PROGRAMOWANIE
W MATLABIE

PODRĘCZNIK. STUDIA PRZYPADKÓW



Tom 16 Kompetencje, Wiedza, Innowacje – Zintegrowany Program Rozwoju WSEI w Lublinie – Etap III

ANNA TATARCZAK

16
TOM

PROGRAMOWANIE W MATLABIE

Część pierwsza
Podręcznik

Anna Tatarczak

Lublin 2021

WYŻSZA SZKOŁA EKONOMII I INNOWACJI W LUBLINIE

Seria wydawnicza:
Kompetencje, Wiedza, Innowacje – Zintegrowany Program Rozwoju WSEI w Lublinie – Etap III

Tom 16

PROGRAMOWANIE W MATLABIE
Cz. I – Podręcznik Cz. II – Studia przypadków

Autor:
Anna Tatarczak
ORCID: 0000-0001-8573-5791

Recenzenci:
dr inż. Paweł Lonkwic
dr Przemysław Adamkiewicz

Korekta:
Magdalena Grela-Tokarczyk

Skład, łamanie:
Marta Krysińska-Kudlak

Projekt okładki:
Patrycja Kaczmarek

Grafika na okładce:
rawpixel.com/Freepik.com
rawpixel.com/Freepik.com

@Copyright by
Innovatio Press, Lublin 2021
Wszelkie prawa zastrzeżone. W sprawie kopiowania lub rozpowszechniania fragmentów albo całości niniejszej pracy należy kontaktować się z wydawcą.

Publikacja oraz recenzje zrealizowano w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Proces wydawniczy (korekta, skład) sfinansowano z dotacji na zadania związane z zapewnieniem osobom niepełnosprawnym warunków do pełnego udziału w procesie kształcenia.

Printed in Poland
Innovatio Press Wydawnictwo Naukowe
Wyższej Szkoły Ekonomii i Innowacji
20-209 Lublin, ul. Projektowa 4
tel.: +48 81 749 17 77, fax: + 48 81 749 32 13
www.wsei.lublin.pl

ISBN – wersja elektroniczna: 978-83-66159-90-7

Spis treści

Wstęp 5

1. Wprowadzenie do Matlaba 6
   1.1. Co to jest Matlab? 7
   1.2. Interfejs Matlaba 8
   1.3. Pomoc programu 13
   1.4. Pliki 15
   1.5. Operacje na plikach 17
   1.6. Proste obliczenia 20
   1.7. Wartości liczbowe 27
   1.8. Operacje logiczne 31
   1.9. Typy danych 33
   1.10. Zadania 36

2. Wektory 39
   2.1. Tworzenie wektorów 40
   2.2. Odwoływanie się do elementów wektora 41
   2.3. Operacje na wektorach 42
   2.4. Zadania 45

3. Macierze 48
   3.1. Definiowanie macierzy 49
   3.2. Odwołania do elementów macierzy 55
   3.3. Operacje macierzowe 58
   3.4. Usuwanie elementów macierzy 64
   3.5. Macierze rzadkie 65
   3.6. Zadania 71

4. Tablice 74
4.1. Tablice numeryczne 75
4.2. Tablice znaków 80
4.3. Tablice struktur 84
4.4. Zadania 87

5. Programowanie 89
5.1. Instrukcja if 90
5.2. Instrukcja switch 92
5.3. Pętle 94
5.4. Instrukcja break/continue 98
5.5. Skrypty 99
5.6. Funkcje 102
5.7. Zadania 106

6. Wykresy i grafika 112
6.1. Wstęp 113
6.2. Grafika dwuwymiarowa 115
6.3. Grafika trójwymiarowa 137
6.4. Graficzny interfejs użytkownika 154
6.5. Zadania 159

Bibliografia 164

Wstęp

Podręcznik zawiera opis głównych zagadnień prezentowanych na wykładzie **Programowanie w Matlabie**. Z wykładem związane są ćwiczenia laboratoryjne. W ich trakcie praktycznie ilustrowane są zagadnienia przedstawiane na wykładzie. Podstawowym narzędziem programowym, stosowanym do opisu poszczególnych procedur obliczeniowych, jak i do obliczeń w laboratorium komputerowym jest Matlab. Program ten jest stosowany zarówno do formułowania i sprawdzania prostych algorytmów numerycznych, jak i do rozwiązywania bardziej złożonych zagadnień z wykorzystaniem gotowych procedur.

Sześć rozdziałów, z których składa się ten podręcznik, jest zarazem manualem oprogramowania Matlab i prezentacją wykorzystania algorytmów. Każdy z rozdziałów książki ma podobną konstrukcję. Podstawową konwencją przyjętą w książce jest minimum teorii, poparte przykładami, które pomogą zrozumieć omawiane zagadnienia. Ostatnią część każdego rozdziału stanowi zestaw zadań przewidzianych do samodzielnego rozwiązania.

W rozdziale pierwszym zostały omówione podstawowe elementy pakietu, niezbędne do zrozumienia dalszej części podręcznika. Po informacjach wprowadzających – rozdziały koncentrują się kolejno na wektorach, macierzach i tabelach. W rozdziale piątym zostały przedstawione podstawowe zasady pisania skryptów i funkcji. W ostatnim rozdziale przeanalizowano możliwości reprezentacji graficznej danych z użyciem oprogramowania. Dołączona bibliografia przedmiotu umożliwi zainteresowanym pogłębienie wiedzy oraz dotarcie do innych podręczników z zakresu omawianego oprogramowania.

1. Wprowadzenie do Matlaba



1.1. Co to jest Matlab?

Matlab to rozbudowane środowisko przeznaczone do obliczeń inżynierskich oraz tworzenia symulacji. Cechuje go duża liczba funkcjonalnych bibliotek oraz toolboxów, możliwości znacząco poszerzane są za pomocą funkcji pisanych przez użytkownika.

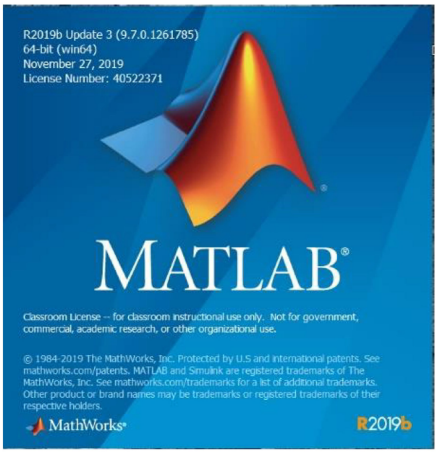
Matlab wykorzystywany jest przede wszystkim do:

- obliczeń matematycznych,
- algorytmów numerycznych,
- modelowania i symulacji,
- analizy danych,
- aplikacji z wykorzystaniem graficznych interfejsów użytkownika.

Matlab znajduje zastosowanie w następujących gałęziach przemysłu:

- telekomunikacja,
- elektronika,
- usługi finansowe,
- automatyzacja,
- przemysł maszynowy.

Zaletą Matlaba jest możliwość interaktywnej pracy z konsoli, gdzie można wydawać pojedyncze polecenia. Istnieje też możliwość tworzenia skryptów oraz dodawania nowych funkcji.



Rysunek 1. Logo Matlaba.



WAŻNE

Przed rozpoczęciem pracy z Matlabem należy pamiętać o podstawowych wskazówkach:

- **wprowadzając polecenia, pamiętaj, że Matlab rozróżnia wielkie i małe litery,**
- **tekst (łańcuchy) zapisuje się w apostrofach,**
- **średnik wpisany na końcu linijki powoduje, że Matlab nie wyświetla wyniku tego polecenia,**
- **komentarze rozpoczynają się znakiem %,**
- **zmienne są tworzone w pamięci po przypisaniu im wartości,**
- **zmienna ans przyjmuje (domyślnie) wynik operacji, którego nie podstawiono pod żadną inną zmienną,**
- **Matlab używa standardowego zapisu liczb z kropką jako separatorem dziesiętnym. W liczbie może wystąpić znak e oznaczający notację wykładniczą; dopuszczalne jest również użycie znaków i lub j dla oznaczenia części urojonej liczby zespolonej,**
- **natychmiastową pomoc na temat polecenia, funkcji czy symbolu możesz wyświetlić, wpisując polecenie help, po którym wpiszesz odpowiednią nazwę,**
- **pracę z Matlabem kończy się przez wpisanie polecenia exit lub quit.**

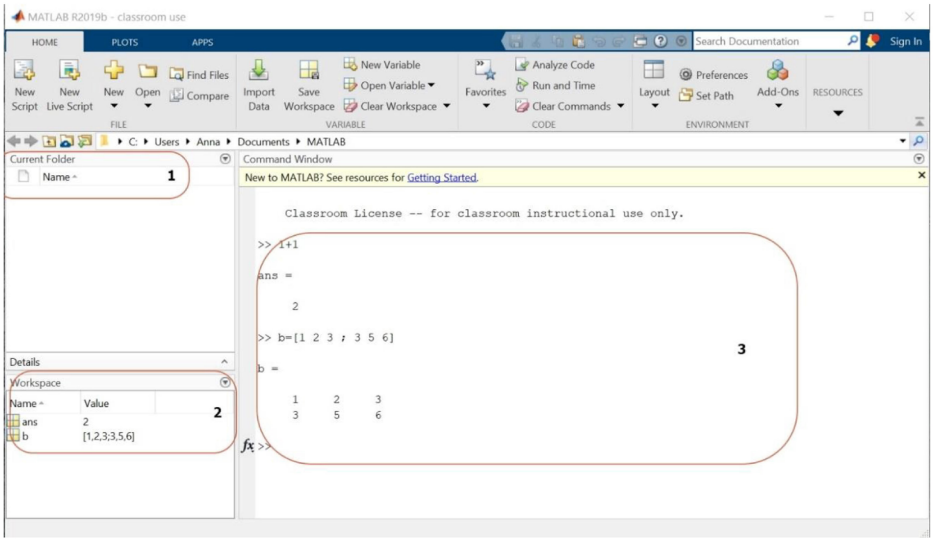
1.2. Interfejs Matlaba

Po uruchomieniu Matlaba pojawia się główne okno – Rysunek 1. O gotowości programu do pracy świadczy widoczny w wierszu poleceń znak >>

W standardowym widoku znajduje się:

1. Przestrzeń robocza (*Workplace*);
2. Bieżący folder (*Current folder*);
3. Okno poleceń (*Command Window*);

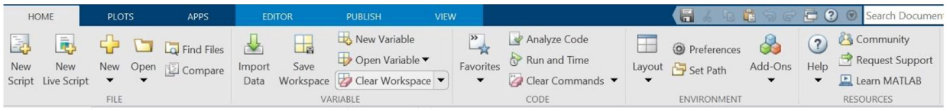
jak również pasek narzędzi wraz z menu programu – patrz Rysunek 2.



Rysunek 2. Główne okno Matlaba.

Okno poleceń (*Command Window*) zawiera menu główne z następującymi opcjami (patrz Rysunek 3):

1. *File* – zawiera opcje, które umożliwiają operacje na M-plikach:
   a) *New/M-file* – otwarcie okna edycyjnego celem utworzenia nowego M-pliku,
   b) *New/Figure* – otwarcie nowego okna graficznego,
   c) *New/Model* – otwarcie okna Untitled dla tworzenia nowego modelu Simulink'a − *Open M-file* ... – modyfikacja istniejącego M-pliku.
2. *Variable* – przenoszenie informacji do bufora poprzez funkcje Cut, Copy, Paste, Clear.
3. *Code* – wybór edytora dla plików oraz parametrów związanych np. z formatem danych numerycznych, kolorem, wielkością i krojem znaków itp.
4. *Environment* – podaje spis otwartych okien Matlaba.
5. *Resources* – interaktywna pomoc.



Rysunek 3. Okno poleceń.

Centralną część pulpitu Matlaba zajmuje *okno poleceń* (ang. *Command Window*). Kiedy w oknie tym wyświetlany jest znak zachęty >>, oznacza to, że Matlab jest gotowy na wprowadzenie polecenia. Polecenia wprowadzamy najczęściej pojedynczo z klawiatury i zatwierdzamy klawiszem *Enter*. Jeśli chcemy napisać kilka poleceń w jednym wierszu, to oddzielamy je średnikami lub przecinkami.

Przykład 1.1.
Zacznijmy od prostego polecenia:

```
>> 1+1 %wydanie polecenia obliczenia wyniku dodawania, naciśnij ENTER
ans =
    2
% otrzymamy powyższą odpowiedź. Ponieważ nie określono nazwy zmiennej wynikowej, % więc wynik został przypisany do zmiennej ans.
```

Przykład 1.2.
Zwróćmy uwagę, jaka jest różnica, jeśli piszemy kilka poleceń w jednym wierszu i oddzielimy je przecinkami lub alternatywnie średnikami.

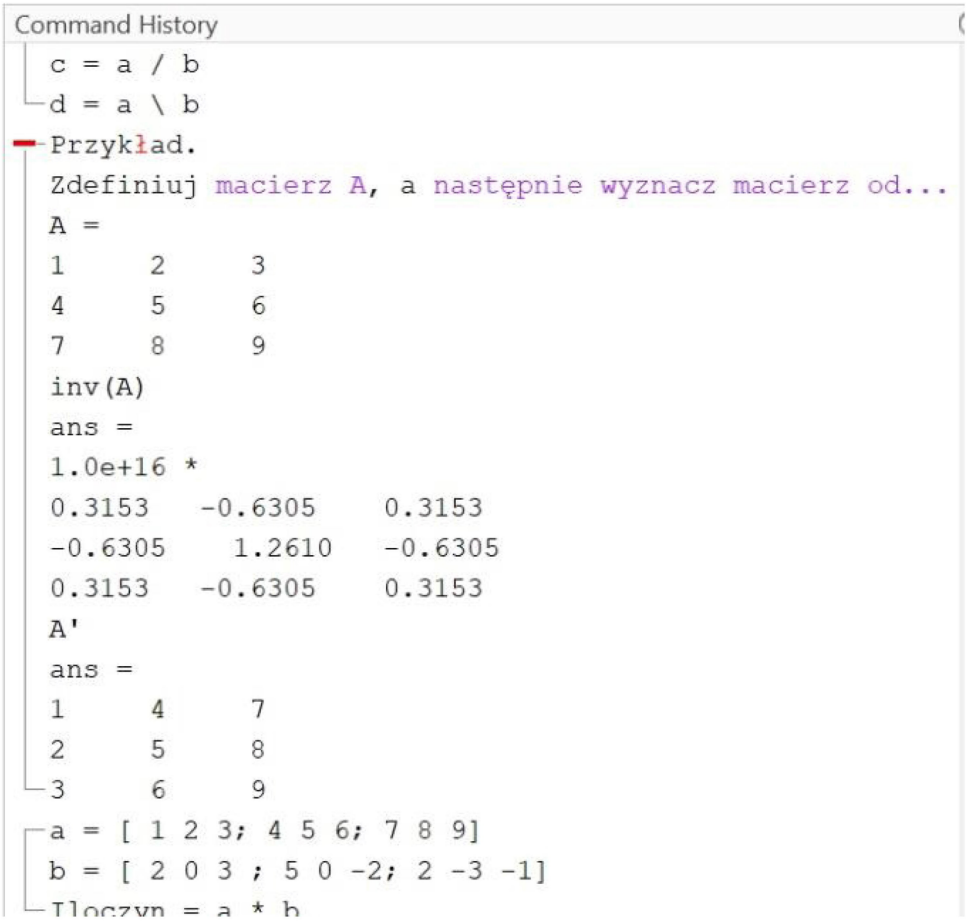
```
>> a=1; b=2; c=3
%użycie średnika (bez ostatniego polecenia) zwróci poniższy wynik
c =
    3
>> a=1; b=2; c=3;
%użycie średnika po każdej instrukcji nie wyświetla żadnego wyniku w oknie poleceń,
% chociaż przypisanie wartości następuje
>> a=1, b=2, c=3
%natomiast użycie przecinka po każdej instrukcji daje poniższy wynik
a =
    1
b =
    2
c =
    3
```

Wszystkie zmienne, jakie zdefiniujemy w czasie aktualnej sesji, przechowywane są w tzw. *przestrzeni roboczej* (ang. *Workspace*).

W oknie tym możemy znaleźć podstawowe informacje o każdej zmiennej (nazwę, typ, aktualną wartość itp.).

Pozostałe panele:

- *Command History* (Historia poleceń), który zawiera wszystkie polecenia wydane podczas pracy z Matlabem – patrz Rysunek 4.
- *Current Directory* (Katalog bieżący), zawierający aktualny główny katalog roboczy.
- *Launch Pad* (Wprowadzone bloki), zawiera strukturę zainstalowanych części Matlaba i toolboxów.



Rysunek 4. Okno Historii poleceń.

W Matlabie mamy możliwość korzystania z funkcji umożliwiających wykonanie operacji na zmiennych z przestrzeni roboczej – patrz Tabela 1.

Tabela 1. Funkcje zarządzające pamięcią.

Funkcja	Opis
who	polecenie to wyświetla w panelu *Command Window* wszystkie zmienne z pamięci roboczej
whos	polecenie to wyświetla w panelu *Command Window* wszystkie zmienne z pamięci roboczej wraz z rozmiarami i typem
clear x	usuwa zmienną x z pamięci
clear all	usuwa wszystkie zmienne z pamięci
clear functions	usuwa wszystkie skompilowane m-funkcje i mex-funkcje z pamięci
clear global	usuwa zmienne zadeklarowane jako globalne

Przykład 1.3.

```
>> a=1; b=2; c=3; %definiujemy zmienne a, b, c
>> who %uzyskamy informacje, jakie są aktualne zmienne robocze w przestrzeni
Your variables are:
a  b  c
>> clear a %polecenie usunie z przestrzeni roboczej zmienną a
>> who %ponownie wyświetlamy aktualne zmienne robocze
Your variables are:
b  c
```

Oprócz tych standardowo widocznych okien, Matlab wykorzystuje również inne, które pojawiają się podczas wykonywania konkretnych czynności. Do takich okien należą:

- okno pomocy (ang. *Help*) otwierane za pomocą klawisza *F1*, które pozwala przeglądać bardzo rozbudowane pliki pomocy,
- okna grafiki (ang. *Figure*), w których prezentowane są różnego rodzaju wykresyi inne wyniki poleceń Matlaba mające postać graficzną,

- okno edytora plików (ang. *Editor*) zawierające prosty edytor tekstów, w którymmożemy tworzyć własne skrypty i funkcje,
- okno profilera pozwalające testować własne funkcje i poprawiać ich wydajność.

Główne funkcje opisujące okno poleceń zostały opisane w Tabeli 2.

Tabela 2. Funkcje obsługujące okno poleceń.

Funkcja	Opis
clc	czyszczenie ekranu
echo on/ ech off	włączenie/wyłączenie echa
diary	włączenie rejestracji historii wykonywanych rozkazów

1.3. Pomoc programu

Program Matlab posiada bardzo rozbudowaną pomoc, z której możemy skorzystać na dwa sposoby: za pomocą interfejsu graficznego (Rysunek 5) lub polecenia help.



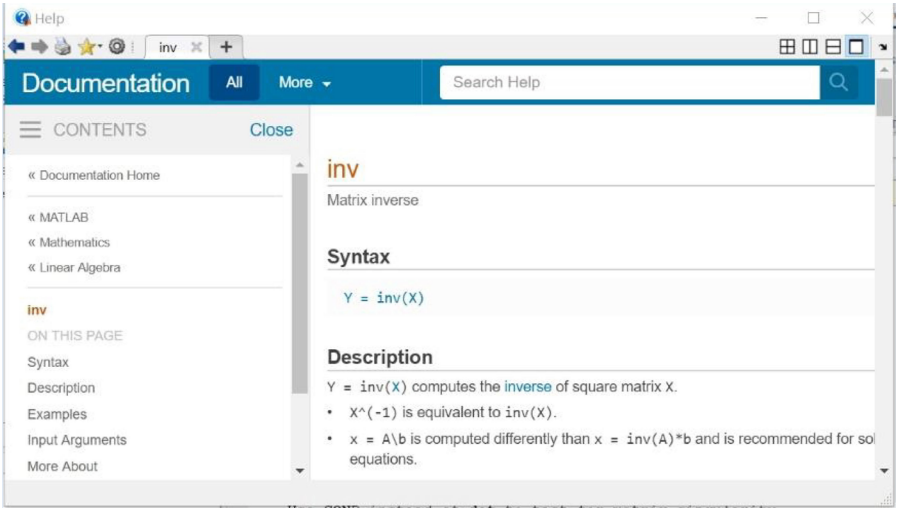
Rysunek 5. Główne okno pomocy.

Wszelkie informacje, jakie uzyskamy, są wyświetlane bezpośrednio w konsoli programu. Uzyskanie informacji dotyczących jakiegoś polecenia ma następującą postać

help nazwa_funkcji

Przykład 1.4.

>> help inv
Po wpisaniu tego polecenia otrzymamy na ekranie konsoli informacje na temat podanej funkcji – Rysunek 6.



Rysunek 6. Okno pomocy dla polecenia inv.

Polecenia dotyczące pomocy programu zostały opisane w Tabeli 3.

Tabela 3. Polecenia związane z pomocą.

Polecenie	Opis
help	lista tematów pomocy
help syntax	pomoc na temat podstaw składni
help aaa	help aaa
help katalog	pomoc na temat wszystkich m-plików z danego katalogu
help	pomoc na temat operatorów
helpwin	lista tematów pomocy w interfejsie graficznym
helpdesk	otwiera pomoc dot. Matlaba

System pomocy Matlaba pozwala także na przeszukiwanie dokumentacji elektronicznej. Służy do tego zakładka *Search* w oknie pomocy. Możemy przeszukiwać dokumentację na kilka sposób:

- opcja *Full Text* – przeszukuje treść dostępnych dokumentów pomocy,
- opcja *Document Titles* – umożliwia przeszukiwanie wśród tytułów zgromadzonych dokumentów,
- opcja *Function Name* – opcja szuka podanej frazy w bazie nazw funkcji dostępnych w Matlabie,
- opcja *Online Knownledge Base* – przeszukiwanie zasobów sieciowych.

1.4. Pliki

Podstawowe typy plików występujące w Matlabie zostały opisane w Tabeli 4., natomiast polecenia związane z plikami w Tabeli 5.

Tabela 4. Rodzaje plików.

Typ pliku	Opis
*.m	m-pliki są to pliki tekstowe, można rozróżnić dwa typy m-plików: skryptowe i funkcyjne
*.mat	pliki, w których zapisane są dane w postaci binarnej
*.mex	pliki skompilowane napisane w języku C lub Fortranie wywoływane w Matlabie
*.fig	pliki, w których zapisywane są wykresy, które można ponownie edytować
*.asv	pliki generowane przez autozapis co 5 minut, mogą służyć do odzyskania pliku

Tabela 5. Polecenia na plikach i katalogach.

Polecenie	Opis
cd path	zmienia aktualny katalog
date	aktualna data
delete	kasuje plik o danej nazwie

Polecenie	Opis
diary	zapisuje historię poleceń w pliku
dir	wypisanie zawartości katalogu
load(‘nazwa pliku’)	wczytuje zmienne z pliku typu *.mat
load(‘nazwa pliku’).txt	wczytuje dane z pliku tekstowego
mkdir	tworzenie nowego folderu
path	wyświetla ścieżkę
pwd	wyświetla aktualny katalog
save(‘nazwa pliku’)	zapisuje zmienne z przestrzeni roboczej do pliku o rozszerzeniu *.mat
save(‘nazwa pliku’,’zmienna1’,’zmienna2’,...)	zapisuje tylko wymienione zmienne do pliku o rozszerzeniu *.mat
type	edycja pliku
what path	wyświetla zawartość katalogu o podanej ścieżce dostępu

Przykład 1.5.

```
>> mkdir Folder %utworzenie nowego katalogu o nazwie Folder
>> chdir Folder %zmiana bieżącego katalogu na katalog Folder
>> edit Dane.m %stworzenie pliku o nazwie Dane.m
>> diary Historia %zapisze historię poleceń i ich rezultaty w pliku Historia
```

Przykład 1.6.

```
>> x=4; y=2;  %definicja zmiennych
>> z=x*y  %obliczenie wyrażenia
%otrzymany wynik
z =
     8
>> save wynik x y % zapisanie zmiennych x i y w pliku o nazwie wynik.
mat
>> load wynik %odczytaj dane zapisane w pliku wynik.mat
```

1.5. Operacje na plikach

W Matlabie istnieje wiele sposobów zapisu danych do pliku.

Najpopularniejsze sposoby to:

1. Korzystając z funkcji fopen i fprintf:

```
fid = fopen(nazwa pliku, uprawnienia);
fprintf(fid, tekst i format zmiennych, zmienne);
fclose(fid);
```

gdzie:
nazwa pliku – łańcuch znaków z nazwą otwieranego pliku,
uprawnienia – łańcuch znaków opisany w Tabeli 6.

Tabela 6. Wartość argumentu uprawnienia funkcji fopen.

Wartość argumentu	Opis
‘r’	odczyt z istniejącego pliku
‘w’	zapis do istniejącego lub utworzonego pliku
‘a’	zapis z dopisywaniem do istniejącej zawartości
‘r+’	odczyt i zapis do istniejącego pliku
‘w+’	odczyt i zapis do istniejącego lub utworzonego pliku
‘a+’	odczyt i zapis z dopisywaniem do istniejącej zawartości istniejącego lub utworzonego pliku

Tabela 7. Rodzaje konwersji używane w argumencie format funkcji fprintf.

Konwersja	Opis
%d	konwersja do zapisu liczb całkowitych
%f	konwersja do zapisu liczb rzeczywistych w formacie stałoprzecinkowym
%e	konwersja do zapisu liczb rzeczywistych w formacie zmiennoprzecinkowym
%g	automatyczny dobór krótszego zapisu

Konwersja	Opis
%c	konwersja do zapisu pojedynczych znaków
%s	konwersja do zapisu łańcuchów znakowych

Tabela 8. Znaki specjalne używane w argumencie format funkcji fprintf.

Znak specjalny	Opis
\b	cofnięcie o jeden znak
\f	nowa strona
\n	nowy wiersz
\r	przesunięcie kursora do początku wiersza
\t	znak tabulatora
\”	znak apostrofu
\\	znak lewego ukośnika
%%	znak procentu

Przykład 1.7.

```
x=1:0.5:10; %zdefiniowanie wektora liczb od 1 do 10, z krokiem 0,5 plik1 = fopen('test.txt', 'r+'); %otworzenie pliku fprintf(plik1, 'Wartość zmiennej x wynosi %g', x); %zapisanie do pliku fclose(plik1); %zamknięcie pliku
```

2. Korzystając z funkcji save:

```
save(nazwa pliku, zmienne ,'-append', format);
```

Tabela 9. Atrybuty plików.

Atrybut	Opis
'-mat'	plik danych Matlaba
'-ascii'	plik tekstowy z zapisem liczb na 8 znakach

Atrybut	Opis
‘-ascii’, ‘-tabs’	plik tekstowy z zapisem liczb na 8 znakach i oddzielaniem liczb tabulatorami
‘-ascii’, ‘-double’	plik tekstowy z zapisem liczb na 16 znakach
‘-ascii’, ‘-double’, ‘-tabs’	plik tekstowy z zapisem liczb na 16 znakach i oddzielaniem liczb tabulatorami

Przykład 1.8.

```
x=1:0.5:10; %zdefiniowanie wektora liczb od 1 do 10 z krokiem 0.5
y=-5:1:5; %zdefiniowanie wektora liczb od -5 do 5 z krokiem 1
save(‘plik2.txt’, ‘x’, ‘y’, ‘-ascii’); %zapisanie zdefiniowanych wektorów do pliku
```

3. Korzystając z funkcji dlmwrite:

dlmwrite(*nazwa pliku, zmienna ,’znak odstępu’*);

Wykonując natomiast operację odczytu danych z pliku, możemy je wykonać na jeden z poniższych sposobów:

1. Korzystając z funkcji fopen i *f*scanf:

fid = fopen(*nazwa pliku*, *uprawnienia*);
[*zmienne*] = fscanf(fid, *format zmiennych*);
fclose(fid); %operacja zamknięcia pliku

gdzie:
nazwa pliku – łańcuch znaków z nazwą otwieranego pliku,
uprawnienia – łańcuch znaków opisany w Tabeli 26.

Polecenie fscan dopuszcza takie same rodzaje konwersji, jak funkcja fprintf (Tabela 7).

Przykład 1.9.

```
test1 = fopen('plik1.txt', 'r+');
test2 = fscanf('test1', '%f  %f');
st = fclose(test1);
```

2. Korzystając z funkcji load:

[zmienne] = load(*nazwa pliku*);

3. Korzystając z funkcji dlmread:

[zmienne] = dlmread(*nazwa pliku, znak odstępu*);

1.6. Proste obliczenia

Matlab może być traktowany jako rozbudowany kalkulator. Program potrafi wykonywać działania we właściwej kolejności. Podstawowe operatory arytmetyczne zostały opisane w Tabeli 10.

Tabela 10. Operatory arytmetyczne.

Operator	Opis
+	dodawanie
-	odejmowanie
*	mnożenie macierzowe
.*	mnożenie tablicowe
/	dzielenie macierzowe
./	dzielenie tablicowe
^	potęgowanie macierzowe
.^	potęgowanie tablicowe

Przed rozpoczęciem pracy z programem warto wpisać w oknie poleceń diary on. Użycie tego polecenia spowoduje, że wszystkie komendy, które wpiszemy w oknie poleceń, zostaną zapisane automa-

tycznie w pliku tekstowym diary. Aby zakończyć zapisywanie, należy natomiast użyć polecenia diary off (patrz Tabela 5).

Przykład 1.10.

Poniżej przykład prostych obliczeń wykonanych przy pomocy Matlaba.

```
>> a = 20; %zmienna a przyjmuje wartość 20
b = -10; %zmienna b przyjmuje wartość -10
%teraz wykonujemy proste obliczenia na zdefiniowanych zmiennych.
%Za każdym razem wynik działania przypisujemy nowej zmiennej
c = a + b
d = a - b
e = a * b
f = a / b
g = a \ b
% teraz otrzymamy wyniki naszych obliczeń
c =
     10
d =
     30
e =
   -200
f =
     -2
g =
   -0.5000
```

Poza obliczeniami zaprezentowanymi w Przykładzie 1.10, Matlab oferuje również funkcje/polecenia służące do wykonywania innych operacji arytmetycznych – patrz Tabela 11.

Tabela 11. Funkcje do operacji arytmetycznych.

Nazwa funkcji	Opis
uplus(a)	Minus jednoargumentowy
plus(a,b)	Dodawanie
uminus(a)	Plus jednoargumentowy

minus(a,b)	Odejmowanie
times(a, b)	Mnożenie skalarne
mtimes(a, b)	Mnożenie macierzowe
rdivide(a, b)	Prawostronne dzielenie skalarne
ldivide(a, b)	Lewostronne dzielenie skalarne
mrdivide(a, b)	Prawostronne dzielenie macierzowe
mldivide(a, b)	Lewostronne dzielenie macierzowe
power(a, b)	Potęgowanie skalarne
mpower(a, b)	Potęgowanie macierzowe

Operatory relacji występujące w Matlabie zostały opisane w Tabeli 12.

Tabela 12. Operatory relacji.

Operator	Opis
<	Mniejszy niż
<=	Mniejszy lub równy niż
>	Większy niż
>=	Większy lub równy niż
==	równy
~=	Nie równy

Oprócz opisanych w Tabeli 12 standardowych operatorów relacyjnych Matlab oferuje też funkcje (patrz Tabela 13), które działają na podobnej zasadzie.

Tabela 13. Operatory relacyjne.

Funkcja	Opis
eq(a, b)	Sprawdza, czy a jest równe b
ge(a, b)	Sprawdza, czy a jest większe lub równe b
gt(a, b)	Sprawdza, czy a jest większe niż b
le(a, b)	Sprawdza, czy a jest mniejsze lub równe b

Funkcja	Opis
lt(a, b)	Sprawdza, czy a jest mniejsze od b
ne(a, b)	Sprawdza, czy a jest różne od b
isequal	Sprawdza równość tablicową

Przykład 1.11.

Możliwości zastosowania funkcji z Tabeli 13 znajdują się poniżej.

```
>> a=10;
>> b=-10;
>> eq(a,b) %funkcja sprawdzi, czy zmienne a oraz b są sobie równe
%jako wynik działania funkcji otrzymujemy zmienną logiczną przyjmu-
jące wartości:
% true (prawda) lub false (fałsz)
ans =
  logical
   0
>> ge(a,b)
%funkcja sprawdza, czy zmienna a jest większa od zmiennej b
%jako wynik działania funkcji otrzymujemy zmienną logiczną przyjmu-
jącą wartości:
% true (prawda) lub false (fałsz)
ans =
  logical
   1
>> ne(a,b)
%funkcja sprawdza, czy wartość a jest różna od wartości b
%jako wynik działania funkcji otrzymujemy zmienną logiczną przyjmu-
jącą wartości:
% true (prawda) lub false (fałsz)
ans =
  logical
   1
```

Tabela 14. Funkcje matematyczne.

Nazwa funkcji	Opis
sqrt(x)	Pierwiastek kwadratowy
power(a,b), pow2(x), nextpow2(x)	Funkcje potęgowe
abs(x)	Wartość bezwzględna
exp(x)	Funkcja wykładnicza
log(x), log2(x), log10(x)	Funkcje logarytmiczne
sin(x), cos(x), tan(x), cot(x)	Funkcje trygonometryczne
asin(x), acos(x), atan(x), atan2(x1,x2), acot(x)	Funkcje trygonometryczne odwrotne
sinh(x), cosh(x), tanh(x),coth(x)	Funkcje hiperboliczne
asinh(x), acosh(x), atanh(x), acoth(x)	Funkcje hiperboliczne odwrotne
rem(x,y), mod(x,y)	Funkcje dzielenia całkowitoliczbowego
round(x), fix(x), floor(x), ceil(x)	Funkcje zaokrągleń liczby
sign(x)	Funkcja znaku
factorial(x)	Funkcja silnia
complex(x,y)	Utworzenie liczby zespolonej z=x+iy

Przykład 1.12.

Poniżej przykłady zastosowania funkcji dotyczących zaokrąglania liczb rzeczywistych.

```
>> round(1.1)
ans =
     1
>> floor(1.9) %Funkcja floor to zaokrąglanie w dół do najbliższej liczby
całkowitej
ans =
     1
>> floor(1.1)
ans =
     1
```

```
>> ceil(1.9) %Funkcja ceil to zaokrąglanie w górę do najbliższej liczby
całkowitej
ans =
     2
>> ceil(1.1)
ans =
     2
>> fix(1.9)
ans =
     1
>> fix(1.1)
ans =
   1
>> rem(21,3);
>> rem(21,3)
ans =
     0
>> rem(21,4)
ans =
     1
>> mod(21,4) %funkcja mod to reszta z dzielenia 21 przez 4
ans =
     1
>> mod(21,3)
ans =
     0
```

Listę dostępnych standardowych funkcji matematycznych możemy wyświetlić poleceniem > > help elfun

Tabela 15. Znaki specjalne.

