1(x1)$, $y2(x2)$ itd.
plot(x1,y1,s1,x2,y2,s2...)	rysuje w jednym oknie wiele wykresów z określeniem dokładnego wyglądu linii każdego z nich

Łańcuch znaków określający sposób rysowania linii może zawierać symbole zestawione w tabeli 4.3. Odnoszą się one do:

- rodzaju linii (kreskowana, ciągła itp.),

- sposobu oznaczenia punktów na wykresie (kółka, trójkąty, punkty itp.),
- koloru linii.

Większość funkcji graficznych Matlaba akceptuje argument *s*, definiujący wszystkie lub tylko niektóre składniki wyglądu linii.

Tabela 4.3. Symbole określające wygląd linii

Znaki	Rodzaj linii
'-'	ciągła (domyślna)
'--'	kreskowana
':'	kropkowana
'-.'	kreska-kropka
	Kolor linii
'y'	żółty
'm'	karmazynowy
'c'	turkusowy
'r'	czerwony
'g'	zielony
'b'	niebieski
'w'	biały
'k'	czarny

Znaki	Oznaczenie punktów
'+'	krzyżyk
'*'	gwiazdka
'.'	kropka
'o'	kółko
'x'	iks
's'	kwadrat
'd'	romb
'p'	gwiazdka pięcioramienna
'h'	gwiazdka sześcioramienna
'v'	trójkąt skierowany do dołu
'^'	trójkąt skierowany do góry
'<'	trójkąt skierowany w lewo
'>'	trójkąt skierowany w prawo

W tworzeniu danych do wykresów może pomóc funkcja **linspace**. Polecenie

linspace(x1,x2,N)

generuje wierszowy wektor *N* liczb rozłożonych równomiernie w przedziale od *x1* do *x2*. Natomiast polecenie

linspace(x1,x2)

generuje domyślnie 100 liczb z przedziału od *x1* do *x2*.

Ćwiczenie 4.2

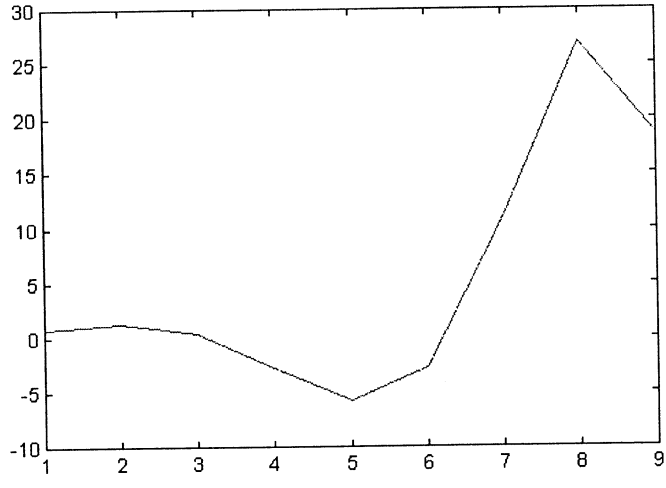
Dany jest 9-elementowy wektor **W**=[0.69 1.24 0.32 -2.8 -5.84 -2.81 10.88 27 18.55]

```
» W=[0.69 1.24 0.32 -2.8 -5.84 -2.81 10.88 27 18.55]
```

Narysuj wykres elementów wektora **W**:

a) względem indeksów wektora

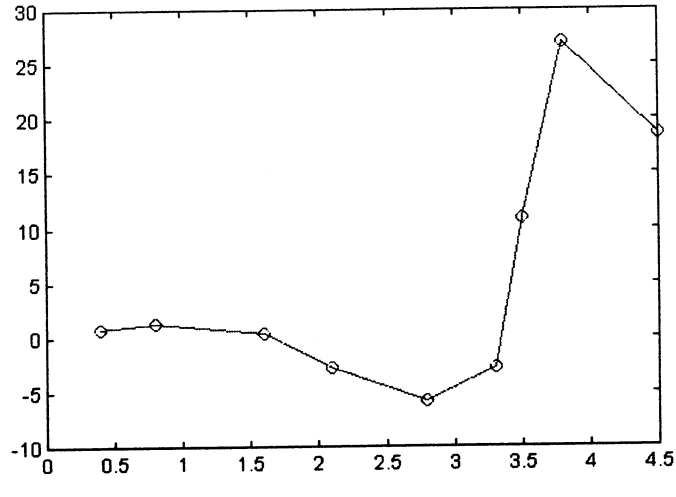
```
» plot(W)
```



Rysunek 4.1. Wykres elementów wektora **W** względem ich indeksów (od 1 do 9)

b) względem elementów wektora **S**=[0.4 0.8 1.6 2.1 2.8 3.3 3.5 3.8 4.5], przy czym wykres ma być narysowany linią ciągłą, a punkty na wykresie zaznaczone kółkami

```
» S=[0.4 0.8 1.6 2.1 2.8 3.3 3.5 3.8 4.5]
» plot(S,W,'-o')
```

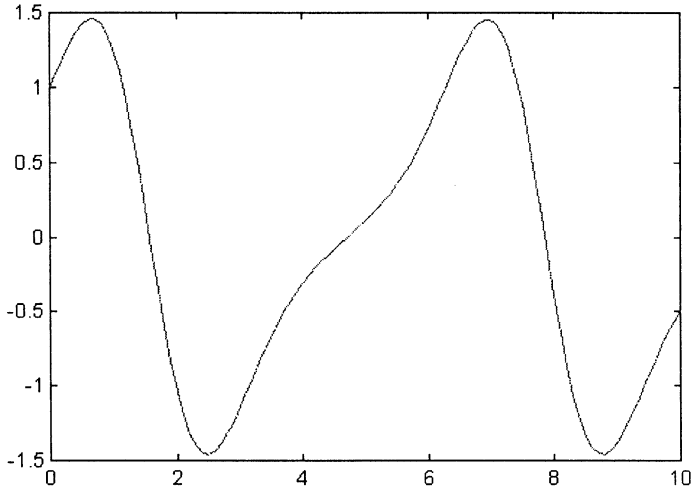


Rysunek 4.2. Wykres linii opisanej elementami wektorów **S** i **W** (punkty zaznaczone kółkami)

Ćwiczenie 4.3

Narysuj wykres funkcji $y = \cos(t)e^{\sin(t)}$ dla wartości t zmieniających się od 0 do 10 z krokiem 0,1.

```
» t=[0:0.1:10];
» y=cos(t).*exp(sin(t));
» plot(t,y)
```



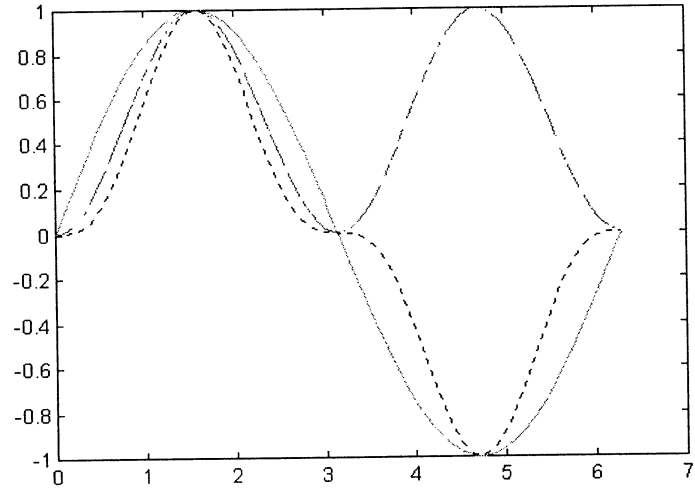
Rysunek 4.3. Wykres funkcji $y = \cos(t)e^{\sin(t)}$ dla $t \in \langle 0,10 \rangle$

Ćwiczenie 4.4

Dla $x \in \langle 0,2\pi \rangle$ narysuj na jednym wykresie przebiegi funkcji:

a) $\sin(x)$ – czerwoną linią ciągłą,
b) $\sin^2(x)$ – niebieską linią kreskowaną,
c) $\sin^3(x)$ – czarną linią kropkowaną.

```
» x=linspace(0,2*pi);
» plot(x,sin(x),'r-',x,sin(x).^2,'b--',x,sin(x).^3,'k:')
```



Rysunek 4.4. Przebiegi trzech funkcji: $\sin(x)$, $\sin^2(x)$, $\sin^3(x)$ na jednym wykresie

Ćwiczenie 4.5

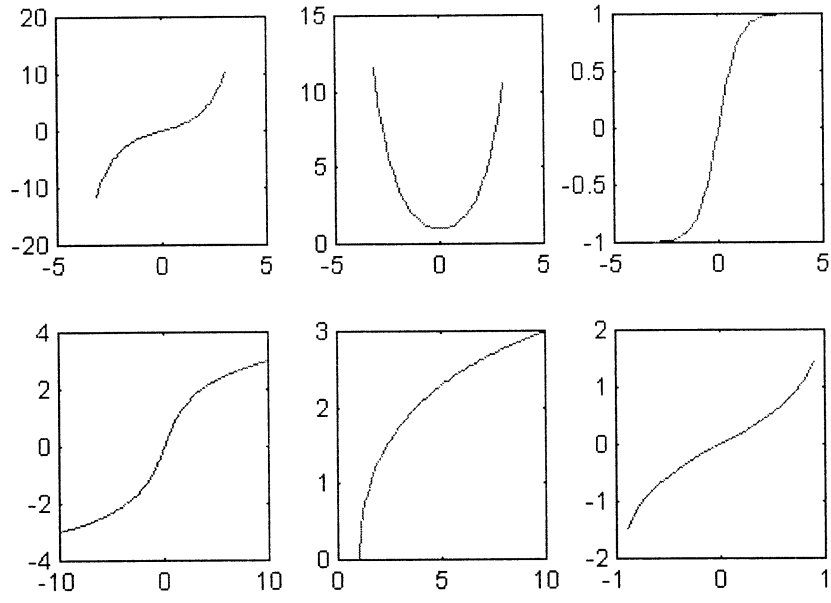
Na jednym rysunku umieść za pomocą funkcji **subplot** 6 wykresów w 2 wierszach i 3 kolumnach. Na wykresach umieść kolejno:

a) w pierwszym wierszu: $\sinh(x)$, $\cosh(x)$, $\tgh(x)$ dla $x \in \langle -\pi, \pi \rangle$;

b) w drugim wierszu: $\operatorname{arcsinh}(y)$ dla $y \in \langle -10, 10 \rangle$, $\operatorname{arcosh}(z)$ dla $z \in \langle 1, 10 \rangle$, $\operatorname{arctgh}(w)$ dla $w \in \langle -1, 1 \rangle$.

Rozwiązaniem może być skrypt o treści:

```
x=-pi:0.1:pi;
subplot(2,3,1)    %uaktywnienie wykresu z lewej w 1. rzędzie
plot(x,sinh(x))
subplot(2,3,2)    %uaktywnienie środkowego wykresu w 1. rzędzie
plot(x,cosh(x))
subplot(2,3,3)    %uaktywnienie wykresu z prawej w 1. rzędzie
plot(x,tanh(x))
subplot(2,3,4)    %uaktywnienie wykresu z lewej w 2. rzędzie
y=-10:0.1:10; z=1:0.1:10; w=-1:0.1:1;
plot(y,asinh(y))
subplot(2,3,5)    %uaktywnienie środkowego wykresu w 2. rzędzie
plot(z,acosh(z))
subplot(2,3,6)    %uaktywnienie wykresu z prawej w 2. rzędzie
plot(w,atanh(w))
```



Rysunek 4.5. Widok sześciu różnych wykresów w jednym oknie graficznym

4.2.2. Opisywanie wykresów

Aby wykres był w pełni czytelny, należy go odpowiednio opisać. Do tego celu służą funkcje zestawione w tabeli 4.4.

Tabela 4.4. Funkcje opisujące wykresy

Funkcja	Opis
xlabel(tekst)	wyświetla łańcuch znaków *tekst* jako opis osi *x* aktywnego wykresu
ylabel(tekst)	wyświetla łańcuch znaków *tekst* jako opis osi *y* aktywnego wykresu
title(tekst)	wyświetla łańcuch znaków *tekst* jako tytuł aktywnego wykresu
text(x,y,tekst)	wyświetla łańcuch znaków *tekst* w miejscu określonym przez współrzędne *x*, *y*
legend(s1,s2,...)	wyświetla legendę; łańcuch znaków *s1* jest opisem odnoszącym się do pierwszego wykresu, *s2* – do drugiego itd.
grid on/off	włącza/wyłącza wyświetlanie pomocniczej siatki współrzędnych

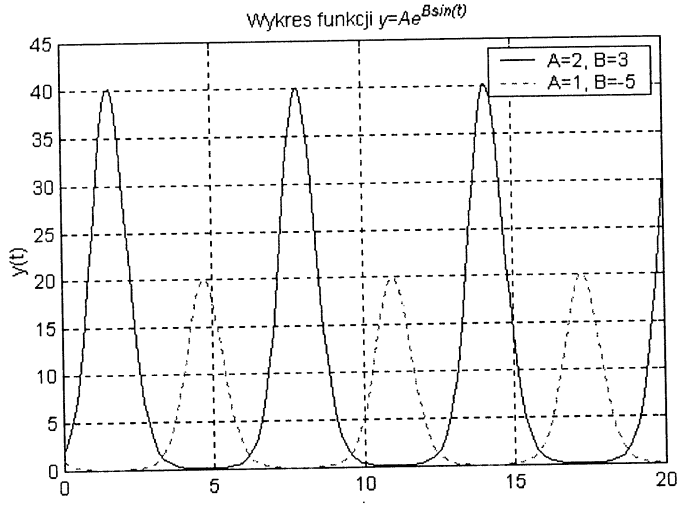
Opis wykresu wyświetlany za pomocą funkcji **xlabel**, **ylabel**, **title**, **text** i **legend** można odpowiednio formatować. Przykładowo tekst '*\alpha*' wyświetla znak α, tekst '*\Sigma*' – znak Σ, a '*\infty*' – ∞. Znak następujący w tekście po znaku '_' zostanie

wyświetlony jako indeks dolny, a po znaku '^' jako indeks górny. Format '\bf' wyświetla kolejny znak za pomocą czcionki pogrubionej, a '\it' pochyłej. Aby dany sposób formatowania odnosił się do grupy znaków, należy je umieścić w nawiasach klamrowych.

Ćwiczenie 4.6

Napisz skrypt rysujący wykres funkcji $y=Ae^{B\sin(t)}$ dla $t\in<0,20>$. Wykonaj dwa wykresy: dla $A=2$, $B=3$ (czarna linia ciągła) oraz $A=1$, $B=-3$ (niebieska linia kropkowana). Na wykresie umieść także siatkę, podpisy pod osiami (oś x – t, oś y – $y(t)$), tytuł *Wykres funkcji* $y=Ae^{B\sin(t)}$ oraz legendę.

```
t=0:0.1:20;
A=2;B=3; y1=A*exp(B*sin(t));
A=1;B=-3; y2=A*exp(B*sin(t));
plot(t,y1,'k-',t,y2,'b:')
xlabel('t')
ylabel('y(t)')
title('Wykres funkcji y=Ae^{Bsin(t)}')
legend('A=2, B=3','A=1, B=-3')
grid on
```



Rysunek 4.6. Wykres z tytułem, opisami osi, legendą i siatką

4.2.3. Skalowanie i nakładanie rysunków

Wyświetlanie wykresu w oknie graficznym powoduje jednoczesne wyczyszczenie okna i usunięcie poprzedniego rysunku. Dodanie wykresu do znajdującego się w oknie umożliwia funkcja **hold**. Polecenie

hold on/off

wyłącza (*on*) lub włącza (*off*) tryb czyszczenie okien.

Sprawdzenie stanu przełącznika **hold** umożliwia funkcja **ishold**, która zwraca wartość 1, gdy jest on aktywny (**hold on**) lub 0 w przeciwnym razie.

Zakres osi na wykresie jest w Matlabie wybierany automatycznie na podstawie zakresu danych tak, aby wszystkie wartości (od najmniejszej do największej) mogły zostać wyświetlone. Domyślnie układ współrzędnych zajmuje cały rysunek, a skala osi dobierana jest automatycznie.

Zmianę wyglądu osi umożliwia funkcja **axis**, którą można wywoływać dla różnych argumentów.

Tabela 4.5. Sposoby wywoływania funkcji **axis**

Polecenie	Opis
axis([xmin xmax ymin ymax])	ustawia zakres osi *x* i *y*
axis auto	włącza tryb automatycznego ustawiania zakresu osi
axis manual	wyłącza tryb automatycznego ustawiania zakresu osi; po włączeniu przełącznika **hold** wszystkie kolejne wykresy będą rysowane w takim samym układzie współrzędnych
axis ij	zmienia układ współrzędnych na macierzowy, z początkiem układu współrzędnych w lewym górnym rogu
axis xy	zmienia układ współrzędnych na kartezjański (domyślny)
axis equal	zmienia skalowanie osi w taki sposób, aby jednostka na podziałce miała ten sam rozmiar na wszystkich osiach
axis image	zmienia rozmiary osi na takie same, jak rozmiary wykresu
axis square	ustawia jednakowy rozmiar wszystkich osi
axis normal	przywraca standardowe ustawienia rozmiaru osi
axis off	ukrywa osie wraz z ich opisem
axis on	przywraca wyświetlanie osi
[s1, s2, s3]=axis('state')	zwraca aktualne ustawienia układu: *s1* – *auto* lub *manual*, *s2* – *on* lub *off* (osie wyświetlane lub nie), *s3* – *xy* lub *ij*
v=axis	zwraca wektor wierszowy v=[*xmin xmax ymin ymax*]

Ćwiczenie 4.7

Narysuj dla x z przedziału <0, 4π>:

a) wykresy funkcji $y = \cos(x)-1$ i $z = \sin\left(\frac{1}{x^2+1}\right)$ w oddzielnych oknach

b) wykresy obu funkcji na jednym rysunku.

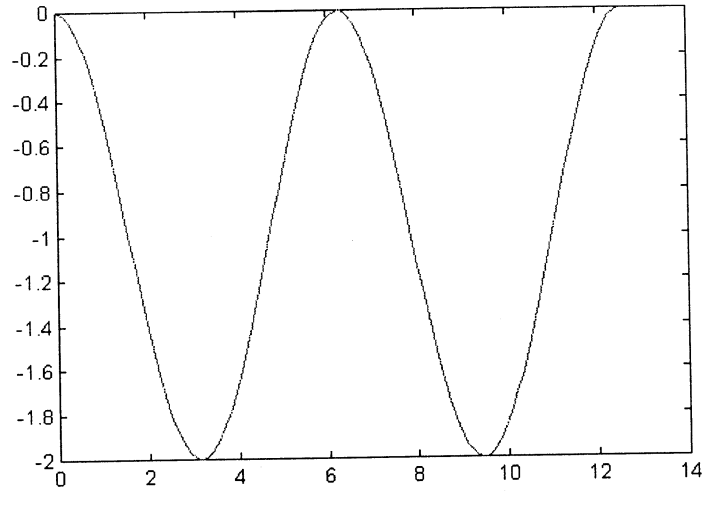
Jeżeli w odpowiednich skryptach zdefiniowano funkcje **fgr1a** i **fgr1b**:

```
function y=fgr1a(x)
y=cos(x)-1;
```

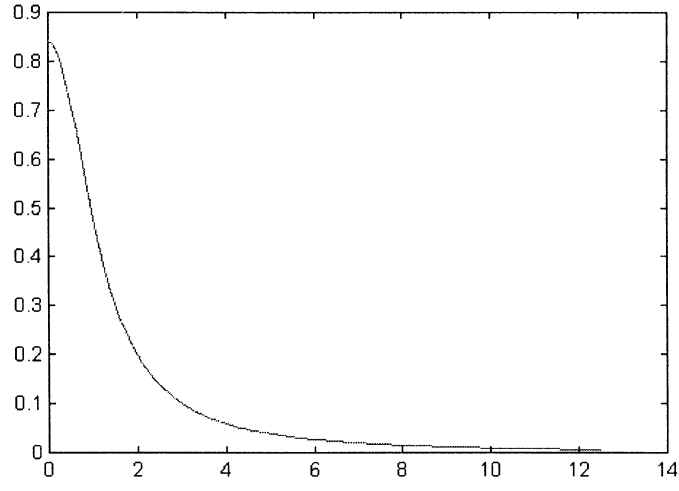
```
function y=fgr1b(x)
y=sin(1./(x.^2+1));
```

to skrypt realizujący punkt a) może wyglądać następująco:

```
close all           % zamknięcie wszystkich okien
x=0:0.1:4*pi;
plot(x, fgr1a(x))
figure              % otwarcie nowego okna
plot(x, fgr1b(x)) % rysowanie wykresu w nowym oknie
```



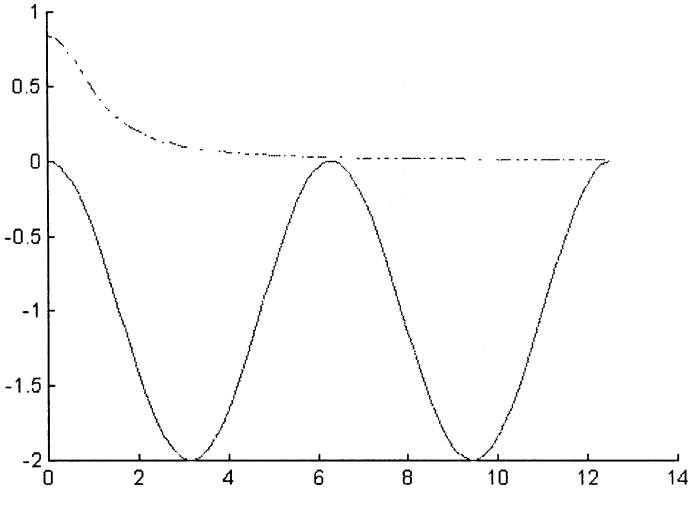
Rysunek 4.7. Wykres funkcji $y = \cos(x)-1$



Rysunek 4.8. Wykres funkcji $z = \sin\left(\frac{1}{x^2+1}\right)$

Natomiast skrypt realizujący punkt b):

```
close all              % zamknięcie wszystkich okien
x=0:0.1:4*pi;
hold on                % wyłączenie trybu czyszczenia
plot(x, fgr1a(x))
plot(x, fgr1b(x),'-.')
hold off               % przywrócenie trybu czyszczenia
```



Rysunek 4.9. Wykresy funkcji: $y = \cos(x)$-1 i $z = \sin\left(\frac{1}{x^2+1}\right)$ przy wyłączonym trybie czyszczenia

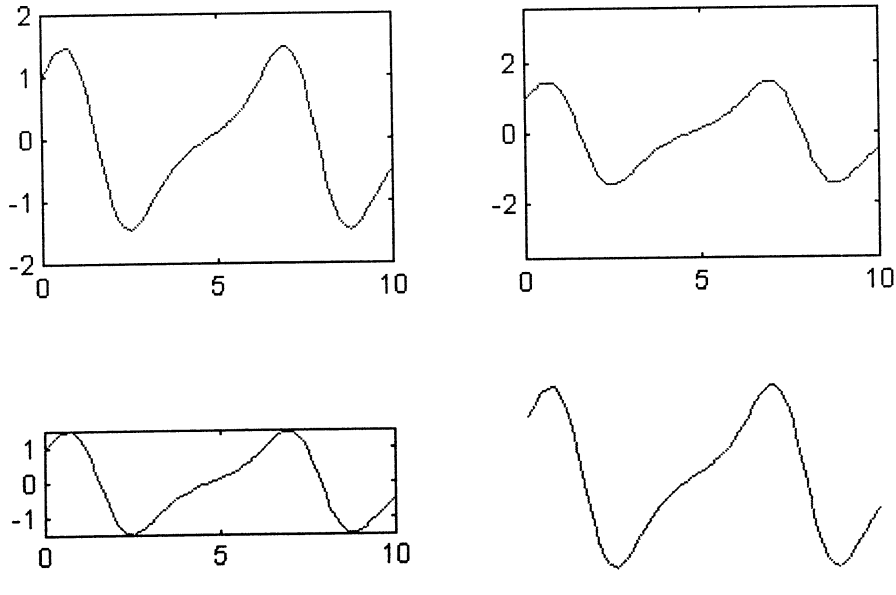
Ćwiczenie 4.8

Wykres funkcji z ćwiczenia 4.3 narysuj kolejno:

a) w domyślnym układzie współrzędnych;
b) w układzie, w którym jednostka ma taki sam rozmiar na obu osiach;
c) w układzie o rozmiarach takich samych, jak rozmiary rysunku;
d) bez układu współrzędnych.

Wszystkie wykresy umieszczono w jednym oknie za pomocą polecenia **subplot**, dzieląc je na 2 wiersze i 2 kolumny.

```
t=[0:0.1:10];
y=cos(t).*exp(sin(t));
subplot(2,2,1)        % 1. wiersz, 1. kolumna
plot(t,y)             % zwykły wykres
subplot(2,2,2)        % 1. wiersz, 2. kolumna
plot(t,y)
axis equal            % realizacja punktu b)
subplot(2,2,3)        % 2. wiersz, 1. kolumna
plot(t,y)
axis image            % realizacja punktu c)
subplot(2,2,4)        % 2. wiersz, 2. kolumna
plot(t,y)
axis off              % realizacja punktu d)
```



Rysunek 4.10. Widok tego samego wykresu dla różnych wywołań funkcji **axis**

4.2.4. Wykreślanie funkcji – funkcja fplot

Funkcja **plot** umożliwia narysowanie wykresu dowolnej funkcji po zapisaniu jej argumentu w postaci wektora. Dokładność takiego rozwiązania jest jednak uzależniona od sposobu dyskretyzacji zadanego przedziału. W celu narysowania możliwie najbardziej precyzyjnego wykresu funkcji (zarówno z biblioteki, jak i utworzonej przez użytkownika w m-pliku) lepiej skorzystać z funkcji **fplot**:

fplot(f, [x0 xk])

gdzie:

f – łańcuch znaków zawierający nazwę funkcji;
x0, *xk* – początek i koniec przedziału rysowania funkcji.

Wywołanie funkcji

[x,y]=fplot(...)

nie powoduje narysowania wykresu, tylko zwraca wektor argumentów **x** i wektor wartości funkcji **y**. Wykres uzyskanych danych można narysować za pomocą polecenia:

plot(x,y)

Ćwiczenie 4.9

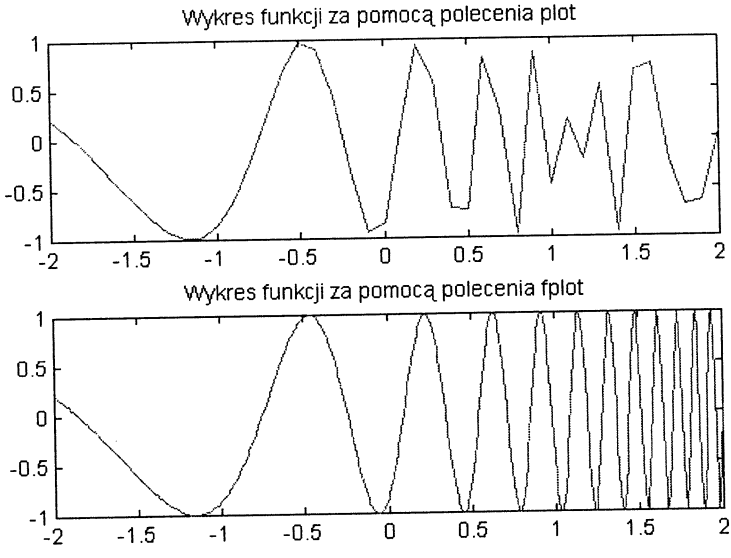
Zdefiniuj funkcję $y = \cos(10\, e^x)$ i narysuj jej wykres w przedziale <-2,2> za pomocą polecenia **fplot** oraz polecenia **plot** dla wektora **x**=-2:0.1:2.

Jeżeli w pliku **fgr1.m** zdefiniowano potrzebną funkcję:

```
function y=fgr1(x)
y=cos(10*exp(x));
```

to zadanie zrealizuje skrypt:

```
x=-2:0.1:2; y=fgr1(x);
subplot(2,1,1)
plot(x,y)
title('Wykres funkcji za pomocą polecenia plot')
subplot(2,1,2)
fplot('fgr1',[-2 2])
title('Wykres funkcji za pomocą polecenia fplot')
```



Rysunek 4.11. Porównanie wykresów tej samej funkcji uzyskanych za pomocą polecenia **plot** i **fplot**

4.2.5. Wykresy w skali logarytmicznej

Do rysowania wykresów w skali logarytmicznej służą funkcje Matlaba zestawione w tabeli 4.6. Opcjonalnym argumentem wszystkich tych funkcji jest łańcuch znaków *s* określający wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3).

Tabela 4.6. Funkcje rysujące wykresy w skali logarytmicznej

Funkcja	Opis
loglog(x,y,s)	rysuje wykres, używając skal logarytmicznych na obu osiach
semilogx(x,y,s)	rysuje wykres, używając skali logarytmicznej tylko na osi *x*
semilogy(x,y,s)	rysuje wykres, używając skali logarytmicznej tylko na osi *y*

W przygotowywaniu tego typu wykresów może pomóc funkcja **logspace**, która generuje wektor wierszowy *N* liczb, rozmieszczonych logarytmicznie między wartościami 10^{x1} a 10^{x2}:

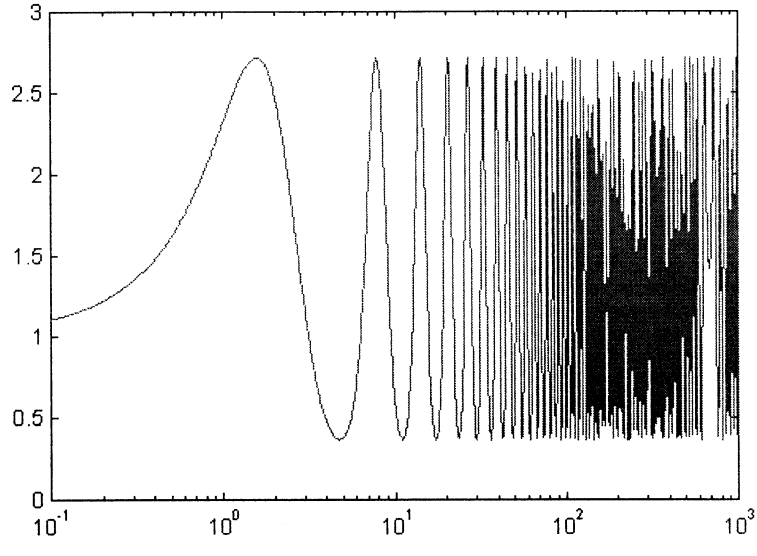
logspace(x1,x2,N)

Wywołanie funkcji bez argumentu *N* spowoduje wygenerowanie wierszowego wektora 50 liczb.

Ćwiczenie 4.10

Używając skali logarytmicznej na osi x narysuj wykres funkcji $y = e^{\sin(x)}$ dla x z przedziału $<10^{-1}, 10^{3}>$.

```
» x=logspace(-1,3,1000);
» semilogx(x,exp(sin(x)))
```



Rysunek 4.12. Wykres przy użyciu skali logarytmicznej na osi x

4.2.6. Wykresy w biegunowym układzie współrzędnych

Do rysowania wykresów w biegunowym układzie współrzędnych (θ,r) służy funkcja **polar** postaci:

polar(theta,r,s)

z argumentami:

theta – wektor kątów (w radianach) dla poszczególnych punktów,
r – wektor odległości poszczególnych punktów od początku układu współrzędnych.

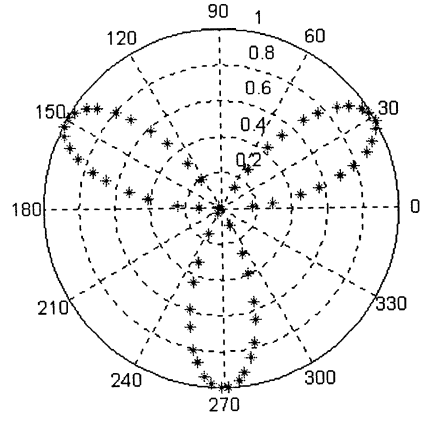
Opcjonalnym argumentem funkcji jest łańcuch znaków s, określający wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3).

Ćwiczenie 4.11

Narysuj w układzie biegunowym wykres funkcji $y = \sin(3t)$ dla t z przedziału $< -\pi, \pi >$.

```
» t=-pi:0.1:pi;
```

```
» polar(t,sin(3*t),'*')
```



Rysunek 4.13. Wykres w biegunowym układzie współrzędnych

4.2.7. Wykresy danych zespolonych

Dane zawierające wartości zespolone można przedstawiać graficznie, wykorzystując funkcje matematyczne wyodrębniające ich części rzeczywiste i urojone. W Matlabie odbywa się to za pomocą funkcji zestawionych w tabeli 4.7. Opcjonalnym argumentem wszystkich funkcji jest łańcuch znaków *s*, określający wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3).

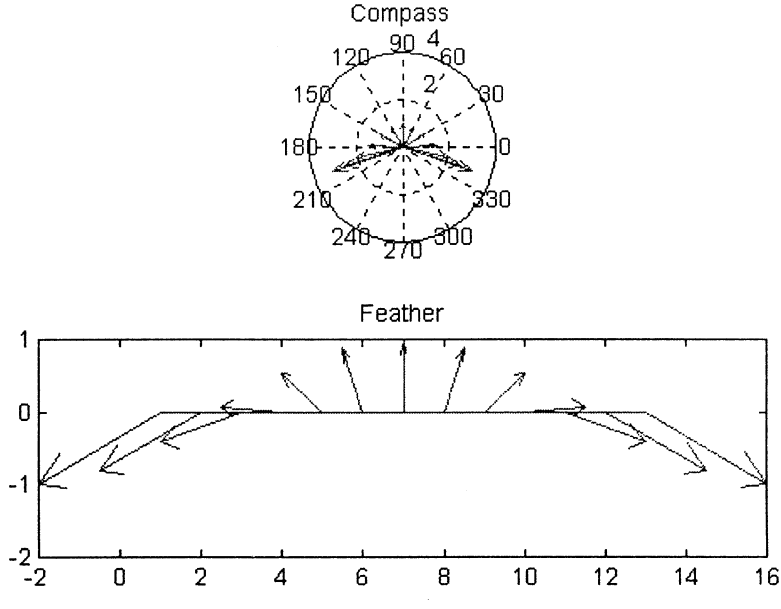
Tabela 4.7. Funkcje rysujące wykresy danych zespolonych

Funkcja	Opis
plot(z,s)	jeżeli **z** jest macierzą o elementach zespolonych, to zostanie narysowany wykres Im(z)=f(Re(z)); równoważne polecenie: **plot(real(z),imag(z),s)**
compass(z,s) lub **compass(x,y,s)**	rysuje wykres, na którym elementy macierzy zespolonej **z** są przedstawione w postaci strzałek o wspólnym początku i grotach w punktach opisanych przez współrzędne **x=real(z)**, **y=imag(z)**; **x** i **y** są współrzędnymi kartezjańskimi, wykres jest rysowany w biegunowym układzie współrzędnych
feather(z,s) lub **feather(x,y,s)**	rysuje wykres, na którym elementy macierzy zespolonej **z** są przedstawione w postaci strzałek o początkach rozmieszczonych równomiernie na osi x; długości strzałek są równe modułom elementów macierzy **z**, a kąty nachylenia strzałek – ich argumentom

Ćwiczenie 4.12

Za pomocą poleceń: **compass** i **feather** narysuj w jednym oknie graficznym wykres funkcji $z = x + jy$, gdzie $y = \cos(x)$, x należy do przedziału <-3,3>.

```
» x=-3:0.5:3;
» subplot(2,1,1)
» compass(x,cos(x))
» title('Compass')
» subplot(2,1,2)
» feather(x,cos(x))
» title('Feather')
```



Rysunek 4.14. Dwa rodzaje wykresów danych zespolonych

4.3. Grafika trójwymiarowa

4.3.1. Wykreślanie linii

Rysowanie wykresów funkcji trzech zmiennych umożliwia funkcja **plot3**, będąca odpowiednikiem funkcji **plot** dla przestrzeni dwuwymiarowej. Polecenie:

plot3(x,y,z,s)

generuje trójwymiarową krzywą złożoną z punktów (x_i, y_i, z_i), których współrzędne zostały określone w wektorach **x**, **y** i **z**. Wektory muszą być tej samej długości.

Opcjonalnym argumentem funkcji jest łańcuch znaków *s* określający wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3).

Wiele wykresów w jednym oknie umieszcza polecenie:

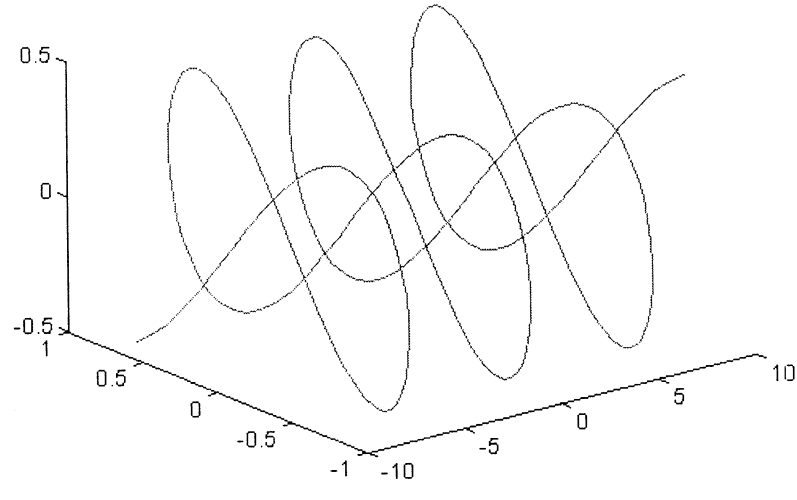
plot3(x1,y1,z1,s1,x2,y2,z2,s2,...)

Funkcję **plot3** można także wykorzystać do wykreślania powierzchni (tabela 4.8).

Ćwiczenie 4.13

Narysuj wykres funkcji $z=\cos(x)\sin(y)$, gdzie x – wektor o wartościach od -10 do 10, $y=\sin(x)$.

```
» x=[-10:0.1:10];
» y=sin(x);
» z=cos(x).*sin(y);
» plot3(x,y,z)
```



Rysunek 4.15. Wykres trójwymiarowej krzywej

4.3.2. Opisywanie wykresów, skalowanie i nakładanie rysunków

Do opisywania wykresów trójwymiarowych stosowane są te same funkcje, co w grafice dwuwymiarowej (tabela 4.4). Oprócz tego funkcja **zlabel** postaci:

zlabel(tekst)

wyświetla łańcuch znaków *tekst* jako opis osi *z* aktywnego wykresu. Natomiast funkcja **text** z trzema argumentami:

text(x,y,z,tekst)

wyświetla łańcuch znaków *tekst* w miejscu określonym przez współrzędne *x*, *y* i *z*.

Zasady nakładania rysunków (**hold**) i zmiany wyglądu osi (**axis**) są w grafice trójwymiarowej takie same, jak w przypadku grafiki dwuwymiarowej (rozdział 4.2.3). Dodatkowe wywołanie funkcji **axis** umożliwia zmianę zakresów wszystkich trzech osi:

axis([xmin xmax ymin ymax zmin zmax])

4.3.3. Wykreślanie powierzchni

Powierzchnia rysowana jest w Matlabie jako wykres funkcji $z=f(x,y)$, przy czym współrzędne punktów (x_i, y_j) określone są za pomocą wektorów **X** i **Y**, gdzie indeksy *ij* przyjmują wartości: **i=1:length(X)**, **j=1:length(Y)**.

Ponieważ tworzymy wykres trójwymiarowy na dwuwymiarowej płaszczyźnie ekranu, na początek należy wygenerować specjalną siatkę na płaszczyźnie XY w tych węzłach, w których szukane są wartości funkcji w osi *z*. Służy do tego funkcja **meshgrid**:

[x,y]=meshgrid(X,Y)

która transformuje obszar opisany przez wektory **X** i **Y** (z przestrzeni 3D) na dwie macierze **x** oraz **y** we współrzędnych ekranowych 2D. Wywołanie:

[x,y]=meshgrid(X)

jest równoważne wywołaniu **meshgrid(X,X)**.

Po przygotowaniu siatki opisanej macierzami **x** i **y** należy obliczyć wartości macierzy **z**, określone zależnością $z_{ij}=f(x_i,y_j)$.

Do rysowania wykresów powierzchni służą funkcje Matlaba zestawione w tabeli 4.8. Ich argumentami są macierze **x**, **y** i **z**. Płaszczyzna jest składana z czworokątów, których wierzchołki leżą w punktach o współrzędnych opisanych macierzami **x**, **y**, **z**.

Tabela 4.8. Funkcje wykreślające powierzchnie

Funkcja	Opis
mesh(x,y,z,c)	rysuje powierzchnię opisaną macierzami **x**, **y**, **z** w postaci kolorowej siatki o polach wypełnionych kolorem tła; elementy macierzy **c** określają kolory linii poszczególnych pól
mesh(x,y,z)	rysuje powierzchnię, przyjmując **c** = **z**

Funkcja	Opis
mesh(z,c) **mesh(z)**	rysuje wykres wartości elementów macierzy **z**, przyjmując **x=1:n**, **y=1:m**, gdzie **[m,n]=size(z)**
meshc(x,y,z,c)	rysuje siatkę identyczną, jak funkcja **mesh**, i umieszcza pod nią wykres poziomicowy; łączy działanie funkcji **mesh** i **contour** (tabela 4.10)
meshz(x,y,z,c)	działa podobnie jak **mesh**, ale dodatkowo w dół od krawędzi wykresu rysowane są linie określające płaszczyzny odniesienia
surf(x,y,z,c)	rysuje różnokolorową powierzchnię opisaną macierzami **x**, **y**, **z**
surf(x,y,z)	rysuje powierzchnię, przyjmując **c = z**
surf(z,c) **surf(z)**	rysuje powierzchnię, przyjmując **x=1:n**, **y=1:m**, gdzie **[m,n]=size(z)**
surfc(x,y,z,c)	łączy działanie funkcji **surf** i **contour** (tabela 4.10)
surfl(x,y,z,s,k)	rysuje powierzchnię z uwzględnieniem odbić światła; opcjonalny argument **s** określa kierunek, z którego pada światło, **k** – wektor określający współczynniki odbicia i rozproszenia
waterfall(x,y,z,c)	działa podobnie jak **meshz**, ale nie rysuje linii odpowiadających kolumnom macierzy
plot3(x,y,z,s)	rysuje linie na powierzchni opisanej wektorami **x**, **y**, **z**; wygląd linii określa opcjonalny argument **s** jak w funkcji **plot** (tabela 4.3)

Ćwiczenie 4.14

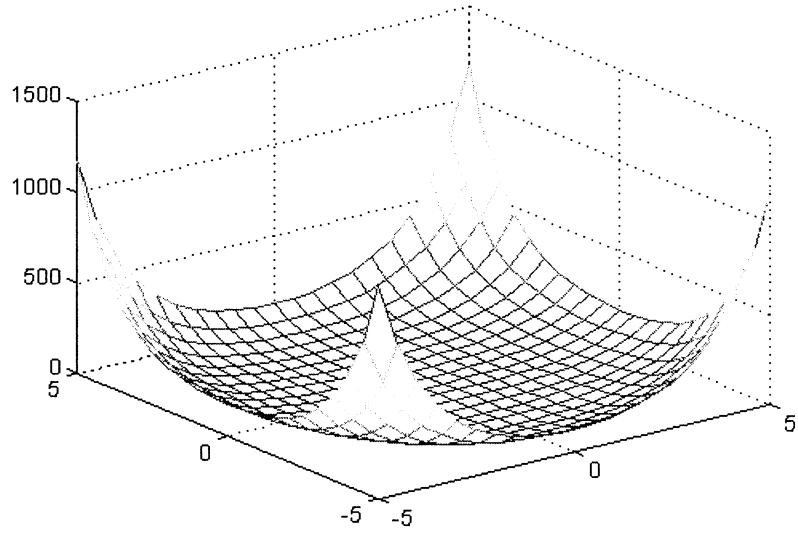
Narysuj wykres powierzchni opisanej funkcją $f(x,y)=(x-y)(x+y)+e^{\sqrt{x^2+y^2}}$ dla $x,y\in<-5,5>$.