Znak	Opis
=	podstawienie wyrażenia pod zmienną
[ ]	używane przy definiowaniu macierzy i przy definiowaniu listy argumentów wyjściowych funkcji
{ }	używane przy definiowaniu i indeksowaniu macierzy komórkowych

Znak	Opis
( )	używane przy zaznaczeniu kolejności wykonywanych działań, przy definiowaniu listy argumentów wejściowych funkcji i przy dostępie do elementów macierzy
,	separator poleceń, argumentów funkcji, indeksów
:	oznaczenie zakresu indeksów macierzowych
%	początek komentarza

Przykład 1.13.

Poniżej kilka prostych działań, ukazujących zastosowanie Matlaba jako kalkulatora.

```
>> 6-11
ans =
   -5
>> 3+4*8
ans =
   35
>> exp(-2)
ans =
   0.1353
>> 2+cos(pi/4)
ans =
   2.7071
```

Aby zapamiętać wyniki obliczeń, wykorzystujemy *zmienne*. W tym celu wybieramy nazwę zmiennej (ciąg znaków zaczynający się od litery) i przypisujemy jej wartość za pomocą operatora.

Przykład 1.14.

```
>> a=4*3-9
a =
     3
>> b=log(3)
b =
    1.0986
>> c=2*a+b
c =
    7.0986
```

Możemy też sami definiować proste funkcje. Używamy w tym celu polecenia

inline('wzor_funkcji')

Przykład 1.15.

```
>> f=inline(,2*x+5') %definiujemy funkcję liniową
f =
     Inline function:
     f(x) = 2*x+5
>> f(2) %obliczamy wartość funkcji f dla argumentu 2
ans =
     9
>> g=inline(,x+y') %definiujemy funkcję dwóch zmiennych
g =
     Inline function:
     g(x,y) = x+y
>> g(-3,9) %obliczamy wartość funkcji w zadanym punkcie
ans =
     6
```

1.7. Wartości liczbowe

Liczby używamy w jednej z poniższych postaci:

- stałopozycyjnej,
- zmiennopozycyjnej z użyciem znaku e poprzedzającego wykładnik potęgi 10.

W Matlabie we wszystkich dostępnych działaniach oraz funkcjach możemy używać liczb zespolonych. Do zapisu części urojonej liczb zespolonych używamy stałej i lub j.

Przykład 1.16.

```
>> z1=1+2i  %w pierwszym zapisie do części urojonej używamy i
z1 =
   1.0000 + 2.0000i
>> z2=3+4j %w drugim zapisie do części urojonej używamy j
z2 =
   3.0000 + 4.0000i
>> z3=4+5*i     %w trzecim zapisie do części urojonej używamy i oraz
symbolu * (mnożenia)
z3 =
   4.0000 + 5.0000i
>> z4=1+2*sqrt(-1)
%w czwartym zapisie do części urojonej używamy pierwiastka kwadra-
towego z liczby -1
z4 =
   1.0000 + 2.0000i
```

Nazwa i oraz j jest zmienną specjalną wbudowaną w program. Pozostałe zmienne specjalne zostały opisane w Tabeli 16.

Tabela 16. Zmienne specjalne.

Zmienna	Opis
eps	podaje dokładność maszynową (w zapisie wykładniczym)
i,j	część urojona liczby zespolonej
inf	nieskończoność
nan	symbol nieoznaczony
nargin	liczba argumentów aktualnych funkcji
pi	wartość liczby
realmax	największa liczba rzeczywista rozpoznawana przez program
realmin	najmniejsza liczba rzeczywista rozpoznawana przez program

Przykład 1.17.

```
>> 1/0 %dzielenie przez zero jest niewykonalne
ans =
   Inf
>> 0/0
ans =
   NaN
>> pi  %wyświetlona zostanie wartość pi z dokładnością do czterech miejsc po przecinku
ans =
    3.1416
>> eps
ans =
   2.2204e-16
>> I %wynikiem jest liczba zespolona
ans =
   0.0000 + 1.0000i
```

W Matlabie są dostępne formaty liczbowe opisane w Tabeli 17. Różnice pomiędzy poszczególnymi formatami zaprezentowane są w Przykładzie 1.18.

Tabela 17. Formaty liczbowe.

Funkcja	Opis
Short	format krótki
short e	format krótki z wykładnikiem
Long	format długi
long e	format długi z wykładnikiem
Bank	format walutowy
Hex	format szesnastkowy
Rat	format przybliżający liczby ułamkami małych liczb całkowitych
+	znak drukowany dla liczb dodatnich

Przykład 1.18.

```
>> format rat
pi
ans =
     355/113
>> format long %w formacie long wyświetlane jest 16 cyfr
>> pi
ans =
   3.141592653589793
>> format long e
>> pi
ans =
     3.141592653589793e+00
>> format short %w formacie short wyświetlane jest 5 cyfr
>> pi
ans =
    3.1416
>> format short e
>> pi
ans =
   3.1416e+00
>> format bank
>> pi
ans =
         3.14
>> format hex
>> pi
ans =
   400921fb54442d18
```

Matlab oferuje także specjalne funkcje sprawdzające, czy podana zmienna czy wyrażenie jest określonego typu (patrz Tabela 18).

Tabela 18. Funkcje do sprawdzania typu zmiennej/wyrażenia.

Funkcja	Opis
isinf	sprawdza, czy wartość jest nieskończona
isfinite	przeciwieństwo poprzedniej funkcji, bo sprawdzamy, czy wartość jest skończona
isnan	sprawdza, czy wartość jest typu NaN
isreal	sprawdza, czy wartość jest typu rzeczywistego
isfloat	sprawdza, czy wartość jest typu zmiennoprzecinkowego
isinteger	sprawdza, czy wartość jest całkowita

1.8. Operacje logiczne

Podstawowe operacje logiczne zostały opisane w Tabeli 19.

Tabela 19. Operacje logiczne.

Operacja	Opis
==	równe
~=	różne
<,<=,>,>=	mniejsze, mniejsze lub równe, większe, większe lub równe
&	iloczyn logiczny AND
|	suma logiczna OR
~	negacja logiczna
xor	funkcja logiczna XOR
any	wartość TRUE(1) jeśli choć jeden element wektora wynikowego jest prawdziwy
all	wartość TRUE(1) jeśli wszystkie elementy wektora wynikowego są prawdziwe
find	funkcja określająca wskaźniki i wartości niezerowych elementów

Przykład 1.19.

```
>> A=[1,2,3; 4,5,6] %zmienna A jest typu tablicowego
A =
     1     2     3
     4     5     6
>> B=[-1,4,3; -7,8,9]
%zmienna B jest typu tablicowego i ma taki sam wymiar jak zmienna A
B =
    -1     4     3
    -7     8     9
>> A>=B
% sprawdzamy, czy każdy element macierzy A jest większy lub równy
od %odpowiedniego elementu macierzy B
%w wyniku otrzymujemy zmienną logiczną
ans =
  2×3 logical array
   1   0   1
   1   0   0
```

Przykład 1.20

```
>> A = [true false]; %zmienna A to wektor o wartościach logicznych
>> B = [true; false]; %zmienna B to wektor o wartościach logicznych
>> C = xor(A,B)
C =
  2×2 logical array
   0   1
1          0

>> A = [0 0 2;0 0 2;0 0 2];
   >> B = any(A)
B =
 1×3 logical array
   0   0   1
```

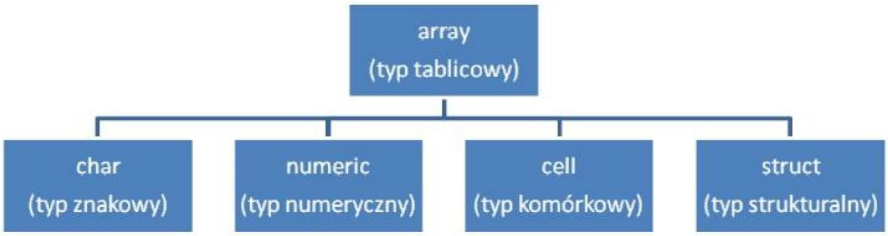
1.9. Typy danych

Podstawowe typy danych dostępne w Matlabie opisane zostały w Tabeli 20.

Tabela 20. Podstawowe typy danych.

Typ danych	Opis
int8	8-bitowa liczba całkowita ze znakiem
uint8	8-bitowa liczba całkowita bez znaku
int16	16-bitowa liczba całkowita ze znakiem
uint16	16-bitowa liczba całkowita bez znaku
int32	32-bitowa liczba całkowita ze znakiem
uint32	32-bitowa liczba całkowita bez znaku
int64	64-bitowa liczba całkowita ze znakiem
uint64	64-bitowa liczba całkowita bez znaku
single	pojedyncza precyzja
double	zmienna podwójnej precyzji
logical	zmienna logiczna – przyjmująca wartości true lub false
char	zmienna znakowa
cell array	zmienna komórkowa
structure	struktura

W Matlabie mamy zdefiniowanych sześć typów (klas) danych, z których każdy może być tablicą wielowymiarową. Na szczycie tej hierarchii znajduje się Tabela (array) – patrz Rysunek 7. Informacje o typach danych uzyskuje się poleceniem: help datatypes.



Rysunek 7. Podstawowe typy danych.

char – typ tablicowy znakowy lub łańcuchowy (każdy znak zapisywany jest 16-bitowo);
numeric – typ numeryczny podwójnej precyzji;
cell – typ komórkowy. Elementy tablic komórkowych mogą zawierać inne tablice;
struct – typ strukturalny. Odwołują się do nazw pól, które mogą zawierać inne tablice.

Do badania stanu i typu zmiennych używamy funkcji opisanych w Tabeli 21.

Tabela 21. Sprawdzanie typu danych.

Funkcja	Opis
is	grupa funkcji do badania stanu i typu zmiennej, wszystkie zwracają 1 jeśli sprawdzany warunek jest prawdą, 0 gdy nie jest spełniony
iscell	zwraca 1, gdy zmienna jest typu cell array, 0, gdy jest innego typu
ischar	zwraca 1, gdy zmienna jest typu znakowego, 0, gdy nie jest
isfloat	zwraca 1, gdy zmienna jest niecałkowita (np. double), 0, gdy jest całkowita
isnumeric	zwraca 1, gdy zmienna jest numeryczna, 0, gdy innego typu
isreal	zwraca 1, gdy zmienna jest rzeczywista, 0, gdy urojona
isscalar	zwraca 1, gdy zmienna jest skalarem, 0, gdy jest wektorem lub macierzą
isstruct	zwraca 1, gdy zmienna jest strukturą, 0, gdy jest innego typu

Funkcja	Opis
isvector	zwraca 1, gdy zmienna jest wektorem, 0, gdy jest macierzą
whos	wypisuje rozmiar, ilość zajętej pamięci, typ danej i atrybuty

Przykład 1.21.

Możliwości zastosowania funkcji opisanych w Tabeli 14 znajdują się poniżej.

```
>> x = 1
isinteger(x) %sprawdza, czy zmienna jest całkowita
isfloat(x) %sprawdza, czy zmienna jest niecałkowita
isvector(x) %sprawdza, czy zmienna jest wektorem
isscalar(x) %sprawdza, czy zmienna jest skalarem
isnumeric(x) %sprawdza, czy zmienna jest numeryczna
x =
     1
ans =
  logical
   0
ans =
  logical
   1
ans =
  logical
   1
ans =
  logical
   1
ans =
  logical
   1
```



1.10. Zadania

1. Znajdź funkcje Matlaba, które wykorzystujemy do zaokrąglania liczb. Następnie użyj odpowiedniej z nich, aby:

a) Wynikiem działania 6/5 była liczba 1
b) Wynikiem działania 6/5 była liczba 2
c) Wynikiem działania 9/5 była liczba 1
d) Wynikiem działania 9/5 była liczba 2
e) Wynikiem działania -6/5 była liczba -1
f) Wynikiem działania- 6/5 była liczba -2
g) Wynikiem działania -9/5 była liczba -1
h) Wynikiem działania -9/5 była liczba -2

2. Zapisz wyniki wyrażeń 6/5, 9/5, -6/5, -9/5 w zmiennej typu double. Dokonaj konwersji wartości tych zmiennych dla wartości typu int8, a wynik konwersji zapisz w nowych zmiennych.

3. Jakie jest przeznaczenie poleceń: clear, clc, home?



4. Utwórz zmienną x=9. Sprawdź jej wartość, a następnie oblicz pierwiastek kwadratowy ze zmiennej x.

5. Wykonaj następujące polecenia:

a) Wyświetl katalog bieżący, jego zawartość i listę dostępnych ścieżek.
b) Zmień katalog bieżący na WSEI:\
c) Utwórz podkatalog Laboratorium i uczyń go katalogiem bieżącym.
d) Utwórz zmienne a=4, b=8 i zapisz je w pliku dane.bin
e) Wyświetl katalog bieżący, jego zawartość i zawartość pliku dane.bin



6. Odszukaj informacje na temat poleceń systemu plików: dir, cd, ls, pwd, copyfile, delete, fileattrib, movefile, mkdir, rmdir, exist.

7. Oblicz wartość następujących wyrażeń:

a) $\frac{(1+i)^3(2-1)}{1+6i} + \frac{1-2i}{(3+i)(2-8i)}$

b) $x = 2 \cdot \sqrt[3]{5},\ y = \frac{3}{5} \cdot 2^4,\ z = e^x \cdot logy.$

c) $|3! \cdot \sin(-2 \cdot \pi)|$

2. Wektory





WAŻNE

Wektorem nazywamy jednowymiarową tablicę liczb.
W Matlabie rozróżniamy dwa rodzaje wektorów:
- **wektory kolumnowe,**
- **wektory wierszowe.**

2.1. Tworzenie wektorów

Zarówno wektory kolumnowe, jak i wierszowe tworzymy, używając znaku nawiasu kwadratowego. Różnica polega tylko na zastosowaniu lub nie znaku średnika.

Przykład 2.1.

Tworzenie wektora kolumnowego.

```
>> w = [1;2;3;4;5] %kolejne wiersze wektora oddzielamy  średnikiem
w =
    1
    2
    3
    4
    5
```

Przykład 2.2.

Tworzenie wektora wierszowego.

```
>> w = [1 2 3 4 5] %kolejne elementy wektora oddzielamy spacją
w =
    1    2    3    4    5
```

2.2. Odwoływanie się do elementów wektora

Odwoływać się do elementów wektora można na kilka sposobów – patrz Przykład 2.3 oraz Przykład 2.4.

Przykład 2.3.

Odwołanie się do elementu wektora poprzez użycie nawiasu okrągłego, wskazującego numer wiersza/kolumny z danym elementem.

```
>> w = [1;2;3;4;5] %tworzenie wektora kolumnowego
w =
     1
     2
     3
     4
     5
>> w(3)
%jaki element znajduje się w trzeciej kolumnie
%numer kolumny zapisujemy w nawiasach okrągłych ()
ans =
     3
```

Przykład 2.4.

Teraz będziemy odwoływali się nie do jednego elementu wektora, ale do kilku. W tym celu zastosujemy konstrukcję

(ElementPoczątkowy:ElementKońcowy)

```
>> wektor = [1 2 3 4 5 6 7 8 9]; %tworzymy wektor wierszowy
wektor2 = wektor(5:7)
%odwołujemy się do elementów od pozycji 5 do pozycji 7
%numery pozycji z odwołania zapisujemy w nawiasach okrągłych z użyciem dwukropka
%wektor wynikowy
wektor2 =
     5     6     7
```

2.3. Operacje na wektorach

Na wektorach możemy wykonywać następujące operacje: dodawanie oraz odejmowanie, iloczyn skalarny, transponowanie, łączenie wektorów.

Przykład 2.5.

Wykonując dodawanie oraz odejmowanie wektorów, musimy pamiętać, żeby wektory były tego samego typu oraz miały taką samą liczbę elementów.

```
>> A=[1 2 3 4 5]; %tworzymy wektor wierszowy
>> B=[6 7 8 9 10]; %tworzymy wektor wierszowy
>> C=A+B %dodajemy odpowiednie elementy wektora A do wektora B
C =
    7    9   11   13   15
>> D=A-B %odejmujemy odpowiednie elementy wektora A od wektora B
D =
   -5   -5   -5   -5   -5
```

Przykład 2.6.

Przemnażając wektor przez skalar, mnożymy każdy element wektora przez daną liczbę.

```
>> A=[1, -2, 3, -4];
%definiujemy wektor wierszowy z użyciem przecinka
>> B=2*A %mnożymy wektor przez skalar
B =
    2   -4    6   -8
```

Transponowanie wektora dokonujemy korzystając z symbolu ‘ – patrz Przykład 2.7.

Przykład 2.7.

```
>> A=[1, 2, 3, 4]; %definiujemy wektor wierszowy z użyciem przecinka
>> TransponowanieA=A' %zapisujemy wektor w postaci kolumnowej – transponowanie
TransponowanieA =
    1
    2
    3
    4
```

W Matlabie mamy możliwość łączenia wektorów, tworząc w ten sposób nowy wektor – patrz Przykład 2.8.

Przykład 2.8.

```
>> a = [1 2 3 4]; %tworzymy wektor wierszowy
>> b = [5 6 7 8]; %tworzymy wektor wierszowy
>> c=[a,b]
%wektor c powstaje z połącznie wektora a oraz wektora b
%wektor c jest zapisany w wierszy – użyliśmy przecinka przy tworzeniu

c =
    1    2    3    4    5    6    7    8
>> d=[a;b]
%wektor d powstaje z połącznie wektora a oraz wektora b
%wektor c jest zapisany w  dwóch wierszach – użyliśmy średnika przy tworzeniu
d =
    1    2    3    4
    5    6    7    8
```

W praktyce inżynierskiej często zachodzi potrzeba obliczenia iloczynu wektorowego lub skalarnego dla wektora A oraz B. Wynikiem takiego działania jest nowy wektor, którego długość (wartość) jest równa iloczynowi długości dwóch wektorów i sinusa kąta między nimi, kierunek otrzymanego wektora jest prostopadły do płaszczyzny wyznaczonej przez dwa wektory, a zwrot nowego wektora określany regułą prawej ręki lub regułą śruby prawoskrętnej.

Istotna jest tu kolejność mnożenia, ponieważ AxB nie równa się BxA. Z kolei wynikiem iloczynu skalarnego jest liczba (skalar) równa

iloczynowi długości dwóch wektorów i cosinusa kąta między nimi.

Iloczyn skalarny w programie Matlab uzyskamy, wykorzystując funkcję dot, natomiast iloczyn wektorowy z użyciem funkcji cross – patrz Przykład 2.9.

Przykład 2.9.

```
>> a=[1 2 3]; %tworzymy wektor wierszowy a
>> b=[4 5 6]; %tworzymy wektor wierszowy b
>> c=dot(a,b) %obliczamy iloczyn skalarny
c =
   32
>> d=cross(a,b) %obliczamy iloczyn wektorowy
d =
   -3     6    -3
```

2.4. Zadania



1. Wygeneruj trzy wektory dziesięcioelementowe X, Y, Z o dowolnych wartościach, następnie:

a) zapisz wektor X do pliku o nazwie wektorX
b) zapisz sumę wektorów Y i Z do pliku o nazwie wektorYZ
c) wyczyść przestrzeń roboczą ze zmiennych X, Y, Z
d) wczytaj wartości zmiennej X z pliku o nazwie wektorX

2. Dla 100-elementowego wektora A oblicz:

a) sumę elementów,
b) średnią elementów,
c) medianę,
d) odchylenie standardowe,
e) liczbę elementów.



3. Zbuduj wektor X = [2 3 4 5] a następnie oblicz $X \cdot X^T$, $X^T \cdot X$

4. Zdefiniuj wektor a = [1,5,-3] oraz wektor b = [9,0,-3].

Dla tych wektorów oblicz ich iloczyn skalarny oraz wektorowy.

5. **Prędkość w ruchu jednostajnie przyśpieszonym dana jest funkcją *V = at + b*.**

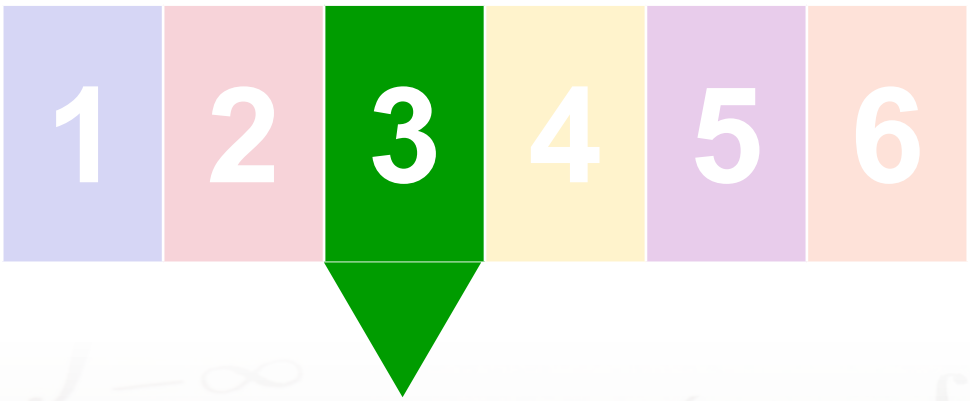


Dla podanych stałych a=0.62 i b=1.5 wyznacz prędkość V dla czasu t=0,1,2,3,4,5,6,7,8,9,10s.

6. **Wygeneruj wektory x i y o elementach od 1 do 20 co 2, a następnie oblicz:**

a) $\frac{x+y}{sinx}$,

b) $\sqrt{x} \cdot (x + y)$.



3. Macierze

Podstawową strukturą danych używaną w Matlabie jest *macierz* (Tabela dwuwymiarowa), oznacza ona prostokątną tablicę o wymiarach wypełnioną liczbami.



WAŻNE

Szczególnymi przypadkami macierzy są *skalary*, które możemy traktować jako macierze wymiaru 1 × 1, oraz *wektory* (macierze wymiaru 1 × *n* lub *n* × 1) – wierszowy lub kolumnowy.

3.1. Definiowanie macierzy

W środowisku Matlab macierz możemy definiować (budować) na kilka sposobów.

Po pierwsze, macierz możemy wygenerować poprzez wypisanie jej elementów. Poszczególne elementy macierzy oddziela się spacjami, a wiersze średnikami lub umieszcza się je w oddzielnych liniach. Listę elementów macierzy należy zapisać w nawiasach kwadratowych.

Przykład 3.1.

```
B=[1 3 4 5; 9 8 2 4] %definiowanie macierzy
%jeżeli za nawiasem kwadratowym nie napiszemy średnika to otrzyma-
my odpowiedź
B =
    1    3    4    5
    9    8    2    4
>> B=[1, 3, 4, 5; 9, 8, 2, 4]
%lub alternatywnie z użyciem przecinka między elementami każdego
wiersza
B =
    1    3    4    5
    9    8    2    4
```

Drugim sposobem definiowania macierzy jest generowanie jej elementów za pomocą polecenia

A=[min:krok:max]

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

Przykład 3.2.

Wygeneruj macierz dwuwierszową o wyrazach od 1 do 5 w pierwszym wierszu i o wyrazach od 6 do 18 (krok 3) w wierszu drugim.

```
B=[1:5; 6:3:18] %definiowanie macierzy
B =
    1    2    3    4    5
    6    9   12   15   18
```

Dla generowania macierzy możemy też zastosować funkcje wspomagające generowanie macierzy (patrz Tabela 22).

Tabela 22. Funkcje generujące macierze.

Nazwa funkcji	Opis
[]	macierz pusta
eye(n)	macierz jednostkowa o wymiarach n x n
eye(n, m)	macierz jednostkowa o wymiarach n x m
linspace	wektor o jednostkowych odstępach liniowych między elementami
logspace	wektor o jednostkowych odstępach logarytmicznych między elementami
meshgrid	macierz definiująca regularną siatkę na płaszczyźnie XY opartą na wektorze poziomym X i pionowym Y
ngrid	macierz definiująca regularną siatkę w przestrzeni n-wymiarowej
ones(n)	macierz złożoną z samych zer o wymiarach n x n
ones(n,m)	macierz złożoną z samych zer o wymiarach n x m
rand(n)	macierz losowa o rozkładzie równomiernym o wymiarach n x n
rand(n,m)	macierz losowa o rozkładzie równomiernym o wymiarach n x m
randn(n)	macierz losowa o rozkładzie normalnym o wymiarach n x n
randn(n, m)	macierz losowa o rozkładzie normalnym o wymiarach n x m
zeros(n)	macierz o wymiarach n x n złożona z samych zer
zeros(n, m)	macierz o wymiarach n x m złożona z samych zer
pascal(n)	generuje macierz o wymiarach n x n, której elementy są liczbami dodatnimi całkowitymi będącymi współczynnikami w trójkącie Pascala

Przykład 3.3.

Utwórz kwadratową macierz jednostkową A o wymiarze 4x4.

```
>> A=eye(4) %w nawiasie okrągłym piszemy wymiar macierzy
%macierz jednostkowa czwartego stopnia
A =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Przykład 3.4.

Utwórz macierz A o wymiarze 2x5 wypełnionej jedynkami.

```
A=ones(2,5)
%pierwsza liczba w nawiasie okrągłym określa liczbę wierszy,
%natomiast druga liczbę kolumn
A =
     1     1     1     1     1
     1     1     1     1     1
```

Kolejną metodą tworzenia macierzy jest utworzenie macierzy z kombinacji macierzy już istniejących. W tym zakresie Matlab oferuje szereg funkcji – patrz Tabela 23.

Tabela 23. Metody tworzenia macierzy.

Funkcja	Opis
blkdiag	konstrukcja macierzy, w której kolejne macierze dołączane są diagonalnie
cat	łączenie macierzy wzdłuż określonego wymiaru
horzcat	łączenie macierzy poziomo (dodawanie kolumn)
repmat	wielokrotne powtórzenie macierzy pionowo i poziomo
vertcat	dołączanie macierzy pionowo (dodawanie wierszy)

Przykład 3.5.

```
>>  A = ones(3); B = rand(3); %macierze A oraz B są tego samego wymiaru
%macierz A składa się z samych jedynek
% elementami macierzy B są liczby losowe
>> C = cat(1, A, B); D = horzcat(A, B);
%macierz C będzie składała się z macierzy i „pod nią” będzie macierz B
%macierz D będzie składała się z macierzy i „obok niej” będzie macierz B
>> C
C =
   1.0000   1.0000   1.0000
   1.0000   1.0000   1.0000
   1.0000   1.0000   1.0000
   0.8147   0.9134   0.2785
   0.9058   0.6324   0.5469
   0.1270   0.0975   0.9575
>> D
D =
   1.0000   1.0000   1.0000   0.8147   0.9134   0.2785
   1.0000   1.0000   1.0000   0.9058   0.6324   0.5469
   1.0000   1.0000   1.0000   0.1270   0.0975   0.9575
```

Przykład. 3.6

```
>> A=[1,2,3;4,5,6];
>> B=[7,8;9,10];
>> C=[11,12,13,14,15;16,17,18,19,20];
%macierze A, B oraz C zostały zdefiniowane
>> D=[A,B;C]
%należy pamiętać o zgodności liczby kolumn w poszczególnych wierszach tworzonej macierzy
D =
    1    2    3    7    8
    4    5    6    9   10
   11   12   13   14   15
   16   17   18   19   20
```

Tabela 24. Wymiar i wyświetlanie macierzy.

Funkcja	Opis
[n,m]=size(A)	zwraca liczbę kolumn n i wierszy m macierzy A
n=length(B)	zwraca wymiar wektora B (lub większy z wymiarów macierzy B)
A lub disp(A)	pokazuje macierz A na ekranie
max(A)	zwraca największy element wektora A; w przypadku gdy A jest macierzą, zwraca wektor wierszowy, którego elementami są maksymalne elementy z każdej kolumny A
min(A)	zwraca najmniejszy element wektora A; w przypadku gdy A jest macierzą, zwraca wektor wierszowy, którego elementami są maksymalne elementy z każdej kolumny A
mean(A)	zwraca średnią arytmetyczną elementów wektora A; w przypadku gdy A jest macierzą, zwraca wektor wierszowy, którego elementami są średnie arytmetyczne elementów z każdej kolumny A

Przykład 3.7.

```
>> A = [-11 0 1; 3 1 0; 5 1 1] %definiowanie macierzy
A =
   -11     0     1
     3     1     0
     5     1     1
>> inv(A) %obliczanie macierzy odwrotnej do macierzy A
ans =
   -0.0769   -0.0769    0.0769
    0.2308    1.2308   -0.2308
    0.1538   -0.8462    0.8462
>> A(1,3) %element stojący w pierwszym wierszu i trzeciej kolumnie
ans =
     1
```

```
>> A(:,2) %druga kolumna macierzy A
ans =
     0
     1
     1
>> min(A)
ans =
   -11     0     0
>> max(A)
ans =
     5     1     1
>> mean(A)
ans =
   -1.0000    0.6667    0.6667
```

3.2. Odwołania do elementów macierzy

Odwołując się do elementów macierzy, używamy dwukropka – patrz Tabela 25.

Tabela 25. Odwołanie się do elementów macierzy

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

Przykład 3.8.

Poniżej przedstawiamy metody odwołania się do podmacierzy.

```
>> A=[-1 -2 -3 -4 -5 -6;1 0 9 8 7 6 ; -1 -1 0 0 1 1] %definiujemy macierz A
A =
   -1   -2   -3   -4   -5   -6
    1    0    9    8    7    6
   -1   -1    0    0    1    1
>>  B=A(:,[1:2 6])
%  utworzenie macierzy B poprzez pobranie z macierzy A kolumn pierw-
szej, drugiej oraz szóstej
B =
   -1   -2   -6
    1    0    6
   -1   -1    1
%  utworzenie macierzy B z elementów macierzy A leżących  na prze-
cięciu wierszy
% 1 i 2 z  kolumnami 1, 3 i 5
>> B=A([1 2],1:2:5)
B =
   -1   -3   -5
    1    9    7
```

Przykład 3.9.

```
>>  A=[1 2 3 4; 5 6 7 8; 9 10 11 12]  %definiowanie macierzy
A =
    1    2    3    4
    5    6    7    8
    9   10   11   12
>> A(2,3)  % dostęp do elementów macierzy w drugim wierszu i trzeciej
kolumnie
ans =
    7
>> A(2,:) %dostęp do całego drugiego wiersza
ans =
    5    6    7    8
```

```
>> A(:,3) % dostęp do całej trzeciej kolumny
ans =
     3
     7
    11
>> A([2 3],:) %dostęp do wybranych wierszy
ans =
     5     6     7     8
     9    10    11    12
>>  A(:,[2 3])  %podobnie dostęp do wybranej kolumny
ans =
     2     3
     6     7
    10    11
>>  A(1:2:end,:) %dostęp do wierszy o numerze nieparzystym
ans =
     1     2     3     4
     9    10    11    12
>> A(3,:)=0   %wyzerowanie całego wiersza trzeciego
A =
     1     2     3     4
     5     6     7     8
     0     0     0     0
>> A(3,:)=[]    %usunięcie całego wiersza trzeciego
A =
     1     2     3     4
     5     6     7     8
```

Przykład 3.10.

Metody usuwania wektora z macierzy.

```
>> A=[-1 -2 -3 -4 -5 -6;1 0 9 8 7 6 ; -1 -1 0 0 1 1] %definiowanie macierzy
A
A =
   -1   -2   -3   -4   -5   -6
    1    0    9    8    7    6
   -1   -1    0    0    1    1
>> A(2,:)=[]        %usuwa drugi wiersz z macierzy A
A =
   -1   -2   -3   -4   -5   -6
   -1   -1    0    0    1    1
>> A(:,1:3)=[] % usuwa trzy pierwsze kolumny z macierzy A
A =
   -4   -5   -6
    0    1    1
```

3.3. Operacje macierzowe

W tym rozdziale omówimy następujące podstawowe i często używane operacje na macierzach:

- dodawanie i odejmowanie macierzy,
- transpozycja macierzy,
- mnożenie macierzy,
- wyznacznik macierzy,
- macierz odwrotna.

Podstawowe operacje macierzowe opisane są w Tabeli 26.

Tabela 26. Operacje macierzowe.

Operacja	Opis
A’	transpozycja macierzy A połączona z zamianą elementów zespolonych na sprzężone do nich
A.’	zwykła transpozycja macierzy zespolonej
A+B	dodawanie macierzy
A-B	odejmowanie macierzy
A*B	mnożenie macierzy
A/B	dzielenie lewostronne
A\B	dzielenie prawostronne
Inv(A)	inwersja macierzy
A^p	podnoszenie macierzy do potęgi skalarnej

Przykład 3.11.

Poniżej metody wykonywania operacji na macierzach.

```
>> A=[1 0 2; -2 -3 1] %definiujemy macierz A
A =
     1     0     2
    -2    -3     1
>> B=[-1 -1 2; -2 -3 21] %definiujemy macierz B
B =
    -1    -1     2
    -2    -3    21
>> A+B %dodając macierze, pamiętamy, aby obie macierze miały ten
sam wymiar
ans =
     0    -1     4
    -4    -6    22
>> A-B  %różnica macierzy
ans =
     2     1     0
     0     0   -20
```

```
>> A-6 %odjęcie od elementów macierzy liczby 6
ans =
   -5   -6   -4
   -8   -9   -5
```

Przykład 3.12.

Zdefiniuj macierz $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, a następnie wyznacz macierz odwrotną do niej i dokonaj transpozycji.

```
A =
    1    2    3
    4    5    6
    7    8    9
>> inv(A) %macierz odwrotna
ans =
  1.0e+16 *
   0.3153   -0.6305    0.3153
  -0.6305    1.2610   -0.6305
   0.3153   -0.6305    0.3153
>> A' %transponowanie macierzy
ans =
    1    4    7
    2    5    8
    3    6    9
```

Przykład 3.13.

Poniżej przykład mnożenia dwóch macierzy. Przypomnijmy, że na ogół operacja mnożenia macierzy nie jest przemienna, czyli A·B≠B·A, nawet jeżeli w obu wypadkach mnożenie jest wykonywalne.

```
>> a = [ 1 2 3; 4 5 6; 7 8 9]
b = [ 2 0 3 ; 5 0 -2; 2 -3 -1]
Iloczyn = a * b
a =
     1     2     3
     4     5     6
     7     8     9
b =
     2     0     3
     5     0    -2
     2    -3    -1
Iloczyn =
    18    -9    -4
    45   -18    -4
    72   -27    -4
```

Duże znaczenie w obliczeniach inżynierskich mają macierze kwadratowe, czyli takie, w których liczba wierszy jest równa liczbie kolumn. Dla macierzy kwadratowej możemy obliczyć m.in. wyznacznik oraz macierz odwrotną.

Przykład 3.14.

Oblicz wyznacznik macierzy kwadratowej $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$ z użyciem funkcji det.