Pierwszą czynnością jest przygotowanie danych do wykresu:

```
» [x,y]=meshgrid(-5:0.5:5,-5:0.5:5);
» z=(x-y).*(x+y)+exp(sqrt(x.^2+y.^2));
```

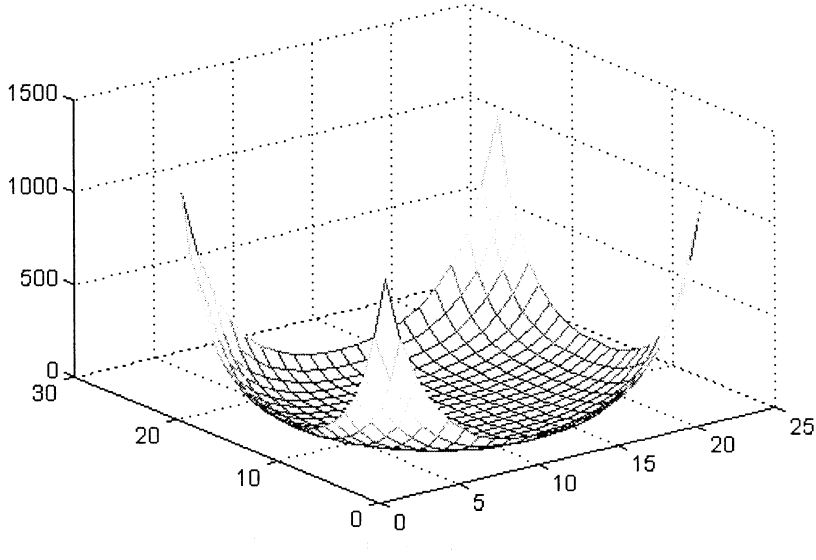
Na rysunkach 4.16 i 4.17 przedstawiono wykresy powierzchni uzyskane dla różnych wywołań funkcji **mesh**.

```
» mesh(x,y,z)
```



Rysunek 4.16. Wykres powierzchniowy macierzy **z** dla siatki opisanej wektorami **x** i **y**

```
» mesh(z)
```



Rysunek 4.17. Wykres powierzchniowy macierzy **z** dla siatki opisanej indeksami elementów macierzy

Ćwiczenie 4.15

Narysuj w oddzielnych oknach wykresy funkcji: $f(x,y) = \sin(x)\cos(y)$ dla $x,y \in \langle -\pi, 2\pi \rangle$ za pomocą poleceń kolejno: **mesh**, **meshc**, **meshz**.

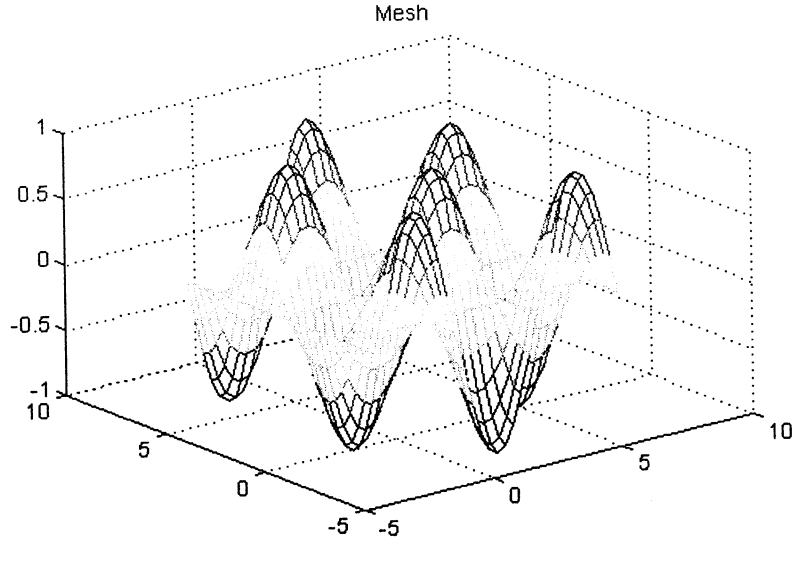
Funkcja zdefiniowana jest w pliku **fgr2.m** jako:

```
function z=fgr2(x,y)
z=sin(x).*cos(y);
```

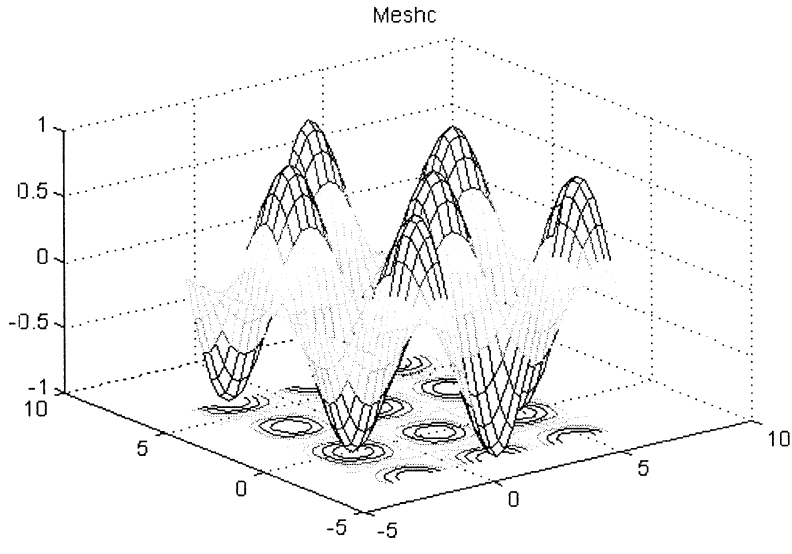
Skrypt realizujący zadanie widoczny jest poniżej.

```
close all              %zamknięcie wszystkich okien
X=(-1:0.1:2)*pi;
[x,y]=meshgrid(X); %przygotowanie danych do wykresu
z=fgr2(x,y);           %obliczenie wartości z=f(x,y)
mesh(x,y,z), title('Mesh')
figure(2)              %uaktywnienie drugiego okna
meshc(x,y,z), title('Meshc')
figure(3)              %uaktywnienie trzeciego okna
meshz(x,y,z), title('Meshz')
```

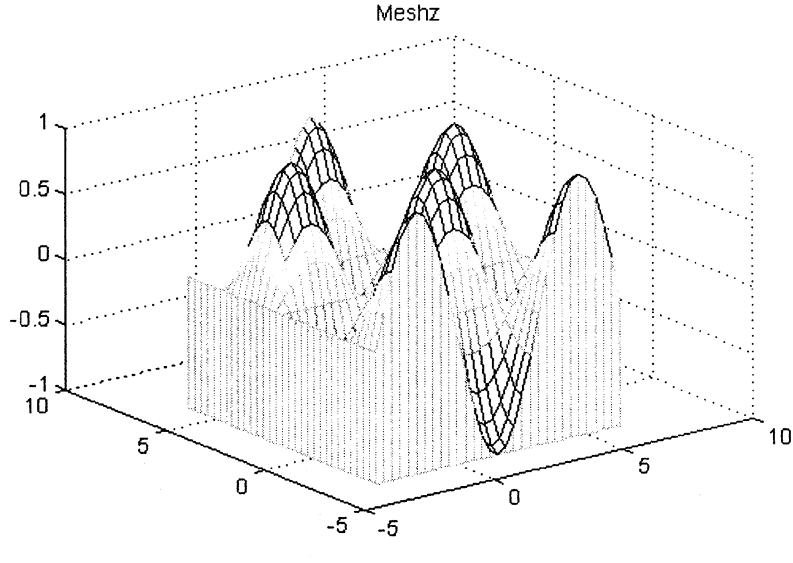
Efektem skryptu będzie narysowanie w trzech kolejnych oknach wykresów widocznych na rysunkach 4.18-4.20.



Rysunek 4.18. Wykres funkcji $f(x,y) = \sin(x)\cos(y)$ utworzony za pomocą polecenia **mesh**



Rysunek 4.19. Wykres funkcji $f(x,y) = \sin(x)\cos(y)$ utworzony za pomocą polecenia **meshc**



Rysunek 4.20. Wykres funkcji $f(x,y) = \sin(x)\cos(y)$ utworzony za pomocą polecenia **meshz**

Ćwiczenie 4.16

Napisz skrypt, który rysuje wykresy funkcji: $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ dla $x,y \in <-3,3>$ w jednym oknie graficznym: kolejno za pomocą poleceń **mesh, surf, waterfall** i **plot3**.

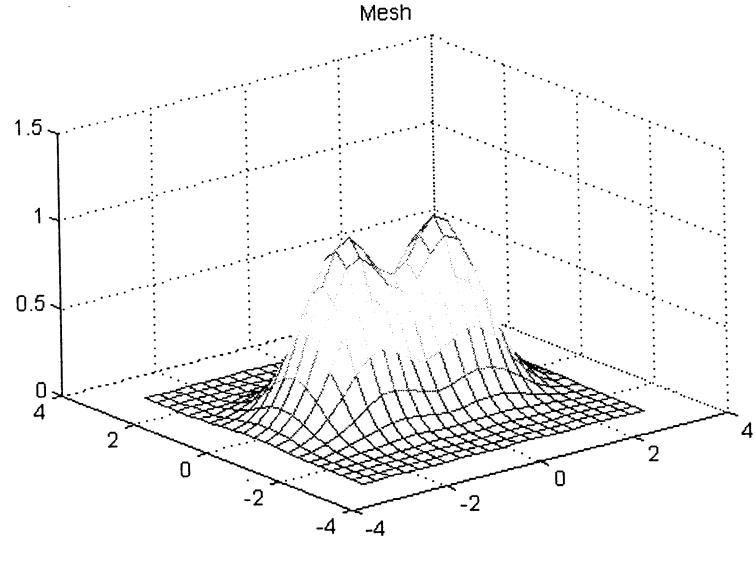
Funkcja zdefiniowana jest w pliku **fgr3.m** jako:

```
function z=fgr3(x,y)
z=exp(-(x-1).^2-y.^2)+exp(-(x+1).^2-y.^2);
```

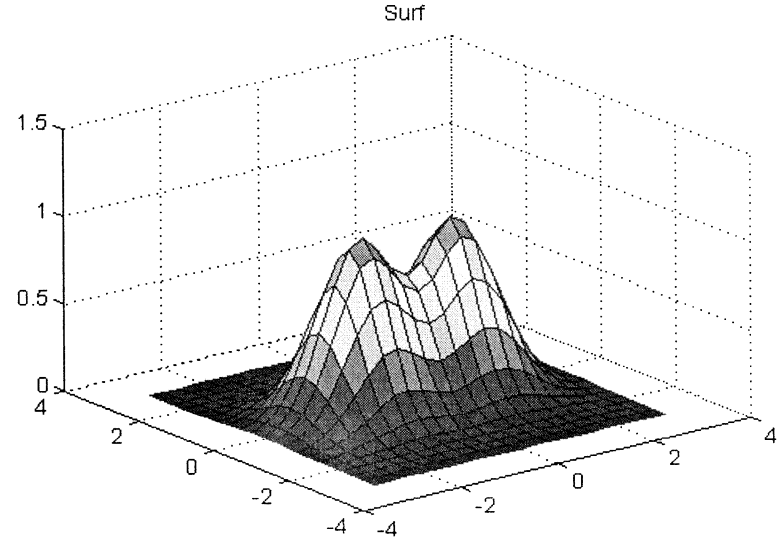
Skrypt powinien zawierać polecenia:

```
close all;          % zamknięcie wszystkich okien
w=-3:0.3:3;
[x,y]=meshgrid(w,w);  % przygotowanie danych do wykresu
z=fgr3(x,y);          % obliczenie wartości z=f(x,y)
mesh(x,y,z); title('Mesh'), pause
figure(1)             %uaktywnienie pierwszego okna
surf(x,y,z); title('Surf'), pause
figure(1)
waterfall(x,y,z); title('Waterfall'), pause
figure(1)
plot3(x,y,z); title('Plot3')
```

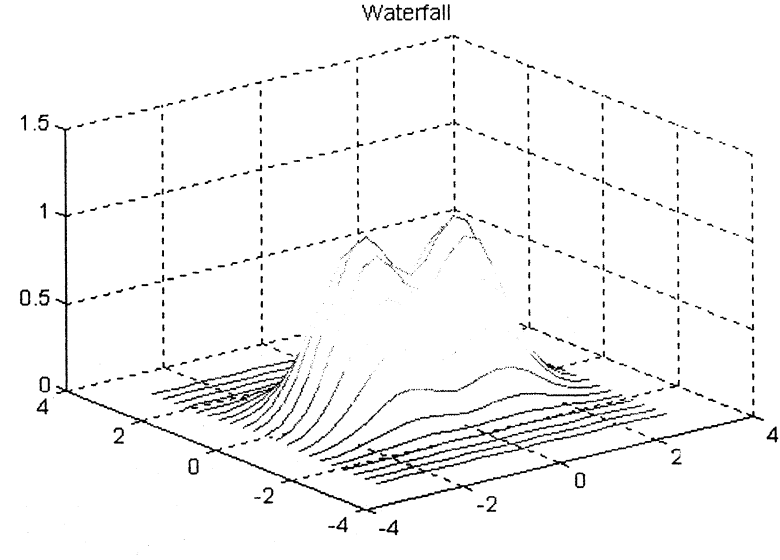
Efektem wykonania skryptu będzie narysowanie kolejno (w tym samym oknie graficznym) wykresów widocznych na rysunkach 4.21-4.24.



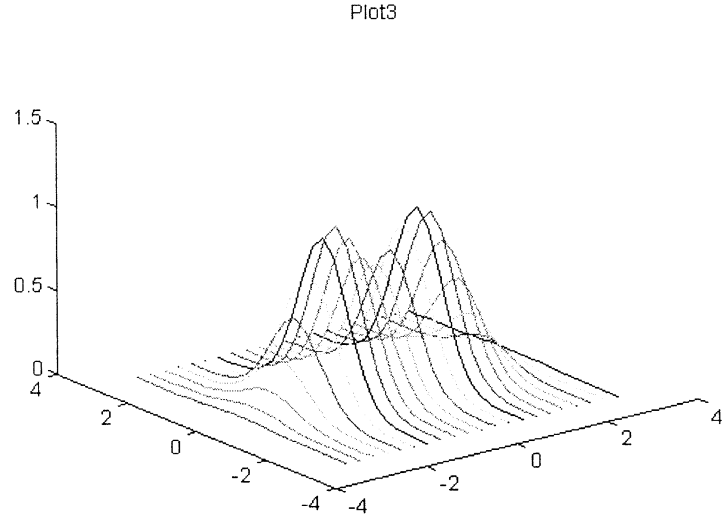
Rysunek 4.21. Wykres funkcji $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ narysowany za pomocą polecenia **mesh**



Rysunek 4.22. Wykres funkcji $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ narysowany za pomocą polecenia **surf**



Rysunek 4.23. Wykres funkcji $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ narysowany za pomocą polecenia **waterfall**



Rysunek 4.24. Wykres funkcji $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ narysowany za pomocą polecenia **plot3**

4.3.4. Mapy kolorów

Mapa kolorów jest macierzą trójkolumnową, której elementami są liczby rzeczywiste z zakresu 0,0–1,0. Każdy wiersz macierzy jest wektorem RGB (akronim ang. *red green blue*) definiującym dany kolor za pomocą intensywności trzech podstawowych kolorów: czerwonego, zielonego i niebieskiego. Funkcja **colormap** pozwala odczytać lub zmienić mapę kolorów przypisaną aktywnemu rysunkowi. Polecenie:

m=colormap

zwraca aktualną mapę kolorów **m**. Zmiana aktualnej mapy kolorów na mapę **m** następuje po wydaniu polecenia:

colormap(m)

Standardową mapę kolorów przywraca polecenie:

colormap('default')

Wybrane mapy opisu kolorów, dostępne w Matlabie, zestawiono w tabeli 4.9.

Tabela 4.9. Mapy kolorów

Mapa	Opis
gray	mapa odcieni szarości
hot	mapa kolorów ciepłych – od czarnego, przez odcienie czerwonego, pomarańczowego i żółtego, aż do białego

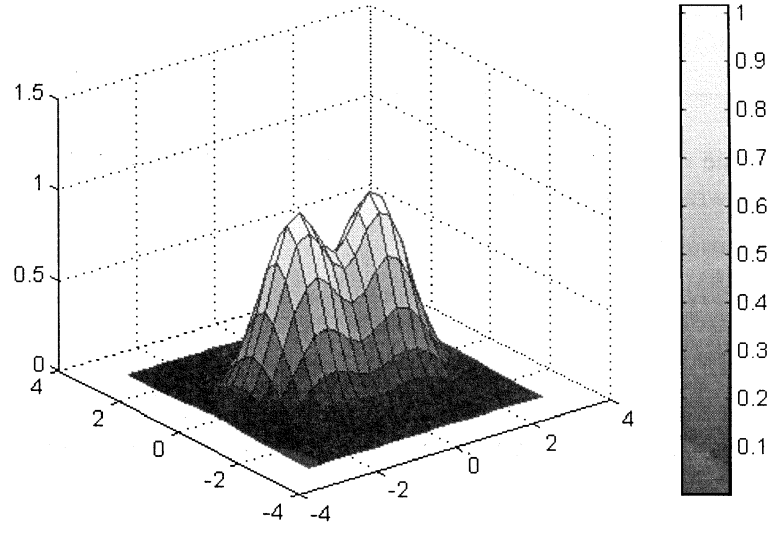
Mapa	Opis
cool	mapa kolorów zimnych – od turkusowego do karmazynowego
autumn	mapa kolorów zmieniających się od czerwonego, przez pomarańczowy, do żółtego
summer	mapa odcieni kolorów żółtego i zielonego
hsv	standardowa mapa kolorów w systemie HSV; każdy wiersz macierzy zawiera 3 liczby z przedziału <0,1> opisujące odcień, nasycenie i jaskrawość; odcień zmienia się od koloru czerwonego, przez żółty, zielony, turkusowy, niebieski, karmazynowy, z powrotem do czerwonego; nasycenie zerowe oznacza, że kolory będą odcieniami szarości

Skalę kolorów można wyświetlić na wykresie za pomocą funkcji **colorbar**.

Ćwiczenie 4.17

Za pomocą funkcji **surf** narysuj wykres funkcji **fgr3** z ćwiczenia 4.16 z użyciem mapy kolorów **gray**. Na wykresie umieść skalę kolorów.

```
close all;                % zamknięcie wszystkich okien
w=-3:0.3:3;
[x,y]=meshgrid(w,w);      % przygotowanie danych do wykresu
z=fgr3(x,y);              % obliczenie wartości z=f(x,y)
colormap(gray);           % zmiana mapy kolorów
surf(x,y,z);              % wykres w wybranej mapie kolorów
colorbar                  % wyświetlenie skali kolorów
```



Rysunek 4.25. Wykres funkcji z ćwiczenia 4.16 narysowany dla mapy kolorów **gray** z dodaną skalą kolorów

4.3.5. Wykresy poziomicowe

Wykres funkcji $z=f(x,y)$ można w Matlabie przedstawić jako wykres ***poziomicowy***. Poziomice to linie łączące punkty, w których wartość z jest jednakowa. Przed utworzeniem wykresu poziomicowego należy wygenerować siatkę na płaszczyźnie XY za pomocą funkcji **meshgrid**, podobnie jak w wypadku wykresów powierzchniowych.

Do rysowania wykresów poziomicowych służą funkcje Matlaba zestawione w tabeli 4.10.

Tabela 4.10. Funkcje rysujące wykresy poziomicowe

Funkcja	Opis
contour(z)	rysuje dwuwymiarowy wykres poziomicowy elementów macierzy **z** na siatce określonej ich indeksami; liczba poziomic i wartości, dla których rysowane są poziomice, przyjmowane są automatycznie
contour(x,y,z)	rysuje wykres poziomicowy elementów macierzy **z** na siatce określonej wektorami **x** i **y**; liczba poziomic i wartości, dla których rysowane są poziomice, przyjmowane są automatycznie
contour(...,n)	rysuje n poziomic
contour(...,v)	rysuje poziomice dla wartości określonych wektorem **v**
contour3(x,y,z,n,v)	na trójwymiarowym wykresie rysuje poziomice powierzchni opisywanej przez elementy macierzy **z**, umieszczając je na odpowiadających im wysokościach nad płaszczyzną odniesienia; domyślnie rysowanych jest n=10 poziomic
contourf(x,y,z,n,v)	rysuje dwuwymiarowy wykres n poziomic, wypełniając przestrzenie między nimi kolorem; rodzaj koloru zależy od ustalonej mapy kolorów

Dodawanie opisu poziomic do wykresu umożliwia funkcja **clabel**. Argumenty funkcji C i h zwraca funkcja rysująca wykres poziomicowy, np.

```
[C,h]=contour(x,y,z)
clabel(C,h)
```

Ćwiczenie 4.18

Napisz skrypt, który rysuje wykresy funkcji: $f(x,y)=e^{\cos(x)}e^{\cos(y)}$ dla $x, y \in \langle -\pi, \pi \rangle$ w tym samym oknie graficznym kolejno za pomocą poleceń **contour**, **contour3** i **mesh**.

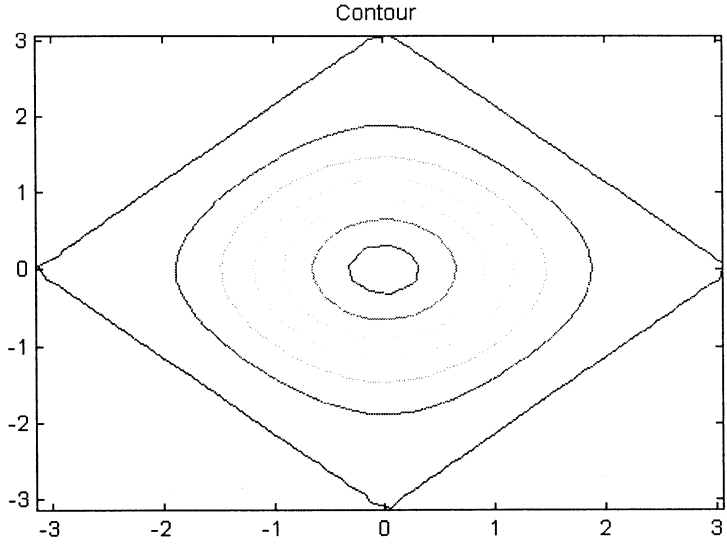
Funkcja zdefiniowana jest w pliku **fgr4.m** jako:

```
function z=fgr4(x,y)
z=exp(cos(x)).*exp(cos(y));
```

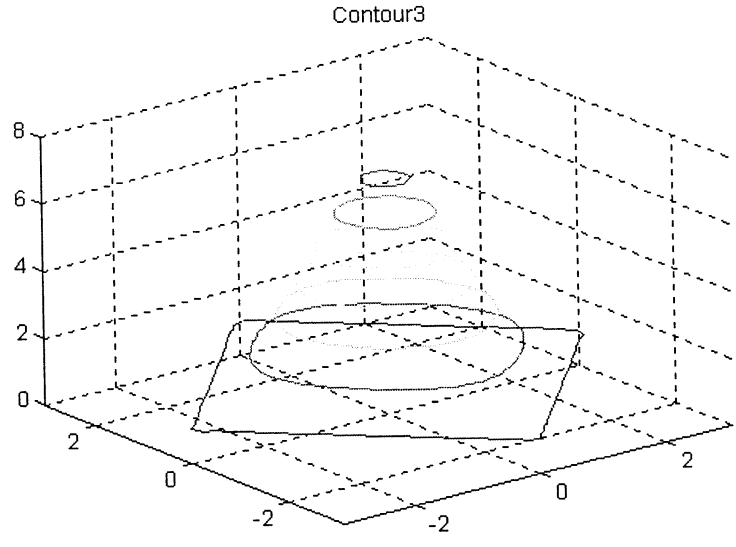
Skrypt powinien zawierać polecenia:

```
close all;                    %zamknięcie wszystkich okien
[x,y]=meshgrid(-pi:0.2:pi); %przygotowanie danych do wykresu
z=fgr4(x,y);                  %obliczenie wartości funkcji
contour(x,y,z); title('Contour'), pause
figure(1)                     %uaktywnienie pierwszego okna
contour3(x,y,z); title('Contour3'), pause
figure(1)
mesh(x,y,z); title('Mesh'), pause
```

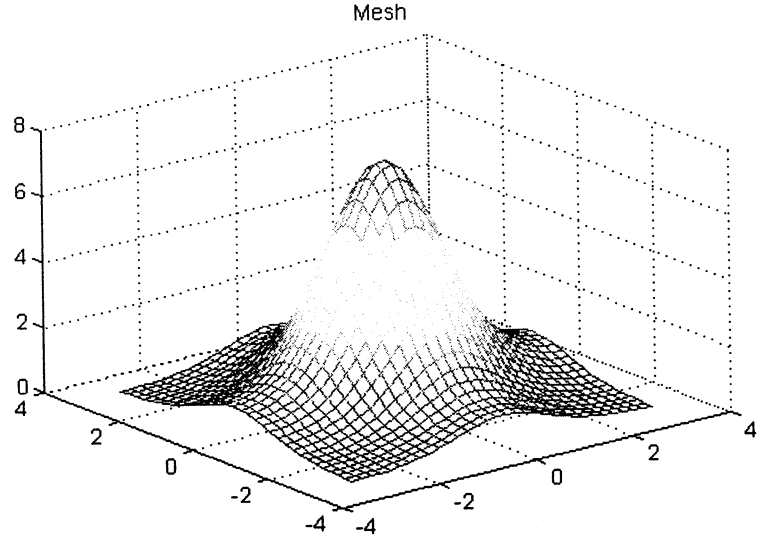
Efektem wykonania powyższych poleceń będzie narysowanie kolejno (w tym samym oknie graficznym) wykresów widocznych na rysunkach 4.26–4.28.



Rysunek 4.26. Wykres funkcji $f(x,y)=e^{\cos(x)}e^{\cos(y)}$ narysowany za pomocą funkcji **contour**



Rysunek 4.27. Wykres funkcji $f(x,y)=e^{\cos(x)}e^{\cos(y)}$ narysowany za pomocą funkcji **contour3**

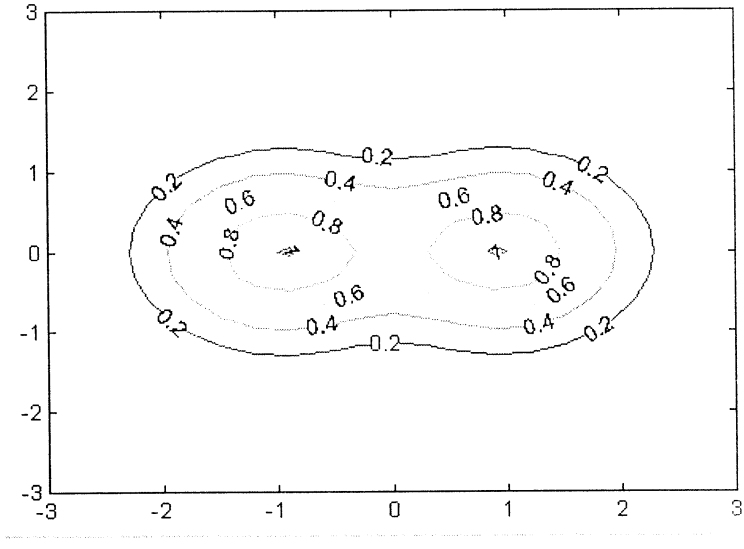


Rysunek 4.28. Wykres funkcji $f(x,y)=e^{\cos(x)}e^{\cos(y)}$ narysowany za pomocą funkcji **mesh**

Ćwiczenie 4.19

Za pomocą funkcji **contour** narysuj poziomicowy wykres funkcji **fgr3** z ćwiczenia 4.16. Dodaj opis poziomic.

```
close all                    %zamknięcie wszystkich okien
w=-3:0.3:3;
[x,y]=meshgrid(w,w);         %przygotowanie danych do wykresu
z=fgr3(x,y);                 %obliczenie wartości z=f(x,y)
[C,h]=contour(x,y,z);        %wykres poziomicowy
clabel(C,h)                  %dodanie opisu poziomic
```



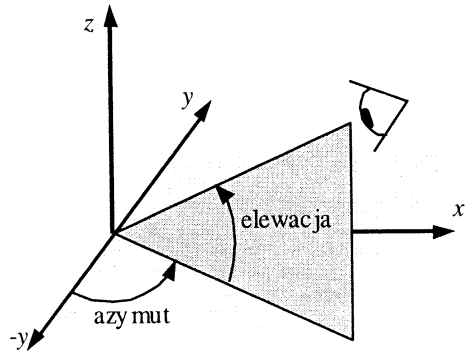
Rysunek 4.29. Poziomicowy wykres funkcji $f(x,y) = e^{-(x-1)^2-y^2} + e^{-(x+1)^2-y^2}$ z opisem poziomic

4.3.6. Zmiana kierunku obserwacji wykresu

Funkcja **view** umożliwia zmiany kierunku, z którego oglądany jest aktywny układ współrzędnych. W poleceniu:

view(az,el)

argumenty *az* i *el* określają odpowiednio: azymut i elewację (kąt podniesienia) położenia oka obserwatora (rysunek 4.30). ***Azymut*** to kąt odmierzany w płaszczyźnie XY, rosnący w kierunku przeciwnym do kierunku ruchu wskazówek zegara. ***Elewacja*** to kąt między płaszczyzną XY a prostą łączącą oko obserwatora ze środkiem wykresu. Kąty podawane są w stopniach.



Rysunek 4.30. Azymut i elewacja

Wywołanie funkcji **view**:

view(2)

ustawia kierunek obserwacji jak dla wykresów dwuwymiarowych, tzn. az=0^0, el=90^0. W wywołaniu:

view(3)

przyjmowany jest standardowy kierunek obserwacji wykresów trójwymiarowych, tzn. az=-37.5^0, el=30^0. Natomiast polecenie:

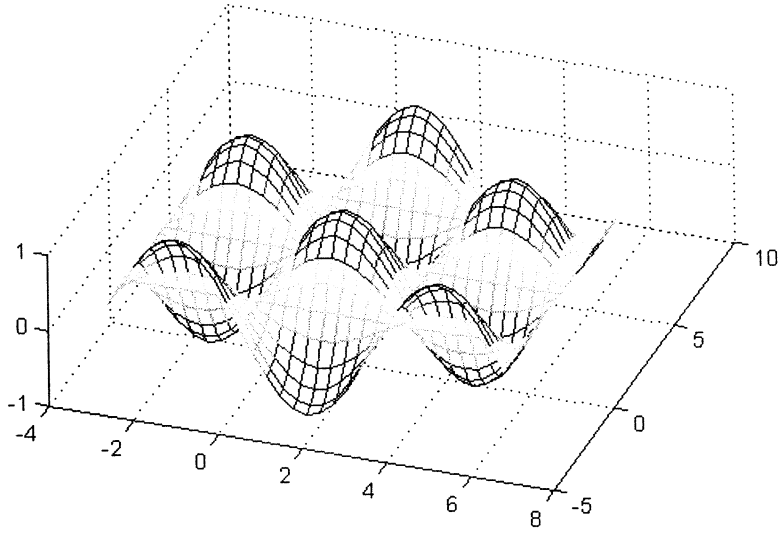
[az,el]=view

zwraca aktualne ustawienia azymutu i elewacji.

Ćwiczenie 4.20

Za pomocą funkcji **mesh** narysuj wykres funkcji **fgr2** z ćwiczenia 4.15. Zmień kierunek obserwacji wykresu, ustawiając azymut na 20^0, a elewację na 60^0.

```
close all              %zamknięcie wszystkich okien
X=(-1:0.1:2)*pi;
[x,y]=meshgrid(X); %przygotowanie danych do wykresu
z=fgr2(x,y);           %obliczenie wartości z=f(x,y)
mesh(x,y,z)            %wykres funkcji mesh
view(20,60)            %zmiana kierunku obserwacji
```



Rysunek 4.31. Wykres funkcji z ćwiczenia 4.15 ze zmienionym kierunkiem obserwacji

4.4. Prezentacja danych dyskretnych

4.4.1. Wykresy słupkowe

Wykresy słupkowe mogą reprezentować dane dyskretne w postaci poziomych lub pionowych słupków, dwu- lub trójwymiarowych. Funkcje rysujące takie wykresy zestawiono w tabeli 4.11.

Tabela 4.11. Funkcje rysujące wykresy słupkowe

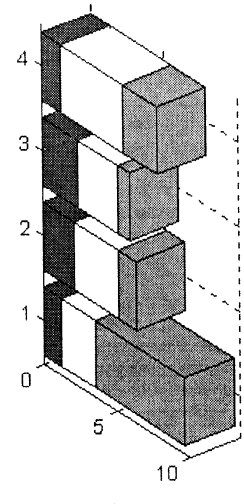
Funkcja	Opis
bar(x,y)	rysuje słupki o wysokości kolejnych elementów wektora **y** w punktach określonych przez elementy wektora **x** (uporządkowane rosnąco); jeśli **y** jest macierzą, wykresy każdego wiersza macierzy są grupowane;
bar(y)	rysuje wykres słupkowy wektora **y**, przyjmując **x=1:length(y)**; zakres osi *x* odpowiada liczbie elementów wektora **y** lub wierszy macierzy **y**
bar(..., s)	łańcuch znaków *s* określa wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3)
bar(..., w)	rysuje słupki o szerokości *w* i odpowiednio dobranej odległości pomiędzy nimi; domyślnie *w*=0.8

Funkcja	Opis
bar(..., styl)	określa *styl* rysowania słupków w przypadku, gdy **y** jest macierzą: *'group'* (domyślny; dla macierzy **y** o rozmiarze *nxm* rysuje *n* grup, każdą składającą się z *m* słupków) lub *'stack'* (do każdego wiersza macierzy **y** rysowany jest jeden słupek o wysokości odpowiadającej sumie wartości elementów w danym wierszu)
barh(x,y,s,w,styl)	działa jak funkcja **bar**, ale rysuje słupki w poziomie
bar3(x,y,s,w,styl)	rysuje trójwymiarowe słupki; argument *styl* może mieć wartości: *'detached'* (domyślny; dla każdego wiersza macierzy **y** wyświetla słupki jeden za drugim), *'grouped'* (dla macierzy **y** o rozmiarze *nxm* wyświetla *n* grup, każdą składającą się z *m* słupków), *'stacked'* (dla każdego wiersza macierzy **y** rysowany jest jeden słupek o wysokości odpowiadającej sumie wartości elementów w danym wierszu)
bar3h(x,y,s,w,styl)	rysuje trójwymiarowe słupki w poziomie

Ćwiczenie 4.21

Narysuj trójwymiarowy wykres słupkowy elementów macierzy: $\mathbf{y} = \begin{bmatrix} 1.2 & 2.3 & 6.1 \\ 2.1 & 3.0 & 1.2 \\ 2.4 & 2.7 & 0.8 \\ 1.3 & 4.2 & 2.3 \end{bmatrix}$.

```
» y=[1.2 2.3 6.1; 2.1 3.0 1.2; 2.4 2.7 0.8; 1.3 4.2 2.3];
» bar3h(y,'stacked')
```



Rysunek 4.32. Trójwymiarowy wykres słupkowy elementów trójkolumnowej macierzy (każdy poziomy słupek ma wyodrębnione wartości trzech elementów w danym wierszu macierzy)

4.4.2. Wykresy schodkowe i odcinkowe

Funkcja **stairs** działa podobnie jak **bar**, ale zamiast słupków rysuje „schodki". Jest stosowana przede wszystkim do prezentacji na wykresie dyskretnych sygnałów cyfrowych. Dane dyskretne można także przedstawić w postaci odcinków zakończonych kółkiem (domyślnie) lub innym znakiem z tabeli 4.3. Służy do tego funkcja **stem** (wykres dwuwymiarowy) lub **stem3** (wykres trójwymiarowy).

Tabela 4.12. Funkcje rysujące wykresy schodkowe i odcinkowe

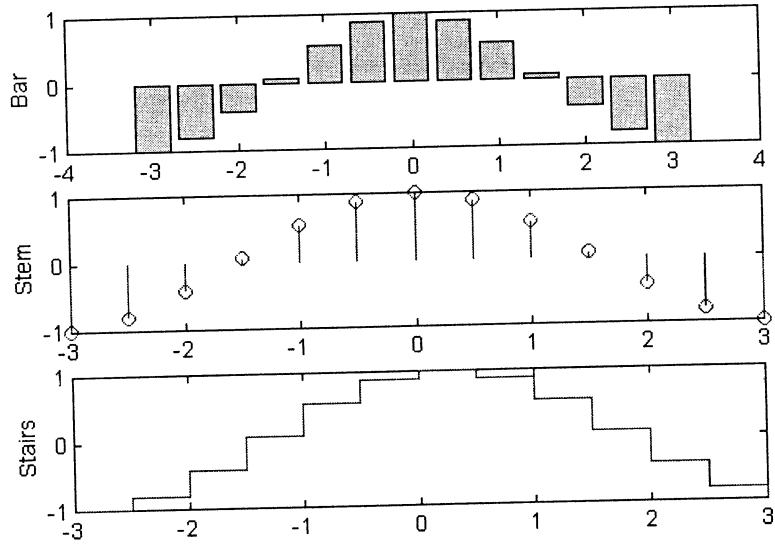
Funkcja	Opis
stairs(x,y)	rysuje „schodki" o wysokości kolejnych elementów wektora **y** w punktach określonych przez elementy wektora **x** (uporządkowane rosnąco); jeśli **y** jest macierzą, osobny wykres rysowany jest dla każdego wiersza macierzy
stairs(y)	rysuje wykres schodkowy wektora **y**, przyjmując **x=1:length(y)**
stairs(...,s)	łańcuch znaków *s* określa wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3)
stem(x,y)	na wykresie dwuwymiarowym rysuje odcinki o wysokości kolejnych elementów wektora **y** w punktach określonych przez elementy wektora **x** (uporządkowane rosnąco); jeśli **y** jest macierzą, odcinki dla każdego wiersza macierzy są dodawane
stem(y)	rysuje wykres odcinkowy wektora **y**, przyjmując **x=1:length(y)**
stem3(x,y,z)	na wykresie trójwymiarowym rysuje odcinki o wysokości kolejnych elementów wektora **z** w punktach określonych wektorami **x** i **y** (o elementach uporządkowanych rosnąco); jeśli **z** jest macierzą, odcinki dla każdego wiersza macierzy są dodawane
stem3(z)	rysuje wykres odcinkowy wektora **z** przyjmując **x=1:n**, **y=1:m**, gdzie **[m,n]=size(z)**
stem(...,'fill') **stem3(...,'fill')**	wypełnia znaki kończące odcinki
stem(...,s) **stem3(...,s)**	łańcuch znaków *s* określa wygląd rysowanej linii, jak w wypadku funkcji **plot** (tabela 4.3)

Ćwiczenie 4.22

Napisz skrypt, który demonstruje efekty działania poleceń **bar**, **stem** oraz **stairs** na przykładzie funkcji $y=\cos(x)$, gdzie **x = -3:0.5:3**. Wykresy umieść w jednym oknie graficznym.

```
close all
x=-3:0.5:3; y=cos(x);
```

```
subplot(3,1,1)
bar(x,y), ylabel('Bar')
subplot(3,1,2)
stem(x,y), ylabel('Stem')
subplot(3,1,3)
stairs(x,y), ylabel('Stairs')
```



Rysunek 4.33. Porównanie wykresów tych samych danych uzyskanych za pomocą funkcji **bar**, **stem** i **stairs**

4.4.3. Wykresy kołowe

Wykresy kołowe pozwalają przedstawić udział poszczególnych wartości w sumie całkowitej. Matlab umożliwia rysowanie wykresów kołowych na płaszczyźnie lub w trzech wymiarach (tabela 4.13).

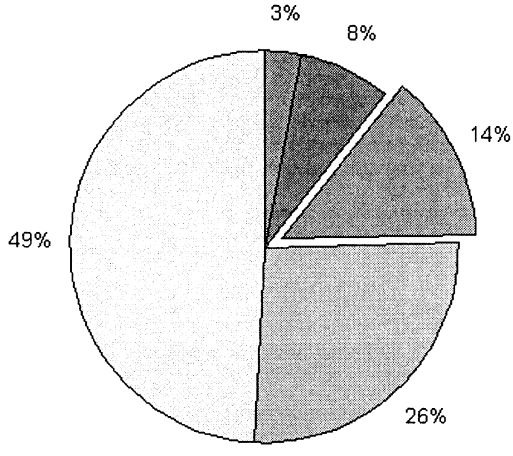
Tabela 4.13. Funkcje rysujące wykresy kołowe

Funkcja	Opis
pie(x)	rysuje wykres kołowy wektora **x**; jeśli wartość **sum(x)** jest większa bądź równa 1, to powierzchnia wycinka odpowiadającego *i*-temu elementowi wektora **x** jest równa **x(i)/sum(x)**; dla **sum(x)**<1 rysuje koło bez brakującego wycinka
pie3(x)	rysuje wykres kołowy z efektem trójwymiarowym
pie(x,r) **pie3(x,r)**	wyróżnia poprzez wysunięcie wybrane wycinki koła; **r** musi być zero-jedynkowym wektorem tej samej długości co **x** – wycinek koła odpowiadający elementowi x_i jest wysunięty, jeśli r_i=1

Ćwiczenie 4.23

Narysuj wykres kołowy wektora **x**=[32 17 9 5 2]. Wyróżnij wycinek odpowiadający trzeciemu elementowi wektora. Zmień mapę kolorów na **cool**.

```
» x=[32 17 9 5 2];
» w=[0 0 1 0 0];
» pie(x,w)
» colormap('cool')
```



Rysunek 4.34. Przykład wykresu kołowego

4.4.4. Wykresy warstwowe

Wykres warstwowy rysuje krzywą określoną wektorami **x** i **y** (podobnie jak funkcja **plot**) i wypełnia obszar pod krzywą. Może być używany zarówno do prezentacji danych dyskretnych, jak i ciągłych.

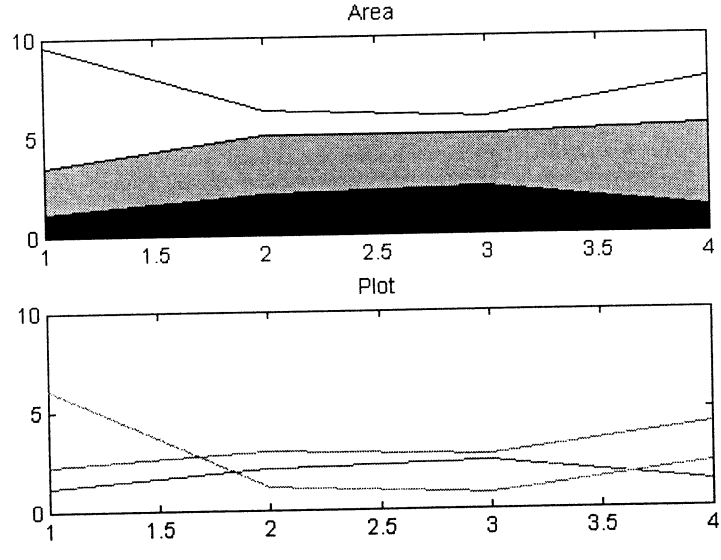
Tabela 4.14. Funkcje rysujące wykresy warstwowe

Funkcja	Opis
area(x,y)	rysuje wykres warstwowy wektora **y** w punktach określonych przez elementy wektora **x** (uporządkowane rosnąco); jeśli **y** jest macierzą, rysuje osobną krzywą dla każdej kolumny macierzy, przy czym każda kolejna krzywa rysowana jest względem poprzedniej, tzn. dla pierwszej krzywej **y1=y(:,1)**, dla drugiej krzywej **y2=y(:,2)+y1**, dla trzeciej krzywej **y3=y(:,3)+y2** itd.
area(y)	rysuje wykres warstwowy wektora **y**, przyjmując **x=1:length(y)**
area(...,ymin)	rozpoczyna wypełnianie wykresu od wartości *y=ymin*

Ćwiczenie 4.24

Napisz skrypt rysujący wykres warstwowy macierzy **y** z ćwiczenia 4.21. Użyj mapy kolorów **hot**. Porównaj wykres z wynikiem działania funkcji **plot**. Oba wykresy narysuj dla takich samych zakresów osi współrzędnych.

```
close all
y=[1.2 2.3 6.1; 2.1 3.0 1.2; 2.4 2.7 0.8; 1.3 4.2 2.3];
subplot(2,1,1)
area(y), title('Area')
colormap('hot')
w=axis;        %przypisanie wektorowi w zakresów osi x i y
               %wykresu warstwowego
subplot(2,1,2)
plot(y), title('Plot')
axis(w)   %ustalenie zakresu osi jak na wykresie warstwowym
```



Rysunek 4.35. Porównanie wykresu warstwowego z wykresem uzyskanym za pomocą funkcji **plot**

4.4.5. Histogramy

Histogramy są specjalnymi wykresami słupkowymi, wykorzystywanymi do graficznego przedstawienia rozkładu liczebności elementów wektora. W Matlabie histogramy można rysować w kartezjańskim lub biegunowym układzie współrzędnych.

Tabela 4.15. Funkcje rysujące histogramy

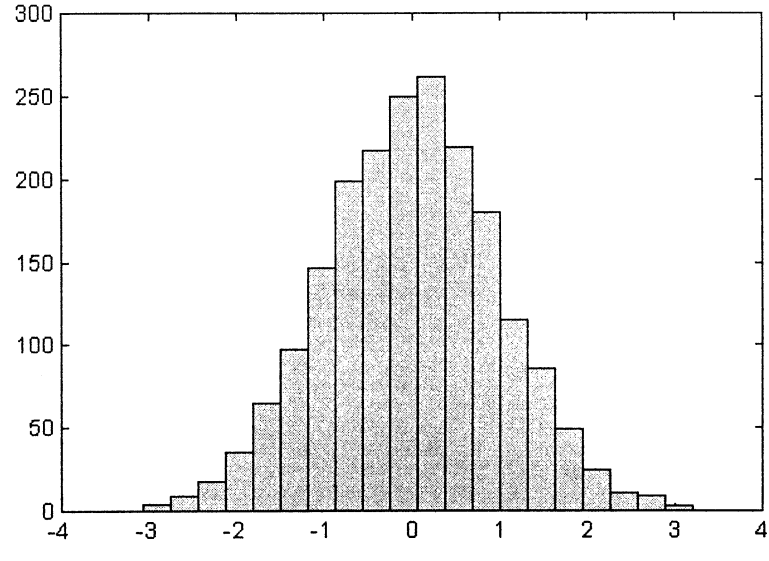
Funkcja	Opis
hist(y)	rysuje histogram w kartezjańskim układzie współrzędnych; łączy elementy wektora **y** w zależności od ich wartości w 10 grup, wyświetlanych w postaci słupków o wysokości równej liczbie elementów należących do danej grupy; zakres osi *x* odpowiada zakresowi wartości elementów wektora **y**
hist(y,x)	jeśli **x** jest wektorem, rysuje **length(x)** słupków o środkach w punktach określonych elementami wektora **x**
hist(y,n)	jeśli **n** jest liczbą, rysuje *n* słupków
rose(theta)	rysuje histogram w biegunowym układzie współrzędnych; łączy elementy wektora **theta** (kąty w radianach) w zależności od ich wartości w 20 grup
rose(theta,x)	jeśli **x** jest wektorem, rysuje **length(x)** grup
rose(theta,n)	jeśli **n** jest liczbą, rysuje *n* grup

Ćwiczenie 4.25

Za pomocą funkcji **randn** wygeneruj wierszowy wektor **y** o 2000 elementach. Narysuj histogram wektora **y** składający się z 20 słupków.

```
» y=randn(1,2000);
» hist(y,20)
```

Otrzymany wykres (rys. 4.36) odpowiada krzywej Gaussa (rozkładowi normalnemu).