```
>> A= [ 1 2 3; 4 5 6; 7 8 0] %zdefiniowanie macierzy
det(A) %obliczenie wyznacznika
A =
     1     2     3
     4     5     6
     7     8     0
ans =
   27.0000
```

Dzielenie macierzy, tak jak iloczyn, może być lewostronne i prawostronne. Dzielenie prawostronne: x=a / b jest rozwiązaniem równania: $x \cdot b = a$. Dzielenie lewostronne macierzy: x=a\b jest roz-

wiązaniem równania $b \cdot x = a$. Poniżej (patrz Przykład 3.15) użycie polecenia dzielenia prawo- oraz lewostronnego macierzy. Operacje te są wykorzystywane przy rozwiązywaniu układów równań.

Przykład 3.15.

```
>> a = [ 1 2 3 ; 4 5 6; 7 8 9];
b = [ 7 6 4 ; 3 2 1; 8 9 10];
c = a / b %dzielenie prawostronne macierzy
d = a \ b %dzielenie lewostronne macierzy
c =
    0.0000   -0.6364    0.3636
    0.0000   -0.3636    0.6364
    0.0000   -0.0909    0.9091
d =
   1.0e+16 *
    2.8373    3.4678    3.7830
   -5.6745   -6.9355   -7.5660
    2.8373    3.4678    3.7830
```

Oprócz standardowych działań na macierzach, Matlab oferuje też szereg funkcji do operacji macierzowych – patrz Tabela 27 oraz Przykład 3.16, Przykład 3.17.

Tabela 27. Operacje macierzowe.

Funkcja	Opis
intersect(A,B)	Przecięcie dwóch tablic. Funkcja zwraca wartości wspólne dla A oraz B, w posortowanym porządku.
intersect(A,B,'wiersze')	Traktuje każdy wiersz tablicy A oraz każdy wiersz tablicy B jako oddzielny element i zwraca te wiersze, które są wspólne dla A i B. Wiersze wynikowe są posortowane.
ismember(A,B)	Przyjmuje wartość 1, jeśli elementy A są znalezione w B.
ismember(A,B,'wiersze')	Traktuje każdy wiersz tablicy A oraz każdy wiersz tablicy B jako oddzielny element. Wynikiem jest wektor, który przyjmuje wartość 1, gdy wiersze macierzy A są również wierszami macierzy B.

Funkcja	Opis
issorted(A)	Przyjmuje wartość logiczną 1, jeśli elementy A są posortowane.
setdiff(A,B)	Zwraca te wartości z macierzy A, które nie są elementami macierzy B. Macierz wynikowa jest posortowana.
setdiff(A,B, ‘wiersze’)	Traktuje każdy wiersz tablicy A oraz każdy wiersz tablicy B jako oddzielny element. Wynikiem są te wiersze macierzy A, które nie są jednocześnie wierszami macierzy B. Wiersze macierzy wynikowej są posortowane.
setxor	operacja OR na dwóch macierzach
union	operacja sumy dwóch macierzy
unique	wartości unikatowe macierzy

Przykład 3.16.

```
>> a = [1 2 3 4 5 6 7 8 9] %zdefiniowanie macierzy
b = [ 7 8 9 10 11 12 13 14] %zdefiniowanie macierzy
u = union(a, b) %połączenie dwóch macierzy (ich elementów)
i = intersect(a, b) %elementy wspólne dla dwóch macierzy
s = setdiff(a, b) %elementy macierzy a, które nie są elementami macierzy b
a =
    1    2    3    4    5    6    7    8    9
b =
    7    8    9   10   11   12   13   14
u =
    1    2    3    4    5    6    7    8    9   10   11   12   13   14
i =
    7    8    9
s =
    1    2    3    4    5    6
```

Przykład 3.17.

```
>> A = [9 2 9 5 6 2]; %definiowanie macierzy
>> C=unique(A)
%funkcja zwraca te same wartości, co macierz A, ale bez powtórzeń
i posortowane
C =
     2     5     6     9
>> issorted(A) %sprawdza, czy macierz  A jest posortowana
ans =
  logical
   0
>> issorted(C) %sprawdza, czy macierz C jest posortowana
ans =
  logical
   1
```

3.4. Usuwanie elementów macierzy

Możemy również usunąć wybrany fragment macierzy A poprzez przypisanie mu wartości w postaci macierzy pustej, używając w tym celu nawiasu kwadratowego [].

Przykład 3.18.

```
>> A=[1 2 3; 4 5 6; 7 8 9] %definiowanie macierzy
A =
     1     2     3
     4     5     6
     7     8     9
>> A(2,:)=[]        % usunięcie drugiego wiersza macierzy A
A =
     1     2     3
     7     8     9
```

3.5. Macierze rzadkie

Macierzą pełną nazywamy macierz, w której przeważającą liczbą elementów są liczby niezerowe. Przeciwieństwem macierzy pełnej są macierze rzadkie.

Macierzą rzadką (ang. *sparse matrix*) nazywamy macierz, w której większość elementów stanowią zera, np. macierz przekątniowa. Rzadkie macierze występują podczas rozwiązywania wielu równań z dziedziny teorii sieci elektrycznych i systemów energetycznych, mechanice płynów czy ruchu ciepła, teorii grafów itd.



UWAGA

Matlab wyświetla w inny sposób macierze rzadkie niż macierze pełne.
Na ekranie wyświetlane są tylko numery wierszy i kolumn zawierające elementy niezerowe.

Do zamiany macierzy gęstej na macierz pełną wykorzystujemy funkcję sparse – patrz Przykład 3.19 oraz Tabela 28.

Przykład 3.19.

```
>> A=[1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]
% definiowanie macierzy jednostkowej stopnia czwartego
A =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> sparse (A)
% uproszczony zapis macierzy A z informacją, gdzie są elementy różne
od zera
ans =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
```

Tabela 28. Funkcje generujące macierze rzadkie.

Funkcja	Opis
sparse	generowanie macierzy rzadkiej z macierzy pełnej
speye	macierz jednostkowa
sprand	macierz losowa o rozkładzie równomiernym
sprandn	macierz losowa o rozkładzie normalnym
sprandsym	macierz symetryczna losowa
spdiags	macierze diagonalna i pasmowa

Tabela 29. Operacje na macierzach rzadkich.

Funkcja	Opis
condest	oszacowanie wskaźnika uwarunkowania
etreeplot	wykres przedstawiający drzewo eliminacji
full	przekształcenie macierzy rzadkiej w pełną
find	wyszukiwanie elementów niezerowych
issparse	sprawdza, czy macierz jest macierzą rzadką

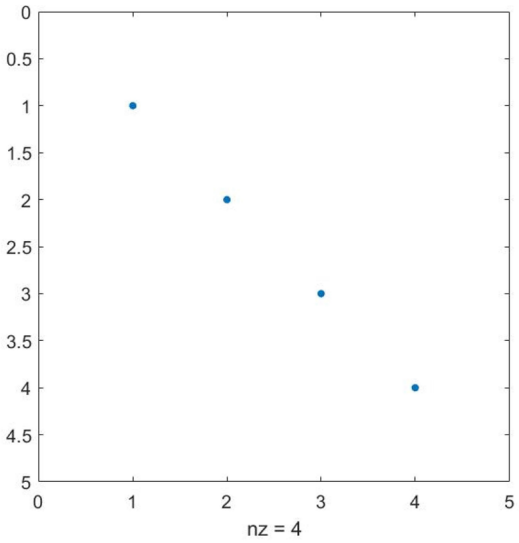
Funkcja	Opis
nnz	liczba elementów niezerowych
nonzeros	elementy niezerowe
nzmax	maksymalna liczba elementów niezerowych
spones	zamiana elementów niezerowych na jedynki
spalloc	rezerwowanie pamięci dla elementów niezerowych
spy	wizualizacja elementów niezerowych macierzy

Przykład 3.20.

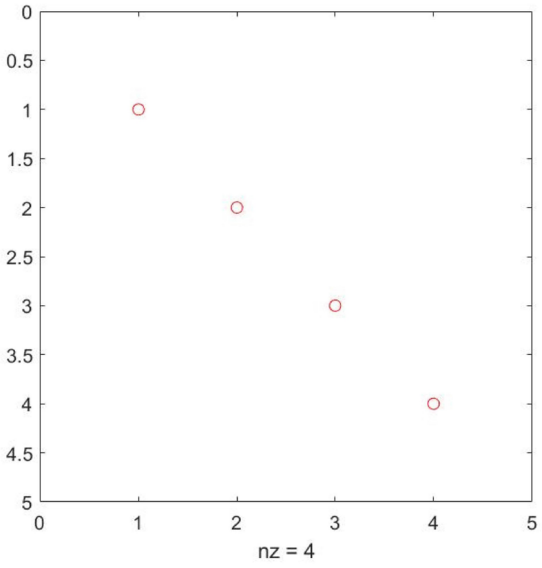
```
>> S = sprand(6,6,0.2)
%macierz losowa o rozkładzie równomiernym z zadanymi parametrami
S =
   (1,2)       0.1386
   (2,4)       0.2575
   (3,4)       0.8407
   (1,5)       0.1493
   (6,5)       0.2435
   (3,6)       0.2543
   (4,6)       0.8143
>> A = full(S) %zapisanie macierzy S w postaci macierzy rzadkiej
A =
         0    0.1386         0         0    0.1493         0
         0         0         0    0.2575         0         0
         0         0         0    0.8407         0    0.2543
         0         0         0         0         0    0.8143
         0         0         0         0         0         0
         0         0         0         0    0.2435         0
>> whos
  Name      Size              Bytes  Class                                Attributes
  A         6x6                 288  double
  P         1x1                   0  Matlab.graphics.chart.primitive.  Line
  S         6x6                 168  double                               sparse
  x         1x63                504  double
```

Przykład 3.21.

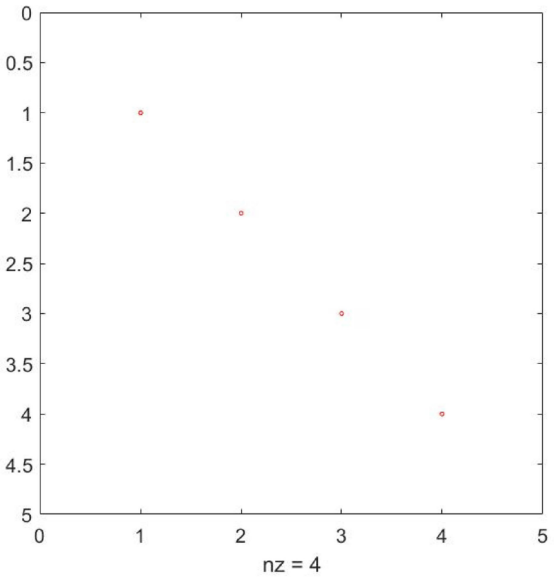
```
>> B=[1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]
%zdefiniowanie macierzy rzadkiej
B =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> sparse(B)
ans =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
>> spy(B)
%wizualizacja elementów niezerowych macierzy B
```



>> spy(B,’ro’) % ustalenie koloru wykresu (czerwony) i rodzaju znacznika (kółko)

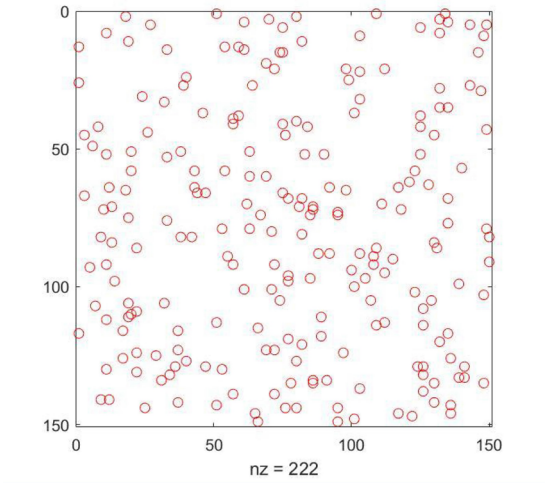


>> spy(B,’ro’,2) % ustalenie koloru wykresu (czerwony), rodzaju znacznika (kółko) oraz wielkości znacznika

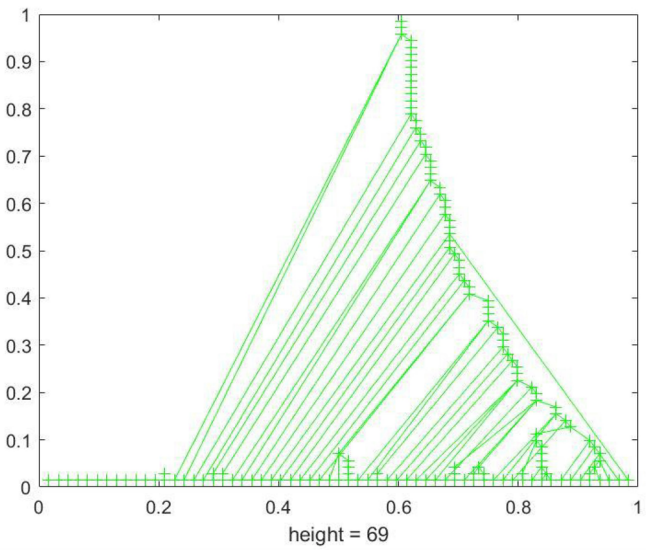


Przykład 3.22.

```
>> A=sprandn(150,150,0.01) %zdefiniowanie macierzy rzadkiej
figure(1)
spy(A,'ro')
```




```
figure(2)
etreeplot(A,'g+') % wykres przedstawiający drzewo eliminacji
```



3.6. Zadania



1. Utwórz macierz $A = \begin{bmatrix} 1 & 2 & 3 \\ -4 & -5 & -6 \\ 7 & 8 & 9 \end{bmatrix}$

a) wyznacz kolejno sumy: po każdym wierszu, po każdej kolumnie i po dwóch przekątnych,
b) wygeneruj macierz B o rozmiarze 3x3 wartości pseudolosowych z zakresu -10 do 10,
c) wyznacz najmniejszą wartość z całej macierzy B,
d) wykonaj działania: *A* + *B*, *A* – *B*, *A* · *B*,
e) usuń pierwszy wiersz z macierzy A oraz drugą kolumnę z macierzy B,
f) obróć macierze otrzymane w podpunkcie f) o 90 stopni.

.....
.....
.....
.....
.....
.....
.....
.....
.....

2. Wygeneruj macierz A o wymiarach 6x6 złożoną z liczb losowych z przedziału 0-200.

a) zapisz macierz A do pliku o nazwie macierz A
b) wyczyść przestrzeń roboczą ze zmiennej A,
c) wczytaj zmienną A z pliku o nazwie macierz A

.....
.....
.....
.....
.....
.....
.....
.....



3. Dla następującej macierzy A

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Wykonaj następujące operacje:

a) zdefiniuj zmienną A,
b) A, A(:,2), A(3,:), A(2,3), A(1:2, 3:4), A(:), A(:,:)
c) [A A], [A; A]
d) A(:,2)=[], A(:, 3)=[]

……
……
……
……
……
……
……
……
……
……

4. Zapoznaj się z poleceniami zeros, ones, eye, diag, rand. Następnie wykonaj operacje: ones(4,4), eye(5).

……
……
……
……
……
……
……
……
……
……
……
……
……
……



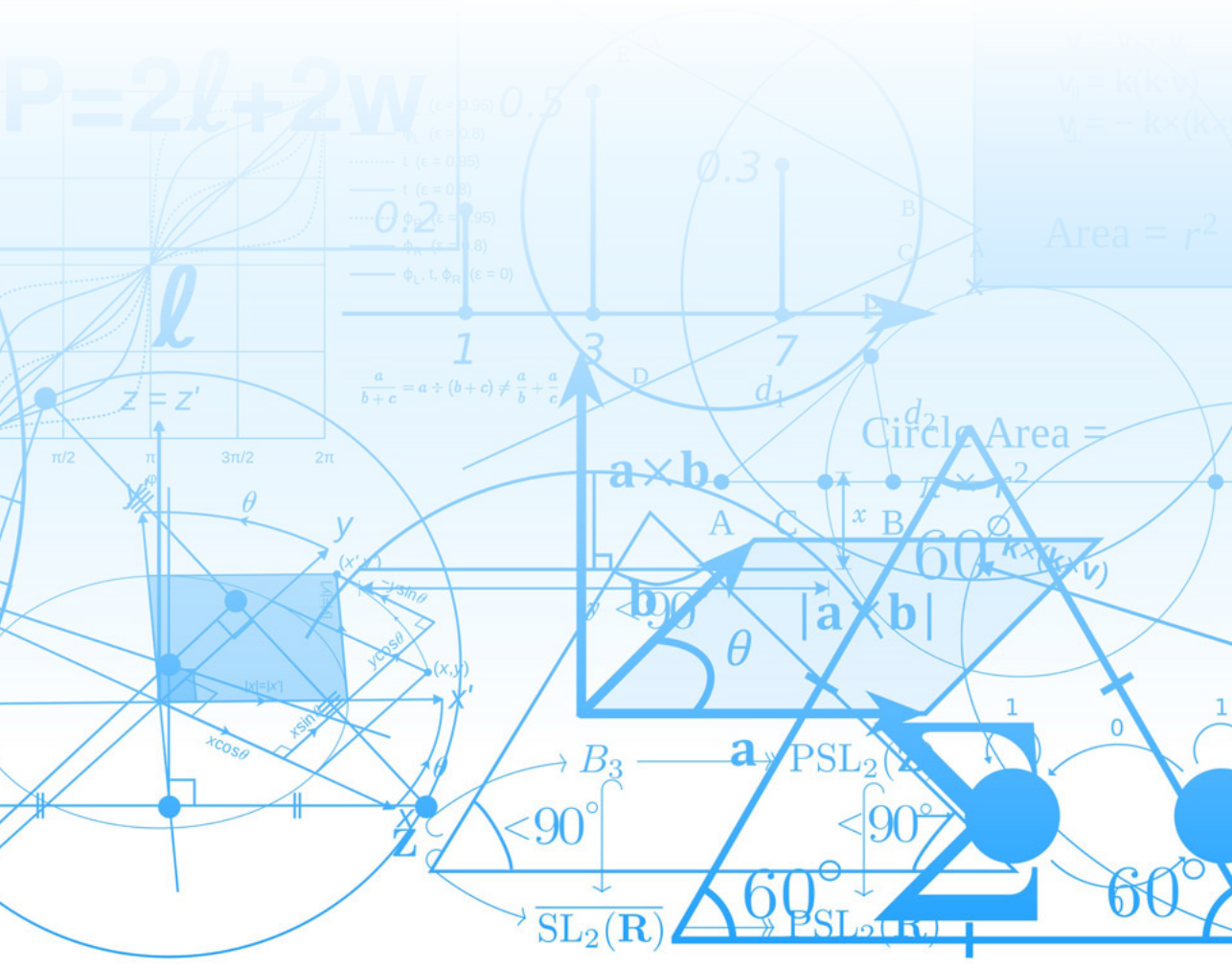
5. **Dla zdefiniowanej macierzy** $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

a) wyświetl jej drugi wiersz,
b) wyświetl jej trzecią kolumnę,
c) wyświetl jej pierwszy wiersz i czwartą kolumnę, wynik przypisz zmiennej B.

6. **Zdefiniuj macierze** $A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ **oraz** $B = \begin{bmatrix} -3 & 2 & 3 \\ 4 & 0 & 6 \\ 5 & 8 & 9 \end{bmatrix}$

a) oblicz iloczyny macierzowe $A \cdot B$, $B \cdot A$
b) wyznacz macierz $C = \frac{A+B^T}{3}$
c) wyznacz macierz $(A \cdot B)^{-1}$

4. Tablice



4.1. Tablice numeryczne

Program Matlba operuje nie tylko na macierzach, ale również na Tabelach.



WAŻNE

Tabela komórek (ang. cell array) jest klasą, która umożliwia przechowywanie różnych typów danych w ramach jednego obiektu.

Każda komórka w tablicy identyfikowana jest (jak w przypadku macierzy) parą indeksów (wiersz, kolumna).

Podobnie jak w przypadku macierzy, można do tablicy komórek **zastosować indeksowanie liniowe**, w którym cała **Tabela traktowana jest jako zbiór połączonych kolumn**.

Każda komórka w tablicy może zawierać dowolne postaci danych (np. macierze liczbowe, ciągi znaków itp.).

Matlab oferuje szereg funkcji pozwalających wykonać operacje na Tabelach. Na przykład można utworzyć mową macierz z kombinacji macierzy istniejących bez stosowania operatora konstrukcji [] – patrz Tabela 30.

Tabela 30. Funkcje tablicowe.

Funkcja	Opis
length	zwraca długość wektora lub największy wymiar macierzy
ndims	liczba wymiarów tablicy
numel	liczba elementów tablicy
size	wymiar tablicy

Funkcja	Opis
blkdiag	konstrukcja macierzy, w której kolejne macierze dołączane są diagonalnie
diag	macierz diagonalna o elementach na przekątnej z wektora x
flipdim	obrót macierzy wokół wybranej osi
fliplr	obrót macierzy wokół osi pionowej
flipud	obrót macierzy wokół osi poziomej
repmat	tworzy macierz złożoną z m pionowych i n poziomych powtórzeń macierzy A
reshape	zwraca macierz o n wierszach i m kolumnach z kolejnych elementów kolumn z macierzy A
rot90	obrót macierzy w taki sposób, że ostatnia kolumna staje się pierwszym wierszem
issorted	zwraca informację, czy elementy tablicy są posortowane
sort	sortuje elementy tablicy
sortrows	sortuje wiersze macierzy prostokątnej w porządku rosnącym względem elementów w pierwszej kolumnie
squeeze	usuwa pojedynczy wymiar
transpose	obrót macierzy wokół głównej przekątnej, zamiana wektorów kolumnowych na wierszowe i vice-versa
vectorize	dodanie kropki w wyrażeniach zawierających symbole mnożenia, dzielenia i potęgowania

Przykład 4.1.

```
>> A = [6 0 -7 1 3 8 -10 4 -2]; %definiowanie macierzy
B = sort(A) %elementy macierzy A zostaną posortowane rosnąco
B =
   -10    -7    -2     0     1     3     4     6     8
>> C = sort(A,'descend') ) %elementy macierzy A zostaną posortowane
malejąco
C =
     8     6     4     3     1     0    -2    -7   -10
```

Przykład 4.2.

```
>> A = [1 2 3; 4 5 6; 2 3 -9; 8 4 -2]  %utworzenie macierzy
A =
    1    2    3
    4    5    6
    2    3   -9
    8    4   -2
>> B = sortrows(A) %posortowanie macierzy względem wierszy
B =
    1    2    3
    2    3   -9
    4    5    6
    8    4   -2

>> C = sortrows(A,2)
%sortowanie wierszy macierzy na podstawie elementów znajdujących
się w drugiej kolumnie
C =
    1    2    3
    2    3   -9
    8    4   -2
    4    5    6
>> D = sortrows(A,[1 3])
%sortowanie elementów macierzy najpierw na podstawie elementów
z pierwszej, a później z trzeciej kolumny
D =
    1    2    3
    2    3   -9
    4    5    6
    8    4   -2
```

Przykład 4.3.

```
>> A = cat(3,{,a' ,b' ,c';'d' ,e' ,f';'g' ,h' ,i'},{,j' ,k' ,l';'m' ,n' ,o';'p' ,q' ,r'})
  3×3×2 cell array
A(:,:,1) =
    {'a'}    {'b'}    {'c'}
    {'d'}    {'e'}    {'f'}
    {'g'}    {'h'}    {'i'}
A(:,:,2) =
    {'j'}    {'k'}    {'l'}
    {'m'}    {'n'}    {'o'}
    {,p'}    {,q'}    {,r'}
>> B = rot90(A,3) %obrót macierzy A o 3*90 stopni
  3×3×2 cell array
B(:,:,1) =
    {'g'}    {'d'}    {'a'}
    {'h'}    {'e'}    {'b'}
    {'i'}    {'f'}    {'c'}
B(:,:,2) =
    {'p'}    {'m'}    {'j'}
    {'q'}    {'n'}    {'k'}
    {'r'}    {'o'}    {'l'}
```

Przykład 4.4.

```
>> A = {,a' ,b' ,c'; ,d' ,e' ,f'; ,g' ,h' ,i'} %macierz składająca się ze znaków
A =
  3×3 cell array
    {'a'}    {'b'}    {'c'}
    {'d'}    {'e'}    {'f'}
    {'g'}    {'h'}    {'i'}
>> B = fliplr(A) %obrót macierzy wokół osi pionowej
B =
  3×3 cell array
    {'c'}    {'b'}    {'a'}
    {'f'}    {'e'}    {'d'}
    {'i'}    {'h'}    {'g'}
```

Przykład 4.5.

Utwórz macierz kwadratową stopnia piątego z następującymi elementami na przekątnej: 2, 1, -1, -2, 5 (pozostałe elementy macierzy to zera).

```
>> A = [2 1 -1 -2 -5];
>> B = diag(A)  %utworzenie macierzy diagonalnej z macierzy A
B =
     2     0     0     0     0
     0     1     0     0     0
     0     0    -1     0     0
     0     0     0    -2     0
     0     0     0     0    -5
```

Przykład 4.6.

```
>> A = diag([100 200 300])
%utworzenie macierzy diagonalnej z elementami 100, 200, 300 na przekątnej
A =
   100     0     0
     0   200     0
     0     0   300
>> B = repmat(A,2) %utworzenie macierzy B, która jest powtórzeniem macierzy A
B =
   100     0     0   100     0     0
     0   200     0     0   200     0
     0     0   300     0     0   300
   100     0     0   100     0     0
     0   200     0     0   200     0
     0     0   300     0     0   300
C = repmat(A,2,3)
```

```
C =
   100     0     0   100     0     0   100     0     0
     0   200     0     0   200     0     0   200     0
     0     0   300     0     0   300     0     0   300
   100     0     0   100     0     0   100     0     0
     0   200     0     0   200     0     0   200     0
     0     0   300     0     0   300     0     0   300
```

Przykład 4.7

```
>> A = 11:20; %liczby od 11 do 20 (z krokiem 1)
B = reshape(A,[5,2])
%utworzenie macierzy składającej się z 5 wierszy oraz 2 kolumn, będą-
cą powtórzeniem macierzy A
B =
    11    16
    12    17
    13    18
    14    19
    15    20
```

4.2. Tablice znaków

Tablice znaków w obliczeniach inżynierskich wykorzystujemy do opisywania danych i rysunków. Tablice znaków definiujemy, zapisując je w apostrofach.

Rozdzielnie łańcuchów znakiem średnika powoduje utworzenie tablicy znaków.

Każdy znak łańcucha traktowany jest jako element tablicy, co umożliwia wykonywanie na nich określonych operacji.

Tabela 31. Funkcje wykonujące operacje na łańcuchach.

Funkcja	Opis
blanks	generuje łańcuch spacji
char	łączy łańcuchy pionowo, uzupełniając krótsze spacjami do równej długości
deblank	kasuje wszystkie spacje w łańcuchu
int2str	zamiana liczby całkowitej na łańcuch
lower	zamiana liter dużych na małe
mat2str	zamiana macierzy na łańcuch
num2str	zamiana liczby rzeczywistej na łańcuch
sprintf	zamiana liczby na łańcuch w wybranym formacie
sscanf	zamiana łańcucha na liczbę o wybranym formacie
strcat	łączy łańcuchy poziomo
str2mat	zamiana łańcucha na macierz
str2num	zamiana łańcucha na liczbę rzeczywistą
upper	zamiana liter małych na duże

Przykład 4.8.

```
>> Znaki = [‚a’ ‚b’ ‚c’
‚d’ ‚e’ ‚f’
‘g’ ‘h’ ‘i’]
Znaki =
  3×3 char array
    ‘abc’
    ‘def’
    ‘ghi’
>> size(Znaki)
ans =
     3     3
```

Przykład 4.9.

Napisz tekst ‘Witam Wszystkich’ małymi literami.

```
>> lower(‘Witam, Wszystkich.’) %zamiana liter na małe
ans =
    ‘witam, wszystkich.’
```

Tabela 32. Funkcje przetwarzające łańcuchy znakowe.

Funkcja	Opis
strcmp	porównuje dwa łańcuchy znakowe, jeżeli są identyczne, zwraca 1, jeśli nie, 0
strncmp	porównuje pierwsze n znaków w łańcuchu
strvcat	łączy w pionie łańcuchy znakowe
deblank	usuwa spacje z końca łańcucha
strtrim	usuwa początkowe i końcowe spacje z łańcucha
strjust	wyrównuje łańcuch do: prawej (j=’right’), lewej (j=’left’) lub środka (j=’center’)

Przykład 4.10.

```
>> s1 = ,Kurs’; %łańcuch znaków
s2 = {,Kurs’,’programowania’;
      ,w’,’Matlabie’}; %łańcuch znaków
znajdz = strcmp(s1,s2) %funkcja znajduje łańcuch s1 w łańcuchu s2
%funkcja przyjmuje wartości logiczne
znajdz =
  2×2 logical array
   1   0
   0   0
```

Przykład 4.11

```
>> s1 = ,Lublin, Czechów; %łańcuch znaków
s2 = ,Lublin, Czuby'; %łańcuch znaków
tf = strncmp(s1,s2,6) %funkcja porównuje 6 pierwszych znaków w łań-
cuchach s1 oraz s2
tf =
  logical
   1
```

Przykład 4.12.

Funkcja strjust wyjustowuje tekst do środka, prawej lub lewej strony.

```
>> S1 = {,Euler    ,;
      ,Fibonacci';
      ,    Gauss'} %Tabela znaków
S1 =
  3×1 cell array
    {'Euler    '}
    {,Fibonacci'}
    {,    Gauss'}
>> C2 = strjust(S1,'center') %wyjustowanie napisów do środka
C2 =
  3×1 cell array
    {'  Euler  '}
    {,Fibonacci'}
    {,  Gauss  ,}
>>  C3 = strjust(S1,'left') %wyjustowanie napisów do lewej strony
C3 =
  3×1 cell array
    {'Euler    '}
    {,Fibonacci'}
    {,Gauss    ,}
```

```
>>  C4 = strjust(S1,'right') %wyjustowanie napisów do prawej strony
C4 =
  3×1 cell array
    {'    Euler'}
    {'Fibonacci'}
    {'    Gauss'}
```

4.3. Tablice struktur

Tabela struktur (ang. structure array) jest tablicą wektorów (struktur o jednakowych zestawach pól), w których poszczególne pola (ang. fields) identyfikowane są za pomocą nazw. Poszczególne pola danej struktury a również pola o tych samych identyfikatorach w poszczególnych strukturach mogą zawierać dowolne typy danych.



UWAGA

Tabela struktur różni się od tablicy komórek identyfikacją komórek danych za pomocą nazw, a nie indeksów liczbowych.

W odróżnieniu od tablic liczbowych, wpisywaną zawartość tablic komórkowych ujmujemy w nawiasy klamrowe { }.

Tablicę struktur możemy zdefiniować na dwa sposoby:

1. Poprzez przypisanie danych – patrz Przykład 4.13.

Przykład 4.13.

```
>> ocena.Nazwisko = ‘Kowalski’;
>> ocena.Imie = ‘Jan’;
>> ocena.Ocena = 3
ocena =
  struct with fields:
    Nazwisko: ‘Kowalski’
        Imie: ‘Jan’
       Ocena: 3
```

2. Z użyciem funkcji struct – patrz Przykład 4.14.

Przykład 4.14

Zastosowanie funkcji struct do tworzenia tablicy struktur.

```
>> ocena = struct(‘Nazwisko’, ‘Kowalski’, ‘Imie’, ‘Jan’, ‘Ocena’, 3)
ocena =
  struct with fields:
    Nazwisko: ‘Kowalski’
        Imie: ‘Jan’
       Ocena: 3
```

Przykład 4.15.

```
>>  s.a1 = 3+3i;
>> s.a2 = ,trzy’;
>> s.a3 = magic(3)
s =
  struct with fields:
    a1: 3.0000 + 3.0000i
    a2: ‘trzy’
    a3: [3×3 double]
>> whos s
  Name      Size            Bytes  Class     Attributes
  s         1x1               624  struct
```

Przykład 4.16.

Zastosowanie funkcji struct do tworzenia tablicy struktur.