Rysunek 4.36. Przykład histogramu

4.5. Grafika rastrowa

4.5.1. Obrazy statyczne

Matlab umożliwia wyświetlanie obrazów przechowywanych w plikach graficznych (w formatach np. JPG, BMP, TIFF, PCX), a także definiowanie macierzy opisujących obrazy. Niektóre standardowe funkcje obsługi obrazów przedstawione zostały w tabeli 4.16. Daleko bardziej zaawansowaną obróbkę obrazów umożliwia biblioteka Image Processing Toolbox.

Tabela 4.16. Funkcje obsługujące obrazy

Funkcja	Opis
image(c)	wyświetla w aktywnym układzie współrzędnych obraz opisany macierzą **c** – każdemu elementowi macierzy **c** odpowiada jeden punkt; elementy macierzy **c** są traktowane jako indeksy kolorów w mapie kolorów; wygląd obrazu zależy od rodzaju użytej mapy kolorów
imagesc(c)	skaluje dane opisane macierzą **c** do pełnego zakresu aktualnej mapy kolorów i wyświetla obraz
A=imread(plik,f)	wczytuje do macierzy **A** obraz z pliku graficznego o nazwie określonej łańcuchem znaków *plik* i formacie *f* (np. *'jpg'*); domyślnie Matlab szuka pliku **plik.f** w katalogu bieżącym – jeśli plik znajduje się w innym katalogu, należy podać w nazwie ścieżkę dostępu
imwrite(A,plik,f)	zapisuje obraz określony macierzą **A** w pliku graficznym o podanej nazwie w formacie *f* (jak w wypadku funkcji **imread**)

Ćwiczenie 4.26

Wczytaj i wyświetl obraz z pliku **trees.bmp** znajdującego się w katalogu bieżącym. Użyj mapy kolorów **hot**. Ukryj osie.

```
x=imread('trees','bmp');  %wczytanie obrazu do macierzy x
image(x)                  %wyświetlenie macierzy x
colormap(gray)            %zmiana mapy kolorów
axis off                  %ukrycie osi
```

Rysunek 4.37. Obraz rastrowy

4.5.2. Animacje

Matlab umożliwia tworzenie filmów animowanych z sekwencji rysunków lub obrazów. Poszczególne klatki są przechowywane w specjalnej macierzy, a następnie wyświetlane jedna po drugiej, dając złudzenie animacji.

Tabela 4.17. Funkcje obsługujące animacje

Funkcja	Opis
X=moviein(n,id,p)	generuje macierz **X** o *n* kolumnach (*n* – liczba rysunków w animacji); opcjonalne argumenty funkcji: *id* – identyfikator obiektu graficznego, z którego ma być zapisany rysunek (domyślnie rysunek aktywny), **p** – czteroelementowy wektor postaci [*lewy dolny szerokość wysokość*] opisujący prostokąt, który ma być zapisany (rozmiary w pikselach, domyślnie cały rysunek)
X=getframe	zapisuje w macierzy **X** aktualny rysunek
X=getframe(id)	zapisuje w macierzy **X** rysunek o identyfikatorze *id*
X=getframe (id, p)	zapisuje w macierzy **X** obszar prostokątny opisany wektorem **p** z rysunku o identyfikatorze *id*
movie(X)	odtwarza film zapisany w macierzy **X**
movie(X,m)	odtwarza *m* razy film zapisany w macierzy **X**; jeśli $m<0$, to film jest odtwarzany $2m$ razy: za każdym razem w przód i w tył

Ćwiczenie 4.27

Napisz skrypt symulujący rozchodzenie się fal za pomocą wykresu półokręgu $y=\sqrt{R^2-x^2}$, gdzie R – promień okręgu, $x \in <-R,R>$. Na kolejnych klatkach filmu promień R powinien się zmieniać od wartości 0.1 do 2.

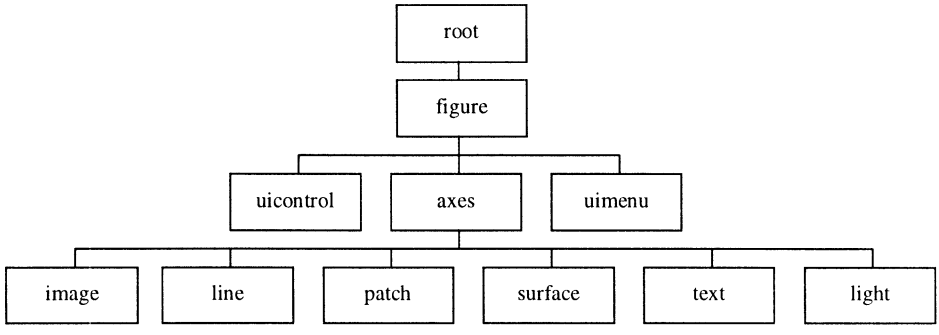
```
clear all; close all;
r=0.1:0.1:2;          %określenie kolejnych wartości promienia
rl=length(r);         %określenie liczby klatek filmu
m=moviein(rl);        %generowanie macierzy m
for i=1:rl,           %TWORZENIE KOLEJNYCH KLATEK FILMU
  R=r(i);             %promień kolejnego okręgu ma wartość r(i)
  x=-R:0.01:R; y=sqrt(R.^2-x.^2);  %obliczenie x,y
  plot(x,y)           %rysowanie wykresu y(x)
  axis equal   %ustawienie jednakowych jednostek na obu osiach
  axis([-2 2 0 2])  %ustawienie zakresów osi
  m(:,i)=getframe;  %zapisanie bieżącego wykresu
                    %w postaci klatki filmu
                    %w kolejnej kolumnie macierzy m
end                 %KONIEC TWORZENIA KLATEK
movie(m,5)          %odtworzenie filmu 5 razy
close               %zamknięcie aktywnego okna
```

4.6. Obiektowy system graficzny

4.6.1. Struktura obiektowego systemu graficznego

Każdy fragment rysunku to pewien ***obiekt*** graficzny o unikatowym ***identyfikatorze***. Każdy obiekt zawiera strukturę danych – rekord, w którym przechowywane są jego parametry i związane z nim dane. Struktura ta zawiera pola, w których można umieścić identyfikatory innych obiektów – ***przodków*** (tylko jeden dla danego obiektu) i ***potomków*** (dany obiekt może mieć wielu potomków). W ten sposób obiekty są uporządkowane w postaci drzewa (rys. 4.38).

Korzeniem drzewa obiektów (ang. *root*) jest obiekt **root**, którego potomkiem jest obiekt **figure**. Ten z kolei może być przodkiem obiektów: **uicontrol**, **axes** i **uimenu**. Potomkami obiektu **axes** są: **image**, **line**, **patch**, **surface**, **text** i **light**. Charakterystykę poszczególnych obiektów graficznych przedstawiono w tabeli 4.18.



Rysunek 4.38. Hierarchia obiektów graficznych

Tabela 4.18. Obiekty graficzne

Obiekt	Opis
root	obiekt o identyfikatorze 0, tworzony automatycznie w chwili uruchomienia pakietu; można go utożsamiać z ekranem komputera
figure	rysunek – oddzielne okno graficzne; obiekt o identyfikatorze będącym liczbą całkowitą (1 – pierwsze okno, 2 – drugie okno itd.); zawiera wszystkie pozostałe obiekty graficzne; na ekranie (**root**) może się znajdować wiele okien
axes	układ współrzędnych – prostokątny fragment obszaru okna graficznego lub cały obszar; wszystkie funkcje graficzne rysują wykresy wewnątrz danego układu współrzędnych; w jednym oknie może się znajdować wiele wykresów
uicontrol	element graficznego systemu komunikacji z użytkownikiem (GUI) – np. przycisk, suwak, pole edycyjne
uimenu	element rozwijanego menu dodanego przez użytkownika do GUI
image	obraz – element grafiki rastrowej
line	linia – linia łącząca punkty na płaszczyźnie, element wykresów dwu- i trójwymiarowych; podstawowy obiekt graficzny
patch	płat (wycinek) – wielokąt wypełniony kolorem
surface	powierzchnia – obiekt 3D, składający się z wielu wypełnionych czworokątów
text	napis – łańcuch znaków
light	źródło światła oddziałujące na obiekty wewnątrz układu współrzędnych; wpływa na wygląd wykresów trójwymiarowych

4.6.2. Podstawowe operacje na obiektach graficznych

Na obiektach graficznych dozwolone są dwie grupy operacji:

- tworzenie i usuwanie obiektów,
- zmiany w rekordzie danych związanych z obiektem.

Pola rekordu związanego z obiektem nazywane są ***własnościami*** obiektu. Własność to zmienna określająca parametr wyglądu obiektu, której można przyporządkować pewną ***wartość***.

Odpowiednie obiekty tworzone są automatycznie w momencie wywołania funkcji graficznej wysokiego poziomu. Można je również tworzyć bezpośrednio za pomocą nazw funkcji, takich samych jak nazwy poszczególnych typów obiektowych. Wywołanie takiej funkcji ma ogólnie postać:

id=funkcja(własność,wartość,...)

Argumentami funkcji mogą być pary *własność-wartość*, przy czym *własność* określana jest za pomocą łańcucha znakowego. Funkcja zwraca zawsze identyfikator nowo utworzonego obiektu *id*. Przykłady tworzenia obiektów przedstawione zostały w tabeli 4.19.

Tabela 4.19. Funkcje tworzące obiekty

Polecenie	Opis
id=figure	tworzy nowe okno graficzne i nadaje mu identyfikator *id*
id=axes	tworzy układ współrzędnych i nadaje mu identyfikator *id*
id=axes('Position',p)	tworzy układ współrzędnych określając jego położenie za pomocą własności *'Position'*; **p** – czteroelementowy wektor postaci [*lewy dolny szerokość wysokość*] opisujący prostokątny obszar zajmowany przez układ; rozmiary podawane są w jednostkach określonych własnością *'Units'* (tabela 4.23)
id=line(x,y) **id=line(x,y,z)**	tworzy obiekt typu **line** w aktywnym układzie współrzędnych i nadaje mu identyfikator *id*
id=text(x,y,tekst) **id=text(x,y,z,tekst)**	tworzy obiekt typu **text** w aktywnym układzie współrzędnych i nadaje mu identyfikator *id*

Identyfikatory obiektów graficznych zwracają także specjalne funkcje Matlaba, poszukujące ich według określonych własności (tabela 4.20).

Tabela 4.20. Poszukiwanie obiektów według własności

Polecenie	Opis
id=gco	zwraca identyfikator *id* aktywnego obiektu w bieżącym oknie (**gco** – akronim ang. *get current object*); obiekt aktywny to obiekt ostatnio kliknięty myszą; jeśli żaden obiekt nie został kliknięty – obiektem aktywnym jest okno
id=gco(idf)	zwraca identyfikator *id* aktywnego obiektu w oknie o podanym identyfikatorze *idf*
id=gcf	zwraca identyfikator aktywnego okna (**gcf** – ang. *get current figure*)
id=gca	zwraca identyfikator aktywnego układu współrzędnych (**gca** – akronim ang. *get current axes*)

Identyfikator obiektu *id* można wykorzystać do wykonywania niektórych operacji. Przykładowo polecenie

axes(id)

uaktywnia układ współrzędnych o identyfikatorze *id*. Inne operacje na obiektach o znanym identyfikatorze zestawiono w tabelach 4.21 i 4.22.

Tabela 4.21. Odczytywanie i zmiana własności obiektów

Polecenie	Opis
get(id)	zwraca listę wszystkich własności obiektu o identyfikatorze *id* oraz ich wartości
wartość=get(id,własność)	zwraca *wartość* danej *własności* obiektu o identyfikatorze *id*
set(id,własność,wartość,...)	zmienia *własność* obiektu o identyfikatorze *id*, nadając jej nową *wartość*
set(id)	wyświetla listę własności obiektu, które użytkownik ma prawo zmieniać
reset(id)	przywraca standardowe wartości własności obiektu o identyfikatorze *id*

Tabela 4.22. Usuwanie obiektów

Polecenie	Opis
delete(id)	usuwa obiekt o identyfikatorze *id* oraz jego wszystkich potomków
close(id)	zamyka okno o identyfikatorze *id*
clf	usuwa wszystkie obiekty z aktywnego okna (**clf** – akronim ang. *clear figure*)
cla	usuwa wszystkie obiekty z aktywnego układu współrzędnych (**cla** – akronim ang. *clear axes*)

Ćwiczenie 4.28

Wykonaj polecenia:

a) wyświetl informacje o wszystkich własnościach obiektu **root**

```
» get(0)
```

b) wyświetl bieżące ustawienia własności *ScreenSize* obiektu **root**

```
» get(0,'ScreenSize')
```

4.6.3. Własności wybranych obiektów

Z uwagi na ograniczoną objętość książki, zaprezentujemy jedynie podstawowe własności wybranych obiektów, co jednak pozwoli dostrzec możliwości pakietu.

Podstawową własnością wszystkich obiektów graficznych jest *'Type'*. Jej wartością jest łańcuch znaków będący nazwą typu obiektu – np. *'root'*, *'text'* itp. Niektóre inne własności wybranych obiektów zostały przedstawione w tabelach 4.23-4.26.

Tabela 4.23. Własności obiektu **figure**

Własność	Opis
'Color'	kolor tła okna; specyfikacja np. jak w tabeli 4.3
'MenuBar'	przełącznik decydujący, czy widoczne jest menu okna (*'figure'*, wartość domyślna), czy nie (*'none'*)
'Name'	łańcuch znaków określający tytuł okna (domyślnie łańcuch pusty ' ')

Tabela 4.24. Własności obiektu **axes**

Własność	Opis
'Box'	przełącznik decydujący, czy układ współrzędnych jest widoczny (*'on'*), czy nie (*'off'*, wartość domyślna)
'Color'	kolor tła prostokąta obejmowanego przez układ współrzędnych (*'none'* lub specyfikacja np. jak w tabeli 4.3); domyślnie biały
'FontAngle'	kąt nachylenia czcionki – *'normal'* (domyślnie), *'italic'* lub *'oblique'*
'FontName'	nazwa czcionki, która zostanie użyta do wyświetlenia wartości liczbowych na osiach, tytule wykresu i etykietach osi (np. *'Times'*, *'Arial'*, *'Courier'*)
'FontSize'	rozmiar czcionki w jednostkach określonych własnością *'FontUnits'*
'FontUnits'	jednostka rozmiaru czcionki: *'points'* (domyślnie), *'normalized'*, *'inches'*, *'centimeters'* lub *'pixels'* (patrz opis własności *'Units'*)

Własność	Opis
'GridLineStyle'	typ linii, którą zostanie narysowana siatka pomocnicza (*'none'* lub specyfikacja jak w tabeli 4.3); domyślnie linia kropkowana
'LineWidth'	grubość linii osi układu w punktach; domyślnie 0,5
'PlotBoxAspectRatio'	trzyelementowy wektor określający proporcje osi układu współrzędnych; własność ustalana domyślnie przez funkcję graficzną
'Position'	czteroelementowy wektor postaci [*lewy dolny szerokość wysokość*], opisujący prostokątny obszar zajmowany przez układ; rozmiary podawane są w jednostkach określonych własnością *'Units'*
'Units'	jednostki wykorzystywane do ustalenia własności *'Position'* – *'inches'* (cale), *'centimeters'*, *'characters'* (rozmiary znaków domyślnej czcionki), *'normalized'* (zmieniające się od 0,0 do 1,0 na każdej osi), *'points'* (punkty – 1 punkt=1/72 cala), *'pixels'*; domyślnie *'normalized'*
'XTick', *'YTick'*, *'ZTick'*	położenie oznaczeń wartości na odpowiednich osiach (wektory)

Tabela 4.25. Własności obiektu **line**

Własność	Opis
'Color'	kolor linii (specyfikacja jak w tabeli 4.3)
'LineStyle'	typ linii (*'none'* lub specyfikacja jak w tabeli 4.3)
'LineWidth'	grubość linii w punktach (domyślnie 0.5)

Tabela 4.26. Własności obiektu **text**

Własność	Opis
'Color'	kolor liter (specyfikacja jak w tabeli 4.3)
'FontAngle'	nachylenie czcionki – *'normal'* (domyślnie), *'italic'* lub *'oblique'*
'FontName'	nazwa czcionki (np. *'Times'*, *'Arial'*, *'Courier'*)
'FontSize'	rozmiar czcionki w jednostkach określonych własnością *'FontUnits'*
'FontUnits'	jednostka rozmiaru czcionki: *'points'* (domyślnie), *'normalized'*, *'inches'*, *'centimeters'* lub *'pixels'* (patrz opis własności *'Units'* obiektu **axes**)
'FontWeight'	grubość liter: *'light'* – czcionka delikatna, *'normal'* – normalna, *'demi'* – lekko pogrubiona, *'bold'* – pogrubiona; domyślnie *'normal'*
'HorizontalAlignment'	określa, w jaki sposób napis jest umieszczany względem punktu ustalonego przez własność *'Position'* (np. *'left'* – na prawo od punktu, tzn. wyrównany w lewo do punktu, *'center'*, *'right'*)

Własność	Opis
'Position'	położenie tekstu w układzie współrzędnych
'Rotation'	kąt o jaki jest obrócony wyświetlany tekst

Ćwiczenie 4.29

Za pomocą polecenia **mesh** narysuj trójwymiarowy wykres funkcji $z(x,y)=\sin(x)+2\cos(y)$ w przedziale $x,y\in\langle -2\pi,2\pi\rangle$. Dodaj siatkę. Zakres skali na obu osiach ustaw od -2π do 2π.

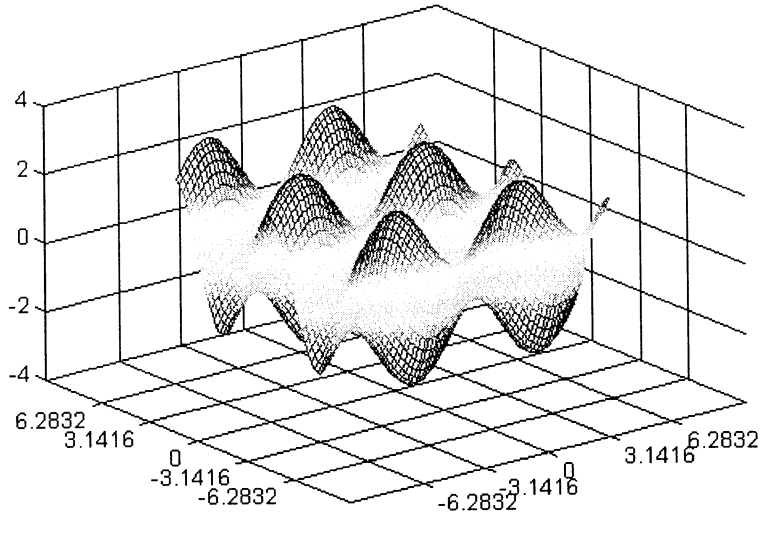
Zmień własności obiektu **axes**:

a) *'GridLineStyle'* (typ linii siatki) na linię ciągłą,

b) *'XTick'* i *'YTick'* (położenie oznaczeń na osiach) określ wektorem $[-2\pi, -\pi, 0, \pi, 2\pi]$.

Odpowiedni skrypt może mieć postać:

```
clf                                  %czyszczenie aktywnego okna
[x,y]=meshgrid(-2*pi:0.2:2*pi);
z=sin(x)+2*cos(y);
mesh(x,y,z)
grid on;                             %dodanie do rysunku siatki
axis=([-2*pi,2*pi,-2*pi,2*pi]); %ustawienie zakresu osi
xt = -2*pi:pi:2*pi;
yt = -2*pi:pi:2*pi;
set(gca,'GridLineStyle','-'); %zmiana własności obiektu
set(gca,'XTick',xt);                 %określonego przez funkcję gca
set(gca,'YTick',yt);
```



Rysunek 4.39. Przykład zmiany własności obiektu **axes**

4.7. Graficzny system komunikacji z użytkownikiem (GUI)

Graficzny system komunikacji z użytkownikiem (ang. *graphical user interface*, GUI) dodaje do grafiki Matlaba ***interaktywność*** – możliwość reagowania na polecenia użytkownika. Za pomocą GUI można tworzyć np. przyciski, suwaki, pola edycyjne, a także dodawać własne menu. Wszystkie elementy GUI użytkownik może obsługiwać myszą. Każdemu elementowi można przypisać obsługę zdarzeń, realizowaną przez wykonanie odpowiednich poleceń Matlaba.

W kolejnych rozdziałach omówimy główne zasady programowania GUI. Matlab umożliwia także graficzne projektowanie GUI poprzez uruchomienie polecenia Show GUI Layout Tool w menu File okna poleceń pakietu.

4.7.1. Tworzenie obiektów w GUI

Graficzny system komunikacji z użytkownikiem składa się z dwóch rodzajów obiektów: **uicontrol** oraz **uimenu**.

Funkcja **uicontrol** jest wykorzystywana do tworzenia elementów GUI. Jej wywołanie może mieć postać:

id=uicontrol(własność, wartość,...)

lub

id=uicontrol(idf, własność, wartość,...)

Argumentami funkcji są pary *własność–wartość* określane jak w rozdziale 4.6.2. Opcjonalny argument *idf* określa identyfikator przodka (rysunku, okna itp.), wewnątrz którego zostanie utworzony dany obiekt graficzny o identyfikatorze *id*.

Za pomocą funkcji **uimenu** można tworzyć dodatkowe elementy menu głównego lub podmenu. Funkcję wywołuje się podobnie jak w przypadku funkcji **uicontrol**:

id=uimenu(własność, wartość,...)

lub

id=uimenu(idf, własność, wartość,...)

Argumentami funkcji są także pary *własność–wartość*. Opcjonalny argument *idf* może określać identyfikator przodka (nazwę menu głównego lub podmenu), wewnątrz którego zostanie utworzony dany element menu o identyfikatorze *id*.

4.7.2. Właściwości obiektów w GUI

W tabeli 4.27 przedstawione zostały niektóre własności obiektów **uimenu** i **uicontrol**. Do modyfikacji własności mogą także służyć polecenia **set** i **get** (tabela 4.21).

Tabela 4.27. Własności modyfikowane za pomocą funkcji **uicontrol**

Własność	Opis
'Style'	typ elementu; możliwe wartości: *'pushbutton'* (przycisk), *'togglebutton'* (przełącznik), *'edit'* (pole edycyjne), *'checkbox'* (pole wyboru z możliwością wyboru kilku opcji), *'radiobutton'* (pole wyboru z możliwością wyboru tylko jednej opcji), *'text'* (pole tekstowe), *'frame'* (ramka), *'slider'* (suwak), *'listbox'* (lista), *'popupmenu'* (lista rozwijana)
'CallBack'	czynność wykonywana po uaktywnieniu elementu przez kilknięcie myszą; może to być łańcuch znaków zawierający dowolne polecenie lub kilka poleceń Matlaba albo łańcuch znaków będący nazwą skryptu Matlaba
'Position'	czteroelementowy wektor postaci: [*lewy dolny szerokosc wysokosc*] określający położenie lewego dolnego rogu elementu względem lewego dolnego rogu okna, a także szerokość i wysokość elementu; rozmiary podawane są w jednostkach określonych własnością *'Units'*
'String'	łańcuch znaków określający np.: – etykietę wyświetlaną na obiekcie (w przypadku przycisków, przełączników i wyłączników) – elementy listy oddzielone pionową kreską (w przypadku list rozwijanych) – edytowany tekst (w przypadku pól edycyjnych)
'Units'	jednostki wykorzystywane do ustalenia własności *'Position'*; określane jak w wypadku obiektu **axes** (tabela 4.24); domyślnie *'points'*
'Value'	liczba określająca aktualny stan elementu, np.: – w wypadku suwaków jest to liczba z przedziału od *Min* do *Max*, wybrana myszą – w wypadku list rozwijanych jest to numer wybranej pozycji na liście
'Min', 'Max'	granice, w jakich zmienia się wartość własności *'Value'* przycisków, przełączników, wyłączników i suwaków

Tabela 4.28. Własności modyfikowane za pomocą funkcji **uimenu**

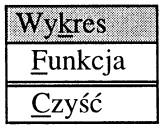
Własność	Opis
'Label'	ciąg znaków określający nazwę elementu menu; jeśli przed jedną z liter zostanie umieszczony znak '&', spowoduje to wyróżnienie danej litery w nazwie (podkreślenie); dany element menu można będzie uruchomić szybko jednoczesnym naciśnięciem klawisza Alt i podkreślonej litery

Własność	Opis
'Separator'	oddziela parametry podmenu poziomą kreską; może przybierać wartości *'on'* lub *'off'* (włączenie/wyłączenie separatora)
'CallBack'	czynność wykonywana po uaktywnieniu elementu menu (jak w wypadku obiektów tworzonych poleceniem **uicontrol**)

Ćwiczenie 4.30

Dla danych a, b, c wykonaj wykres funkcji $f(x) = \sin(ax^3+bx+c)$. Ponadto:

a) zmień nazwę rysunku w pasku tytułowym okna na: „*Wykresy funkcji*";
b) dodaj element menu głównego o nazwie „*Wykres*" (wyróżniona ma być litera *k*);
c) do elementu menu dodaj rozwijane podmenu o postaci pokazanej poniżej;



rysowanie wykresu funkcji $f(x) = \sin(ax^3+bx+c)$

czyszczenie wykresu

d) na dole rysunku dodaj: suwak, pole edycyjne i listę rozwijaną; wszystkie te elementy będą zmieniały wartości parametru a; ponadto ustaw:
   - wartość minimalną suwaka: 0, wartość maksymalną: 10,
   - etykietę w polu edycji: „a= " oraz wartość parametru a,
   - etykietę menu rozwijanego zawierającą trzy pozycje: „0.1", „1", „10".

Wartość parametru a, przyjmowana do obliczeń oraz wyświetlana w polu edycji, obliczana będzie według wzoru: $a_1 \cdot skala$ (a_1 – wartość wybrana na suwaku, *skala* – wybrana pozycja z listy rozwijanej).

Rozwiązanie zadania umieszczone zostanie w kilku plikach. Poniżej przedstawiamy ich zawartość.

1. Plik główny **obiekty.m**.

```
clf; clear all;
global a b c skala;
a=1; b=1; c=1;     %początkowe wartości parametrów a, b, c
skala=1;           %początkowa wartość parametru skala
set(gcf,'Name','Wykresy funkcji'); %zmiana własności
                                   %obiektu figure
%TWORZENIE LISTY ROZWIJANEJ
Przel = uicontrol(gcf,'Style','popupmenu',...
   'String','0.1|1|10', ...      %etykieta listy
   'Position',[420 7 70 17], ... %określenie położenia obiektu
   'CallBack','przelicz');       %po uaktywnieniu obiektu
                          %uruchamiany będzie skrypt przelicz.m
```

```
%TWORZENIE SUWAKA
Suwak = uicontrol('Style','slider');     %tworzenie suwaka
set(Suwak,'Position',[50 7 100 17]);     %określenie
                                         %położenia obiektu
set(Suwak,'CallBack','ustaw');       %po uaktywnieniu obiektu
                                %uruchamiany będzie skrypt ustaw.m
set(Suwak,'Min',0,'Max',1);          %ustalenie zakresów suwaka
set(Suwak,'Value',a);       %przypisanie początkowej wartości
                            %suwaka parametru a
%TWORZENIE POLA EDYCYJNEGO
Edit1 = uicontrol('Style','edit');  %tworzenie pola edycyjnego
set(Edit1,'Position',[200 7 200 17]);       %określenie
                                            %położenia obiektu
set(Edit1,'String',[' a=  ',num2str(a)])
                          %ustalenie treści edytowanego tekstu
%TWORZENIE MENU
idm=uimenu('Label','Wy&kres');   %tworzenie elementu menu idm
uimenu(idm,'Label','&Funkcja','CallBack','wykres_f');
             %tworzenie podmenu do elementu idm
             %po kliknięciu uruchamiany będzie plik wykres_f.m
uimenu(idm,'Label','&Czysc','CallBack','cla','Separator','on')
             %tworzenie drugiego podmenu do elementu idm
             %po kliknięciu uruchamiana będzie funkcja cla
             %separator jest włączony
```

2. Plik **przelicz.m** – przeliczanie skali z listy rozwijanej utworzonej w pliku **obiekty.m**.

```
global a skala
val1 = get(Przel,'Value');
       %odczytanie wartości z listy rozwijanej
if(val1 == 1)          %jeśli wybrano pierwszą pozycję z listy
      skala = 0.1;
elseif (val1 == 2)     %jeśli wybrano drugą pozycję z listy
      skala = 1;
elseif (val1 == 3)     %jeśli wybrano trzecią pozycję z listy
      skala = 10;
end
```

3. Plik **ustaw.m** – zmiana wartości parametru *a* na podstawie wskazań suwaka i listy.

```
global a skala;
a1 = get(Suwak,'Value');
%odczytanie aktualnej wartości ustawionej na suwaku
a=a1.*skala;  %przeliczenie nowej wartości parametru a
set(Edit1,'String',[' a=  ',num2str(a)]);
         %wyświetlenie nowej wartości a w oknie edycyjnym
```

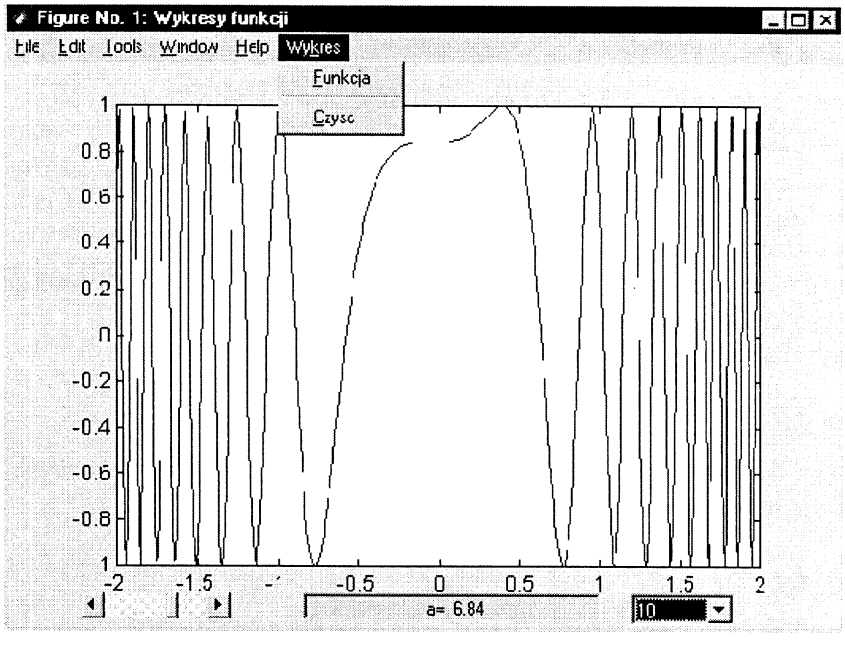
4. Plik **wykres_f.m** – wykres funkcji **fsin3**.

```
global a b c skala
fplot('fsin3',[-2 2])
```

5. Plik **fsin3.m** – obliczenie wartości funkcji $f(x)$.

```
function y=fsin3(x)
global a b c
y=sin(a*x.*x.*x+b*x.*x+c);
```

Uruchomienie skryptu **obiekty.m** spowoduje wywołanie pozostałych plików. Na rysunku 4.40 widoczne są wszystkie utworzone elementy GUI.



Rysunek 4.40. Okno graficzne z dodanymi elementami GUI

4.8. Zadania do samodzielnego wykonania

Zadanie 4.1

Za pomocą polecenia **subplot** umieść w jednym oknie graficznym dwa okienka na dwa sposoby:

a) 

b) 

W jednym okienku przedstaw wykres funkcji $\sin(x)$, a w drugim wykres funkcji $\cos(x)$, w obu przypadkach $x \in \langle -2\pi, 2\pi \rangle$.

Zadanie 4.2

Przyjmując dane: **x**=[0 1 2 3 4 5 6] oraz **y**=[0 0.95 3.9 9.4 15.2 22.3 37.1]:

a) narysuj wykres wartości $y(x)$, oznaczając je kółkami;
b) oblicz wartości $y_2(x)=x^2$ i oznacz je na wykresie kwadracikami, łącząc jednocześnie linią ciągłą;
c) zakres skali na osi x ustaw od 0 do 6, a na osi y – od 0 do 40;
d) dodaj opisy osi ('*oś x*', '*oś y*'), tytuł ('*Porównanie*') oraz legendę, określając pierwszy wykres jako '*pomiary*', a drugi – '*obliczenia*';
e) narysuj siatkę.

Zadanie 4.3

Przyjmując dane: **x**=[1 2 3 4] i **y**=[2.2 6.5 0 4.1] narysuj kolejno w różnych oknach trzy rodzaje wykresów dyskretnych: **bar**, **stem** i **stairs**.

Zadanie 4.4

Oblicz wartości macierzy $z(x,y) = (x-5)^2 - (y-5)^2$ dla $x,y=1,2,...,10$

Narysuj i porównaj wykresy wartości **z**:

a) poziomicowe – **contour**, **contour3**,
b) powierzchniowe – **surf** (**surfc**, **surfl**), **mesh** (**meshc**, **meshz**), **waterfall**.

Zadanie 4.5

Za pomocą polecenia **surf** narysuj wykres funkcji $z(x,y)= (x-y)(x+y)+ e^{\sqrt{x^2+y^2}}$ w przedziale $x,y \in \langle -5,5 \rangle$. Dodaj siatkę. Zakres skali na obu osiach ustaw od -5 do 5.

Zadanie 4.6

Zmień własności aktywnych obiektów z zadania 4.5, ustawiając wartości:

a) *GridLineStyle* – na linię kropkowaną,
b) *XTick* i *YTick* – na [-4, -2, 0, 2, 4].

Zadanie 4.7

W lewym dolnym rogu okna z wykresem z zadania 4.6 utwórz przycisk o nazwie „Czyść", którego naciśnięcie spowoduje wyczyszczenie aktywnego układu współrzędnych.

Zadanie 4.8

Odczytaj dane zapisane w pliku tekstowym **temperatura.txt** (zadanie 3.3) i sporządź wykresy *Tc(t)* i *Tk(t)*, gdzie **t=1:length(Tc)**, każdy innym kolorem linii. Zmodyfikuj wykres, zmieniając właściwości obiektów graficznych:

a) zmień nazwę rysunku w pasku tytułowym okna na: *Wykresy temperatury*'
b) dodaj element menu głównego o nazwie *Przebiegi* (wyróżniona ma być litera *P*);
c) dodaj rozwijane podmenu, składające się z dwóch elementów – *Tc(t)* i *Tk(t)*; uaktywnienie jednego z elementów podmenu będzie powodowało rysowanie właściwego wykresu.

5. Obliczenia numeryczne

Wykonując obliczenia numeryczne [2,7] za pomocą pakietu Matlab, zakładamy, że:

- operujemy na liczbach, dopuszczając możliwość występowania niedokładności w ich reprezentacji,
- podczas obliczeń mogą występować niedokładności.

Wobec powyższego obliczenia numeryczne cechują m.in. następujące właściwości:

- przy danym sposobie postępowania znana jest ilość pamięci potrzebna do przechowania struktur danych wykorzystywanych w trakcie obliczeń,
- niewielkie błędy mogą się pojawiać na każdym etapie obliczeń,
- należy się liczyć z tym, że wynik będzie obarczony pewną niedokładnością, która może okazać się nie do zaakceptowania.

5.1. Miejsca zerowe i minima funkcji, pierwiastki wielomianów

5.1.1. Poszukiwanie miejsc zerowych i minimów funkcji

Funkcje Matlaba zestawione w tabeli 5.1 umożliwiają znajdowanie miejsc zerowych i minimów nieliniowych funkcji matematycznych.

Tabela 5.1. Funkcje wyznaczające miejsca zerowe i minima funkcji matematycznych

Funkcja	Opis
x1=fzero(f,x0)	zwraca miejsce zerowe *x1* nieliniowej funkcji jednej zmiennej $f(x)$; argument *x0* określa początkowe przybliżenie wartości szukanego miejsca zerowego (punkt startowy); *f* jest łańcuchem znaków określającym funkcję Matlaba: nazwę funkcji (standardowej lub zapisanej w m-pliku) lub definicję funkcji **inline**
x1=fminbnd(f,x0,xk)	zwraca wartość *x1*, dla której nieliniowa funkcja jednej zmiennej $f(x)$ osiąga minimum; argumentami funkcji są: liczby *x0* i *xk*, określające początek i koniec przedziału poszukiwań, oraz *f* – łańcuch znaków, jak w przypadku funkcji **fzero**

Funkcja	Opis
X=fminsearch(f,X0)	zwraca wektor wartości **X**, dla których nieliniowa funkcja wielu zmiennych osiąga minimum; argumentami funkcji są: wektor **X0**, określający punkt startowy poszukiwań oraz *f* – łańcuch znaków, jak w przypadku funkcji **fzero**

Wszystkie funkcje z tabeli 5.1 można wywoływać z opcjonalnym argumentem *opcje*, np.

x1=fzero(f,x0,opcje)

Opcje określają parametry wywołania tych funkcji. Domyślne ustawienia parametrów można zmieniać za pomocą funkcji **optimset**:

opcje = optimset(parametr,wartość,...)

Argumentami funkcji **optimset** są pary *parametr–wartość*, przy czym *parametr* określany jest za pomocą łańcucha znaków. Wybrane *parametry* zestawiono w tabeli 5.2.

Tabela 5.2. Parametry zmieniane za pomocą funkcji **optimset**

Parametr	Opis
'Diagnostics'	wydruk diagnostyki minimalizowanej funkcji lub rozwiązywanego równania; dopuszczalne wartości: '*on*' lub '*off*' (domyślna)
'Display'	sposób wyświetlania wyników: bez wyświetlania ('*off*'), wyświetlanie wyników po każdej iteracji ('*iter*'), wyświetlanie tylko ostatecznego rozwiązania ('*final*' – wartość domyślna)
'MaxFunEvals'	maksymalna dozwolona liczba obliczeń wartości funkcji (liczba całkowita dodatnia)
'MaxIter'	maksymalna dozwolona liczba iteracji (liczba całkowita dodatnia)
'TolX'	dokładność wykonywania obliczeń (liczba dodatnia, np. 2.2e-016)

Wywołując funkcję **optimset** bez argumentów:

```
» optimset
```

uzyskujemy listę parametrów i ich możliwych wartości. Z kolei wywołanie funkcji w postaci np.

```
» optimset('fminbnd')
```

wyświetli nazwy parametrów oraz ich domyślne wartości dla konkretnej funkcji **fminbnd**.

Wartości parametrów niewymienionych jako argumenty funkcji **optimset** pozostają niezmienione. Z kolei ustawienie wartości dowolnego parametru na [] powoduje przyjęcie domyślnej wartości dla tego parametru.

Ćwiczenie 5.1

Znajdź miejsca zerowe wymienionych niżej funkcji w otoczeniu podanego punktu.

a) $f(x)=\cos(2x-\pi)$ – w otoczeniu punktu $x=3$,

```
» f = inline('cos(2*x-pi)');          (definicja funkcji inline)
» x = fzero(f,3)
Zero found in the interval: [2.3212, 3.48].
x = 2.3562
```

b) $f(x)=\text{abs}(x)+1$ w otoczeniu punktu $x=1$,

```
» fzero(inline('abs(x)+1'), 1)
Exiting fzero: aborting search for an interval containing a
sign change because NaN or Inf function value encountered
during search (Function value at -Inf is Inf)
Check function or try again with a different starting value.
ans = NaN
```

Otrzymaliśmy komunikat informujący o przerwaniu obliczeń, a jako wynik zwrócona została wartość **NaN** (funkcja $f(x)=\text{abs}(x)+1$ nie ma miejsc zerowych)

c) $f(x)=\cos(x)$ w otoczeniu punktu $x=3$,

```
» fzero('cos', 3)                     (nazwa funkcji standardowej)
Zero found in the interval: [1.08, 4.3576].
ans = 1.5708
```

d) $f(x)=\cos(x)$ w otoczeniu punktu $x=1.6$ z wyświetleniem informacji o kolejnych iteracjach .

```
» fzero('cos', 1.6, optimset('Display','iter'))
Func-count      x          f(x)          Procedure
    1         1.6       -0.0291995       initial
    2         1.55475    0.0160505       search

   Looking for a zero in the interval [1.5547, 1.6]

    3          1.5708  -1.0276e-006      interpolation
    4          1.5708  4.41237e-011      interpolation
    5          1.5708  6.12323e-017      interpolation
    6          1.5708 -6.04901e-016      interpolation
Zero found in the interval: [1.5547, 1.6].
ans =
    1.5708
```

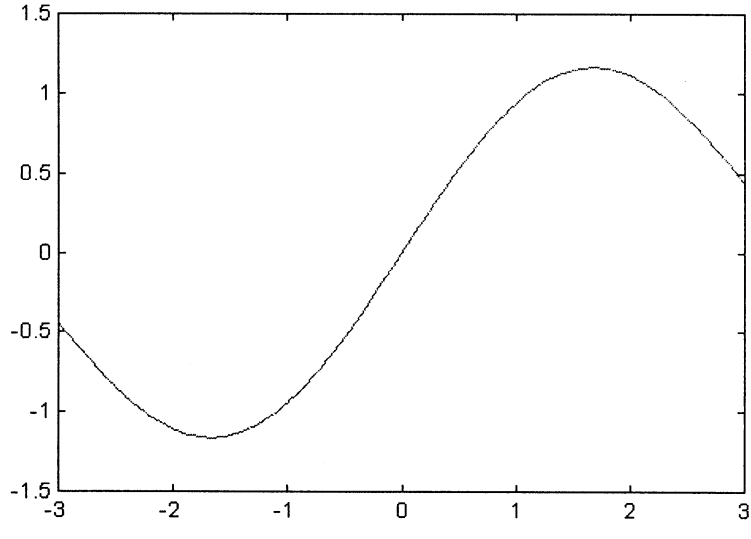
Obliczenia zakończono po sześciu iteracjach. Dla każdej iteracji wydrukowana została informacja o bieżącej wartości szukanego miejsca zerowego, wartość funkcji w znalezionym punkcie oraz krótki komentarz do stosowanej procedury obliczeniowej.

Ćwiczenie 5.2

Dla funkcji jednej zmiennej $f(x)=\sin(x)+x/10$:

a) narysuj jej wykres,

```
» f1=inline('sin(x)+x/10');
» fplot(f1,[-3,3])
```



Rysunek 5.1. Wykres funkcji $f(x)=\sin(x)+x/10$

b) znajdź minimum funkcji w przedziale <-3,3>.

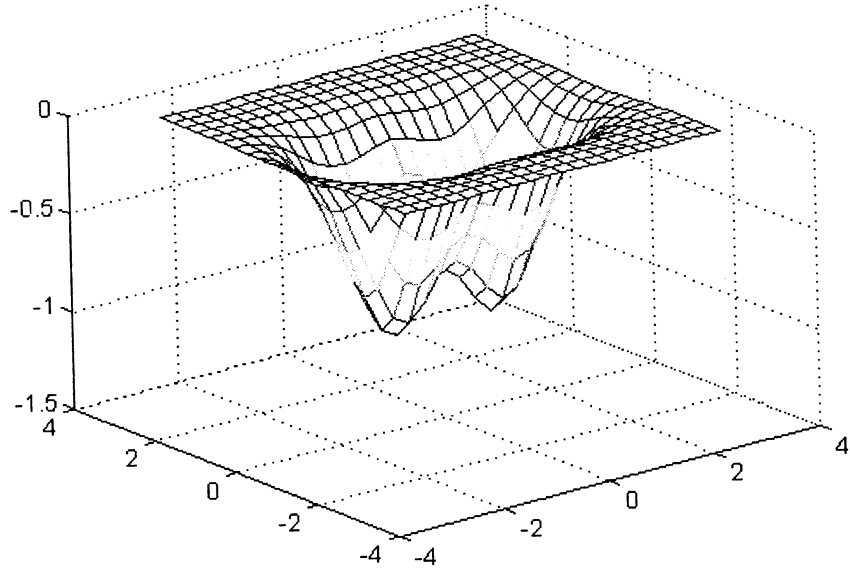
```
» x1=fminbnd(f1,-3,3), f1(x1)
x1 =
   -1.6710
min =
   -1.1621
```

Znalezione zostało minimum funkcji o wartości –1.1621 w punkcie $x1$=-1.6710.

Ćwiczenie 5.3

Na rysunku 5.2 przedstawiony został wykres funkcji dwóch zmiennych

$f(x,y)=-e^{-(x-1)^2-y^2}-e^{-(x+1)^2-y^2}$.