```
>>  s = struct('a1', 3+3i, 'a2', 'trzy', 'a3', magic(3))
s =
  struct with fields:
    a1: 3.0000 + 3.0000i
    a2: 'trzy'
    a3: [3×3 double]
```

4.4. Zadania



1. W poniższym tekście zamień duże litery na małe, a małe litery na duże.

Celem tych zajęć jest zapoznanie się ze środowiskiem MATLAB. Podstawowym zadaniem MATLABa jest wykonywanie złożonych obliczeń numerycznych i wizualizacja wyników. Może on wykonywać także symboliczne obliczenia algebraiczne (tak jak Mathematica), jednak należy pamiętać, że przeznaczony jest głównie do obliczeń numerycznych.



2. Zaproponuj tablicę struktur, która będzie przechowywała dane dotyczące wyników egzaminu z poniższej tabeli

Student ID	Liczba punktów	Ocena
Student1	34	2
Student2	56	3
Student3	89	4,5
Student4	90	5
Student5	23	2
Student6	78	4

1 2 3 4 5 6

5. Programowanie

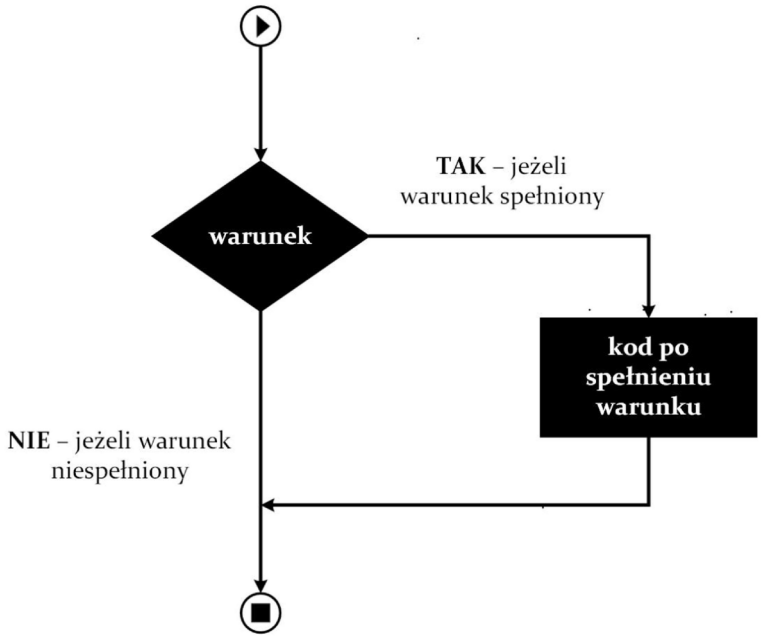


5.1. Instrukcja if

Instrukcja if pozwala na podjęcie określonych działań w zależności od spełnienia/niespełnienia określonych warunków. Niemal w każdym programie musimy sprawdzać, czy dane informacje są poprawne itp., dlatego z if'a korzysta się bardzo często. Instrukcje warunkowe pozwalają na wybranie bloku instrukcji, który zostanie wykonany. Instrukcja if oblicza wartości wyrażenia logicznego i w zależności od uzyskanego wyniku wykonuje właściwą grupę instrukcji – patrz Rysunek 8.

Najprostsza składnia instrukcji if wygląda następująco:

```
if wyrażenie_logiczne
   blok instrukcji
end
```



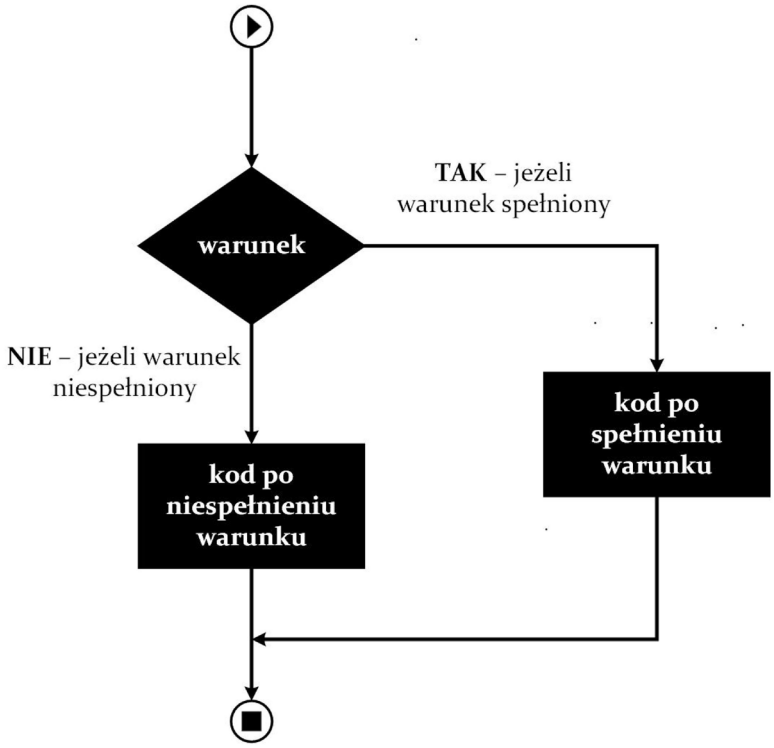
Rysunek 8. Składnia instrukcji if.

Przykład 5.1.

```
if (x>0) %sprawdź warunek za pomocą instrukcji if
disp('x dodatni') %jeśli warunek jest prawdziwy
end %koniec instrukcji warunkowej
```

Jeżeli wartość wyrażenia logicznego jest prawdą, wówczas wykonywany jest blok instrukcji pomiędzy if i end. Dodatkowe warunkowanie wykonania instrukcji if uzyskujemy dzięki wyrażeniom else i elseif:

```
if wyrażenie_logiczne_if
  blok instrukcji_if
else
  blok instrukcji_else
end
```



Rysunek 9. Składnia instrukcji if-else.

```
if wyrażenie_logiczne_if
  blok instrukcji_if
elseif wyrażenie_logiczne_elseif
  blok instrukcji_elseif
else
  blok instrukcji_else
end
```

Przykład 5.2.

```
if (x>0) %sprawdź warunek za pomocą instrukcji if
disp('x dodatni') %jeśli warunek jest prawdziwy
else %warunek if jest fałszywy
disp('x niedodatni')
end %koniec instrukcji warunkowej
```

Przykład 5.3.

Zastosowanie konstrukcji elseif. Sprawdź, czy liczba x jest dodatnia, ujemna czy równa zero i wyświetl odpowiedni komunikat.

```
if (x>0) %sprawdź warunek za pomocą instrukcji if
disp('x dodatni') %jeśli warunek jest prawdziwy
elseif (x<0)
disp('x ujemny')
else %wtedy x jest równy zero
disp('x równy zero')
end %koniec instrukcji warunkowej
```

5.2. Instrukcja switch

Struktura instrukcji wyboru switch składa się z serii przypadków case i opcjonalnego przypadku otherwise.



WAŻNE

Po słowie kluczowym switch następuje wyrażenie sterujące. Wartość tego wyrażenia jest porównywana z każdą z etykiet case. Jeśli wynik porównania jest prawdziwy, wykonane zostaną instrukcje po tym przypadku case. Instrukcja break powoduje, że wykonanie programu jest kontynuowane od pierwszej instrukcji po strukturze switch.

Składnia instrukcji jest następująca:

```
switch wyrażenie_logiczne
case wartość1
   instrukcje
case wartość2
  instrukcje
otherwise
  instrukcje
end
```

Przykład 5.4,

Zmiennej **ocena** przypisz odpowiadającą wartość w zależności od liczby zdobytych punktów.

Punty	Ocena
1	1
2	2
3	3
4	4
5	5

```
switch punkty
case 1
Ocena =  1
case 2
Ocena =  2
case 3
Ocena = 3
case 4
Ocena = 4
case 5
Ocena = 5
otherwise %wartość punktów była spoza dopuszczalnego przedziału
disp('dopuszczalne tylko punkty = 1, 2, 3, 4 lub 5 ')
end koniec instrukcji case
```

5.3. Pętle

W programowaniu pętla to jedna z trzech podstawowych konstrukcji programowania strukturalnego. Umożliwia cykliczne wykonywanie ciągu instrukcji aż do momentu zajścia warunku zakończenia pętli. Pętle dostępne w Matlabie: while, for.



UWAGA

Znajomość działania wszystkich pętli ułatwia znacznie czytanie kodu – różnią się one bowiem drobnymi detalami, które sprawiają, że każda pętla jest znacznie wygodniejsza od pozostałych w określonych sytuacjach.

Instrukcja while jest pętlą wykonującą cyklicznie umieszczony wewnątrz kod, pod warunkiem, że wyrażenie warunkowe zwraca wartość TRUE. Składnia instrukcji jest następująca:

```
while wyrażenie
        instrukcje
end
```

Przed każdym wykonaniem instrukcji następuje sprawdzenie warunku. Pętla zostanie przerwana, gdy wartość warunku będzie równa FALSE – patrz Rysunek 10.

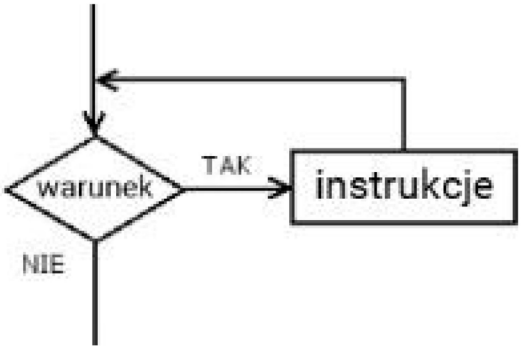
W celu przerwania wykonywania instrukcji należy wywołać instrukcje break wewnątrz instrukcji while.

UWAGA

Przestrzegamy przed pętlą nieskończoną, gdy warunek jest źle sformułowany. Jeśli program zawiesi się, można go przerwać przez CTRL+C.

Warto także zauważyć, że pętla może nie uruchomić się ani razu (gdy od samego początku warunek (warunki) będzie fałszywy).

Należy także zwrócić uwagę, że niepoprawna konstrukcja pętli może spowodować jej zapętlenie (wykonywanie bez końca).



Rysunek 10. Schemat działania pętli while.

Pętlę while w **praktyce stosuje się zawsze tam**, gdzie mamy potrzebę przetwarzania nieokreślonej liczby danych.

Przykład 5.5.

Wyświetl wartość zmiennej x od 5 do 0.

```
x = 6;
while x > 0
x = x - 1;
disp(x)
end %koniec instrukcji while
%uruchomienie instrukcji daje poniższy rezultat
5
4
3
2
1
0
```

Pętla for, podobnie jak wszystkie pozostałe pętle, umożliwi nam powtórzenie określonych operacji tak długo, jak warunek końcowy jest spełniony. Za pomocą pętli for możliwe jest powtarzalne wykonywanie bloków instrukcji. Jedno powtórzenie nazywamy iteracją.

Jeśli znana jest liczba iteracji, wykorzystuje się pętlę for. Pętla for ma następującą składnię:

```
for zmienna_iterowana = wyrażenie
   blok instrukcji
end
```

Pętlę for używamy praktycznie zawsze, gdy znamy ilość danych, jaką mamy wczytać, wypisać lub zmienić. Jeśli chcemy policzyć średnią z określonej liczby liczb, wczytać określoną ilość danych z pliku lub wypisać określoną ilość danych na ekran, pętla for jest do tego po prostu idealna.

Przykład 5.6.

Prosta instrukcja ilustrująca zastosowanie pętli for do obliczania sumy.

```
n = 10; %zmienna n wynosi 10
i  = rand(1,n);%losowa wartość
suma = 0;
for k = 1 : n
suma =suma + i(k);
end
suma
```

Zagnieżdżenie pętli, podobnie jak instrukcji warunkowej, polega na wywołaniu jednej pętli wewnątrz drugiej. Oznacza to, że na jedną iterację pętli zewnętrznej zostanie wykonany cały przebieg pętli wewnętrznej. Instrukcje iteracyjne w Matlabie można dowolnie zagnieżdżać. Im więcej zagnieżdżeń, tym większa złożoność obliczeniowa algorytmu.

Przykład 5.7.

Opracowany algorytm zawiera dwie pętle for.

```
for k  = 1 : 3 %pętla zagnieżdżona od 1 do 3
for n = 1 : 2 %druga pętla od 1 do 2
suma =  k + n;
disp( suma)
end %koniec drugiej pętli
end %koniec pierwszej pętli
    2       %wynik to 6-elementowa kolumna
    3
    3
    4
    4
    5
```

5.4. Instrukcja break/continue

Instrukcja break powoduje natychmiastowe zakończenie wykonywania pętli; for, while, oraz instrukcji switch.

Jeżeli mamy do czynienia z **kilkoma pętlami**, zagnieżdżonymi jedna wewnątrz drugiej, to instrukcja break powoduje przerwanie tylko tej najbardziej wewnętrznej pętli, w której występuje. To przerwanie pętli z wyjściem o jeden poziom wyżej.

Przykład 5.8.

Zastosowanie instrukcji break.

```
>> x = 1;
%pętla while
while (x < 10 ) %dopóki x jest mniejsze od 10
fprintf('wartość x: %d\n', x);
x = x+1; %zwiększ x o jeden
if( x > 5)
% teraz zakończymy pętle, stosując instrukcję break
break;
end %koniec instrukcji if
end %koniec pętli while
%poniżej wynik działania kodu
wartość x: 1
wartość x: 2
wartość x: 3
wartość x: 4
wartość x: 5
```

Instrukcja continue występuje zwykle wewnątrz pętli **while, for**. Powoduje ona zaniechanie wykonywania instrukcji będących treścią pętli, jednak sama pętla nie zostaje przerwana. Instrukcja continue przerywa tylko bieżący obieg pętli i zaczyna następny obieg, kontynuując pracę pętli.

Przykład 5.9.

Zastosowanie instrukcji continue.

```
>> x = 1;
%rozpoczynamy pętlę while
while x < 10 %dopóki x jest mniejszy id 10
if x == 5
% omiń ten etap iteracji
x = x + 1; % zwiększ x o jeden
continue;
end %koniec instrukcji if
fprintf('wartość x: %d\n', x);
x = x + 1; % zwiększ x o jeden
end %koniec instrukcji while
%poniżej wynik działania kodu
wartość x: 1
wartość x: 2
wartość x: 3
wartość x: 4
wartość x: 5
wartość x: 6
wartość x: 7
wartość x: 8
wartość x: 9
```

5.5. Skrypty

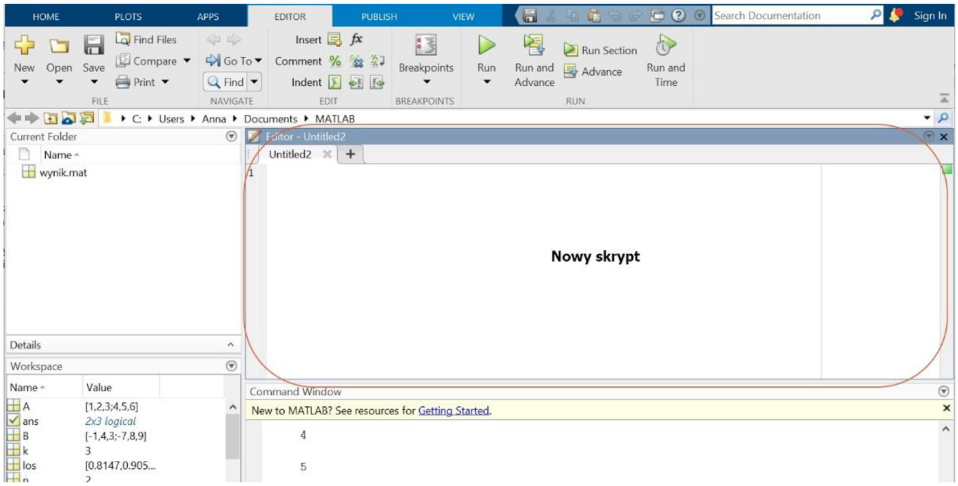
Skrypty zawierają sekwencje poleceń Matlaba, które są wykonywane kolejno linia po linii, począwszy od początku pliku, tak jakby każde z nich zostało wprowadzone i zlecone do wykonania w *Oknie Poleceń*.

Aby otworzyć nowy skrypt, klikamy zakładkę *Home*, a następnie *New Script* – patrz Rysunek 11. Wówczas Matlab otworzy kolejną zakładkę dla nowego skryptu – patrz Rysunek 12.

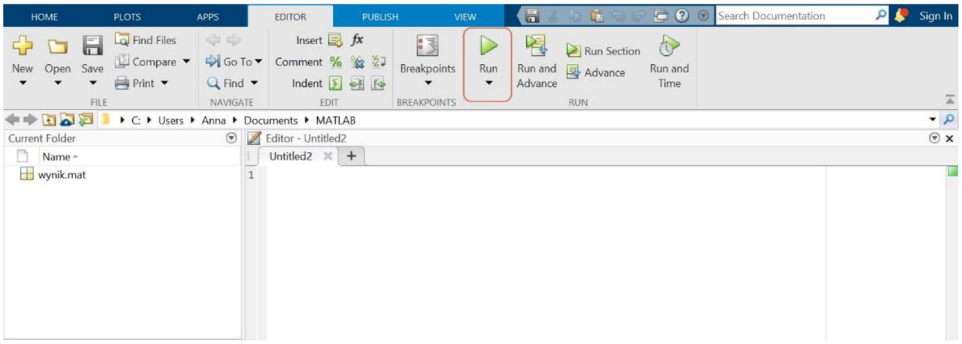
Po wpisaniu treści m-pliku (oraz zapisaniu go pod nazwą z rozszerzeniem '.m'), można go natychmiast uruchomić za pomocą narzędzia *Run* na zakładce *Editor Paska Narzędzi* (Rysunek 13) lub naciskając klawisz *F5*.



Rysunek 11. Tworzenie skryptu.



Rysunek 12. Nowy skrypt.



Rysunek 13. Uruchomienie skryptu.

Przykład 5.10.

Uruchom skrypt o poniższej treści i zapisz go do pliku Przyklad1.m.

```
A = [1 2 3;  4 5 6]; %zdefiniowanie macierzy
B = [7 8 9; 10 11 12]; %zdefiniowanie macierzy
A+B %skrypt oblicza sumę zdefiniowanych macierzy
%skrypt należy zapisać do pliku Przyklad1.m
%uruchamiamy skrypt z konsoli
>> Przyklad1
ans =
    8    10    12
   14    16    18
```

Poniżej czytelnik znajdzie kilka prostych skryptów – patrz Przykłady 5.11 – 5.13.

Przykład 5.11.

Skrypt z zastosowaniem instrukcji warunkowej if – else. Skrypt należy zapisać w pliku *.m oraz uruchomić go z poziomu konsoli.

```
%skrypt wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik
%na ekran
x = input(‚Podaj liczbę rzeczywistą\nx = ,);
if x<0
 disp(‘liczba ujemna’)
else
 disp(‘liczba nieujemna’)
end %koniec instrukcji if
```

Przykład 5.12.

Skrypt z zastosowaniem instrukcji warunkowej if – elseif – else. Skrypt należy zapisać w pliku *.m oraz uruchomić go z poziomu konsoli.

```
%skrypt wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik
%na ekran
x = input(,Podaj liczbę rzeczywistą\nx = ,);
if x<0
 disp('liczba ujemna')
elseif x == 0
    disp(,x jest zerem')
else
 disp(,liczba dodatnia')
end
```

Przykład 5.13.

Skrypt z zastosowaniem pętli for. Skrypt należy zapisać w pliku *.m oraz uruchomić go z poziomu konsoli.

```
n = input('Podaj liczbę naturalną n\nn = '); %Wczytanie liczby naturalnej
for zmienna = -10:1:n, %pętla for od -10 do n z krokiem 1
disp(zmienna) %wypisanie kolejnych wartości zmiennej
end %koniec instrukcji for
```

5.6. Funkcje

Jednym z rodzajów m-plików są pliki funkcyjne, zwane skrótowo funkcjami. Definicja własnej funkcji musi zaczynać się od słowa kluczowego function i dalej zawiera:

- nazwę zmiennej zwracającej wartość obliczoną przez funkcję np. MojaFunkcja. Jeżeli takich zmiennych jest więcej, ich listę umieszcza się w nawiasie kwadratowym [ ],
- znak podstawienia (=),

- wybraną przez programistę nazwę funkcji z umieszczoną w nawiasach półokrągłych listą parametrów wejściowych, od których funkcja zależy np. global i j k,
- dobrym zwyczajem jest zamieszczenie komentarza informującego, co ta funkcja robi, jakie znaczenie mają parametry wejściowe itp.,
- potem jest ciąg instrukcji do wykonywania przez funkcję, czyli tzw. ciało funkcji,
- definicję funkcji kończy słowo kluczowe end.

function [zmienne_wyjsciowe] = nazwa_funkcji(zmienne_wejsciowe)

Kończąc budowanie funkcji, musimy ją zapisać w m-plikach. Pamiętajmy, aby nazwy funkcji były takie same, jak nazwy plików, w których są zapisane. Taka systematyka bardzo ułatwi pracę z wieloma funkcjami. Dobrym zwyczajem jest też komentowanie pisanego kodu.

Wywołując funkcję, podstawiamy jej wartość pod zmienną np. x = MojaFunkcja(1,2,3). Po wywołaniu funkcji z argumentami wejścia w oknie *Command Window* otrzymamy wynik operacji podstawiony pod zmienną domyślną ans.

Przykład 5.14.

Tworzymy skrypt Moja.m obliczający sumę funkcji sinus oraz cosinus. Funkcję należy zapisać w pliku *.m oraz uruchomić go z poziomu konsoli.

```
function  Suma   =  Moja(x, y)
% oblicza sumę funkcji trygonometrycznych
Suma = sin(x) + cos(y)
End %koniec funkcji
```

Po zapisaniu tego kodu w pliku Moja.m korzystamy z tej funkcji, wywołując ją z linii komend.

```
>> z=Moja(0,1) %uruchomienie funkcji z wartości 0 oraz 1
Suma =
   0.5403
z =
   0.5403
```

Przykład 5.15.

Napiszemy teraz funkcję, która oblicza pierwiastki dla zadanego równania kwadratowego. Funkcję zapisujemy w pliku fkwadratowa.m, a jej działanie sprawdzamy z poziomu konsoli.

```
function [x1, x2] = fkwadratowa(a, b, c)
% funkcja ma dwie wartości, dlatego użyliśmy nawiasów kwadratowych
delta = b^2 - 4*a*c; %obliczanie delty równania
x1 = (-b-sqrt(delta))/2*a; %wzór na pierwszy pierwiastek równania
x2 = (-b+sqrt(delta))/2*a; %wzór na drugi pierwiastek równania
end %koniec funkcji
```

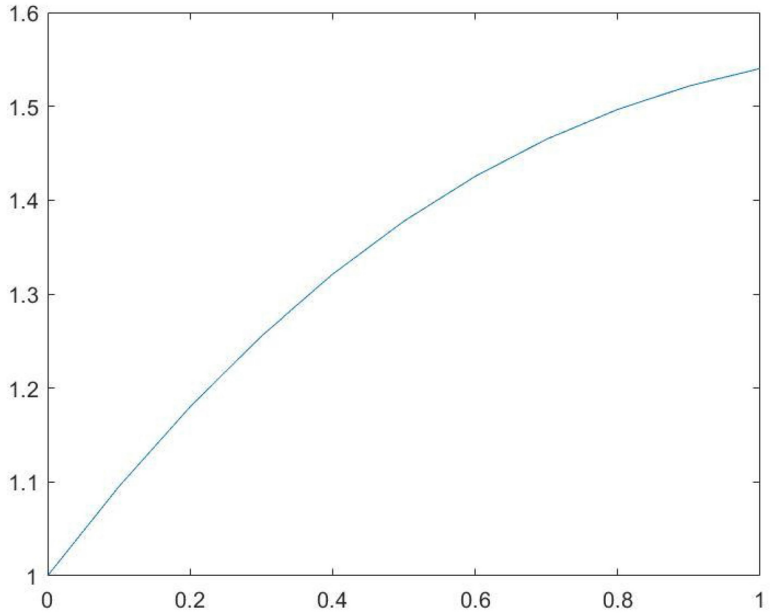
Jeśli funkcje nie są zbyt skomplikowane, to można zastosować szybszy sposób programowania, np. za pomocą tzw. funkcji anonimowych. Są one tworzone i przechowywane poprzez uchwyty funkcji (czyli zmienne będące identyfikatorem funkcji). Funkcje anonimowe tworzy się poleceniem

nazwa_funkcji = @(lista_wejsciowa) wyrazenie_matematyczne

gdzie nazwa_funkcji jest uchwytem funkcji. Lista wejściowa może zawierać pojedynczą zmienną lub wiele zmiennych oddzielonych przecinkami.

Przykład 5.16.

```
funkcjaAnonimowa = @(x) cos(x) + x;
z=funkcjaAnonimowa(10); % użycie funkcji dla pojedynczego argumentu
x=0:0.1:1; %x przyjmuje wartości od 0 do 1 z krokiem 0.1
y=funkcjaAnonimowa(x); % użycie funkcji dla wektora
plot(x,y) %wykres
%wywołanie funkcji
```





5.7. Zadania

1. Omów, czym różni się funkcja od skryptu. O czym należy pamiętać, zapisując skrypty w Matlabie?

.....
.....
.....
.....
.....
.....
.....
.....

2. Zapisz w pliku dane.txt 10-elementowy wektor liczb pseudolosowych.

.....
.....
.....
.....
.....
.....
.....
.....

3. Oblicz wartość funkcji f, określonej wzorem:

$$f(x) = \begin{cases} x^2 - 2, x < 0 \\ 5, x = 0 \\ x + 3, x > 0 \end{cases}$$

Użyj instrukcji warunkowej if.

.....
.....
.....
.....
.....
.....
.....
.....

4. Oblicz wartość funkcji f, określonej wzorem:



$$f(x) = \begin{cases} -1, x < 0 \\ 0, x = 0 \\ 1, x > 0 \end{cases}$$

Użyj instrukcji case.

5. Napisz funkcję sprawdzającą, czy dana liczba jest parzysta.

6. Napisz funkcję, która wczytuje z klawiatury liczbę, aż do chwili, gdy będzie ona należała do przedziału.



7. Napisz instrukcję switch wyświetlającą na podstawie zmiennej całkowitej nr nazwę miesiąca słownie.

……
……
……
……
……
……
……

8. Napisz instrukcję, która na podstawie zmiennej całkowitej ocena wyświetla jedną z informacji: brak promocji do następnej klasy, promocja do następnej klasy, promocja z ocena celująca

……
……
……
……
……
……
……

9. Napisz instrukcję switch, która wyznaczy i zwróci jako wartość ocenę na podstawie punktów zdobytych za kolokwium według następujących reguł:

0-10 pkt – 2,
11-13 pkt – 3,
14-16 pkt – 4,
17-18 pkt – 5.

……
……
……
……
……
……
……
……

10. Napisz instrukcję, która liczy sumę n początkowych liczb nieparzystych.



11. Napisz instrukcję, która liczy sumę n początkowych liczb naturalnych podzielnych przez 5 oraz 40.

12. Napisz instrukcję, która liczy sumę n początkowych liczb naturalnych kończących się jedną z cyfr: 2, 3 lub 9.



13. Napisz skrypt, który będzie generował losowo liczbę całkowitą z przedziału [-1000, 1000] i będzie wyprowadzał informację, czy liczba jest parzysta czy nieparzysta.

14. Mając dany wektor x=[1 5 7 2 -1 0 -15 -9 5 2], napisz polecenia wykonujące następujące operacje:

a) zamień dodatnie elementy wektora x na zera,
b) pomnóż nieparzyste elementy wektora x przez 5,
c) utwórz wektor y złożony z elementów x pomnożonych przez -9,
d) wymień zerowe elementy na liczbę 100,
e) policz sumę kwadratów elementów wektora x.

Wykonując powyższe polecenia, zastosuj pętle.

15. Napisz skrypt, który oblicza objętość oraz pole powierzchni całkowitej prostopadłościanu. Długość, szerokość i wysokość powinny być wprowadzane z klawiatury.



16. Opracuj funkcje silnia(n), która policzy i wyświetli wartość silni dla parametru n.

17. Napisz funkcję ZamianaNaWielkie(napis), której parametrem wejściowym będzie łańcuch znakowy napis zawierający ciąg dowolnych liter, a wyjściowym drugi łańcuch znakowy. Funkcja ma przekształcić wejściowy łańcuch znakowy napis tak, aby uzyskany w wyniku przekształcenia łańcuch wyjściowy składał się z takiej samej sekwencji liter, ale aby były to jedynie wielkie litery, niezależnie od wielkości liter użytych w łańcuchu wejściowym napis.

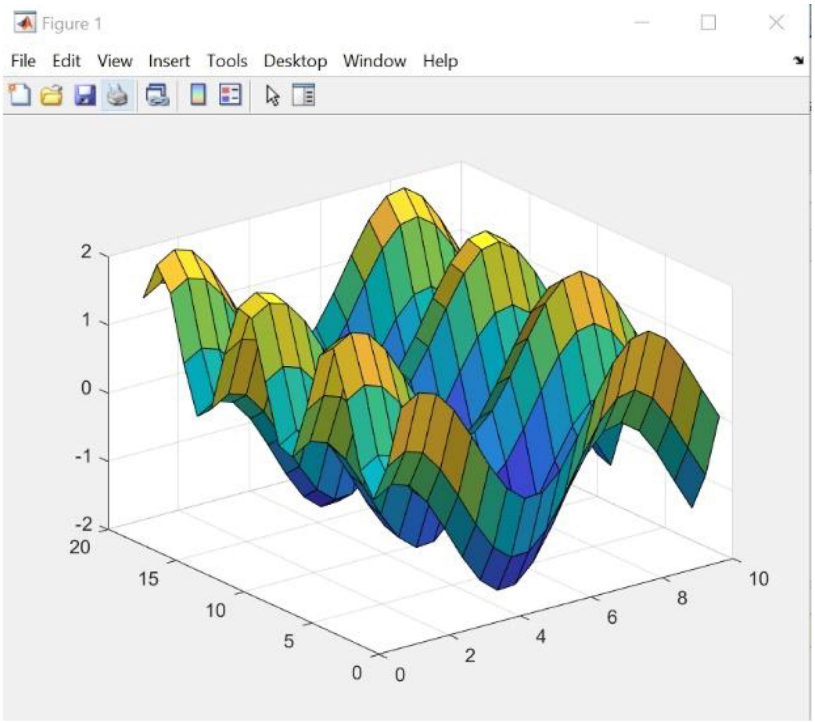
1 2 3 4 5 6

6. Wykresy i grafika



6.1. Wstęp

Grafika w Matlabie wyświetlana jest w oknach graficznych. Jednocześnie może być otwartych kilka okien graficznych, ale **tylko jedno z nich jest oknem aktywnym** (czyli takim, w którym wyświetlane są wyniki działania funkcji graficznych). Jeśli żadne okno graficzne nie jest otwarte, to funkcje graficzne automatycznie tworzą nowe. Każde okno graficzne ma unikalny numer wyświetlany w jego nagłówku, np. Figure 1, Figure 2 – patrz Rysunek 14.



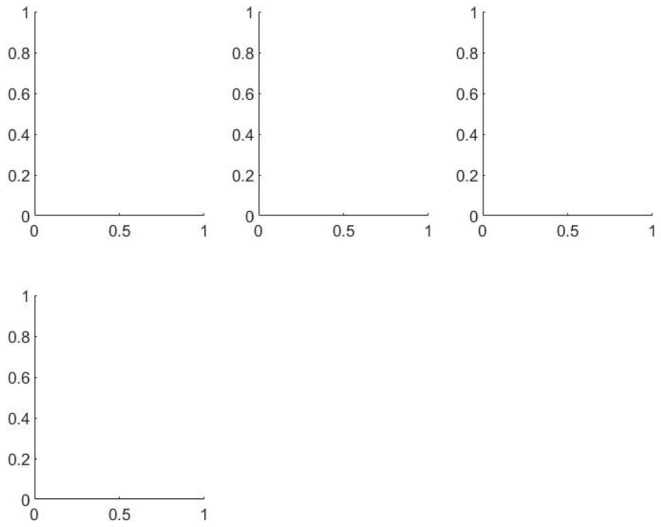
Rysunek 14. Przykład okna graficznego.

Jedno okno graficzne możemy podzielić na kilka mniejszych, stosując funkcję subplot (n,m,p). Funkcja ta dzieli okno graficzne na mniejsze prostokątne okienka, umieszczając je w n-wierszach i m-kolumnach, p - numer aktywnego okienka, w każdym okienku można umieścić odrębny wykres; okna graficzne są numerowane od lewej do prawej, wierszami od góry do dołu; funkcja ta służy także do przełączania się pomiędzy wykresami w podzielonym oknie graficznym

Przykład 6.1.

Podziel okno wykresu na cztery podokna.

```
subplot(2,3,1)
subplot(2,3,2)
subplot(2,3,3)
subplot(2,3,4)
```



Do obsługi okien graficznych stosujemy funkcje opisane w Tabeli 33.

Tabela 33. Obsługa okien graficznych.

Funkcja	Opis
figure	tworzy nowe okno graficzne
figure(n)	tworzy nowe okno graficzne o numerze n
close	zamyka aktywne okno graficzne
close(n)	zamyka aktywne okno graficzne o numerze n
close all	zamyka wszystkie okna graficzne
clf	czyści zawartość aktywnego okna graficznego

6.2. Grafika dwuwymiarowa

W programie Matlab mamy szereg funkcji pozwalających na graficzne przedstawienie danych – patrz Tabela 34.

Tabela 34. Funkcje do grafiki dwuwymiarowej.

Polecenie	Opis
area	wykres funkcji z wypełnionym obszarem pod nią
bar	wykres słupkowy
barth	wykres słupkowy poziomy
comet	animowany wykres funkcji typu ogon komety
compass	wykres przedstawiony na tarczy kompasu
contour	wykres poziomic
errorbar	wykres funkcji zaznaczając jednocześnie odchylenie
ezplot	wykres dowolnej funkcji
fplot	wykres zadanej funkcji wbudowanej lub zewnętrznej
hist	wykres histogramu w prostokątnym układzie współrzędnych
logplot	wykres, w którym obie osie są logarytmiczne
pie	wykres kołowy
plot	wykres z dwiema osiami liniowymi
plotyy	wykres z dwiema różnymi osiami y
rose	histogram w biegunowym układzie współrzędnych
scatter	wykres punktowy
semilogx	wykres z logarytmiczną skalą osi x i liniowa skalą osi y
semilogy	wykres z liniową skalą osi x i logarytmiczną osi y
stairs	wykres schodkowy
stem	odległość wartości y od osi x

Dla wymienionych funkcji w Tabeli możemy stosować różnego rodzaju style linii, punkty oraz kolory – patrz Tabela 35.

Tabela 35. Atrybuty funkcji.

Symbol	Opis
-	linia ciągła
--	linia przerywana
:	linia kropkowana
-	linia przerywano-kropkowana
+	punkt w stylu znak plus
*	punkt w stylu gwiazdka
.	punkt w stylu kropka
x	punkt w stylu krzyżyk
s	punkt w stylu kwadrat
d	punkt w stylu romb
^	punkt w stylu trójkąt
p	punkt w stylu pentagrama
h	punkt w stylu hexagrama
r	kolor krzywej lub punktu czerwony
g	kolor krzywej lub punktu zielony
b	kolor krzywej lub punktu niebieski
c	kolor krzywej lub punktu cyan
m	kolor krzywej lub punktu megenta
y	kolor krzywej lub punktu żółty
k	kolor krzywej lub punktu czarny
w	kolor krzywej lub punktu biały

Wykresy uzyskane za pomocą wyżej wymienionych funkcji mogą być uzupełniane różnymi elementami opisującymi wykresy za pomocą tekstu i innych elementów objaśniających. Służą do tego funkcje opisane w Tabeli 36, jak również proste opisy można dodawać, używając *Figure Palette*.

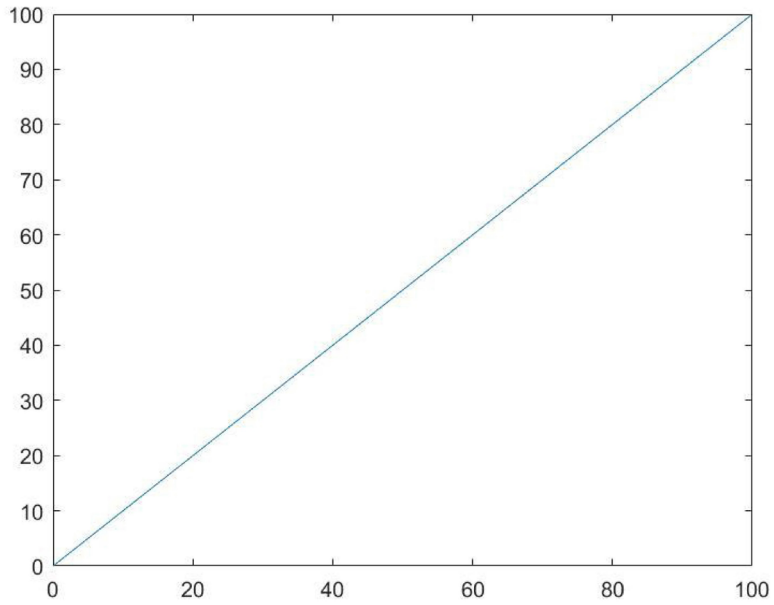
Tabela 36. Atrybuty funkcji graficznych.

Funkcja	Opis
grid	wprowadzenie lub ukrycie na wykresie siatki prostokątnej Funkcja może mieć argumenty: • grid on – wprowadzenie siatki głównej, • grid off – ukrycie siatki, • grid minor – wprowadzenie lub ukrycie siatki pomocniczej
axis	ustawienie lub pobranie parametrów osi bieżącego wykresu Funkcja może mieć argumenty: • axis on/off – wyświetlenie lub ukrycie osi, • axis equal – zrównanie jednostek na obu osiach, • axis square – wykres na obszarze kwadratowym, • axis normal – usunięcie działania equal i square, • axis auto/manual – włączenie lub wyłączenie automatycznego doboru podziałki na osiach, • axis state – zwraca informację o ustawieniach osi. • axis([xmin xmax ymin ymax]) – określenie ręczne zakresów osi, • axis xy/ij – osie w układzie kartezjańskim lub „macierzowym”.
hold	włączenie lub wyłączenie „zamrożenia” rysunku, do umieszczenia na nim kolejnych wykresów Funkcja może mieć argumenty: • hold on – włączony tryb dodawania nowych wykresów, • hold off – włączony tryb zastępowania istniejącego wykresu nowym.
figure(h)	wybór rysunku o identyfikatorze *h* jako bieżącego; jeśli rysunek nie istnieje, to zostaje zainicjowany pusty bieżący rysunek
title(‚*napis*’)	wprowadzenie tytułu na wykresie
xlabel(, *napis* ‘)	wprowadzenie opisu osi x
ylabel(, *napis* ‘)	wprowadzenie opisu osi y
legend(, *napis 1*’,’ *napis 2*’...)	wprowadzanie pola legendy na bieżącym rysunku

W przykładach zilustrowano wybrane wykresy opisane w Tabeli 31, wraz z możliwością zastosowania różnego rodzaju linii itd. (Tabela 34 oraz Tabela 35).

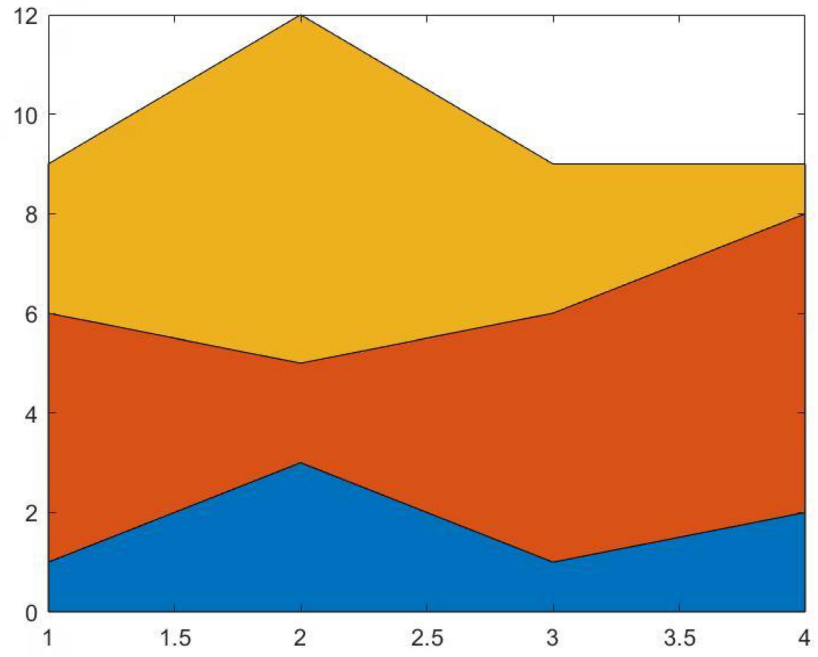
Przykład 6.2.