Rysunek 5.2. Wykres funkcji $f(x,y) = -e^{-(x-1)^2-y^2} - e^{-(x+1)^2-y^2}$

Zdefiniuj funkcję **inline** oraz znajdź jej minimum w otoczeniu punktu (1,1).

```
»   f2=inline('-exp(-(x(1)-1).^2 -x(2).^2)-
»                    exp(-(x(1)+1).^2 -x(2).^2)');
»   x0=fminsearch(f2,[1 1]), min=f2(x0)
Optimization terminated successfully:
 the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
 and F(X) satisfies the convergence criteria using
OPTIONS.TolFun of 1.000000e-004
x0 =
    0.9575   -0.0000
min =
   -1.0199
```

Znaleziono minimum lokalne o wartości -1.0199 w punkcie (0.9575, 0.0000)

Ćwiczenie 5.4

Zdefiniuj funkcję z ćwiczenia 5.3 w pliku oraz znajdź jej minimum w otoczeniu punktu (–1,–1).

Plik z definicją funkcji **fp.m** może wyglądać następująco:

```
function z=fp(x)
z=-exp(-(x(1)-1).^2-x(2).^2)-exp(-(x(1)+1).^2-x(2).^2);
```

Minimum wyznaczymy podobnie jak z ćwiczeniu 5.3, podając jako pierwszy argument polecenia łańcuch znaków określający funkcję z pliku **fp.m**, czyli:

```
»   x0=fminsearch('fp',[-1 -1]), min=fp(x0)
Optimization terminated successfully:
  the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
   and F(X) satisfies the convergence criteria using
OPTIONS.TolFun of 1.000000e-004
x0 =
   -0.9575    0.0000
min =   -1.0199
```

Ćwiczenie 5.5

Zdefiniuj funkcję $f(x)=\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6$. Ponadto:

a) narysuj jej wykres w przedziale <-20,20>;
b) oblicz wektor wartości funkcji w punktach -2,-1, 2;
c) znajdź minimum funkcji w przedziale <3,4>;
d) znajdź miejsca zerowe funkcji w otoczeniu punktów 3 i 4.

Na początek definiujemy funkcję o nazwie **fx** i zapisujemy ją w pliku **fx.m**:

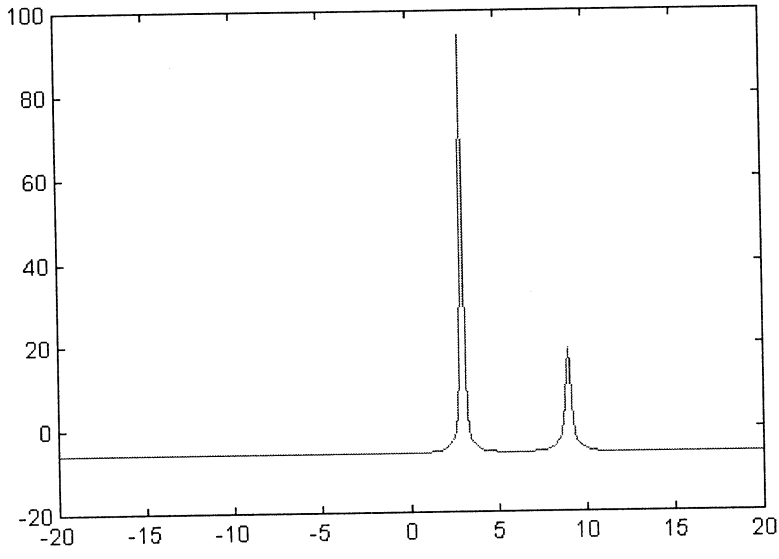
```
function y=fx(x)
y=1./((x-3).^2+0.01)+1./((x-9).^2+.04)-6;
```

Natomiast skrypt będzie miał postać:

```
x= -20:0.1:20;                          % rozwiązanie punktu a
plot(x,fx(x))
x=[-2 1 2];                             % rozwiązanie punktu b
y=fx(x)
x0=fminbnd ('fx',3,4), min=fx(x0)       % rozwiązanie punktu c
z1=fzero('fx',3)                        % rozwiązanie punktu d
z2=fzero('fx',4)
```

Efektem działania skryptu będzie wykres (rys. 5.3) oraz wartości **y**, *x0*, *min*, *z1*, *z2*.

```
y =   -5.9518   -5.7350   -4.9895
x0 =    3.9999
min =   -4.9699
Zero found in the interval: [2.52, 3.3394].
z1 =    2.6033
Zero found in the interval: [3.36, 4.4525].
z2 =    3.3969
```



Rysunek 5.3. Wykres funkcji $f(x)=\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6$

5.1.2. Wyznaczanie pierwiastków wielomianów

Operacje na wielomianach umożliwiają funkcje zestawione w tabeli 5.3. W funkcjach tych przyjmuje się, że wielomian n-tego stopnia ma ogólną postać:

$$W(x,a)=a_1x^n+a_2x^{n-1}+...+a_nx+a_{n+1}$$

gdzie $a_1, a_2,..., a_{n+1}$ – współczynniki wielomianu, uszeregowane według malejących potęg zmiennej x.

Tabela 5.3. Funkcje wyznaczające pierwiastki wielomianów

Funkcja	Opis
a=poly(r)	zwraca wektor **a** współczynników $a_1, a_2,..., a_{n+1}$ wielomianu $W(x,a)$ o pierwiastkach podanych w postaci wektora **r**=[$r_1, r_2,..., r_n$]
p=polyval(a,x0)	zwraca wartości wielomianu $W(x,a)$ w punkcie $x=x0$; współczynniki wielomianu określa wektor **a**, przy czym muszą być one uszeregowane w kolejności od najbardziej znaczącego (a_1) do wyrazu wolnego (a_{n+1}); jeśli **x0** jest wektorem (macierzą), wartości wielomianu obliczane są dla wszystkich elementów wektora **x0**
r=roots(a)	zwraca wektor **r** pierwiastków wielomianu $W(x,a)$; **a** – wektor współczynników wielomianu $a_1, a_2,..., a_{n+1}$.

Ćwiczenie 5.6

Wyznacz pierwiastki wielomianów:

a) $W(x) = x^2 + 3x - 4$

```
» roots([1 3 -4])
ans =
    -4
     1
```

b) $W(x) = 3x^5 - 2x^4 + 5x^2 + 2x - 7$

```
» a=[3 -2 0 5 2 -7]; r=roots(a)
r =
   0.7987 + 1.1879i
   0.7987 - 1.1879i
  -0.9372 + 0.5729i
  -0.9372 - 0.5729i
   0.9437
```

W przypadku a) otrzymamy dwa pierwiastki rzeczywiste, w przypadku b) – jeden pierwiastek rzeczywisty i cztery zespolone.

5.2. Metody numeryczne algebry liniowej

5.2.1. Własności macierzy

W praktyce numerycznej istotną sprawą jest określenie takich wielkości, jak rząd, wyznacznik, norma czy też wartości własne macierzy [2]. Ze względu na dość niekorzystne własności numeryczne ogólnego zadania wyznaczania wartości i wektorów własnych, funkcje przedstawione w tabeli 5.4 nie są niezawodne.

Tabela 5.4. Funkcje wyznaczające wielkości charakteryzujące macierze

Funkcja	Opis
rank(A)	oblicza rząd macierzy **A**
det(A)	oblicza wyznacznik macierzy kwadratowej **A**
norm(A)	oblicza normę macierzy **A**
cond(A)	oblicza liczbę warunkową macierzy **A**
L=eig(A)	zwraca wektor **L**, zawierający wartości własne macierzy kwadratowej **A**
[V, D] = eig(A)	wyznacza macierz **D**, zawierającą na przekątnej wartości własne macierzy **A**, oraz macierz **V** wektorów własnych odpowiadających tym wartościom (zachodzi zależność **A·V=V·D**).

Ćwiczenie 5.7

Oblicz rząd, wyznacznik i liczbę warunkową macierzy $\mathbf{A}=\begin{bmatrix}1 & 1 & 1\\ 1 & 2 & 3\\ 1 & -1 & 1\end{bmatrix}$.

```
» A=[1 1 1; 1 2 3; 1 -1 1];
» wyznacznik=det(A), rzad=rank(A), liczba_war=cond(A)
wyznacznik =
             4
rzad =
       3
liczba_war =
             6.9152
```

5.2.2. Układy równań liniowych

Podstawowym zagadnieniem liniowej algebry numerycznej są układy równań liniowych. Ich efektywne rozwiązywanie umożliwia metoda oparta na eliminacji Gaussa oraz rozkładzie LU [2]. W Matlabie metody te realizują funkcje przedstawione w tabeli 5.5.

Tabela 5.5. Funkcje rozwiązujące układy równań liniowych

Funkcja	Opis
[L,U]=lu(A)	dokonuje rozkładu LU macierzy **A**, tzn. znajduje macierze **L** i **U** takie, że **A=L·U**, przy czym **L** – macierz trójkątna dolna, **U**-macierz trójkątna górna
x=inv (A)*b lub **x=A\b**	rozwiązuje układ równań **A·x=b** (**b** musi być wektorem kolumnowym)
x=b/A	rozwiązuje układ równań **x·A=b** (**b** musi być wektorem wierszowym)

Ćwiczenie 5.8

Dokonaj rozkładu LU macierzy **A** zdefiniowanej z ćwiczeniu 5.7. Sprawdź, czy dla znalezionych macierzy **L** i **U** spełniona jest równość **A=L·U**.

```
» [L,U]=lu(A)
L =
    1.0000         0         0
    1.0000   -0.5000    1.0000
    1.0000    1.0000         0
U =
     1     1     1
     0    -2     0
```

```
         0     0     2
»   L*U
ans =
         1     1     1
         1     2     3
         1    -1     1
```

Ćwiczenie 5.9

Napisz skrypt rozwiązujący układ równań liniowych:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$$

```
A=[1 1 1; 1 2 3; 1 -1 1];
b=[3 1 2]'; % transpozycja wektora na kolumnowy
x=A\b;              % lub x=inv(A)*b;
disp('Rozwiązanie:')
disp(x)
```

Efektem działania skryptu będzie rozwiązanie podane w postaci wektora kolumnowego $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$.

```
Rozwiazanie:
    3.7500
    0.5000
   -1.2500
```

Ćwiczenie 5.10

Napisz skrypt rozwiązujący układ równań liniowych dla danych wpisywanych z klawiatury.

```
clear all;
disp(' Rozwiązywanie układu równań postaci Ax=b')
disp(' gdzie A jest macierzą m x n, x szukanym rozwiązaniem')
disp(' b - wektorem wyrazów wolnych')
disp(' ')
disp(' Podaj rozmiary macierzy A :')
m=input('Podaj liczbę wierszy macierzy A: m=')
n=input('Podaj liczbę kolumn macierzy A: n=')
disp(' Podaj elementy macierzy A:')
for i = 1:m
```

```
   for j = 1:n
      a(i,j) = input('podaj kolejny element:')
   end
end
disp(' Podaj elementy wektora b:')
for j = 1:m
   b(j) = input('podaj kolejny element:')
end
x=a\b' % wektor b jest wierszowy - konieczna jest transpozycja
```

5.3. Interpolacja i aproksymacja

Standardowe procedury Matlaba realizują interpolację za pomocą wielomianów pierwszego i trzeciego stopnia oraz funkcji sklejanych stopnia trzeciego. Wybrane polecenia do interpolacji i aproksymacji zestawiono w tabeli 5.6.

Tabela 5.6. Funkcje interpolujące

Funkcja	Opis
yi=interp1(x,y,xi,metoda)	zwraca wektor **yi**, będący wartościami funkcji jednej zmiennej $y=f(x)$ w punktach określonych wektorem **xi**; węzły interpolacji (punkty, w których znane są wartości funkcji) określają wektory **x** i **y**; *metoda* – łańcuch znaków określający metodę interpolacji
zi=interp2(x,y,z,xi,yi,metoda)	zwraca macierz **zi**, zawierającą wartości funkcji dwu zmiennych $z=f(x,y)$ w punktach określonych wektorami **xi** i **yi**; węzły interpolacji określają macierze **x, y** i **z**
vi=interp3(x,y,z,v,xi,yi,zi,metoda)	interpolacja funkcją trzech zmiennych, analogicznie jak **interp2**
vi=interpn(x1,x2,x3,...,v,y1,y2,y3,...)	interpolacja funkcją *n* zmiennych, analogicznie jak **interp2**

Interpolację można przeprowadzić za pomocą metod m.in.:

- *'linear'* – interpolacja liniowa,
- *'spline'* – interpolacja funkcjami sklejanymi stopnia trzeciego,
- *'cubic'* – interpolacja wielomianami stopnia trzeciego.

Wszystkie metody interpolacji wymagają, aby ciąg **x** był monotoniczny (rosnący lub malejący). Jeżeli dodatkowo jest to ciąg arytmetyczny, to w celu przyspieszenia poszukiwań można stosować metody: '**linear*', '**cubic*' lub '**spline*'.

Standardową metodą aproksymacji w Matlabie jest aproksymacja średniokwadratowa wielomianami wybranego stopnia. Polecenie:

a=polyfit(x,y,n)

znajduje wektor **a** współczynników wielomianu stopnia *n* najlepiej dopasowanego (w sensie aproksymacji średniokwadratowej [2]) do danych wektorów **x**, **y**. Wartość wielomianu aproksymującego w dowolnym punkcie *x0* (wektorze **x0**) można wyznaczyć, korzystając z polecenia **polyval(a,x0)** omówionego w rozdziale 5.1.

Ćwiczenie 5.11

Przyjmując dane z tabeli:

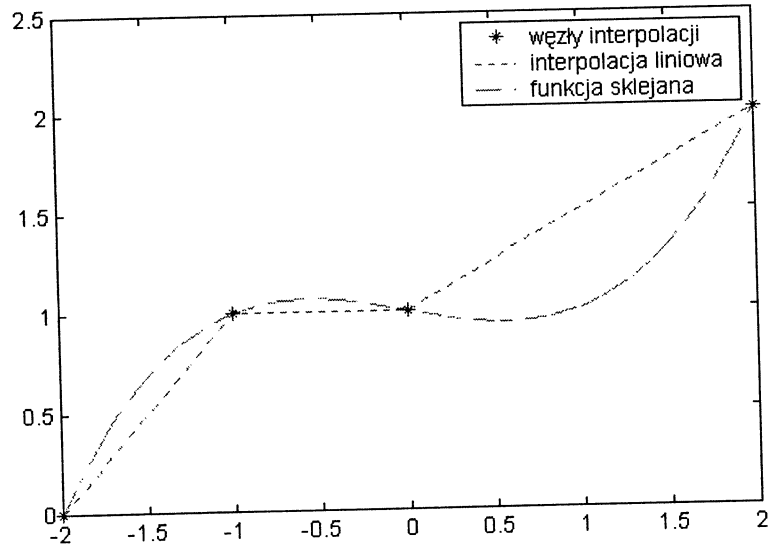
x	-2	-1	0	2
y	0	1	1	2

znajdź wartości funkcji interpolującej w punktach –2, -1.9, -1.8, ... , 0, ..., 1.8, 1.9, 2. Zastosuj interpolację liniową i funkcjami sklejanymi. Dane oraz wyniki wyświetl na jednym rysunku. Znajdź wartość funkcji interpolującej w punkcie 1.85 dla obu metod.

Skrypt realizujący powyższe zadanie może wyglądać następująco:

```
clear all   %wyczyszczenie pamięci
%określenie węzłów interpolacji:
x=[-2 -1 0 2];
y=[ 0 1 1 2];
%określenie wektora xi, dla którego poszukujemy
%wartości funkcji yi=f(xi):
xi=-2:0.1:2;
%wykonanie interpolacji liniowej:
yi_lin=interp1(x,y,xi,'linear');
%interpolacja funkcjami sklejanymi:
yi_spline=interp1(x,y,xi,'spline');
% wykresy danych (x,y) i efektów interpolacji (xi, yi_lin)
% oraz (xi,yi_spline)
plot(x,y,'*',xi,yi_lin,':',xi,yi_spline,'--')
axis([-2 2 0 2.5])      %zwiększenie zakresu osi y
                        %aby rysunku nie zasłoniła legenda
legend('węzły interpolacji','interpolacja liniowa',...
       'funkcja sklejana')
```

Efekt wykonania skryptu obrazuje rysunek 5.4.



Rysunek 5.4. Wyniki interpolacji dla danych z ćwiczenia 5.11

W celu znalezienia wartości jedynie w punkcie *xi*=1.85 należy wykonać powyższy skrypt, przyjmując *xi*=1.85 oraz wyświetlić wartość *yi*.

W wyniku takich modyfikacji skryptu otrzymamy szukane wartości:

```
yi_lin =
    1.9250
yi_spline =
    1.7469
```

Ćwiczenie 5.12

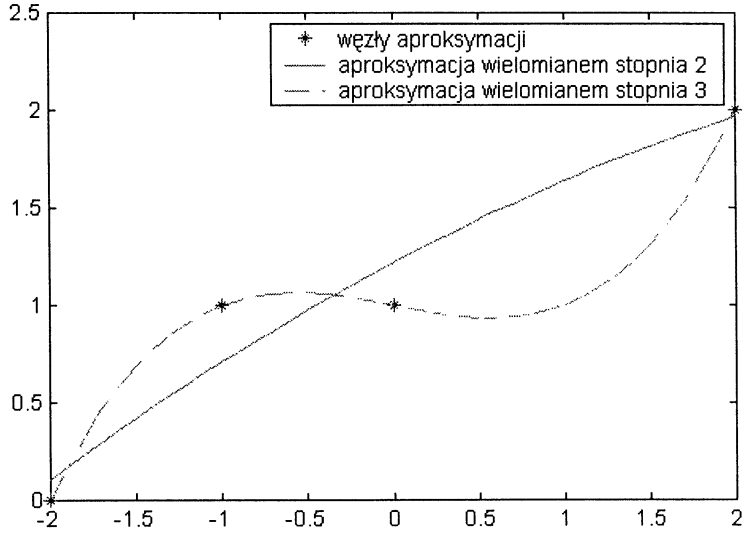
Przyjmując dane *x*, *y* z ćwiczenia 5.11, znajdź współczynniki wielomianu aproksymującego stopnia drugiego i trzeciego. Wyniki wyświetl na jednym rysunku.

Skrypt realizujący powyższe zadanie może wyglądać następująco:

```
clear all
%określenie węzłów aproksymacji:
x=[-2 -1 0 2];
y=[ 0 1 1 2];
% obliczenie współczynników wielomianu apr. stopnia 2:
a2=polyfit(x,y,2);
% obliczenie współczynników wielomianu apr. stopnia 3:
a3=polyfit(x,y,3);
%przygotowanie pomocniczego wektora wartości x
xi=-2:0.1:2;
% obliczenie wartości wielomianów w punktach xi
```

```
y2=polyval(a2,xi);
y3=polyval(a3,xi);
% wykres węzłów i efektów aproksymacji
plot(x,y,'*',xi,y2,'-',xi,y3,'--')
axis([-2 2 0 2.5])          % jak z ćwiczeniu5.11
legend('węzły aproksymacji',...
       'aproksymacja wielomianem stopnia 2',...
       'aproksymacja wielomianem stopnia 3')
```

Efekt wykonania skryptu obrazuje rysunek 5.5.



Rysunek 5.5. Wyniki aproksymacji dla danych z ćwiczenia 5.11

5.4. Całkowanie numeryczne

Funkcje Matlaba realizujące całkowanie numeryczne [2] zestawiono w tabeli 5.7.

Tabela 5.7. Funkcje realizujące całkowanie

Funkcja	Opis
Q=quad(f,a,b) **Q=quad(f,a,b,tol)** **Q=quad(f,a,b,tol,tr)**	oblicza wyrażenie $\int_a^b f(x)dx$ za pomocą prostej kwadratury Simpsona

Funkcja	Opis
Q=quad8(f,a,b,tol,tr)	oblicza wyrażenie $\int_a^b f(x)dx$ za pomocą złożonej kwadratury Newtona-Cotesa (rzędu 8); daje lepsze rezultaty niż prosta kwadratura **quad**

Parametrami metod są:

- *f* – łańcuch znaków określający funkcję Matlaba: nazwę funkcji (standardowej lub zapisanej w m-pliku) albo definicję funkcji **inline**,
- *a*, *b* – przedział całkowania,
- *tol* – wymagana względna tolerancja błędu (domyślnie, jeśli brak parametru: 10^{-3}),
- *tr* – jeśli parametr jest różny od 0, to dodatkowo kreślony jest wykres obrazujący postęp w obliczeniach.

Ćwiczenie 5.13

Oblicz całki:

a) $\int_1^{10} \sin(x)dx$,

```
»   quad('sin',1,10)
ans =
    1.3794
```

b) $\int_3^4 \left[\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6\right]dx$

Funkcja podcałkowa została zdefiniowana z ćwiczeniu5.5, w pliku **fx.m**, stąd:

```
»   format long
»   quad('fx',3,4)
ans =
   8.74457774894817
»   quad8('fx',3,4)
ans =
   8.74454500297259
```

Zwiększamy dokładność obliczeń, dodając parametr $tol=0.1\cdot10^{-10}$, i obserwujemy postępy obliczeń, dodając parametr $tr\neq0$.

```
»   quad8('fx',3,4,0.1e-10,1)
ans =
   8.74456519926256
```

5.5. Układy równań różniczkowych zwyczajnych

5.5.1. Wprowadzenie

Dowolne równanie różniczkowe zwyczajne rzędu n

$$\frac{d^n y}{dt^n} = f\left(t, y, \frac{dy}{dt}, ..., \frac{d^{n-1} y}{dt^{n-1}}\right) \tag{5.1}$$

z warunkami początkowymi:

$$y(t_0) = y_0, \ \frac{dy}{dt}(t_0) = y_1, ..., \frac{d^n y}{dt^n}(t_0) = y_n \tag{5.2}$$

można za pomocą metody ***zmiennych stanu*** przedstawić jako układ n równań różniczkowych rzędu pierwszego. Dla układu równań różniczkowych n-tego rzędu można określić n zmiennych stanu y_1, y_2, ..., y_n. Wyboru zmiennych stanu można w zasadzie dokonywać dowolnie. Najczęściej dobiera się je tak, aby każda kolejna zmienna była pochodną po czasie poprzedniej [1,4]:

$$\begin{cases} y_1 = y \\ y_2 = \dfrac{dy_1}{dt} = \dfrac{dy}{dt} \\ y_3 = \dfrac{dy_2}{dt} = \dfrac{d^2 y}{dt^2} \\ ... \\ y_n = \dfrac{dy_{n-1}}{dt} = \dfrac{d^{n-1} y}{dt^{n-1}} \end{cases} \tag{5.3}$$

Równanie (5.3) można inaczej zapisać jako:

$$\begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = y_3 \\ ... \\ \dfrac{dy_n}{dt} = f(t, y_1, ... y_n) \end{cases} \tag{5.4}$$

z warunkami początkowymi:

$$y_1(t_0) = y_{1,0}, \ y_2(t_0) = y_{2,0}, ..., y_n(t_0) = y_{n,0}. \tag{5.5}$$

Zagadnienie powyższe można przedstawić w postaci wektorowej:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ \dots \\ f(t, y_1, \dots, y_n) \end{bmatrix} \tag{5.6}$$

z wektorem warunków początkowych:

$$\begin{bmatrix} y_1(t_0) \\ \dots \\ y_n(t_0) \end{bmatrix} = \begin{bmatrix} y_{1,0} \\ \dots \\ y_{n,0} \end{bmatrix} \tag{5.7}$$

Równanie (5.1) sprowadza się w ten sposób do *n* równań postaci:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0. \tag{5.8}$$

Metody numerycznego rozwiązywania układów równań różniczkowych polegają na poszukiwaniu rozwiązania, startując z punktu, w którym jest ono znane. Jeśli znamy rozwiązanie równania (5.8) w punkcie t_i, to rozwiązanie w kolejnym kroku t_{i+1} można zaproksymować jako [6]:

$$y(t_{i+1}) = y(t_i)+h\Delta(t_i, y(t_i), h, f) \tag{5.9}$$

gdzie:

h – krok całkowania; $h=t_{i+1}-t_i$

Δ – pewna funkcja

Ze względu na sposób rozwiązania równania (5.9) wyróżnia się metody:

- ***jednokrokowe*** – do wyznaczenia wartości $y(t_{i+1})$ wystarczy znajomość tylko poprzedniej wartości $y(t_i)$; metody takie wymagają wielu małych kroków i nie zawsze dają dobre wyniki;
- ***wielokrokowe*** – w każdym kroku funkcja obliczana jest w kilku punktach przedziału;
- ***o zmiennym kroku całkowania*** – w każdym kroku sprawdza się dokładność rozwiązania i jeśli błąd jest zbyt duży, obliczenia są powtarzane dla zmniejszonego kroku.

Układy równań różniczkowych mogą być ***sztywne*** (źle uwarunkowane). Dzieje się tak, jeśli macierz Jacobiego obliczona w aktualnym punkcie pracy ma wartości własne, różniące się o kilka rzędów wielkości. W rozwiązaniu takiego układu występują jednocześnie bardzo duże i bardzo małe stałe czasowe.

Najczęściej stosowanymi metodami numerycznego rozwiązywania układów równań różniczkowych (metodami ***całkowania***) są: metoda Rungego-Kutty, Adamsa-

Bashfortha-Moultona, a w przypadku układów sztywnych – metoda Geara. Należy jednak podkreślić, że żadna metoda całkowania nie jest najlepsza do wszystkich rodzajów równań różniczkowych.

Układy fizyczne opisane za pomocą równań różniczkowych zwyczajnych (tzw. ***układy dynamiczne***) można spotkać w rozmaitych dziedzinach nauki (fizyka, chemia, biologia, elektrotechnika, mechanika, medycyna itp.). Specjalnie do rozwiązywania układów dynamicznych służy wspomniany we wstępie program Simulink.

5.5.2. Rozwiązywanie układów równań różniczkowych zwyczajnych w Matlabie

Rozwiązanie układu równań różniczkowych zwyczajnych realizują w Matlabie funkcje ODE (akronim ang. *ordinary differential equations*). Rozróżnia się grupy funkcji do rozwiązywania układów równań dobrze uwarunkowanych oraz źle uwarunkowanych (sztywnych).

Tabela 5.8. Funkcje ODE

Nazwa funkcji	Opis
Układy równań dobrze uwarunkowanych	
ode45	używa jednokrokowej metody Rungego-Kutty rzędu 4-5; najlepsza do szukania „pierwszego przybliżenia" rozwiązania
ode23	używa jednokrokowej metody Rungego-Kutty rzędu 2-3; mniej skuteczna od funkcji **ode45**
ode113	używa wielokrokowej metody Adamsa-Bashfortha-Moultona; najlepsza do rozwiązywania układów równań dobrze uwarunkowanych
Układy równań źle warunkowanych (sztywnych)	
ode15s	używa wielokrokowej metody Geara, nie zawsze efektywnej; zalecana, jeśli funkcja **ode45** nie dała zadowalających rezultatów i podejrzewa się, że zagadnienie jest sztywne
ode23s	używa zmodyfikowanej metody Rosenbrocka rzędu 2 (jednokrokowej); zalecana w sytuacjach, w których nie sprawdza się funkcja **ode15s**
ode23t	implementacja reguły trapezów; zalecana do zagadnień lekko sztywnych
ode23tb	implementacja metody TR-BDF2; użycie zalecane jak w wypadku funkcji **ode23s**

Sposób wywołania każdej z funkcji ODE jest jednakowy:

[T, Y]=funkcja_ODE(plik_ODE, przedział, y0)

Argumentami funkcji ODE są:

- *plik_ODE* – łańcuch znaków, zawierający nazwę specjalnego m-pliku z definicją układu równań różniczkowych;
- **przedział** – wektor określający przedział czasu całkowania; w przypadku dwuelementowego wektora [*t0*, *tk*] całkowanie będzie wykonywane od chwili czasu *t0* do *tk*; w przypadku wektora **przedział** o liczbie elementów większej od 2 rozwiązanie jest zwracane w chwilach czasu określonych w wektorze;
- **x0** – wektor kolumnowy określający warunki początkowe danego zagadnienia.

Funkcja ODE zwraca w najprostszym przypadku dwa parametry wyjściowe:

- **T** – wektor kolumnowy z wartościami czasu, dla których obliczane było rozwiązanie;
- **Y** – macierz rozwiązań; każda kolumna macierzy jest wektorem reprezentującym wartości jednej ze zmiennych stanu w punktach określonych wektorem **T**.

Plik ODE musi zawierać funkcję Matlaba o dowolnej nazwie i dwóch argumentach:

- *t* – skalar,
- **y** – kolumnowy wektor zmiennych stanu.

W ciele funkcji należy zdefiniować układ równań różniczkowych zwyczajnych pierwszego stopnia, przyporządkowując parametrowi wyjściowemu **F** wektor kolumnowy, którego pierwszy element odpowiada pochodnej po czasie pierwszej zmiennej stanu, drugi – pochodnej po czasie drugiej zmiennej stanu itd.

Plik ODE musi więc mieć następującą postać:

```
function F = plik_ODE(t,y)
%ewentualne komentarze i definicje
F=...       %definicja układu równań różniczkowych
```

Ćwiczenie 5.14

Rozwiąż równanie różniczkowe rzędu drugiego: $\frac{d^2y}{dt^2} + y = 0$ z warunkami początkowymi: y(0)=0, $\frac{dy}{dt}(0) = 1$. Przyjmij czas całkowania od 0 do 10 s.

Równanie to można przedstawić w postaci:

$$\frac{d^2y}{dt^2} = -y$$

Wprowadzając następujące zmienne stanu:

$$\begin{cases} y_1 = y \\ y_2 = \dfrac{dy_1}{dt} = \dfrac{dy}{dt} \end{cases}$$

analizowane równanie można sprowadzić do układu równań różniczkowych zwyczajnych pierwszego rzędu:

$$\begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = -y_1 \end{cases}$$

co w postaci wektorowej zapisuje się następująco:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ -y_1 \end{bmatrix}$$

Uzyskany układ równań można teraz zapisać w pliku ODE o nazwie np. **rownanie.m**:

```
function F = rownanie(t,y)
%rozwiązuje równanie y''+y=0
F=[y(2); -y(1)];
```

Do rozwiązania problemu można użyć funkcji **ode45**, jako mającej najbardziej wszechstronne zastosowanie. Ponieważ wektor warunków początkowych ma postać:

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} y(0) \\ \dfrac{dy}{dt}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

funkcję ODE należy wywołać następująco:

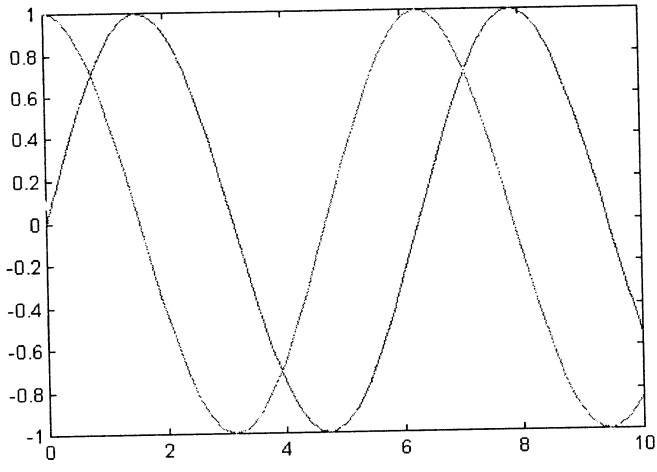
```
» [T,Y]=ode45('rownanie',[0 10],[0; 1])
```

Wynikiem działania funkcji jest macierz **Y**, której pierwsza kolumna zawiera wartości zmiennej stanu y_1, a druga – wartości zmiennej stanu y_2, obliczone dla chwil czasu zapisanych w wektorze **T**. Polecenie:

```
» plot(T,Y)
```

spowoduje jednoczesne narysowanie wykresów obu zmiennych stanu – $y_1=y(t)$ oraz $y_2=\dfrac{dy}{dt}(t)$, jak na rysunku 5.6.

Można wykazać, że matematycznym rozwiązaniem omawianego zagadnienia jest funkcja $y(t)=A\cos(t)+B\sin(t)$, gdzie A i B – stałe zależne od wartości warunków początkowych.



Rysunek 5.6. Rozwiązanie równania różniczkowego $\frac{d^2y}{dt^2} + y = 0$

Ćwiczenie 5.15

Klasyczny model *predator–prey* (łowca–ofiara), który opisuje zachowanie dwóch gatunków zwierząt żyjących w tej samej niszy ekologicznej, można opisać równaniem Lotki-Volterry [1]:

$$\begin{cases} \frac{dN_1}{dt} = aN_1 - bN_1N_2 \\ \frac{dN_2}{dt} = -cN_2 + dN_1N_2 \end{cases}$$

gdzie N_1 – ofiary (zwierzęta roślinożerne), N_2 – drapieżniki (zwierzęta mięsożerne), a – współczynnik przyrostu liczby zwierząt roślinożernych przy braku drapieżników, c – współczynnik spadku liczby zwierząt mięsożernych przy braku ofiar, b i d – współczynniki określające wzajemne oddziaływanie między gatunkami. Przeanalizuj zachowanie systemu, jeśli a=3,5, b=0,1, c=4, d=0,06, a warunki początkowe $N_1(0)$=100, $N_2(0)$=20.

Ponieważ zagadnienie zostało opisane dwoma równaniami różniczkowymi stopnia pierwszego, najprościej będzie przyjąć za zmienne stanu N_1 i N_2:

$$\begin{cases} y_1 = N_1 \\ y_2 = N_2 \end{cases}$$

skąd:

$$\frac{d}{dt}\begin{bmatrix} N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} ay_1 - by_1y_2 \\ -cy_2 + dy_1y_2 \end{bmatrix}$$

Uzyskany układ równań został zapisany w pliku ODE **pred** o treści:

```
function wynik = pred(t,x)
%oblicza model predator-prey
a=3.5; b=0.1; c=4; d=0.06;
x12=x(1).*x(2);
wynik=[a*x(1)-b*x12; -c*x(2)+d*x12];
```

Przyjmując wektor warunków początkowych:

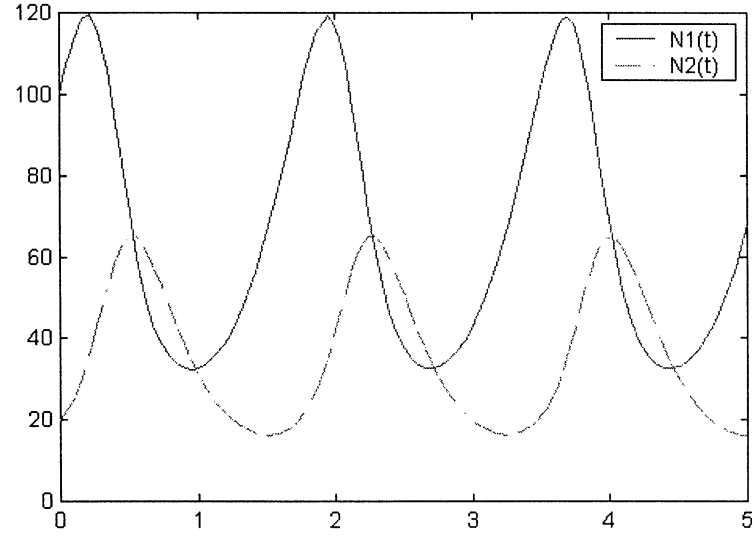
$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 100 \\ 20 \end{bmatrix}$$

funkcję ODE można wywołać np.:

```
» [T,Y]=ode45('pred', [0 5], [100; 20]);
```

Załóżmy, że chcemy umieścić wykresy $N_1(t)$ i $N_2(t)$ na jednym rysunku, przy czym zwierzęta roślinożerne (zmienna N_1) będą zaznaczone linią ciągłą, a drapieżniki (zmienna N_2) – linią przerywaną. W poleceniu **plot** trzeba oddzielić dyrektywy dla każdej ze zmiennych stanu (każdej kolumny macierzy **Y**). Do wykresu można dodać legendę.

```
» plot(T,Y(:,1),'-',T,Y(:,2),'--')
» legend('N1(t)','N2(t)')
```



Rysunek 5.7. Wynik symulacji dla modelu *predator-prey*

Można teraz przeanalizować uzyskany rysunek i zastanowić się, jak zmiana liczby zwierząt roślinożernych wpływa na liczbę zwierząt mięsożernych i odwrotnie.

Ćwiczenie 5.16

Napisz skrypt umożliwiający analizę ćwiczenia 5.16 dla warunków początkowych wpisywanych z klawiatury.

```
%Skrypt uruchamia funkcję pred z modelem predator-prey
clear all          %Usunięcie wszystkich zmiennych
%Wczytanie warunków początkowych
n10=input('N1(0)=');
n20=input('N2(0)=');
%Uruchomienie obliczeń i wyświetlenie wyników
[T,Y]=ode45('pred', [0 5], [n10; n20]);
plot(T,Y(:,1),'-',T,Y(:,2),'--')
legend('N1(t)','N2(t)')
```

Ćwiczenie 5.17

Równanie Duffinga dane jest wzorem:

$$\frac{d^2\phi}{dt^2} + \omega_0\phi + h\phi^3 = F_m \cos\omega_1 t$$

Rozwiąż równanie, przyjmując F_m=0.3, ω_1=1, ω_0=-1, h=2 oraz zerowe warunki początkowe. Całkowanie przeprowadź w czasie 40 s.

Do równania wprowadzono zmienne stanu:

$$\begin{cases} y_1 = \phi \\ y_2 = \dfrac{d\phi}{dt} \end{cases}$$

uzyskując układ równań:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ F_m \cos(\omega_1 t) - \omega_0 y_1 - h y_1^3 \end{bmatrix}$$

Wektor warunków początkowych ma natomiast postać:

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

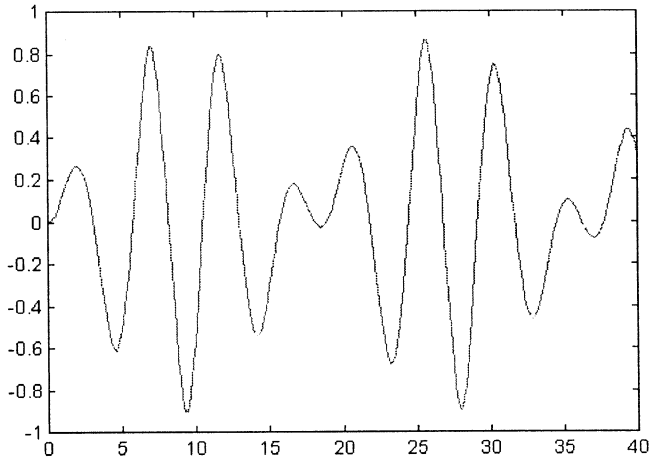
Otrzymany układ równań został zapisany w pliku ODE **duffing**. Tym razem wszystkie parametry rozwiązywanego równania będą określane w skrypcie, dlatego muszą być globalne:

```
function f=duffing(t,y)
%Oblicza rownanie Duffinga
global Fm w1 w0 h
f=[y(2); Fm*cos(w1*t)-w0*y(1)-h*y(1).^3];
```

A oto skrypt wywołujący funkcję **duffing**:

```
clear all
global Fm w1 w0 h
Fm=0.3; w1=1; w0=-1; h=2;
tp=0; tk=40; %określenie początku i końca przedziału symulacji
[t,x]=ode45('duffing', [tp tk], [0; 0]);
plot(t,x(:,1))      %wykres pierwszej zmiennej stanu - fi(t)
```

Równanie Duffinga opisuje zjawiska zachodzące w obwodzie składającym się z szeregowo połączonej idealnej cewki nieliniowej i kondensatora, zasilanych napięciem zmiennym sinusoidalnie. Drgania wymuszone powstające w takim obwodzie widoczne są na wykresie (rys. 5.8).



Rysunek 5.8. Wynik symulacji dla modelu opisanego równaniem Duffinga

5.6. Zadania do samodzielnego rozwiązania

Zadanie 5.1

Wyznacz miejsce zerowe funkcji sin(x), startując z punktu x=3.

Zadanie 5.2

Wyznacz pierwiastki wielomianów:

a) $W(x) = x^2 + 3x - 4$

b) $W(x) = -2x^4 + 3x^2 + x - 6$

Zadanie 5.3

Rozwiąż układ równań:

$$\begin{cases} a+2b+3c+4d=1 \\ 3a+4b+5c+6d=2 \\ 2a+3b+5c=4 \\ 3a+5b+2c+d=6 \end{cases}$$

Zadanie 5.4

Oblicz rząd i wyznacznik macierzy z zadania 5.1.

Zadanie 5.5

Utwórz wektor wartości funkcji $y(x)=\cos(3x)+\cos(x)$ w punktach –10:0.5:10. Przeprowadź interpolację za pomocą trzech dostępnych metod. Na jednym wykresie przedstaw węzły interpolacji i trzy uzyskane przebiegi funkcji. Dla danych x, y sprawdź działanie funkcji **polyfit**, wykorzystując różne stopnie wielomianu aproksymującego.

Zadanie 5.6

Oblicz całki:

a) $\int_{-2}^{2}(x-1)^2\,dx$,

b) $\int_{-\pi/2}^{\pi/2}[x+2\sin(x)]dx$.

Zadanie 5.7

Rozwiąż równanie różniczkowe:

$$\frac{dy}{dt}=-\left(1+\sin^2(y)\right)y$$

przyjmuje różne wartości warunku początkowego $y(0)\neq 0$. Całkowanie przeprowadź w czasie 3 s.

Ponieważ równanie jest stopnia pierwszego, zarówno wektor zmiennych stanu, jak i wektor warunków początkowych będą jednoelementowe (innymi słowy – będą liczbami).

Bibliografia

[1] Cavallo A., Setola R., Vasca F., *Using Matlab, Simulink and Control System Toolbox*. 1996, Prentice Hall Europe.

[2] Dahlquist G., Björck A., *Metody numeryczne*. Warszawa 1983, PWN.

[3] Kernighan B.W., Ritchie D.M., *Język ANSI C*. Warszawa 1998, WNT.

[4] *Matlab. The Language of Technical Computing*. 1996, The Math Works Inc.

[5] Mrozek B., Mrozek Z., *Matlab 5.x, Simulink 2.x*. 1998, Wydawnictwo PLJ.

[6] Osowski S., *Modelowanie układów dynamicznych z zastosowaniem języka Simulink*. Warszawa 1997, Oficyna Wydawnicza Politechniki Warszawskiej.

[7] Zalewski A., Cegieła R., *Matlab – obliczenia numeryczne i ich zastosowania*. Poznań 1996, Wydawnictwo Nakom.

Skorowidz

E

F

G

H

I

J

K

L

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Wydawnictwo MIKOM

Prowadzimy wysyłkową sprzedaż naszych książek. Wystarczy podać listę zamawianych tytułów (katalog wysyłamy bezpłatnie). Zamówienie zostanie zrealizowane w ciągu kilku dni.

1. Książki oferujemy w naszych cenach detalicznych (niższych niż u innych dostawców). **Przy jednorazowym zamówieniu na kwotę przekraczającą 600 złotych są stosowane katalogowe ceny hurtowe.**
2. Odbiorca opłaca zryczałtowane koszty wysyłki.
3. Na życzenie wysyłamy wszystkim zainteresowanym aktualny katalog.

Aktualne ceny detaliczne naszych książek są podane na dołączonym do katalogu druku zamówieniu. Zamówienia można składać listownie, telefonicznie oraz za pośrednictwem Internetu. W Internecie można też zamówić informacje o nowościach.

Prosimy o zamówienia na adres firmy wysyłkowej

WITKOM

ul. Andrzejowska 3

02-312 Warszawa

Realizujemy zamówienia telefoniczne i internetowe

tel./fax: **(22) 823 70 77, 823 94 61**

Strona WWW: **http://www.mikom.com.pl**

http://www.mikom.pl

e-mail: **zamowienia@mikom.pl**

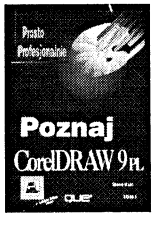
Wydawnictwo MIKOM

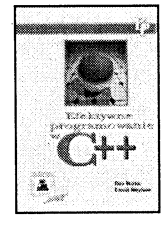
poleca swoje książki

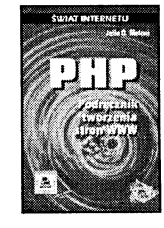
... z każdym bitem serca

Nasze serie wydawnicze:

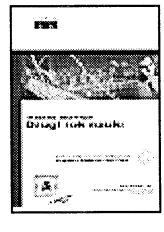
- „Poznaj" – obszerne książki na temat popularnych pakietów
- „W przykładach" – programowanie na konkretnych przykładach
- „Biblioteka programisty" – praktyka programowania
- „Świat Internetu" – różne aspekty Internetu i sieci

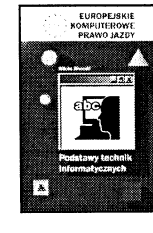






- „Non Stop" – podręczniki dla profesjonalistów
- „Cisco" – podręczniki o produktach firmy Cisco
- „Szkoła, Internet, Intranet" – Internet i sieci w edukacji
- „Europejskie Komputerowe Prawo Jazdy" – kurs podstawowy





- „Podręczniki" – przeznaczone głównie dla szkół i uczelni, ale nie tylko
- „Bazy danych" – wszystko o bazach danych i językach baz danych
- „Od A do Z" – proste poradniki encyklopedyczne
- „Ćwiczenia z ..." – ćwiczenia na niemal każdy temat