```
>> x = [0:5:100]; %zakres liczbowy zmiennej x od 0 do 100 z krokiem 5
y = x; %funkcja liniowa
plot(x, y) %wykres funkcji
```



Przykład 6.3.

```
>> Y = [1, 5, 3;
     3, 2, 7;
     1, 5, 3;
     2, 6, 1]; %dane wejściowe
figure
area(Y)
```

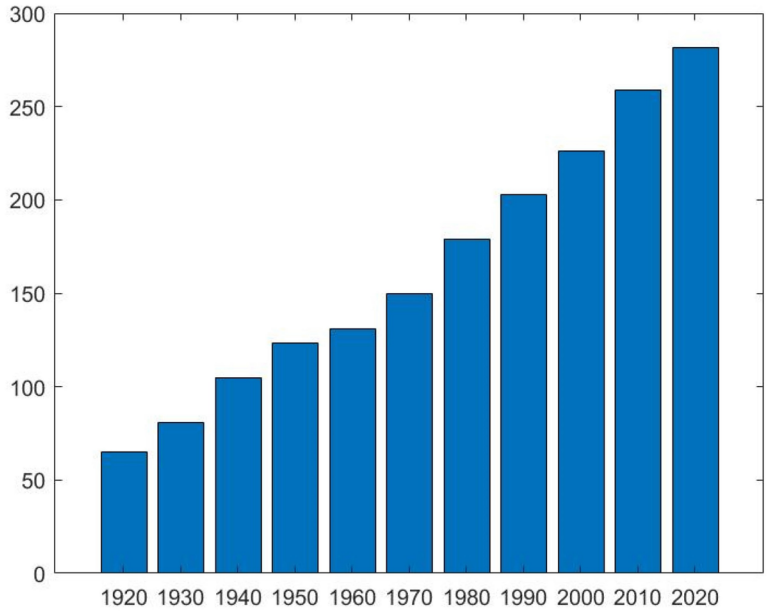


Przykład 6.4.

Zilustruj poniższe dane dotyczące liczby sklepów mięsnych (w tys.) w Lublinie.

Rok	1920	1930	1940	1950	1960	1970	1980	1990	2000	2010	2020
Liczba	65	81	105	123.5	131	150	179	203	226	259	281.5

```
x = 1920:10:2020; %zakres czasowy od roku 1920 do 2020 z krokiem 10 lat
y = [65 81 105 123.5 131 150 179 203 226 259 281.5]; %dane wejściowe
bar(x,y)
```

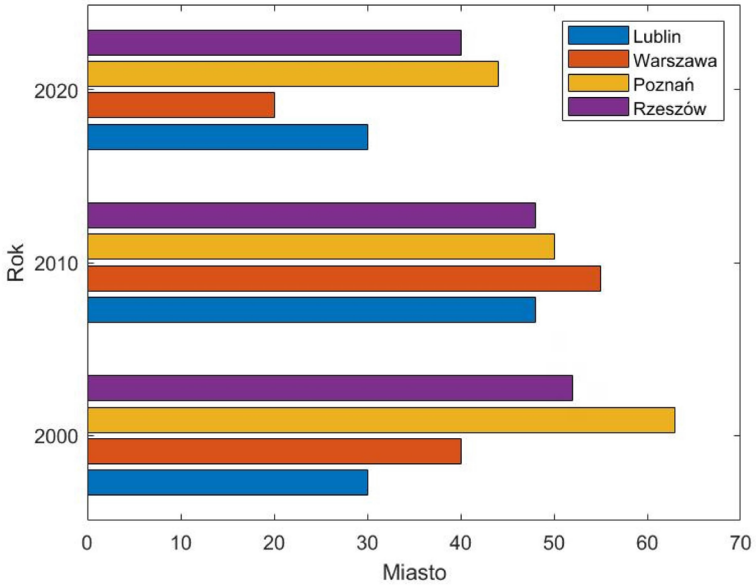


Przykład 6.5.

Zilustruj poniższe dane dotyczące liczby sklepów budowlanych w poszczególnych miastach.

Lublin		Warszawa		Poznań		Rzeszów	
Rok	Liczba sklepów	Rok	Liczba sklepów	Rok	Liczba sklepów	Rok	Liczba sklepów
2000	30	2000	40	2000	63	2000	52
2010	48	2010	55	2010	50	2010	48
2020	30	2020	20	2020	44	2020	40

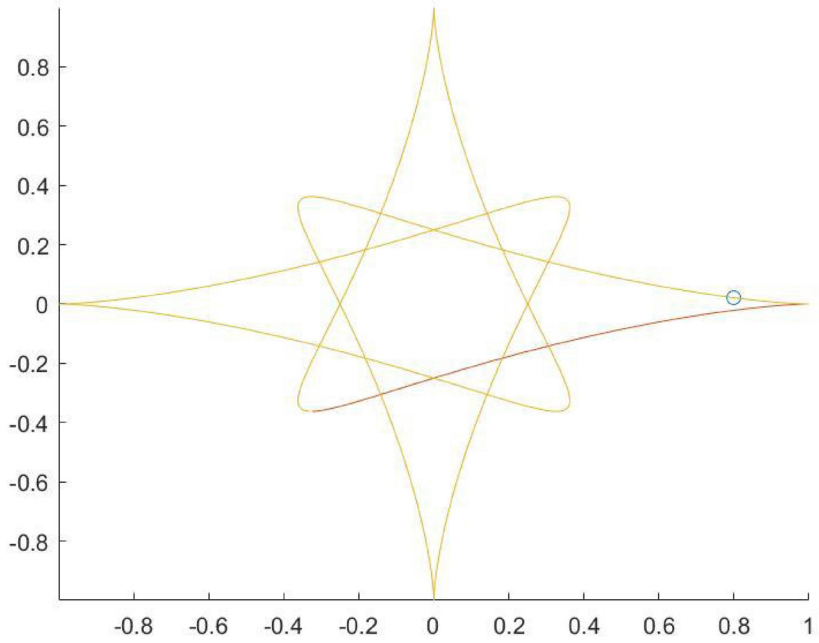
```
>> x = [2000 2010 2020]; %zakres czasowy w latach
y = [30 40 63 52; 48 55 50 48; 30 20 44 40]; %dane wejściowe
barh(x,y)
xlabel('Miasto')
ylabel('Rok')
legend({'Lublin','Warszawa','Poznań','Rzeszów'})
```



Przykład 6.6.

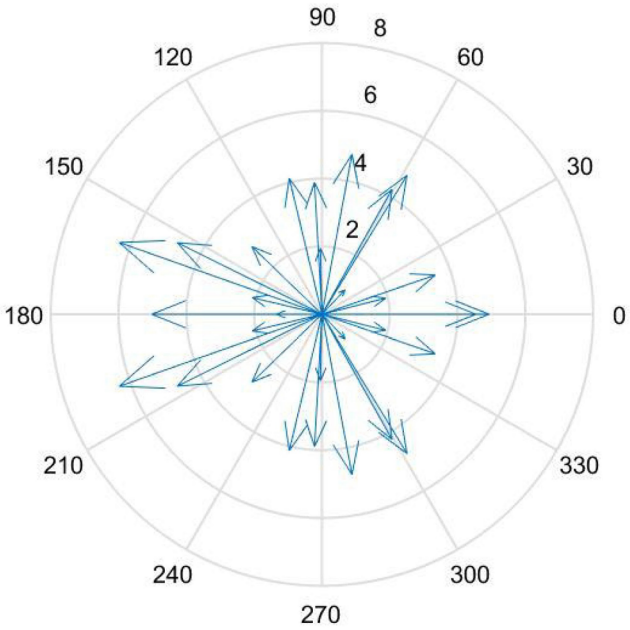
Narysuj wykres funkcji

```
>> t = 0:.01:3*pi;
x = cos(3*t).*(cos(t).^2);
y = sin(3*t).*(sin(t).^2);
comet(x,y);
```



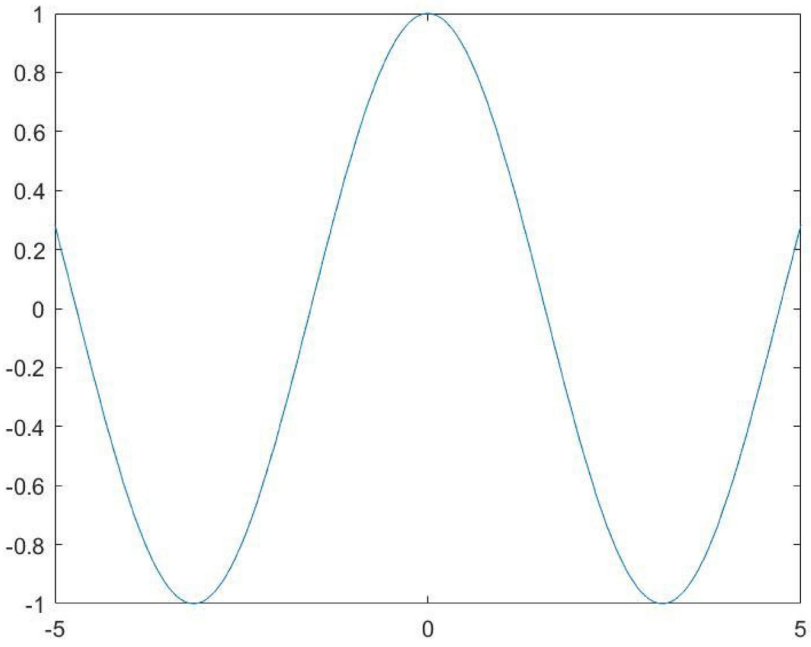
Przykład 6.7.

```
>> rng(0,'twister') % inicjacja losowych liczb
M = randn(30,30);
Z = eig(M);
figure
compass(Z)
```



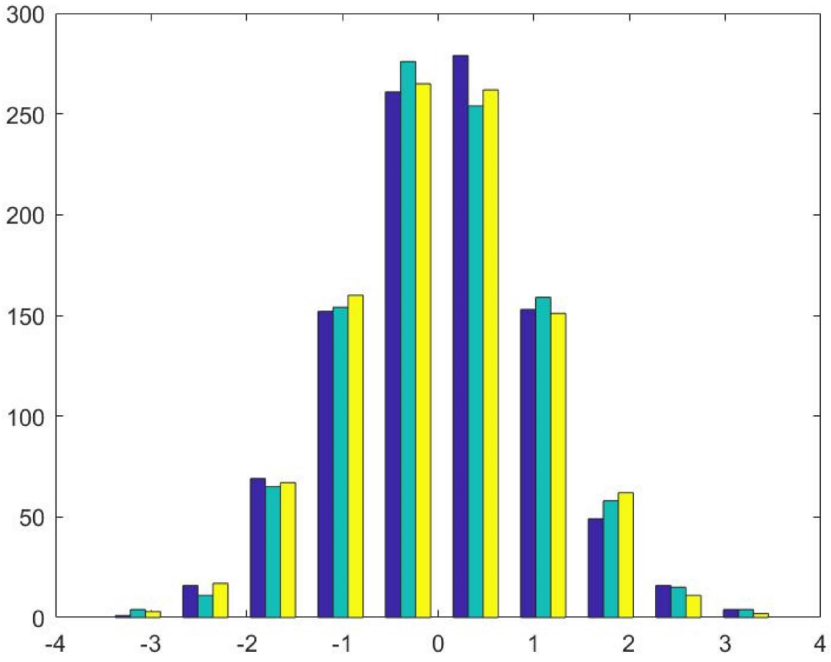
Przykład 6.8.

```
>> fplot(@(x) cos(x))
```



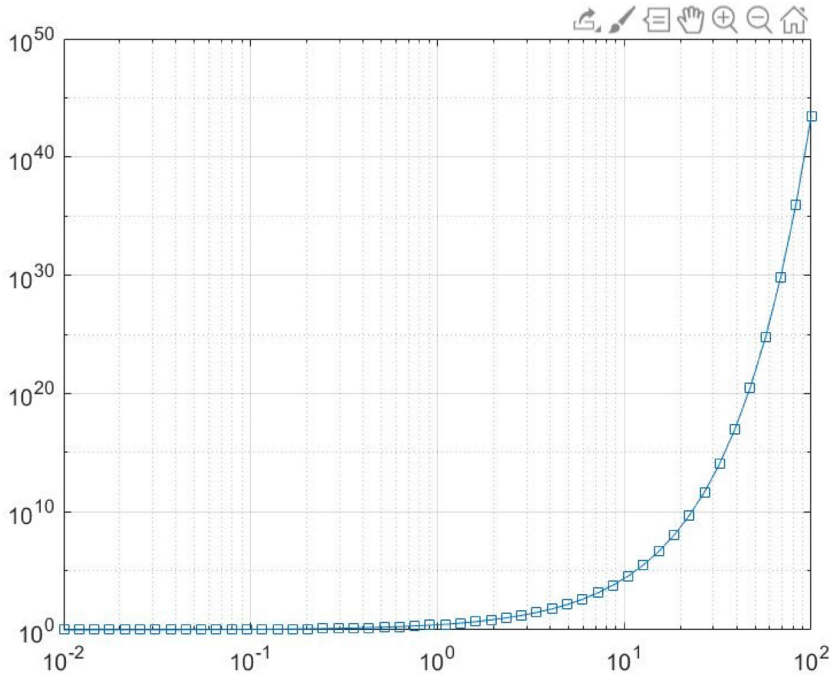
Przykład 6.9.

```
>> x = randn(1000,3); %generator liczb losowych
hist(x)
```



Przykład 6.10.

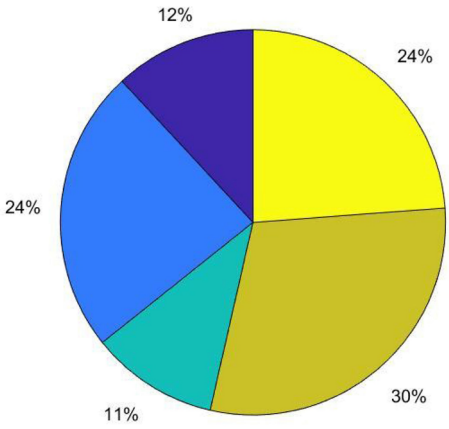
```
>> x = logspace(-2,2);
y = exp(x);
loglog(x,y,'-s')
grid on
```



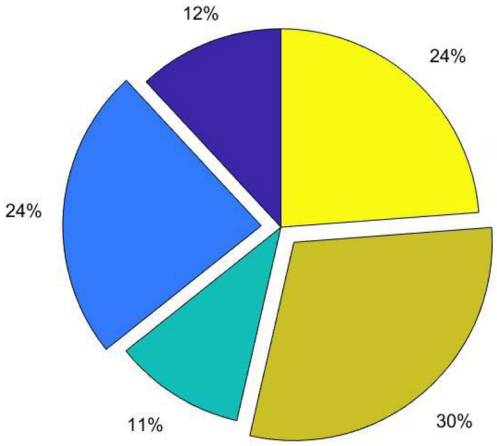
Przykład 6.11.

Zaproponuj reprezentację graficzną dla poniższych danych dotyczących dziennej sprzedaży (w tys. kg) owoców w pewnym sklepie spożywczym w Lublinie: jabłka (2.5), gruszki (1), banany (2), winogrona (2), pomarańcze (0.9).

```
>> X = [1 2 0.9 2.5 2]; %wczytanie danych wejściowych
pie(X)
```



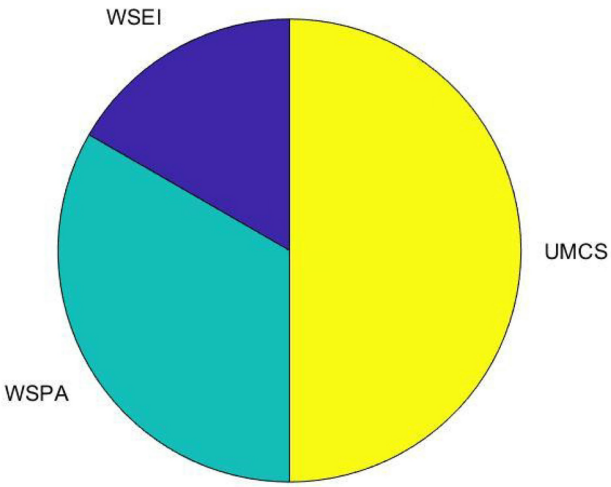

```
>> X = [1 2 0.9 2.5 2];
pie(X)
>> explode = [0 1 0 1 0];
pie(X,explode)
```



Przykład 6.12.

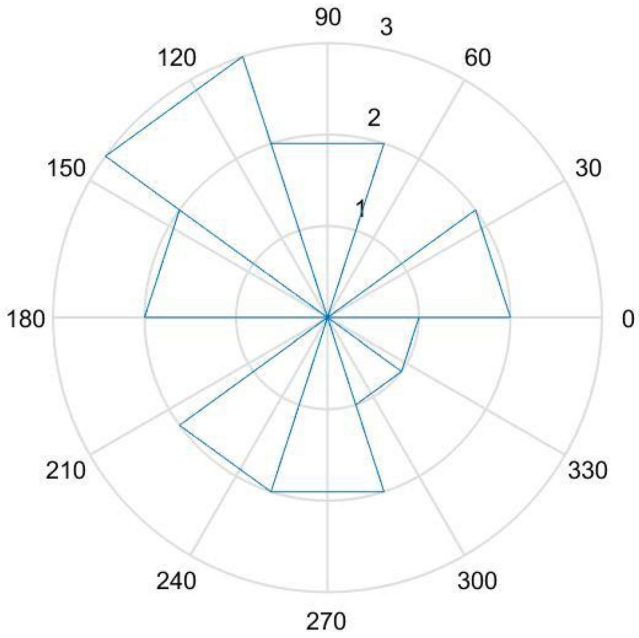
Wiedząc, że liczba studentów kierunku Ekonomia w WSEI jest dwa razy mniejsza niż w WSPA oraz trzy razy mniejsza niż na UMCS, zaproponuj reprezentację graficzną tych danych.

```
>> X = 1:3;
labels = {'WSEI','WSPA','UMCS'}; %etykiety uczelni lubelskich
pie(X,labels)
```



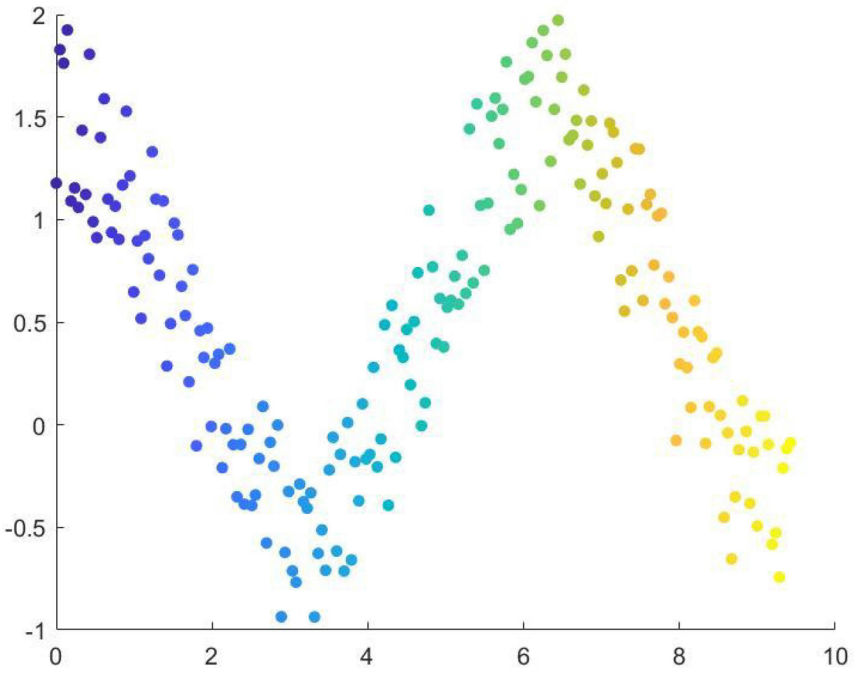
Przykład 6.13.

```
>> x = [0.4 1.4 3.1 2.3 0.4 2.5 3.9 2.8 2.3 1.6 4.6 4.5 6.1 3.9 5.1]; %dane
wejściowe
rose(x,10)
```



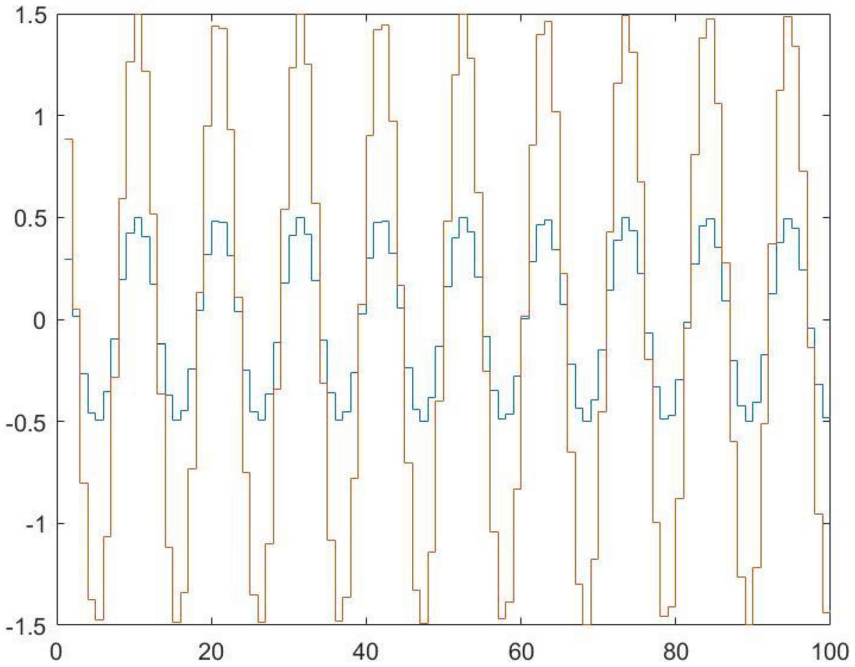
Przykład 6.14.

```
>> x = linspace(0,3*pi,200);
y = cos(x) + rand(1,200);
sz = 25;
c = linspace(1,10,length(x));
scatter(x,y,sz,c,'filled')
```



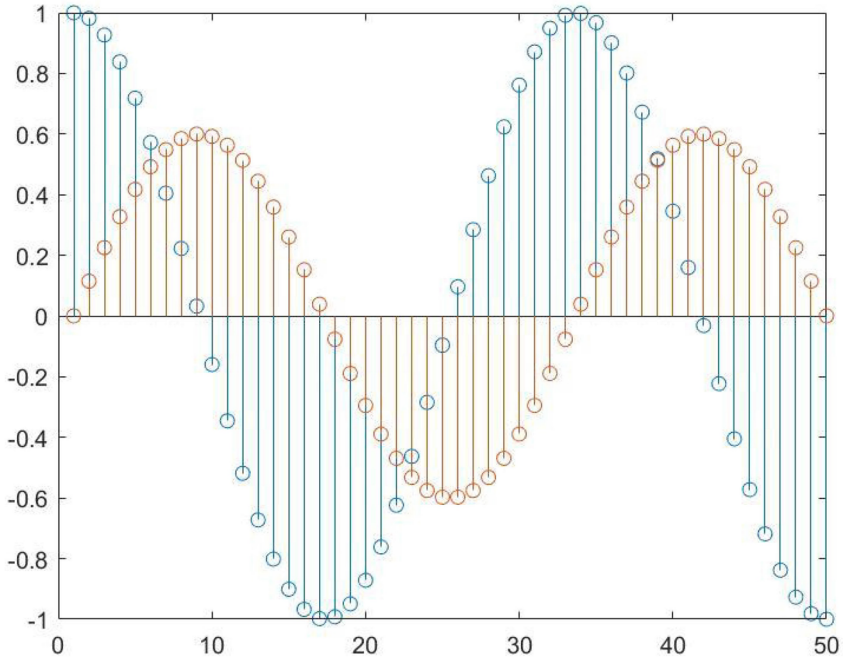
Przykład 6.15.

```
>> X = linspace(0.3*pi,60)';
Y = [0.5*cos(X), 1.5*cos(X)];
figure
stairs(Y)
```



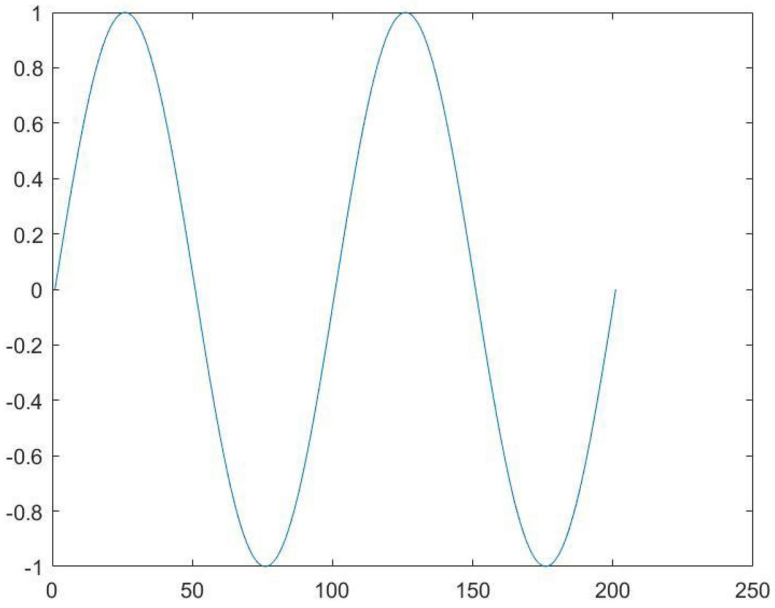
Przykład 6.16.

```
>> figure
X = linspace(0,3*pi,50)';
Y = [cos(X), 0.6*sin(X)];
stem(Y)
```



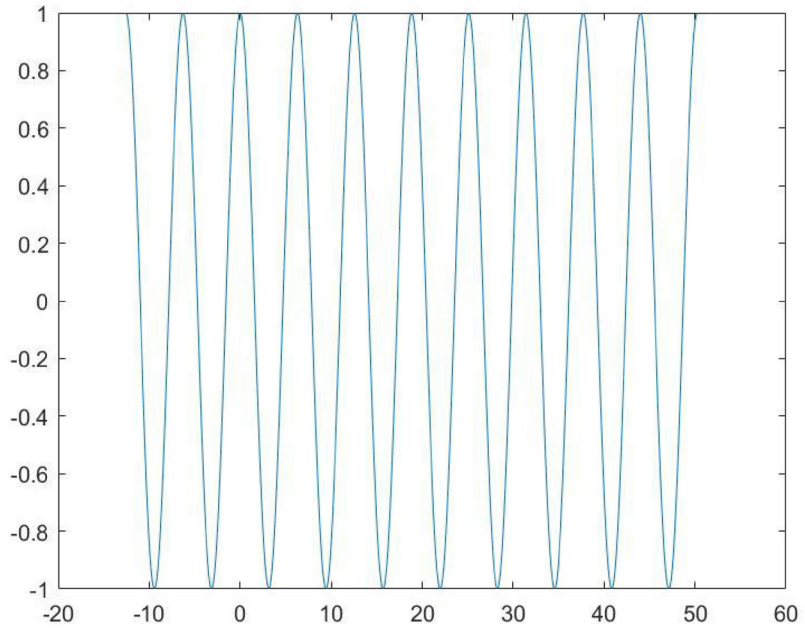
Przykład 6.17.

```
>> t = 2*[-pi:pi/100:pi]; y = sin(t); plot(y);
```



Przykład 6.18.

```
>> t = 4*[-pi:pi/100:4*pi]; y = cos(t); plot(t,y);
```



Liniom wykresu możemy nadać różne właściwości, które są określone przez argumenty funkcji plot – patrz Tabela 37.

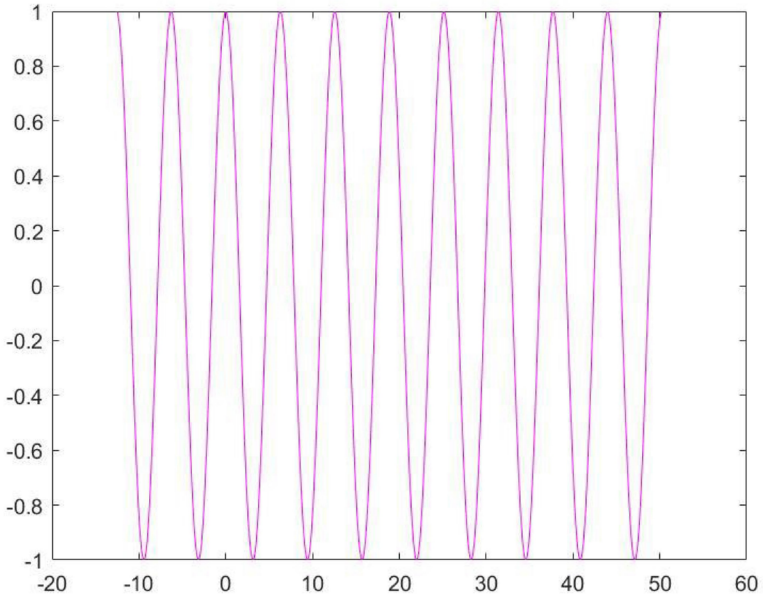
Tabela 37. Style linii wykresów.

Argument	Opis
LineStyle	rodzaj linii
Marker	rodzaj znaczników wartości
LineWidth	kolor linii
MarkerEdgeColor	kolor krawędzi znacznika w przypadku znaczników wypełnianych
MarkerSize	rozmiar znacznika
MarkerFaceColor	kolor wypełnienia znacznika

Możliwości zastosowania różnych stylów linii wykresów opisanych w Tabeli 37 są zaprezentowane w Przykładach 6.19 – 6.20.

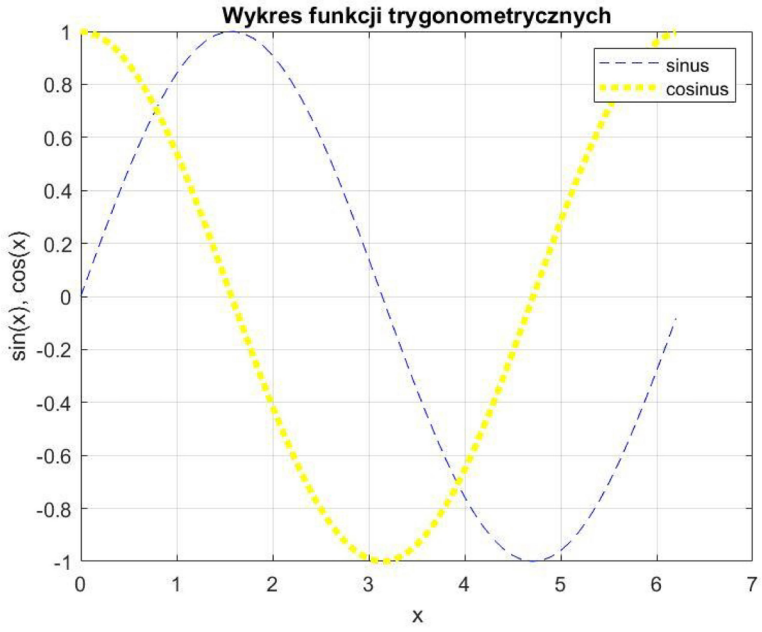
Przykład 6.19.

```
>> t = 4*[-pi:pi/100:4*pi]; y = cos(t); plot(t,y,'-m');
```



Przykład 6.20.

```
>> x=0 : 0.1 : 2*pi;
plot(x,sin(x), '--b');
grid on
title(,Wykres funkcji trygonometrycznych'); %tytuł wykresu
xlabel(,x'); %etykieta na osi X
ylabel(,sin(x), cos(x)'); %etykieta na osi Y
% wykres funkcji cos
hold on % w tym samym układzie
P=plot(x,cos(x), ':y');
set(P,'Linewidth',3);
legend('sinus','cosinus'); %legenda wykresów
```



6.3. Grafika trójwymiarowa

Program Matlab oferuje również szereg funkcji, za pomocą których możemy przedstawiać wyniki w postaci trójwymiarowej.

Funkcje te zostały opisane w Tabeli 38.

Tabela 38. Wykresy 3D oparte na funkcji meshgrid.

Funkcja	Opis
mesh	zobrazowanie powierzchni w postaci kolorowanej siatki
meshc	przedstawienie powierzchni wraz z wykresem warstwicowym poniżej
surf	zobrazowanie powierzchni w postaci zestawu kolorowych czworokątów otoczonych ramkami siatki mesh
surfc	przedstawienie powierzchni wraz z wykresem warstwicowym poniżej
meshz	przedstawienie powierzchni w postaci nieprzezroczystej siatki
surfl	przedstawienie płaskie, w którym kolory punktów reprezentują dane
surface	funkcja niskiego poziomu do tworzenia obiektów graficznych

Do wygenerowania wykresu powierzchniowego (typu z=f(x,y)) służy funkcja meshgrid o składni: [x,y] = meshgrid(X,Y)

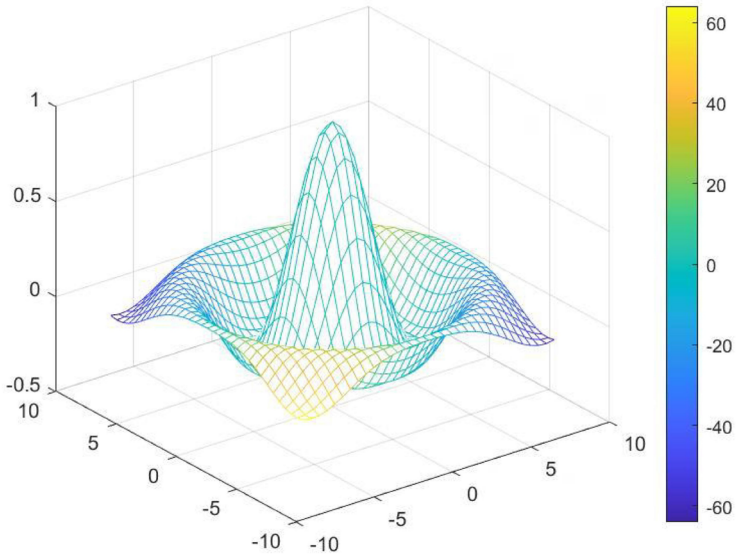
gdzie: X, Y – wektory zawierające wartości w punktach ograniczających płaszczyznę XY,

x, y – macierze określające współrzędne, dla których będą liczone wartości z.

Możliwości zastosowania funkcji meshgrid są zaprezentowane w Przykładach 6.21 – 6.26.

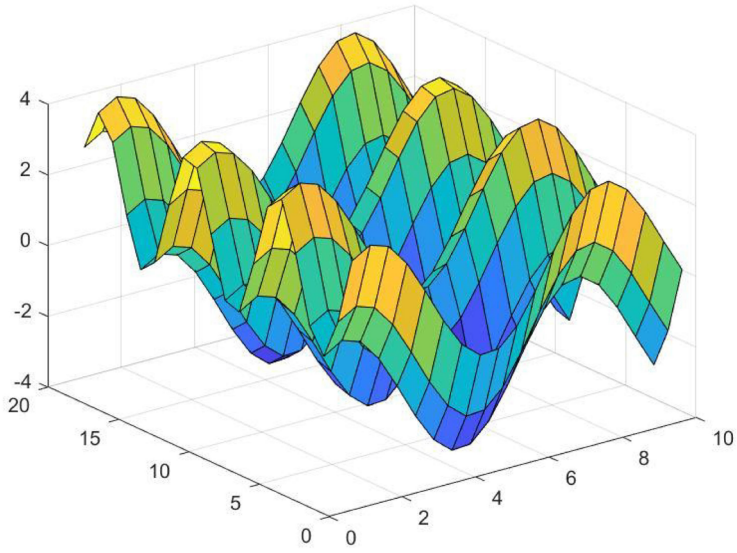
Przykład 6.21.

```
>> [X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
C = X.*Y;
mesh(X,Y,Z,C)
colorbar
```



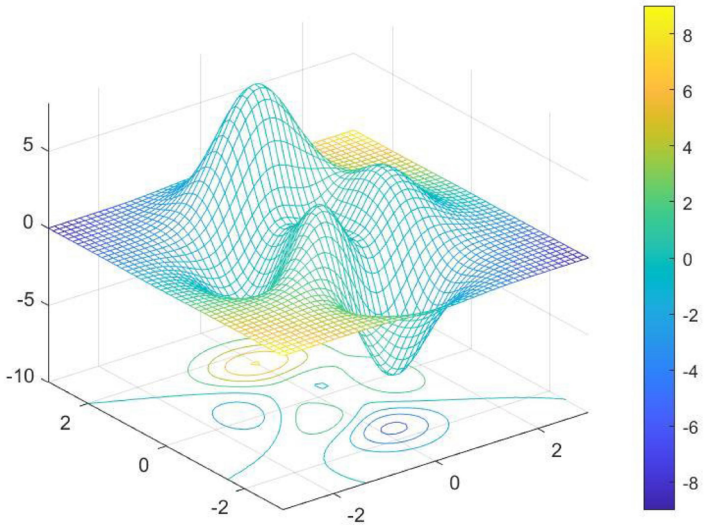
Przykład 6.22.

```
[X,Y] = meshgrid(1:0.5:10,1:20);
Z = 2*sin(X) + 2*cos(Y);
surf(X,Y,Z)
```



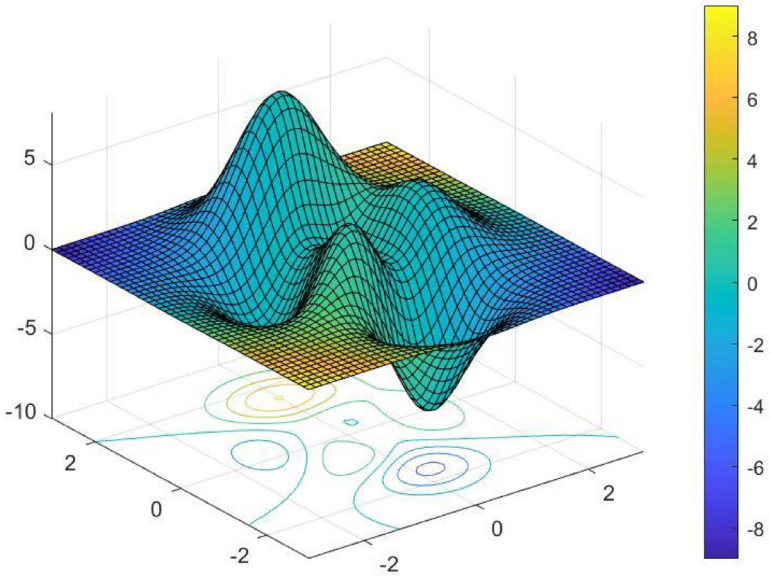
Przykład 6.23.

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
C = X.*Y;
meshc(X,Y,Z,C)
colorbar
```



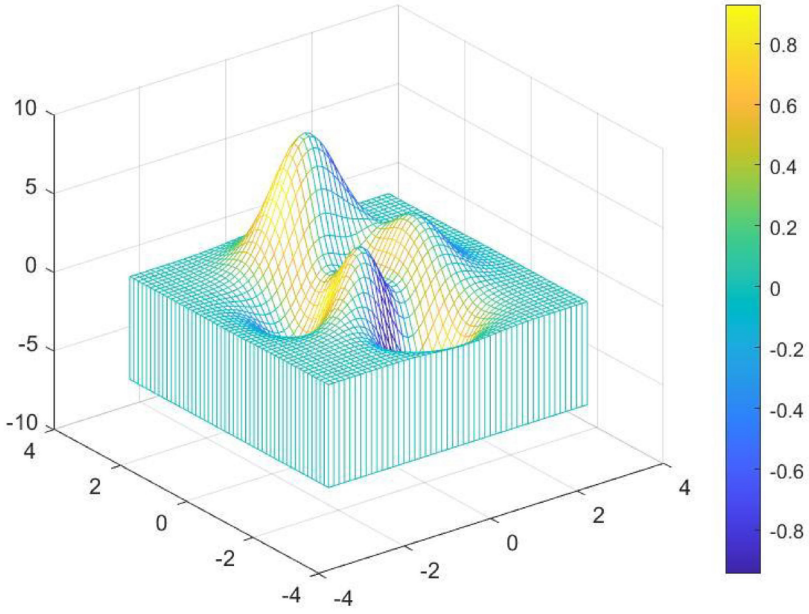
Przykład 6.24.

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
C = X.*Y;
surfc(X,Y,Z,C)
colorbar
```



Przykład 6.25.

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
C = gradient(Z);
meshz(X,Y,Z,C)
colorbar
```



Przykład 6.26.

```
C = [1 2 3; 4 5 6; 7 8 9];
s = pcolor(C);
```

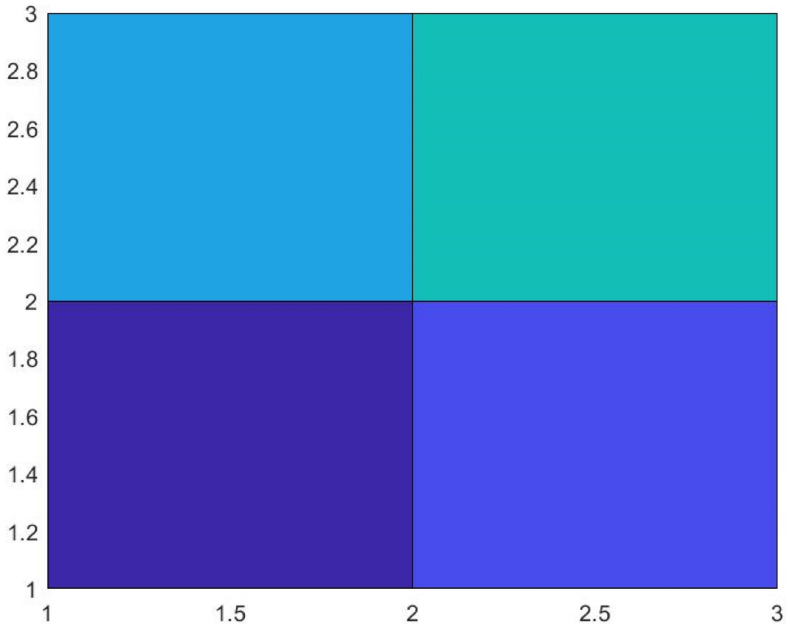
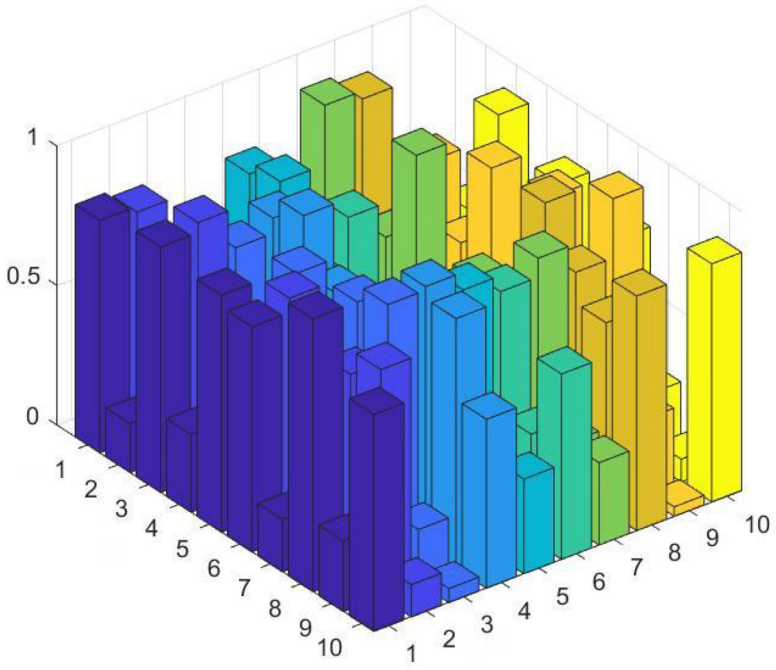


Tabela 39. Inne wykresy 3D.

Funkcja	Opis
bar3	wykres słupkowy
ezplot3	wykres parametryczny
isosurface	izowarstwy dla danych 3D
plot3	linia w 3 wymiarach
scatter3	wykres typu scatter
slice	przekrój przez wykres wolumetryczny

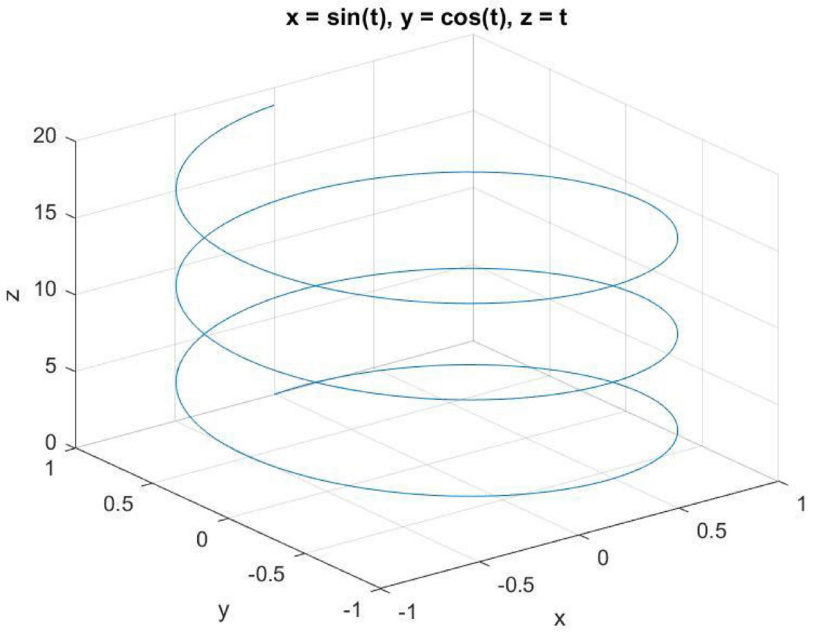
Przykład 6.27.

```
>> x=rand(10); %generowanie danych
bar3(x) % wykres słupkowy
```



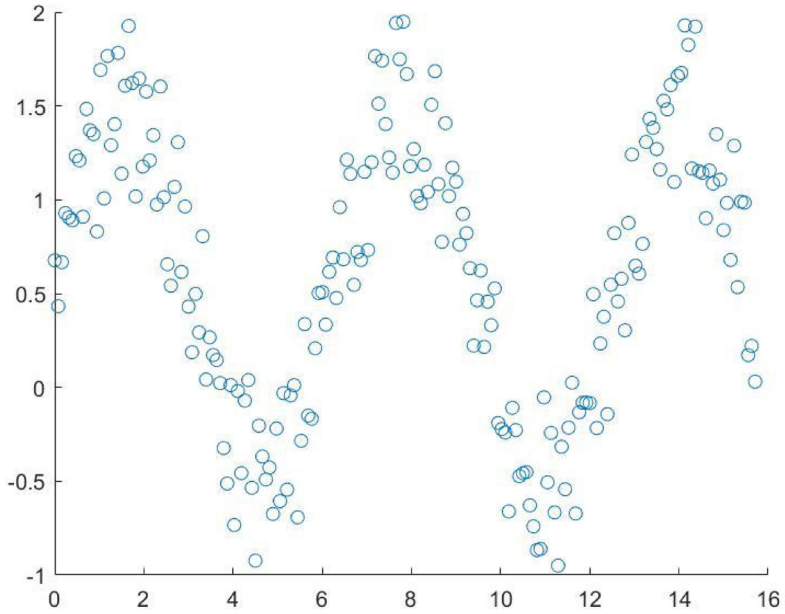
Przykład 6.28.

```
>> t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t) % linia śrubowa
ezplot3('sin(t)','cos(t)','t',[0,6*pi]) % linia śrubowa
```



Przykład 6.29.

```
>> x = linspace(0,5*pi,200);
y = sin(x) + rand(1,200);
scatter(x,y)
```



Przykład 6.30.

```
[xsurf,ysurf] = meshgrid(-5:0.5:5);
zsurf = xsurf.^2-ysurf.^2;
slice(X,Y,Z,V,xsurf,ysurf,zsurf)
```

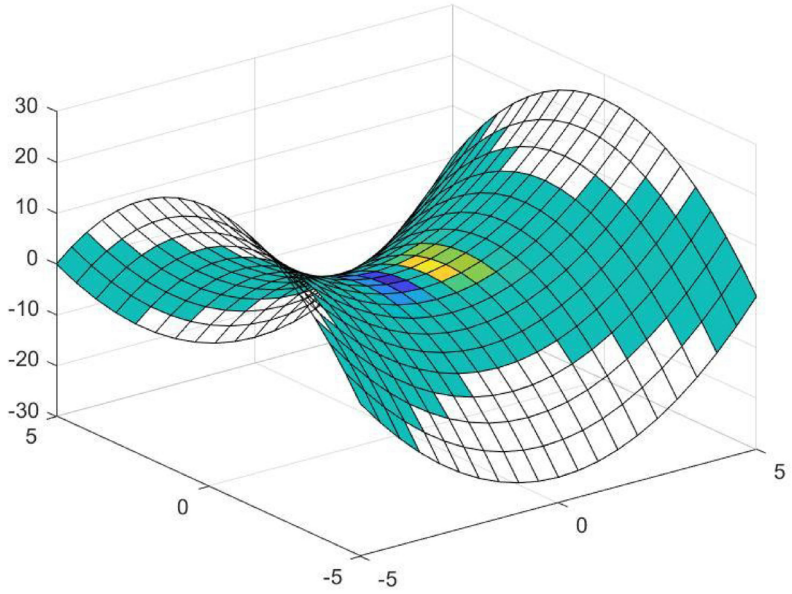


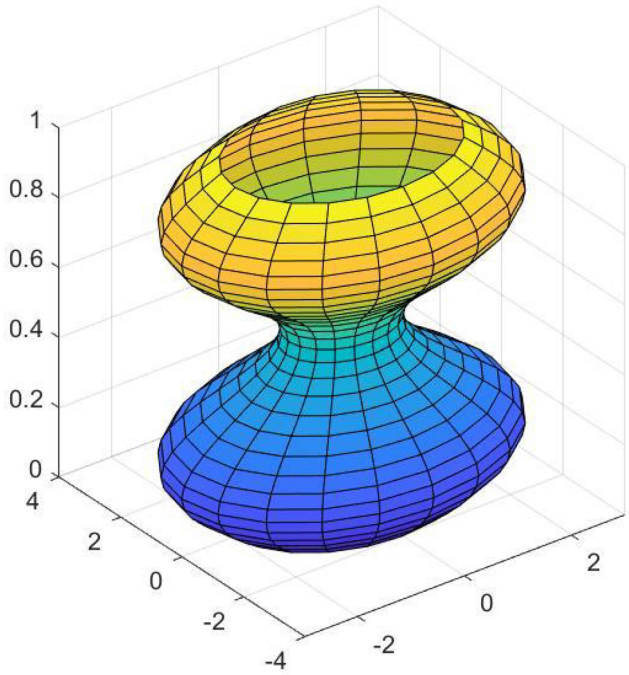
Tabela 40. Obiekty trójwymiarowe.

Funkcja	Opis
cylinder	generacja walca
elipsoid	generacja elipsoidy
fill3	generacja wielokąta
sphere	generacja kuli

Przykłady obiektów trójwymiarowych opisanych w Tabeli 40 są zaprezentowane w Przykładach 6.31 – 6.34.

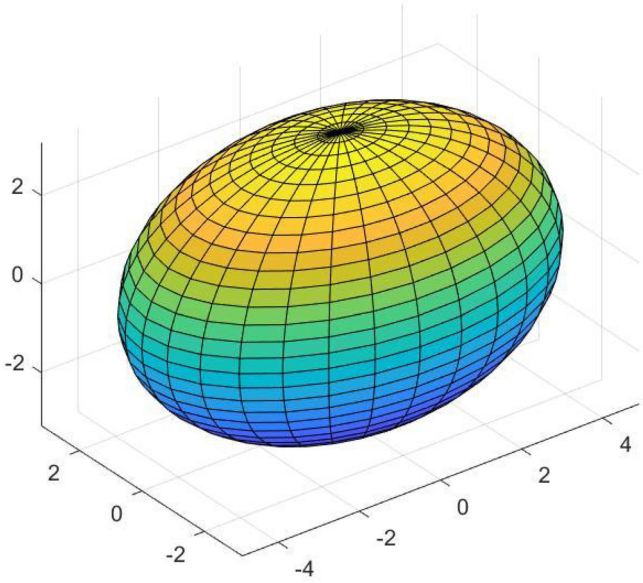
Przykład 6.31.

```
>> t = 0:pi/10:3*pi;
figure
[X,Y,Z] = cylinder(2+sin(t));
surf(X,Y,Z)
axis square
```



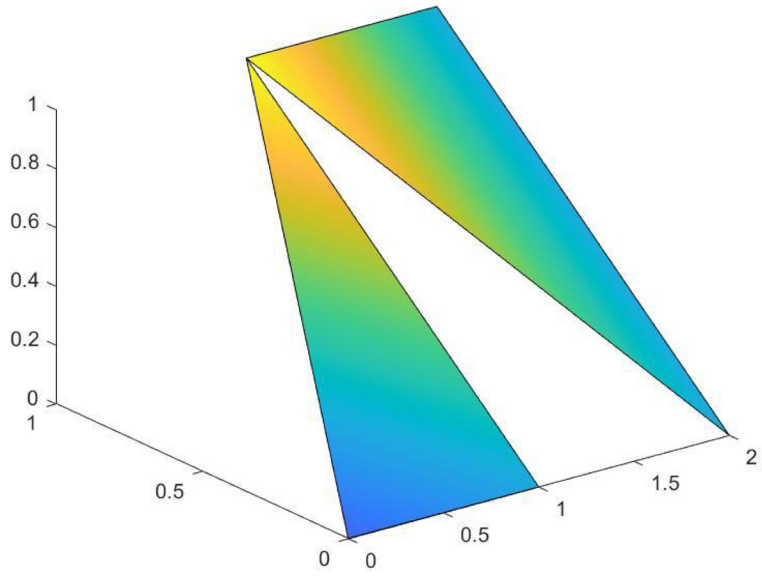
Przykład 6.32.

```
>> [x, y, z] = ellipsoid(0,0,0,4.9,3.25,3.20,30);
figure
surf(x, y, z)
axis equal
```



Przykład 6.33.

```
>> X = [1 1 1 2; 1 1 2 2; 0 0 2 2];
Y = [1 1 1 1; 1 0 1 0; 0 0 0 0];
Z = [1 1 1 1; 1 0 1 0; 0 0 0 0];
C = [0.3000 1.0000 1.0000 0.3000;
     1.0000 0.5000 0.5000 0.1667;
     0.3330 0.3330 0.5000 0.5000];
figure
fill3(X,Y,Z,C)
```



Przykład 6.34.

```
>> figure
surf(x,y,z)
hold on
surf(x+5,y-1,z) % środek w punkcie (-5,1,0)
surf(x,y+2,z-5) % środek w punkcie (0,-2,-5)
```

Tabela 41. Widoki wykresów trójwymiarowych.

Funkcja	Opis
xlabel	opis osi x
ylabel	opis osi y
zlabel	opis osi z
view	zmiana domyślnego punktu obserwacji
view(x,y,z)	określa punkt obserwacji w układzie kartezjańskim
hidden on	wyświetlanie ukrytych krawędzi
hidden off	domyślny, ukrywa niewidoczne krawędzie
shading flat	powierzchnia z dyskretnymi kolorami
shading intern	powierzchnia z wypełnieniem kolorami interpolowanymi
shading faced	powierzchnia z dyskretnymi kolorami i siatką
caxis	przeskalowanie kolorów

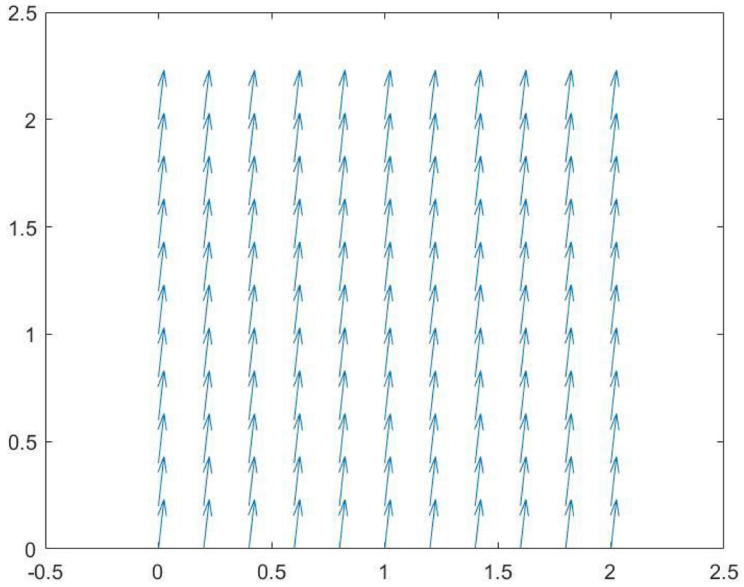
Dane trójwymiarowe możemy przedstawić przez wyświetlanie poszczególnych przekrojów, powierzchni o stałej wartości lub przepływów. Wizualizacja wolumetryczna jest opisana w Tabeli 42, a jej przykłady przedstawione w Przykładach 6.35 – 6.36.

Tabela 42. Wizualizacja wolumetryczna.

Funkcja	Opis
coneplot	pole wektorowe 3D
contourslice	kontury w warstwach
isosurface	powierzchnia o stałej wartości (izopowierzchnia)
slice	płaszczyzna przekroju
streamline	linie przepływu
streamparticles	cząstki wraz z liniami przepływu
streamslice	przepływ w warstwach lub na powierzchniach
streamtube	przepływ pokazany za pomocą walcy
quiver	pole wektorowe 2D

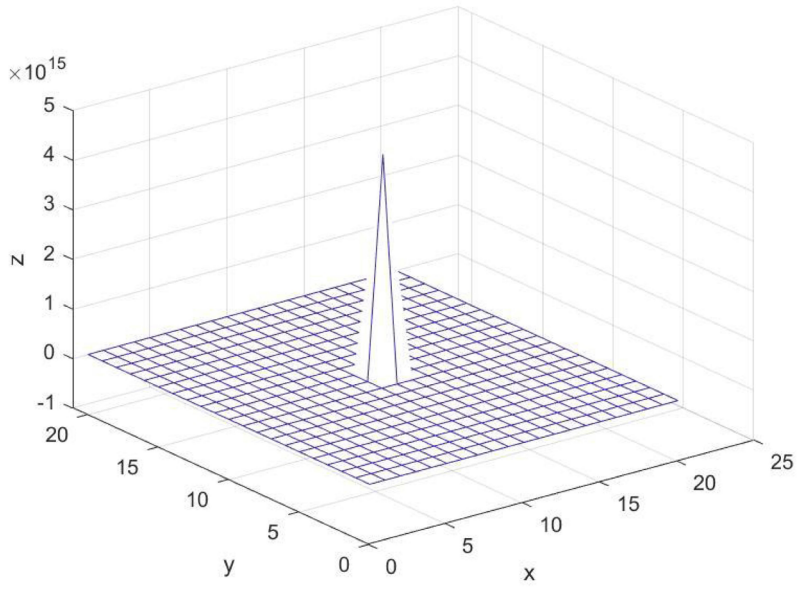
Przykład 6.35.

```
>> [x,y] = meshgrid(0:0.2:2,0:0.2:2);
u = cos(x).*2*y;
v = sin(x).*3*y;
quiver(x,y,u,v)
```



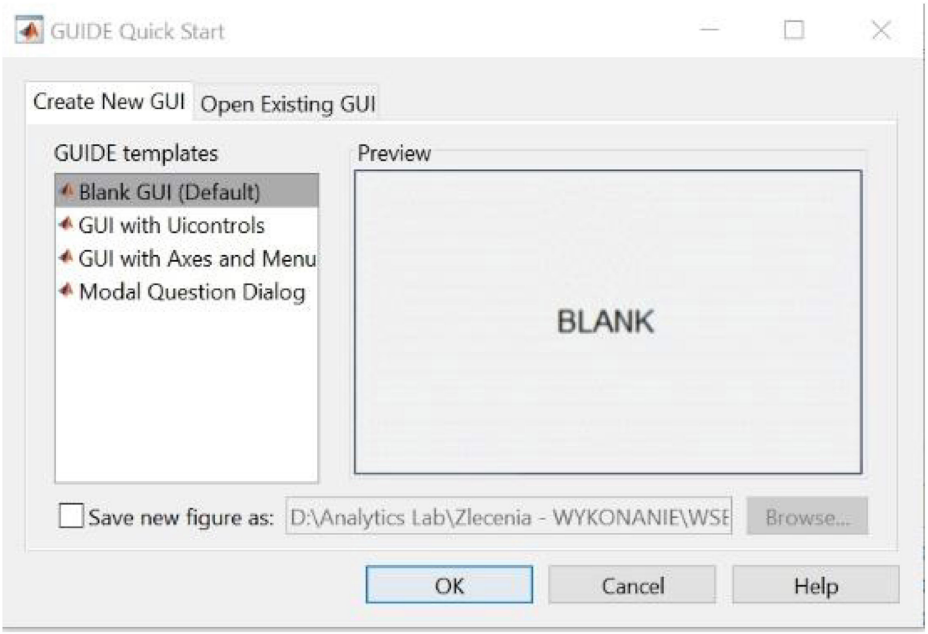
Przykład 6.36.

```
% Wyświetlenie tekstu na ekranie
disp(‚M plik wykreślający powierzchnię’)
disp(‘’)
%
[X,Y]=meshgrid(-5:0.5:5)
% Obliczenie wartości funkcji
r=sqrt(X.^2+Y.^2)+eps;
z=cos(r)./r;
%Wykreślenie funkcji,  i podpisanie osi
mesh(z);
xlabel(‘x’); %etykieta osi X
ylabel(‘y’); %etykieta osi Y
zlabel(‘z’); %etykieta osi Z
```



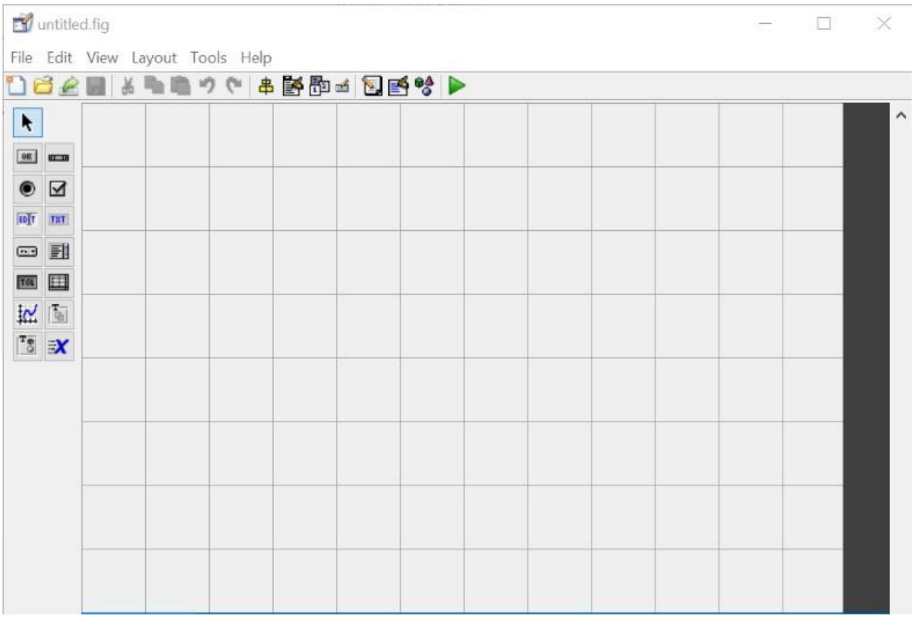
6.4. Graficzny interfejs użytkownika

Pracę z graficznym interfejsem użytkownika zaczynamy od wpisania w linii polecenia guide. Efekt zastosowania tego polecenia jest widoczny na Rysunku 15.



Rysunek 15. Graficzny interfejs użytkownika.

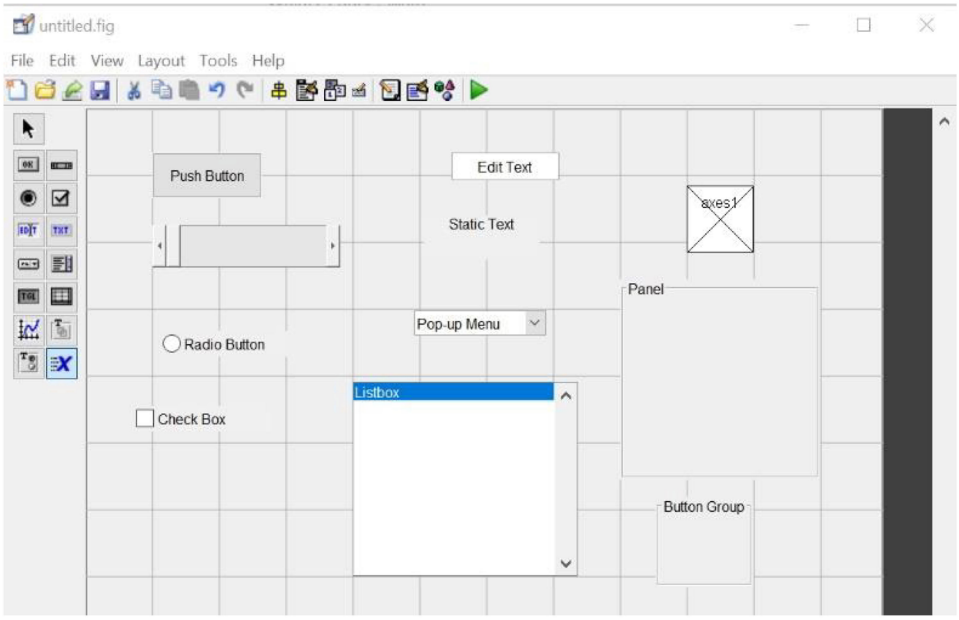
W następnym kroku mamy do wyboru cztery opcje. W celu stworzenia GUI wybieramy opcję *Blank GUI (Default)* i klikamy *OK*. Otworzy nam się pusty projekt. Okno edytora wygląda jak poniżej – Rysunek 15. Z lewej strony okna GUIDE znajduje się menu pozwalające wybrać obiekty używane w GUI. Są one bardzo podobne do obiektów używanych przy projektowaniu stron HTML w JavaScript.



Rysunek 16. Okno edytora GUI.

Najważniejsze elementy graficznego środowiska to (Rysunek 16):

1. *Push button* – standardowy przycisk, wciśnięcie może wywoływać określoną akcję.
2. *Slider* – możemy go ustawić w dwóch płaszczyznach, minimalne przesunięcie suwaka określone jest domyślnie na 0.01x maksymalny zakres.
3. *Radio button* – zgrupowanie ich pozwala uzyskać funkcje jak w ankietach, tylko jedna funkcja może zostać wybrana.
4. *Checkbox* – używany, gdy wymagany jest wielokrotny wybór, najczęściej w połączeniu z panelem, dla większej przejrzystości interfejsu.
5. *Edit text* – okno wprowadzania danych.
6. *Static text* – wyświetlanie danych.
7. *Pop-up menu* – rozwijane menu kontekstowe, możliwość wyboru tylko jednego elementu z listy.
8. *Listbox* – przydatny element, gdy potrzebujemy korzystać z danych wsadowych bądź różnych funkcji rozwiązujących nasz problem.
9. *Axes* – przeznaczeniem tego jest wyświetlanie danych, oprócz wykresów służy do pokazywania animacji oraz statycznych obrazów.
10. *Button group* – pozwala grupować przyciski.

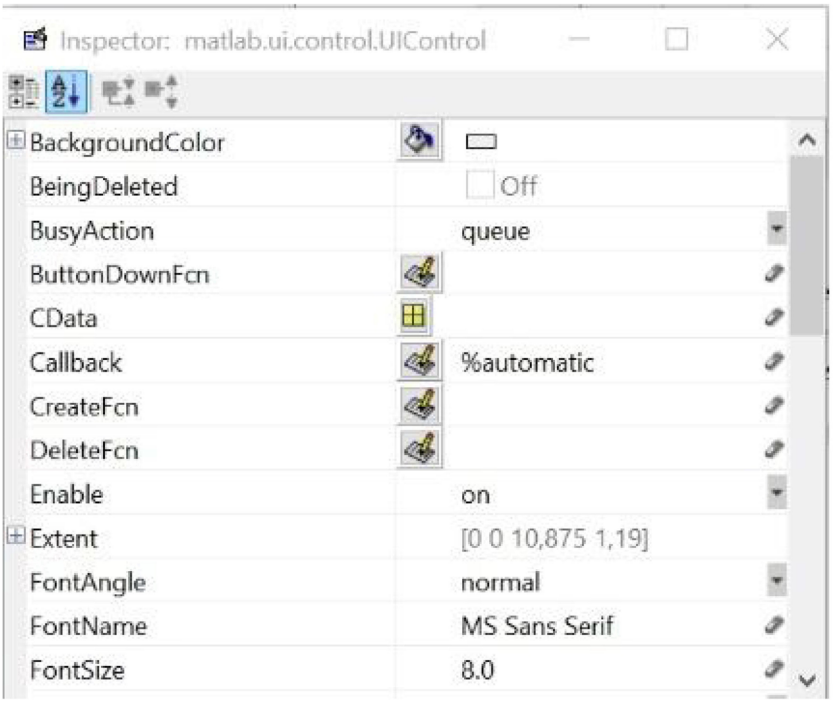


Rysunek 17. Elementy graficznego środowiska.

Budowanie interfejsu graficznego trzeba zacząć od opisanego już definiowania okna graficznego i określenia jego cech. Następnie trzeba zaplanować, jakie kontrolki będą nam potrzebne i zająć się ich definiowaniem.

W celu definiowania poszczególnych kontrolek korzystamy z *Prosperty Inspector*. Okno *Prosperty Inpector* otwieramy (patrz Rysunek 17):

- klikając dwa razy na daną kontrolkę (np. *Push buton*),
- z menu *View* wybieramy opcję *Property Inspector*,
- z lokalnego menu kontekstowego wywołanego prawym przyciskiem myszy wybieramy opcję *Property Inspector*.



Rysunek 18. Okno inspektora właściwości obiektu (*Prospery inspector*).

Najważniejsze cechy dla wszystkich kontrolek zostały opisane w Tabeli 39.

Tabela 43. Wybrane parametry kontrolek

Parametr	Opis
BackgroundColor	kolor tła
ButtonDownFcn	funkcja wywołana po naciśnięciu przycisku myszy
CData	obrazek wyświetlany na kontrolce
Callback	główna funkcja, wywołana, gdy jakieś zdarzenie zostanie wywołane na kontrolce
Enable	określa, czy kontrolka jest włączona czy wyłączona
Extent	wektor określający położenie tekstu na kontrolce, ma postać [0,0,width, height]
FintName	czcionka używana do wyświetlania na kontrolce

Parametr	Opis
ForegroundColor	określa kolor czcionki
KeuPressFcn	funkcja wywołana w momencie, gdy kontrolka jest zaznaczona i zostanie naciśnięty przycisk klawiatury
Position	wektor określający rozmiar i położenie kontrolki, ma postać [left, bottom, width, height]
SeklectionHighlight	określa możliwość podświetlenia zaznaczonej kontrolki
String	tekst wyświetlany na obiekcie
Style	określa rodzaj kontrolki
UIContextMenu	przypisanie do kontrolki menu kontekstowego
Value	wartość kontrolki
Visible	określa, czy kontrolka jest widoczna

6.5. Zadania



1. Jak uzyskać:

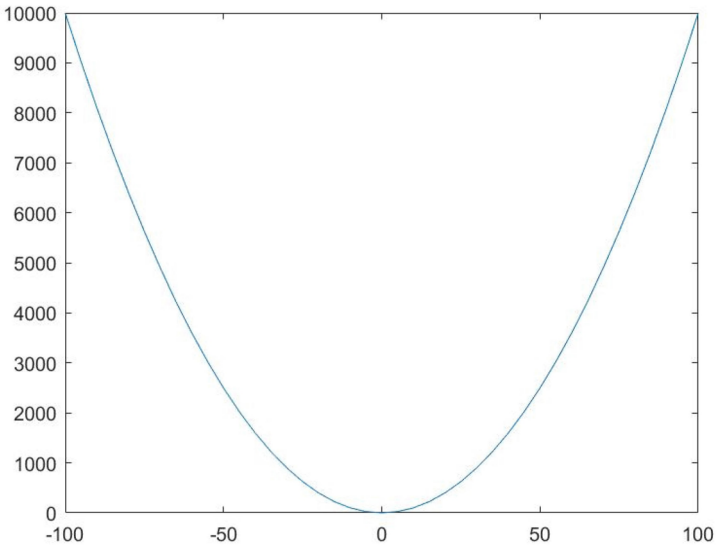
a) wykres słupkowy,
b) siatkę wykresu XY,
c) tytuł wykresu,
d) legendę.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

2. Utwórz nowe okno graficzne, a następnie podziel okno na dwa okienka.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

3. Narysuj wykres funkcji kwadratowej w przedziale [-100, 100] – patrz rysunek poniżej.



4. Dla x z przedziału od 0 do 4*pi narysuj przebieg funkcji sin(x).



5. Dla x z przedziału od 0 do 2*pi narysuj na jednym wykresie trzy przebiegi funkcji:

a) cos(x) – linią czerwoną,
b) sin(2x) – linią niebieską,
c) cos(2x) – linią czarną.

.....
.....
.....
.....
.....
.....

6. W jednym układzie współrzędnych narysuj wykresy dwóch wielomianów:

$$f(x) = 3x^4 + 3x^3 + 8x^2 + 4x + 7, \qquad g(x) = 5x^3 + 5x + 2$$

.....
.....
.....
.....
.....
.....

7. Utwórz wykres funkcji $f(x) = \frac{x^4 - x^2 + 1}{x^4 + x^2 - 1}$ za pomocą polecenia ezplot.

a) Przyjmij krok wektora dziedziny t odpowiedni dla wykreślenia funkcji.
b) Zakres wykresów na osi y zmień poleceniem axis.
c) Porównaj wyniki działania funkcji plot i ezplot.
d) Dodaj opisy osi i tytuły.
e) Skopiuj zawartość okna poleceń programu Matlab do programu Wordpad.
f) Wyczyść zawartość okna poleceń programu Matlab.

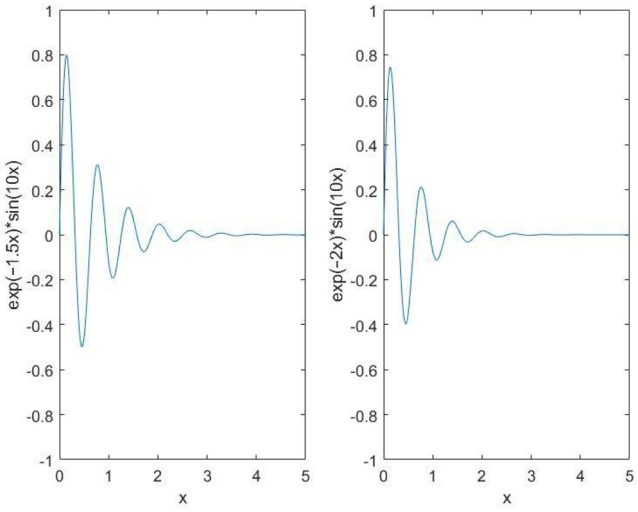
.....
.....
.....
.....
.....
.....

8. Utwórz wykres funkcji $f(x) = x^4 - x^2$, dla $-10 \leq x \leq 10$ za pomocą polecenia plot.

a) Przyjmij krok wektora dziedziny t odpowiedni dla wykreślenia funkcji.
b) Zakres wykresów na osi y zmień poleceniem axis.
c) Porównaj wyniki działania funkcji plot i ezplot.
d) Dodaj opisy osi i tytuły.
e) Skopiuj zawartość okna poleceń programu Matlab do programu Wordpad.
f) Wyczyść zawartość okna poleceń programu Matlab.

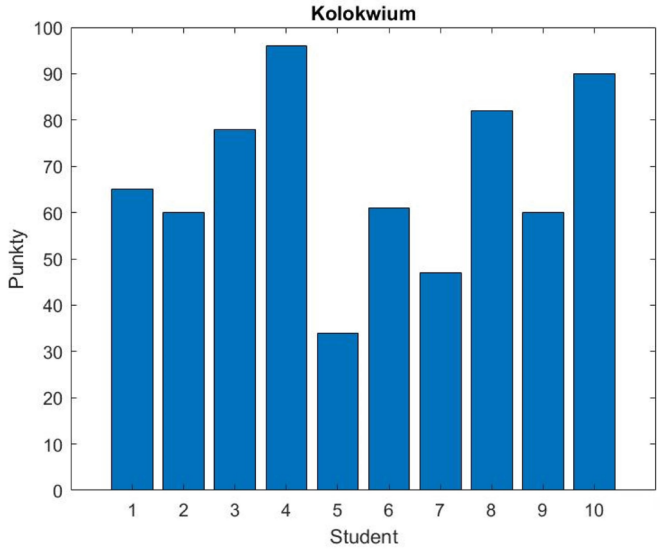
.....
.....
.....
.....
.....
.....
.....

9. Podziel okno graficzne na dwie części, a następnie w pierwszej części narysuj wykres funkcji $y = e^{-1.5x}sin(10x)$, a w drugiej $y = e^{-2x}sin(10x)$. Wykresy są widoczne poniżej.



.....
.....
.....
.....

10. Wyniki punktowe 10 studentów z kolokwium są następujące: 65, 60, 78, 96, 34, 61, 47, 82, 60, 90. Stwórz reprezentację graficzną wyników kolokwiów, taką jak na poniższym rysunku.



.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

11. Za pomocą funkcji ezplot narysuj wykres funkcji *f(x) = xcos(x), x∈[–π, π]*

.....
.....
.....
.....
.....
.....
.....

Bibliografia

1. Banasiak, K.A. (2017). Algorytmizacja i programowanie w MATLABIE. Legionowo: Wydawnictwo BTC.
2. Bjarck, A., Dahlquist, G. (1987). Metody numeryczne. Warszawa: PWN.
3. Brzózka, J., Dorobczyński, L. (2008). MATLAB: środowisko obliczeń naukowo-technicznych. Warszawa: PWN.
4. Dyka, E., Markiewicz, P., Sikora, R. (2006). Modelowanie w elektrotechnice z wykorzystaniem środowiska MATLAB. Łódź: Wydawnictwo Politechnika Łódzka.
5. Kincaid, D., Cheney, E.W. (2006). Analiza numeryczna. Warszawa: Wydawnictwa Naukowo-Techniczne.
6. Kłosowski, P., Ambroziak, A. (2011). Metody numeryczne w mechanice konstrukcji z przykładami w programie Matlab. Gdańsk: Wydaw. Politechniki Gdańskiej.
7. Kula, K. (2009). Zbiór zadań z podstaw automatyki. Rozwiązania analityczne oraz komputerowe w środowisku Matlab. Gdynia: Wydawnictwo Akademii Morskiej w Gdyni.
8. Lubowiecka, I., Ambroziak, A. (2016). MATLAB i jego środowisko. Gdańsk: Wydawnictwo Politechniki Gdańskiej.
9. Mrozek, B., Mrozek, Z. (2010). MATLAB i Simulink: poradnik użytkownika. Gliwice: Helion.
10. Szymczyk, E. (2006). Matlab dla mechaników. Warszawa: Wyd. WAT.
11. Rudra, P. (2007). Matlab 7 dla naukowców i inżynierów. Warszawa: PWN.
12. Regel, W. (2003). Wykresy i obiekty graficzne w Matlab. Warszawa: Mikom.
13. Stachurski, M. (2003). Metody numeryczne w programie Matlab. Warszawa: Mikom.
14. Wanat, K. (1994). Algorytmy numeryczne. Gliwice: Helion.

PROGRAMOWANIE W MATLABIE

Część druga
Studia przypadków

Anna Tatarczak

Lublin 2021

Spis treści

Wstęp 168

Studium przypadku 1. Obliczanie wartości wyrażeń 169
1.1. Przypadek 1 172
1.2. Przypadek 2 173
1.3. Przypadek 3 175
1.4. Przypadek 4 176
1.5. Zadania 178

Studium przypadku 2. Programowanie 179
2.1. Przypadek 1 182
2.2. Przypadek 2 182
2.3. Przypadek 3 183
2.4. Przypadek 4 185
2.5. Przypadek 5 186
2.6. Przypadek 6 187
2.7. Zadania 190

Studium przypadku 3. Macierze i układy równań 191
3.1. Przypadek 1 196
3.2. Przypadek 2 201
3.3. Przypadek 3 203
3.4. Przypadek 4 204
3.5. Zadania 206

Studium przypadku 4. Funkcje 207
4.1. Przypadek 1 208
4.2. Przypadek 2 209
4.3. Przypadek 3 211
4.4. Przypadek 4 213
4.5. Przypadek 5 216
4.6. Zadania 218

Studium przypadku 5. Optymalizacja 219
5.1. Przypadek 1 223
5.2. Przypadek 2 224
5.3. Przypadek 3 225
5.4. Zadania 227

Studium przypadku 6. Interpolacja funkcji 228
6.1. Przypadek 1 230
6.2. Przypadek 2 232
6.3. Przypadek 3 233
6.4. Zadania 235

Studium przypadku 7. Graficzny Interfejs Użytkownika 236
7.1. Przypadek 1. 239
7.2. Przypadek 2 243
7.3. Zadania 249

Wstęp

Niniejsze opracowanie zawiera opis głównych zagadnień prezentowanych na laboratorium z przedmiotu **Programowanie w Matlabie**, który jest przeznaczony dla studentów Wyższej Szkoły Ekonomii i Innowacji w Lublinie. Opracowanie w formie case study przedstawia w sposób skondensowany najważniejsze informacje przekazywane podczas laboratorium. Materiał może więc stanowić użyteczne narzędzie do przygotowania się do zajęć bądź powtórki materiału, ma jednak charakter uzupełniający wobec podstawowego podręcznika.

Pakiet Matlab jest to narzędzie stosowane do przeprowadzania obliczeń i symulacji w wielu dziedzinach nauki. Głównym przeznaczeniem pakietu są obliczenia inżynierskie oraz matematyczne. Program ten pozwala na wykonywanie skomplikowanych obliczeń numerycznych z końcową wizualizacją otrzymanych wyników.

Treść niniejszego opracowania stanowią **wybrane zagadnienia z teorii i praktyki inżynierskiej**. Zostało opisanych siedem głównych case study (każde z podziałem na mini-casy), które są najczęściej wykorzystywane w pracy naukowo-technicznej. Wszystkie rozdziały mają podobną strukturę: na wstępie przedstawiony jest cel ćwiczeń oraz problem, który będzie analizowany. Następnie czytelnik zapozna się z krótkim wstępem teoretycznym i znajdzie wykaz funkcji/poleceń wykorzystywanych w danym rozdziale. Główną treścią każdego rozdziału są rozwiązane przykłady praktyczne ze szczegółowymi opisami w postaci komentarzy. Ponieważ najlepszą nauką języka programowania jest praktyka, więc każdy z case'ów kończy się ćwiczeniami do wykonania.

Wszystkim zainteresowanym pakietem polecam **literaturę** zamieszczoną na końcu opracowania.

Zakłada się, że Czytelnik zna **podstawowy kurs algebry i analizy matematycznej**. Wymagana jest również podstawowa znajomość zasad tworzenia algorytmów obliczeniowych. Doświadczenie w programowaniu w innych językach bardzo pomaga, ale nie jest konieczne. Wykonanie prezentowanych przykładów obliczeniowych wymaga również elementarnej znajomości korzystania z oprogramowania Matlab.

Studium przypadku 1.

Obliczanie wartości wyrażeń

Cel ćwiczenia

- Przedstawienie podstawowych możliwości Matlaba, w szczególności jako zaawansowanego kalkulatora i narzędzia do obliczania wartości wyrażeń.
- Wczytywanie zmiennych od użytkownika i wyświetlanie zmiennych na ekranie.
- Tworzenie i wywoływanie skryptów oraz funkcji.
- Samodzielne rozwiązanie wybranych problemów matematycznych przy użyciu pakietu.

Problem

Zagadnienie, z którym będziemy często spotykać się w pracy inżynierskiej, dotyczy **wyliczania wartości wyrażeń**. Problem ten zazwyczaj nie będzie głównym zadaniem do rozwiązania, a jedynie będzie stanowił część większego problemu. Ważne jest, aby umieć wczytać dane od użytkownika, następnie dla zadanej wartości obliczyć wyrażenie i wynik wyświetlić na ekranie. Opanowanie tego prostego problemu jest kluczowe do dalszego sprawnego programowania w Matlabie.

Wstęp teoretyczny

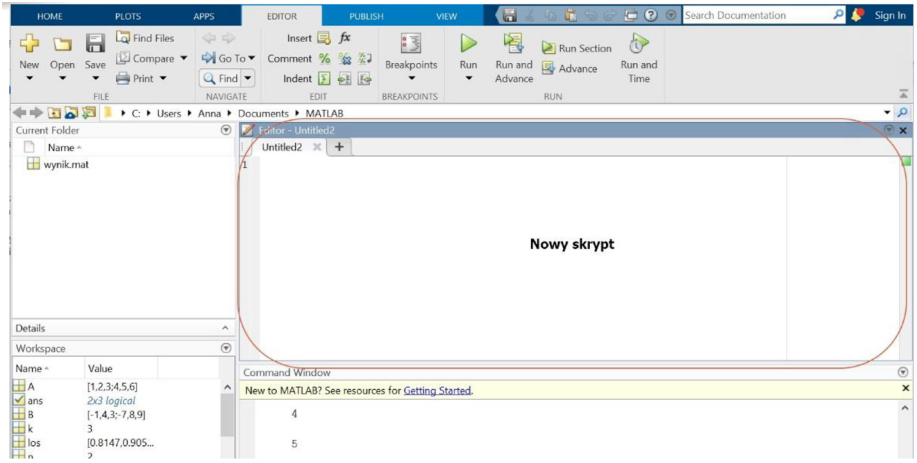
Dobrym zwyczajem pracy z Matlabem jest opracowywanie pewnych stałych elementów kodu w postaci skryptów lub/i funkcji. Skrypty oraz funkcje zawierają ciągi poleceń Matlaba potrzebnych do wykonania danego programu. Zaletą skryptów oraz funkcji jest fakt, że mogą być wykonywane wielokrotnie.

Skrypty zawierają **sekwencje poleceń Matlaba**, które są wykonywane kolejno linia po linii, począwszy od początku pliku, tak jakby każde z nich zostało wprowadzone i zlecone do wykonania w *Oknie Poleceń*.

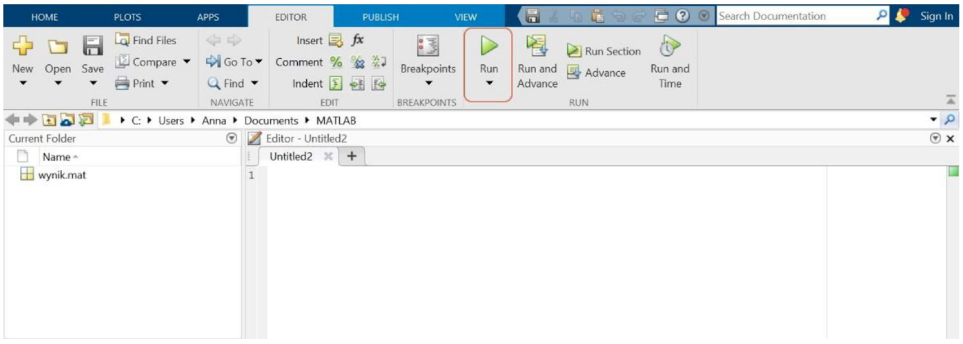
Aby otworzyć nowy skrypt, klikamy zakładkę *Home*, a następnie *New Script* – patrz Rysunek 1. Wówczas Matlab otworzy kolejną zakładkę dla nowego skryptu – patrz Rysunek 2. Po wpisaniu treści m-pliku (oraz zapisaniu go pod nazwą z rozszerzeniem '.m') można go natychmiast uruchomić za pomocą narzędzia *Run* na zakładce *Editor Paska Narzędzi* (Rysunek 3) lub naciskając klawisz *F5*.



Rysunek 1. Tworzenie skryptu.



Rysunek 2. Nowy skrypt.



Rysunek 3. Uruchomienie skryptu.

Jednym z rodzajów m-plików są pliki funkcyjne, zwane skrótowo funkcjami. Definicja własnej funkcji musi zaczynać się od słowa kluczowego function i dalej zawierać:

- nazwę zmiennej zwracającej wartość obliczoną przez funkcję np. MojaFunkcja; jeżeli takich zmiennych jest więcej, ich listę umieszcza się w nawiasie kwadratowym [ ],
- znak podstawienia =,

- wybraną przez programistę nazwę funkcji z umieszczoną w nawiasach półokrągłych listą parametrów wejściowych, od których funkcja zależy, np. global i j k,
- dobrym zwyczajem jest zamieszczenie komentarza informującego, co ta funkcja robi, jakie znaczenie mają parametry wejściowe itp.,
- potem jest ciąg instrukcji do wykonywania przez funkcję, czyli tzw. ciało funkcji,
- definicję funkcji kończy słowo kluczowe end.



WAŻNE

Kod funkcji ma następującą postać:
function [zmienne_wyjsciowe] = nazwa_funkcji(zmienne_wejsciowe)

Kończąc budowanie funkcji, musimy ją zapisać w m-pliku. Pamiętajmy, aby nazwy funkcji były takie same, jak nazwy plików, w których są zapisane. Taka systematyka bardzo ułatwi pracę z wieloma funkcjami. Dobrym zwyczajem jest też komentowanie pisanego kodu.

Wykorzystywane funkcje

Funkcja	Opis
clear	usunięcie zmiennych z pamięci roboczej Matlaba
clc	wyczyszczenie okna komend
sin	funkcja trygonometryczna sinus
cos	funkcja trygonometryczna cosinus
disp(tekst)	wyświetla dany tekst na ekranie
x=input(tekst)	wyświetla łańcuch tekst, oczekuje na wpisanie przez użytkownika danej liczbowej i przypisuje ją zmiennej liczbowej x
ntroot	pierwiastek n-tego stopnia
log	funkcja logarytmiczna
exp	funkcja wykładnicza o podstawie e
abs	wartość bezwzględna

1.1. Przypadek 1

Polecenie:

Dla zadanych wartości x=2.5 oraz y=-6.5 oblicz wartości wyrażeń: . Wartości obliczonych wyrażeń wyświetl na ekranie. Rozwiązanie problemu zaprogramuj w formie skryptu.

Rozwiązanie:

Treść skryptu WartoscWyrazenia.m dla zadanych wartości x=2.5 oraz y=-6.5 poniżej.

```
clear % usunięcie zmiennych z pamięci roboczej Matlaba
clc % wyczyszczenie okna komend
% nadanie wartości zmiennym x i y
x = 2.5;
y = -6.5;
% obliczenie wyrażeń
w1 = x+y;
w2 = x-y;
w3 = x*y;
w4 = sin(2*x)+ cos(y/2);
%wyświetlenie tekstu i wartości zmiennej w oknie komend
disp(‚wyrażenie w1 ma wartość’)
disp(w1)
disp(‚wyrażenie w2 ma wartość’)
disp(w2)
disp(‚wyrażenie w3 ma wartość’)
disp(w3)
disp(‚wyrażenie w4 ma wartość’)
disp(w4)
disp(‚dla x =’)
disp(x)
disp(‘oraz y = ‘)
disp(y)
```

Efekt działania skryptu

```
wyrażenie w1 ma wartość
    -4
wyrażenie w2 ma wartość
     9
wyrażenie w3 ma wartość
  -16.2500
wyrażenie w4 ma wartość
   -1.9531
dla x =
    2.5000
oraz y =
   -6.5000
```

1.2. Przypadek 2

Polecenie:

Teraz zmodyfikujemy skrypt powstały w Przypadku 1, modyfikacja polega na wczytaniu od użytkownika wartości zmiennych x oraz y.

Rozwiązanie:

Dla tak zdefiniowanego problemu kod skryptu WartoscWyrazen2 jest podany poniżej.

```
clear % usunięcie zmiennych z pamięci roboczej Matlaba
clc % wyczyszczenie okna komend
% wczytanie wartości zmiennych x i y
x = input(,Podaj wartość zmiennej x = ,);
y = input(,Podaj wartość zmiennej y = ,);
% obliczenie wyrażeń
w1 = x+y;
w2 = x-y;
w3 = x*y;
w4 = sin(2*x)+ cos(y/2);
%wyświetlenie tekstu i wartości zmiennej w oknie komend
disp(,wyrażenie w1 ma wartość')
disp(w1)
disp(,wyrażenie w2 ma wartość')
disp(w2)
disp(,wyrażenie w3 ma wartość')
disp(w3)
disp(,wyrażenie w4 ma wartość')
disp(w4)
disp(,dla x =')
disp(x)
disp('oraz y = ')
disp(y)
```

Uruchomiony skrypt poniżej.

```
Podaj wartość zmiennej x = 3
Podaj wartość zmiennej y = 6
wyrażenie w1 ma wartość
     9
wyrażenie w2 ma wartość
    -3
wyrażenie w3 ma wartość
    18
wyrażenie w4 ma wartość
   -1.2694
dla x =
     3
oraz y =
     6
```

1.3. Przypadek 3

Polecenie:

Opracuj skrypt, który najpierw wczyta dwie wartości zmiennych x oraz y. Następnie dla wartości x oraz y obliczy wartości wyrażeń: $w1 = \sqrt[3]{\frac{x-y}{2x}}$, $w2 = \cos\left(\frac{1+x^3}{2y}\right)$, $w3 = \frac{log^3(1+x^3)}{xy}$, $w4 = sin\frac{(1+x^2)^2}{x-y}$, $w5 = \frac{log^3(1+x^2)^2}{x+y}$, $w6 = \frac{e^{(2x^6)^{0.3}}}{x+y}$, $w7 = \frac{|x+y|}{\sin(x-y)}$. Wszystkie obliczone wartości wyrażeń mają być wyświetlone na ekranie.

Rozwiązanie:

Sporządzić m-plik jak poniżej.

```
clear %wyczyszczenie przestrzeni roboczej
clc  % czysci ekran command window
%wczytanie danych
x = input(,Podaj wartość zmiennej x = ,);%użytkownik podaje liczbę x
y = input(,Podaj wartość zmiennej y = ,); %użytkownik podaje liczbę y
% obliczenie wyrażeń
w1 = nthroot(((x-y)/(2*x)),3);
w2 = cos(1+x^3)/(2*y);
w3 = (log(1+x^3))^3/(x*y);
w4 = sin((1+x^2)^2)/(x-y);
w5 = (log((1+x^2)^2))^3/(x+y);
w6 = exp(2*(x^6)^0.3/(x+y));
w7 = abs(x+y)/sin(x-y);
%wyświetlenie tekstu i wartości zmiennej w oknie komend
disp(,wyrażenie w1 ma wartość')
disp(w1)
disp(,wyrażenie w2 ma wartość')
disp(w2)
disp(,wyrażenie w3 ma wartość')
disp(w3)
disp(,wyrażenie w4 ma wartość')
disp(w4)
disp(,wyrażenie w5 ma wartość')
disp(w5)
```

```
disp(,wyrażenie w6 ma wartość')
disp(w6)
disp(,wyrażenie w7 ma wartość')
disp(w7)
disp(,dla x =')
disp(x)
disp('oraz y = ')
disp(y)
```

Efekt działania skryptu.

```
Podaj wartość zmiennej x = 2
Podaj wartość zmiennej y = 5
wyrażenie w1 ma wartość
   -0.9086
wyrażenie w2 ma wartość
   -0.0911
wyrażenie w3 ma wartość
    1.0608
wyrażenie w4 ma wartość
    0.0441
wyrażenie w5 ma wartość
    4.7645
wyrażenie w6 ma wartość
    2.7045
wyrażenie w7 ma wartość
  -49.6032
dla x =
     2
oraz y =
     5
```

1.4. Przypadek 4

Polecenie:

Opracuj funkcję, która dla dwóch zadanych wartości (długości boków) obliczy moment bezwładności prostokąta.

Rozwiązanie:

```
%funkcja oblicza momenty bezwładności Ix, Iy prostokąta
function[Ix,Iy]=MomentBezw(b,h);

Ix=(b*h^3)/12
Iy=(h*b^3)/12
```

Wywołujemy opracowaną funkcję

```
>> MomentBezw(2,4)
Ix =
   10.6667
Iy =
    2.6667
ans =
   10.6667
```



1.5. Zadania

Zadanie 1.

Opracuj skrypt, który dla dwóch zmiennych wejściowych oblicza: kwadrat pierwszego argumentu, sumę wszystkich argumentów, iloczyn wszystkich argumentów.

.....
.....
.....
.....
.....
.....
.....

Zadanie 2.

Napisz jedną funkcję do obliczenia dwu wielkości: przebytej drogi s oraz uzyskanej prędkości V spadającego swobodnie ciężarka, gdy dany jest czas t od początku spadania i pomijamy opór powietrza.

.....
.....
.....
.....
.....
.....
.....

Zadanie 3.

Napisz funkcję użytkownika do obliczenia dwu wielkości: składowych Fx, Fy danej siły F, gdy dany jest także kąt jej działania.

.....
.....
.....
.....
.....
.....
.....

Studium przypadku 2.

Programowanie

Cel ćwiczenia

- Wprowadzenie instrukcji warunkowych oraz pętli.
- Opanowanie składni instrukcji warunkowych oraz dwóch najpopularniejszych pętli for oraz while.
- Nauka zastosowania instrukcji if.

Problem

Instrukcje warunkowe są podstawą każdego języka programowania. Używa się ich do wykonania pewnej instrukcji (lub bloku instrukcji), ale tylko w pewnych okolicznościach – jeśli zostanie spełniony określony warunek (lub cały zestaw warunków). Czasem zachodzi potrzeba wykonania danej czynności określoną ilość razy. Z pomocą przychodzi jedna z najczęściej używanych składni w większości języków programowania – pętla for oraz while.

Dopiero poznanie instrukcji **warunkowych oraz pętli** umożliwi pisanie programów w Matlabie, które będą już naprawdę mogły wykonywać zaawansowane obliczenia. Użytkownik będzie mógł na przykład zdecydować, którą opcję chce wybrać, a program w zależności od wybranej opcji będzie mógł wykonać określoną czynność.

Wstęp teoretyczny

Niech $f(x)$=$ax2$ +bx+c, gdzie $a \in R\backslash\{0\}, b \in R, c \in R$, będzie funkcją kwadratową.

Wtedy dla $\Delta = b^2 - 4ac$ mamy

(i) jeżeli $\Delta < 0$ to funkcja f nie ma miejsc zerowych;

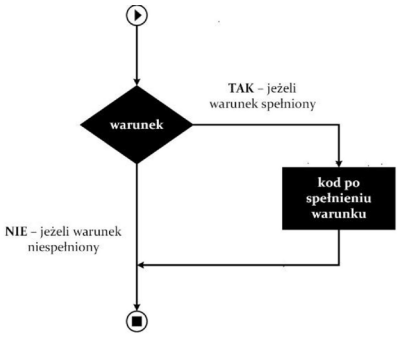
(ii) jeżeli $\Delta = 0$ to funkcja f ma jedno miejsce zerowe równe $x = \frac{-b}{2a}$

(iii) jeżeli $\Delta > 0$ to funkcja f ma dwa różne miejsca zerowe wyrażone wzorami: $x_1 = \frac{-b-\sqrt{\Delta}}{2a}, x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.

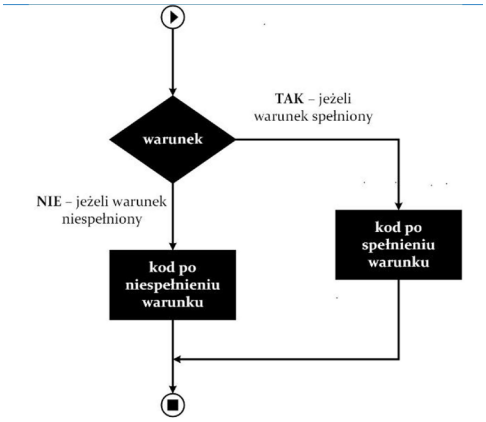
Miejsca zerowe funkcji kwadratowej nazywamy także pierwiastkami tej funkcji.

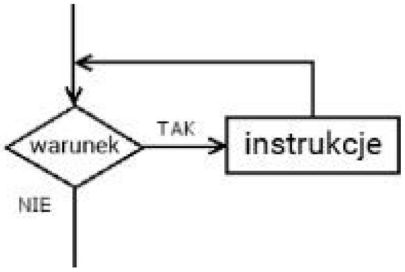
Wykorzystywane funkcje i instrukcje

Funkcja	Opis
x=input(tekst)	wyświetla łańcuch tekst, oczekuje na wpisanie przez użytkownika danej liczbowej i przypisuje ją zmiennej liczbowej x
disp(tekst)	wyświetla dany tekst na ekranie
clear	usuwa zmienną z pamięci
clc	czyszczenie ekranu
sqrt	pierwiastek kwadratowy z danej liczby

Kod instrukcji		Opis składni
Instrukcje warunkowe	if wyrażenie_logiczne blok instrukcji end	 Zasada działania **instrukcji warunkowej if** opiera się na wartości, jaką przyjmuje warunek. W przypadku **TAK**, instrukcje dla bloku **if** zostaną wykonane.

	Kod instrukcji	Opis składni

	if wyrażenie_logiczne_if
	  blok instrukcji_if
	else
	  blok instrukcji_else
	end
	
	if wyrażenie_logiczne_if
	  blok instrukcji_if
	elseif wyrażenie_logiczne_elseif
	  blok instrukcji_elseif
	else
	  blok instrukcji_else
	end
		
		W **instrukcji warunkowej z alternatywą** pojawia się drugi blok instrukcji rozpoczynany słowem kluczowym **else**, wykonany w przypadku, gdy warunek (warunki) jest fałszywy.

Pętle	while wyrażenie instrukcje end	 Przed każdym wykonaniem instrukcji następuje sprawdzenie warunku. Pętla zostanie przerwana, gdy wartość warunku będzie równa FALSE.
	for zmienna_iterowana = wyrażenie blok instrukcji end	W pętli podaje się trzy wyrażenia jako parametry: zmienna_iterowana, czyli ustawienie początkowych wartości dla zmiennych kontrolujących pętlę, sprawdzenie warunku, czyli wyrażenie logiczne kontrolujące pętlę – pętla będzie wykonywana, dopóki ten warunek jest prawdziwy oraz modyfikację zmiennych kontrolujących pętlę – bez tego pętla będzie wykonywała się w nieskończoność (oczywiście wartość tych zmiennych można modyfikować wewnątrz pętli, ale jest to niezalecane).

2.1. Przypadek 1

Polecenie:

Opracuj algorytm, który po podaniu przez użytkownika liczby rzeczywistej wyświetli na ekranie informację, gdy liczba ta będzie ujemna.

Rozwiązanie:

Sporządzić m-plik jak poniżej, z zastosowaniem prostej konstrukcji if.

```
%program wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik na ekran
x = input(‚Podaj liczbę rzeczywistą\nx = ‚);
if x<0
disp(‚liczba ujemna’)
end
```

Uruchomienie pliku Programowanie1.m z poziomu konsoli

```
>> Programowanie1
Podaj liczbę rzeczywistą
x = -9
liczba ujemna
```

2.2. Przypadek 2

Polecenie:

Teraz będziemy rozbudowywali Przypadek 1, w którym wyświetlaliśmy informację, tylko gdy liczba była ujemną. Zadanie polega na wczytaniu liczby i wyświetleniu na ekranie informacji, czy liczba jest ujemna czy jest nieujemna.

Rozwiązanie:

Zastosujemy konstrukcję if – else do opracowania rozwiązania w postaci m-pliku.

```
%program wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik
%na ekran
x = input(,Podaj liczbę rzeczywistą\nx = ,);
if x<0
disp('liczba ujemna')
else
disp(,liczba nieujemna')
end%koniec if
```

```
>> Programowanie2
Podaj liczbę rzeczywistą
x = 4
liczba nieujemna
```

2.3. Przypadek 3

Polecenie:

Dokonamy teraz rozbudowy dwóch poprzednich analizowanych przypadków. Opracuj algorytm, który dla zadanej (wczytanej) liczby wyświetli informację, czy dana liczba jest dodatnia, ujemna czy może równa się zero.

Rozwiązanie:

Do opracowania tego algorytmu zastosujemy najpierw konstrukcję zagnieżdżonej instrukcji warunkowej. Sporządzić m-plik jak poniżej.

```
%program wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik na ekran
x = input(,Podaj liczbę rzeczywistą\nx = ,);
if x<0
disp('liczba ujemna')
else
   if x>0
   disp(,liczba dodatnia')
   else
   disp(,liczba wynosi zero')
   end %koniec if dla x>0
end %koniec if dla x<0
```

```
>> Programowanie3
Podaj liczbę rzeczywistą
x = 7
liczba dodatnia
>> Programowanie3
Podaj liczbę rzeczywistą
x = 0
liczba wynosi zero
>> Programowanie3
Podaj liczbę rzeczywistą
x = -4
liczba ujemna
```

A teraz inne podejście do analizowanego problemu – zastosujemy konstrukcję if-else-elseif. Sporządzić m-plik jak poniżej.

```
%program wczytuje argument funkcji, sprawdza jego wartość i wypisuje
wynik na ekran
x = input(,Podaj liczbę rzeczywistą\nx = ,);
if x<0
disp('liczba ujemna')
elseif x==0
disp(,liczba wynosi zero')
else
disp(,liczba dodatnia')
end %koniec if
```

Rozwiązanie uruchomione z poziomu konsoli.

```
>> Programowanie4
Podaj liczbę rzeczywistą
x = 7
liczba dodatnia
>> Programowanie4
Podaj liczbę rzeczywistą
x = 0
liczba wynosi zero
>> Programowanie4
Podaj liczbę rzeczywistą
x = -8
liczba ujemna
```

2.4. Przypadek 4

Polecenie:

Dla zadanej przez użytkownika liczby naturalnej n wyświetl na ekranie liczby naturalne od 1 do n. Pamiętaj o kontroli błędu przy wczytywaniu liczby.

Rozwiązanie:

Do wyświetlania kolejnych liczb naturalnych wykorzystamy pętlę for. Licznikiem pętli będzie zmienna i od 1 do n, jej zadaniem będzie kontrola zachowania się pętli oraz określenie, ile razy zostanie powtórzony dany ciąg instrukcji. Z kolei kontrola błędu zostanie opracowana z użyciem instrukcji warunkowej if. Sporządzić m-plik jak poniżej.

```
n = input(,Podaj liczbę naturalną n\nn = ,);
if n<0 %kontrola błedu
  disp(,To nie jest liczba naturalna')
else
  for zmienna = 1:1:n, %licznik pętli
  disp(zmienna) %wyświetlanie wartości zmiennej
  end %koniec for
end %koniec if
```

```
>> Programowanie5
podaj liczbę naturalną n
n = 5
     1
     2
     3
     4
     5
>> Programowanie5
podaj liczbę naturalną n
n = -2
To nie jest liczba naturalna
```

2.5. Przypadek 5

Polecenie:

Napisz program, który dla zadanych liczb a, b, c rozwiąże równanie kwadratowe Rozważ trzy przypadki, w zależności od wartości delty.

Rozwiązanie:

```
%program do obliczania pierwiastków trójmianu kwadratowego
clear;
clc; % czyści ekran command window

disp(‚Program oblicza pierwiastki równania kwadratowego
a*x^2+b*x+c=0’)

a=input(‚Podaj współczynnik a:’); %wczytanie współczynnika a
b=input(‚Podaj współczynnik b:’); %wczytanie współczynnika b
c=input(‚Podaj współczynnik c:’); %wczytanie współczynnika c

d=b^2-4*a*c %obliczenie delty
```

```
if d<0 %sprawdzenie ile równanie ma rozwiązań
   disp(‚Równanie nie ma pierwiastków rzeczywistych’)
elseif d==0 %delta wynosi zero
   disp(‚Równanie ma jeden pierwiastek równy:’)
   x0=-b/(2*a) %obliczenie i wyświetlenie pierwiastka
elseif d>0 %delta dodatnia – dwa rozwiązania
   disp(‚Równanie ma 2 pierwiastki rzeczywiste wynoszące:’)
   x1=(-b-sqrt(d))/(2*a) %obliczenie i wyświetlenie pierwiastka
   x2=(-b+sqrt(d))/(2*a) %obliczenie i wyświetlenie pierwiastka
end %koniec if
```

```
Program oblicza pierwiastki równania kwadratowego a*x^2+b*x+c=0
Podaj współczynnik a:2
Podaj współczynnik b:4
Podaj współczynnik c:-9
d =
   88
Równanie ma 2 pierwiastki rzeczywiste wynoszące:
x1 =
  -3.3452
x2 =
   1.3452
```

2.6. Przypadek 6

Polecenie:

Opracuj program, który dla trzech zadanych liczb wyświetli informację, czy można z nich zbudować trójkąt.

Rozwiązanie:

Aby z trzech odcinków zbudować trójkąt, najdłuższy z nich musi być krótszy niż suma długości pozostałych.

```
%Program sprawdzający czy z podanych długości odcinków można zbudować trójkąt

% Przygotowanie pamieci Matlaba i ekranu Command Window
clear              % czysci pamiec Matlaba ze zmiennych, podstawień itp.
clc                % czysci ekran command window

% Wprowadzenie danych i sprawdzenie ich poprawności
a=input(,Podaj długość pierwszego boku:')
% wczytanie długości pierwszego odcinka i podstawienie jej za zmienna „a"

while a<=0
 % pętla „while" użyta do sprawdzenia czy użytkownik nie podał  liczby ujemnej
   disp(,Długość odcinka musi być liczba dodatnia!')
% jeśli taką podał, program przypomni, że długość musi być wartością dodatnią
   a=input(,Podaj jeszcze raz długość pierwszego boku:')
 % i raz jeszcze poprosi o poprawne wprowadzenie długości pierwszego odcinka
end                % zakończenie pierwszej pętli „while"

b=input(,Podaj długość drugiego boku:')
while b<=0         %kontrola poprawności
   disp(,Długość odcinka musi być liczba dodatnia!')
   b=input(,Podaj jeszcze raz długość drugiego odcinka:')
end                % zakończenie drugiej pętli «while"


c=input(,Podaj długość trzeciego boku:')
while c<=0
   disp(,Długość odcinka musi być liczba dodatnia!')
   c=input(,Podaj jeszcze raz długość trzeciego boku:')
end % zakończenie trzeciej pętli «while
```

```
% Obliczenia
 % warunek możliwości zbudowania z 3 odcinków trójkąta: suma długo-
ści dwóch % dowolnych odcinków musi być większa od długości trze-
ciego odcinka
if a+b<=c | b+c<=a | c+a<=b
% „|” oznacza „lub”;
 disp(‚Z podanych długości odcinków nie można zbudować trójkąta’
else
% w przeciwnym wypadku (czyli kiedy suma dwóch będzie zawsze
większa od trzeciego)
   disp(‚Z podanych długości odcinków można zbudować trójkąt’)
end                          % koniec instrukcji sterującej „if”
```

```
Podaj długość pierwszego boku:-4
a =
   -4
Długość odcinka musi być liczba dodatnia!
Podaj jeszcze raz długość pierwszego boku:2
a =
    2
Podaj długość drugiego boku:5
b =
    5
Podaj długość trzeciego boku:8
c =
    8
Z podanych długości odcinków nie można zbudować trójkąta
```



2.7. Zadania

Zadanie 1.
Wyświetl n kolejnych liczb parzystych, gdzie n podajemy z klawiatury.

.....
.....
.....
.....
.....
.....

Zadanie 2.
Napisz program, który dla zadanych parametrów a oraz b wyznaczy miejsce zerowe funkcji liniowej y=ax+b.

.....
.....
.....
.....
.....
.....

Zadanie 3.
Napisz program, który wystawi ocenę z testu. Użyj zagnieżdżonej instrukcji if - else
0 - 39 pkt – ocena niedostateczna
40 - 54 pkt – ocena dopuszczająca
55 - 69 pkt – ocena dostateczna
70 - 84 pkt – ocena dobra
85 - 98 pkt – ocena bardzo dobra
99 - 100 pkt – ocena celująca

.....
.....
.....
.....
.....
.....

Studium przypadku 3.

Macierze i układy równań

Cel ćwiczenia

- Utrwalenie wiadomości dotyczących działań na macierzach.
- Poznanie algorytmów stosowanych do rozwiązywania układów równań.
- Samodzielne rozwiązanie wybranych problemów matematycznych przy użyciu pakietu.

Problem



WAŻNE

Nazwa Matlab pochodzi od słów matrix laboratory, które dobrze oddają to, w czym Matlab jest najlepszy: szeroko rozumiana algebra liniowa. Jednym z ważniejszych tematów z obszaru algebry liniowej są macierze oraz ich zastosowanie do rozwiązywania układów równań.

W praktyce inżynierskiej **układy równań** napotyka się niezwykle często, pojawiają się w szeregu zagadnień projektowych, np. obliczanie temperatury ścianki w problemie przenikania ciepła. Rozwiązywanie tych często dużych układów równań jest pracochłonne. Metod rozwiązywania tego zagadnienia jest wiele, a wybór tej czy innej metody zależy od rodzaju zadania, oczekiwanej dokładności i środków technicznych będących w dyspozycji (szybkość procesora oraz objętość pamięci). Wykorzystujemy więc metody macierzowe lub metody numeryczne.

Wstęp teoretyczny

Rozwiązywanie układów równań w Matlabie sprowadza się do rozwiązania problemu:

Dane: macierze A i B.

Problem: Czy istnieje macierz X, dla której spełnione jest AX=B lub XA=B?

Rozwiązanie: Rozwiązaniem pierwszego równania, o ile istnieje, jest A\B, natomiast drugiego B/A.

Możemy rozróżnić trzy możliwości rozwiązania układu równań dla macierzy A wymiaru mxn.

- Układ oznaczony (jedno rozwiązanie), gdy m=n
- Układ nieokreślony (sprzeczny), gdy m>n
- Układ nieoznaczony (nieskończenie wiele rozwiązań) dla m<n



UWAGA

Zasadniczo układy równań dzielimy na dwa rodzaje: układy liniowe oraz układy nieliniowe.

Nieliniowość może być pochodzenia geometrycznego (np. w mechanice przyjęcie teorii dużych odkształceń czy przemieszczeń) lub fizycznego (nieliniowe związki konstytutywne, gdy materiał nie podlega liniowemu prawu sprężystości).

1. Liniowe układy równań

Rozpatruje się układ n równań liniowych zawierających n niewiadomych:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

który można zapisać w postaci macierzowej AX=B, gdzie A – macierz główna układu, X – wektor niewiadomych, B – wektor wyrazów wolnych:

$$\mathbf{A}=\begin{bmatrix} a_{11}\,a_{12}\,\ldots\,a_{1n} \\ a_{21}\,a_{22}\,\ldots\,a_{2n} \\ \ldots\ldots\ldots\ldots\ldots \\ a_{n1}\,a_{n2}\,\ldots\,a_{nn} \end{bmatrix} \qquad \mathbf{X}=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{B}=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Układem Cramera nazywamy układ równań liniowych AX=B, w którym A jest macierzą nieosobliwą (jej wyznacznik jest różny od zera). Układ Cramera ma dokładnie jedno rozwiązanie. Rozwiązanie to jest określone wzorem:

$$X=\frac{1}{detA}\begin{bmatrix} detA_1 \\ detA_2 \\ \ldots \\ detA_n \end{bmatrix},$$

gdzie:

$$A_j=\begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}.$$

Równość określająca rozwiązanie układu równań liniowych nazywamy wzorem Cramera i ma następującą postać:

$$x_1=\frac{detA_1}{detA_n}, \qquad x_2=\frac{detA_2}{detA_n}, \ldots\ldots, x_n=\frac{detA_n}{detA_n}.$$

Zauważmy, że układ równań posiada rozwiązanie, jeśli wyznacznik macierzy współczynników A jest różny od zera. Jeśli wyznacznik macierzy A jest równy zero, to układ równań jest albo sprzeczny (nie posiada rozwiązań), albo nieoznaczony (posiada więcej niż jedno rozwiązanie).



UWAGA

Twierdzenie Cramera używamy do rozwiązywania układów równań liniowych. Jednak można je stosować tylko i wyłącznie do równań, w których ilość niewiadomych jest równa ilości równań.

Metodą łatwiejszą przy rozwiązywaniu dużych układów równań jest eliminacja Gaussa lub metoda Gaussa-Jordana, jeszcze innym sposobem rozwiązania układu jest zastosowanie macierzy odwrotnej do rozwiązywania układów równań.

Wykorzystywane funkcje

Funkcja	Opis
[x,fval,exitflag,output,jacobian]=fsolve(@fun,x0,options) gdzie fval – wartość funkcji w punkcie zerowym exitflag – wyświetla warunek ukończenia obliczeń output – wyświetla dane o przebiegu optymalizacji jacobian – wyświetla wartość Jakobianu w punkcie rozwiązania options – opcje rozwiązania, takie jak wybór algorytmu	funkcja służy do rozwiązywania nieliniowych układów równań
optimset(parametr1, wartość1, parametr2, wartość2,…)	polecenie modyfikuje strukturę options
clear	usuwa zmienną z pamięci
clc	czyszczenie ekranu
eye	macierz jednostkowa o zadanych wymiarach
zeros	macierz kwadratowa złożona z samych zer o zadanych wymiarach

Funkcja	Opis
ones	macierz złożona z samych zer o zadanych wymiarach
disp(tekst)	wyświetla dany tekst na ekranie
det	wyznacznik macierzy
sum	
x=input(tekst)	wyświetla łańcuch tekst, oczekuje na wpisanie przez użytkownika danej liczbowej i przypisuje ją zmiennej liczbowej x
rand	macierz losowa o rozkładzie równomiernym o zadanych wymiarach
size	rozmiar macierzy

Tabela 1. Nazwy opcji funkcji optimset.

Opcja	Dopuszczalne wartości	Opis
Display	‚off' ‚iter' ‚final' ‚notify'	bez wyświetlania wyników wyświetlanie wyników po każdej iteracji wyświetlanie ostatecznego rozwiązania wyświetlanie wyników w koniecznym przypadku
FunValCheck	‚off' ‚on'	sprawdza, czy wyniki funkcji celu są poprawne on – wyświetla błędy, gdy funkcja celu zwraca wartości złożone lub NaN off – nie wyświetla błędów
MaxIter	Liczba całkowita	maksymalna dozwolona liczba iteracji
ToolFun	Dodatnia wartość skalarna	maksymalna dozwolona liczba obliczeń wartości funkcji
TolX	Dodatnia wartość skalarna	dokładność wykonywania obliczeń argumentu

3.1. Przypadek 1

Polecenie:
Wygeneruj następujące macierze:
B – macierz jednostkowa stopnia piątego
C – macierz zerowa stopnia piątego
D – macierz diagonalna z jedynkami na przekątnej, stopnia piątego
E – macierz jednostkowa wymiaru 3x5
F – macierz zerowa wymiaru 5x4

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Dla tak zadanych macierzy:
a) Wyświetl je na ekranie
b) Oblicz H=B-C
c) W macierzy C kolumna czwarta ma składać się z 2
d) Oblicz I = 2B-3C
e) Ostatni wiersz macierzy B od elementu drugiego ma wynosić 3
f) Drugi wiersz macierzy C ma wynosić -5
g) M=BC (mnożenie tablicowe)
h) N=BC (mnożenie macierzowe)
i) W macierzy C wiersz czwarty oraz piąty ma wynosić 3
j) Podziel macierzowo C/B
k) Oblicz wyznacznik macierzy B
l) Oblicz sumę elementów macierzy B
m) Oblicz iloczyn elementów macierzy B
n) Oblicz macierz transponowaną do macierzy B
o) Wyświetl wszystkie uzyskane wyniki

Rozwiązanie:
Sporządzić m-plik jak poniżej.

```
%tworzenie macierzy wypełnionej jedynkami, zerami i macierzy iden-
tyczności
%Przygotowanie pamięci Matlaba i ekranu Command Window
clear
clc  %czyści ekran command window
B = ones(5);
C = zeros(5);
D = eye(5);
E = ones(3,5);
F = zeros(5,4);
G = eye(3,5);
disp(,ZADANE MACIERZE')
disp(,Macierz B')
disp(B)
disp('Macierz C')
disp(C)
disp('Macierz D')
disp(D)
disp('Macierz E')
disp(E)
disp('Macierz F')
disp(F)
disp('Macierz G')
disp(G)
%działania na macierzach
disp(,DZIAŁANIA NA MACIERZACH')
H = B-C;
disp('Macierz H')
disp(H)
C(:,4)= 2;
disp('Macierz C')
disp(C)
I = 2*B-3*C;
disp(,Macierz I')
disp(I)
B(5,2:end)=3;
disp('Macierz B')
disp(B)
C(2,:)= -5;
disp('Macierz C')
disp(C)
```

```
%mnożenie tablicowe
M = B.*C;
disp('Macierz M')
disp(M)
%mnożenie macierzowe
N = B*C;
disp('Macierz N')
disp(N)
%dzielenie tablicowe
C(4:5,:) = 3;
Dzielenie = C./B;
disp(,Macierz C/B')
disp(Dzielenie)
%dzielenie macierzowe
%Dzielenie = C/B
%wyznacznik macierzy B, dla macierzy kwadratowej B
wyznacznikB = det(B);
disp(,Wyznacznik B')
disp(wyznacznikB)
%suma el. w macierzy
sumaB = sum(sum(B));
disp(,Suma elementów macierzy  B')
disp(sumaB)
%iloczyn el. w macierzy
iloczynB = prod(prod(B));
disp(,Iloczyn elementów macierz B')
disp(iloczynB)
%transpozycja
transB = B';
disp(,Macierz transponowana B')
disp(transB)
```

Efekt działania programu

```
ZADANE MACIERZE
Macierz B
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1

Macierz C
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0

Macierz D
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1

Macierz E
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1

Macierz F
    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0

Macierz G
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    1    1    1    1    1
    1    1    1    1    1
```

DZIAŁANIA NA MACIERZACH
Macierz H

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Macierz C

0	0	0	2	0
0	0	0	2	0
0	0	0	2	0
0	0	0	2	0
0	0	0	2	0

Macierz I

2	2	2	-4	2
2	2	2	-4	2
2	2	2	-4	2
2	2	2	-4	2
2	2	2	-4	2

Macierz B

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	3	3	3	3

Macierz C

0	0	0	2	0
-5	-5	-5	-5	-5
0	0	0	2	0
0	0	0	2	0
0	0	0	2	0

Macierz M

0	0	0	2	0
-5	-5	-5	-5	-5
0	0	0	2	0
0	0	0	2	0
0	0	0	6	0

```
Macierz N
   -5   -5   -5    3   -5
   -5   -5   -5    3   -5
   -5   -5   -5    3   -5
   -5   -5   -5    3   -5
  -15  -15  -15    5  -15

Macierz C/B
    0    0    0    2    0
   -5   -5   -5   -5   -5
    0    0    0    2    0
    3    3    3    3    3
    3    1    1    1    1

Wyznacznik B
    0

Suma elementów macierzy  B
   33

Iloczyn elementów macierzy B
   81

Macierz transponowana B
    1    1    1    1    1
    1    1    1    1    3
    1    1    1    1    3
    1    1    1    1    3
    1    1    1    1    3
```

3.2. Przypadek 2

Polecenie:

Napisz program, który po podanych przez użytkownika wymiarach macierzy generuje zadaną macierz o losowych elementach. Następnie dla wygenerowanej macierzy oblicza minimalny i maksymalny element macierzy.

Rozwiązanie:

```
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear;
clc; % czyści ekran command window
disp(‚OBLICZANIE MINIMALNEGO I MAKSYMALNEGO ELEMENTU MACIERZY’)
n=input(‚Podaj liczbę wierszy macierzy n: ‚); %podanie wymiaru macierzy
m=input(‚Podaj liczbę kolumn macierzy m: ‚); %podanie wymiaru macierzy

A=rand(n,m) %generowanie losowej macierzy o zadanych wymiarach

max=A(1,1); %najpierw zakładamy, że element max to pierwszy element macierzy

%obliczanie elementu max macierzy z użyciem pętli for
for i=1:n
   for j=1:m
      if A(i,j)>max
         max=A(i,j);
      end %end instrukcji if
   end %end pętli for
end%end pętli for
disp(‚Element największy macierzy’)
max %wyświetlenie elementu max


min=A(1,1); %najpierw zakładamy, że element max to pierwszy element macierzy

%obliczanie elementu min macierzy z użyciem pętli for
for i=1:n
   for j=1:m
      if A(i,j)<min
         min=A(i,j);
      end %end instrukcji if
   end %end pętli for
end %end pętli for
disp(‚Element najmniejszy macierzy’)
min %wyświetlenie elementu min
```

Efekt uruchomienia programu

```
OBLICZANIE MINIMALNEGO I MAKSYMALNEGO ELEMENTU MACIERZY
Podaj liczbę wierszy macierzy n: 3
Podaj liczbę kolumn macierzy m: 4
A =
   0.7655   0.4898   0.7094   0.6797
   0.7952   0.4456   0.7547   0.6551
   0.1869   0.6463   0.2760   0.1626
Element największy macierzy
max =
   0.7952
Element najmniejszy macierzy
min =
   0.1626
```

3.3. Przypadek 3

Polecenie:

Napisz program, który rozwiąże następujący układ równań Cramera:

Rozwiązanie:

```
%układ Cramera
A = [1 2 3
   3 1 2
   2 3 1]; %macierz A utworzona z liczb stojących przy niewiadomych
B = [14 11 11]'; %macierz wyrazów wolnych
[n, m] = size(A); %rozmiar macierzy A
for k=1:m
   Rob=A(:,k);
   A(:,k)=B;
   W(k)=det(A);
   A(:,k)=Rob;
End%koniec pętli for
D=det(A); %wyznacznik macierzy A
disp(,Rozwiązanie układu równań wynosi: ,)
X=W./D %zastosowanie wzoru na rozwiązania układu Cramera
```

Rozwiązanie tego układu równań jest następujące

```
>> Macierze2
Rozwiązanie układu równań wynosi:
X =
    1     2     3
```

3.4. Przypadek 4

Polecenie:
Rozwiąż następujący nieliniowy układ równań:

$$\begin{cases} x_1^4 - 2x_2 + 2\sqrt{x_4} = 0 \\ x_3 e^{x_1} + sinx_2 = 0 \\ -3x_3 + 2x_2 = 0 \\ x_2^2 \cdot cosx_4 + 3x_1 = 0 \end{cases}$$

Rozwiązanie:
Rozwiązanie nieliniowego układu równań w postaci kodu funkcji

```
function y=UkladRownan(x) %zdefiniowanie funkcji opisującej układ
równań
y=[ x(1).^4-2.*x(2)+2*sqrt(x(4))
x(3).*exp(x(1))+sin(x(2))
-3.*x(3)+2*x(2)
x(2).^2.*cos(x(4))]+3*x(1);
```

```
%Rozwiązanie nieliniowego układu równań
%liczba niewiadomych – cztery; liczba równań – cztery
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear all; close all;
clc  % czyści ekran command window
%definiujemy punkty startowe możliwie blisko
x0=[0 0 0 0];
%wywołanie solvera wraz z opcjami.
options=optimset(,Display','iter');
x = [fsolve(@UkladRownan,x0,options)]'
```

Iteration	Func-count	f(x)	step	optimality	radius
0	5	23.1744		23.1	1
1	10	1.02485	1	2.58	1
2	15	0.639944	0.368409	2.41	2.5
3	16	0.639944	0.318873	2.41	2.5
54	171	1.2768e-09	2.47424e-11	0.526	2.47e-11
55	176	1.23229e-09	6.18559e-12	0.516	6.19e-12
56	177	1.23229e-09	1.5464e-11	0.516	1.55e-11
57	178	1.23229e-09	3.86599e-12	0.516	3.87e-12
58	183	1.2224e-09	9.66498e-13	0.514	9.66e-13

```
Equation solved, solver stalled.

fsolve stopped because the relative size of the current step is less than
the
value of the step size tolerance squared and the vector of function va-
lues
is near zero as measured by the value of the function tolerance.
<stopping criteria details>
x =
   1.0e-05 *
   0.3770 + 0.0060i
  -0.9557 - 0.0059i
  -0.2600 - 0.0007i
  -0.0000 - 0.0000i
```



3.5. Zadania

Zadanie 1.

Napisz program, który po podanych przez użytkownika wymiarach macierzy generuje zadaną macierz o losowych elementach. Następnie dla wygenerowanej macierzy oblicza:

a) Sumę elementów na przekątnej
b) Sumę elementów każdego wiersza
c) Sumę elementów każdej kolumny
d) Minimalny i maksymalny element macierzy

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Zadanie 2.

Opracuj m-plik, który rozwiąże poniższy układ równań:

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

.....
.....
.....
.....
.....
.....
.....
.....
.....

Studium przypadku 4.

Funkcje

Cel ćwiczenia

- Opanowanie umiejętności związanych z funkcjami, takich jak: rysowanie wykresu funkcji, obliczanie wartości funkcji i obliczanie pierwiastków.
- Samodzielne rozwiązanie wybranych problemów matematycznych przy użyciu pakietu.

Problem

Jednymi z częściej stosowanych zagadnień numerycznych występujących w problemach technicznych a związanych z funkcjami są:

- obliczanie wartości funkcji
- rysowanie wykresu funkcji
- rozwiązywanie równań

Wszystkie wymienione zagadnienia zostaną rozwiązane przy użyciu oprogramowania Matlab.

Wstęp teoretyczny

Załóżmy, że dana jest funkcja rzeczywistego argumentu x. Celem naszych działań jest określenie rozwiązania następującego równania to znaczy, określenie wartości zmiennej x , dla których spełniona jest powyższa zależność. Należy zauważyć, że w ogólnym przypadku zadanie to nie jest proste, gdyż ze względu na nieliniowość funkcji nie jest nawet wiadomo, ilu rozwiązań można oczekiwać. Nie ma ogólnych, jednoznacznych metod rozwiązywania takich zadań. Znane są natomiast metody przybliżone w Matlabie, które opierają się na poszukiwaniu rozwiązań w drodze kolejnych iteracyjnych przybliżeń.

Wykorzystywane funkcje

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; f jest łańcuchem znaków określającym funkcję Matlaba
disp(tekst)	wyświetla dany tekst na ekranie
x=input(tekst)	wyświetla łańcuch tekst, oczekuje na wpisanie przez użytkownika danej liczbowej i przypisuje ją zmiennej liczbowej x
clear	usuwa zmienną z pamięci
clear all	usuwa wszystkie zmienne z pamięci
clc	czyszczenie ekranu
plot	wykres z dwiema osiami liniowymi
xlabel	wprowadzenie opisu osi x
ylabel	wprowadzenie opisu osi y
title	wprowadzenie tytułu na wykresie
legend	wprowadzanie pola legendy na bieżącym rysunku
grid on	wprowadzenie siatki głównej
hold on	włączony tryb dodawania nowych wykresów
MarkerSize	rozmiar znacznika
r*	punkt czerwony w stylu gwiazdka

4.1. Przypadek 1

Polecenie:

Napisz program, który dla zadanego przez użytkownika argumentu obliczy wartość funkcji

$$f(x)= 2x^2 +sin(x^2 +2x+2)$$

Rozwiązanie:

Sporządzić m-plik jak poniżej.

```
%funkcja wczytuje pojedynczy argument, oblicza wartość funkcji dla tego
%argumentu i wypisuje wynik
disp(,f1(x) = 2x^2+sin(x^2+2x+2)')%wyświetlenie funkcji
x = input(,Podaj argument funkcji\nx = ,);%podanie argumentu
%wzór funkcji f1(x) = 2x^2+sin(x^2+2x+2)
f1 = 2*x^2+sin(x^2+2*x+2);
%wypisanie wyniku
disp(,Wartość funkcji f_1 wynosi: ,)
disp(f1)
```

```
>> Funkcje1
f1(x) = 2x^2+sin(x^2+2x+2)
Podaj argument funkcji
x = 2
Wartość funkcji f_1 wynosi:
   7.4560
```

4.2. Przypadek 2

Polecenie:

Opracuj program, który na podstawie zadanych przez użytkownika końców przedziału a, b oraz kroku wyświetli wykres funkcji $f(x) = x^2 - 1, x \in [a, b]$)

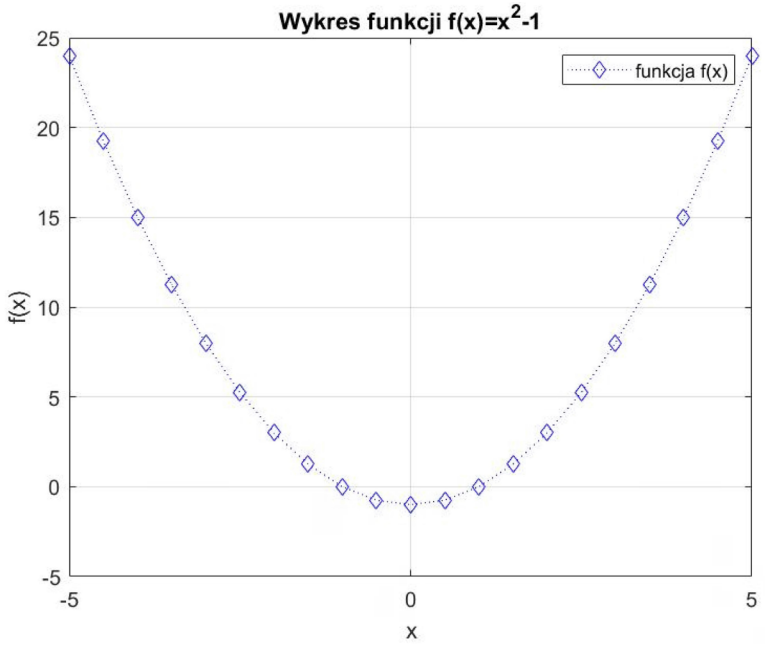
Rozwiązanie:

Sporządzić m-plik jak poniżej.

```
%dane: początek i koniec przedziału tabelaryzacji funkcji oraz krok
%funkcja oblicza wartość dla ciągu argumentów, a następnie wykonany
jest jej wykres
% funkcji f(x) = x^2 – 1
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear x f a b krok
%wczytanie danych
a = input(,Podaj początek przedziału tabelaryzacji\na = ,);
%wczytanie lewego końca przedziału
b = input(,Podaj koniec przedziału tabelaryzacji\nb = ,);
%wczytanie prawego końca przedziału
krok = input(,Podaj krok tabelaryzacji\ndx = ,);
% utworzenie ciągu argumentów x w przedziale [a, b]
x = [a:krok:b];
%obliczenie wartości funkcji dla ciągu argumentów
f = x.^2-1
%wykres funkcji
plot(x,f,'bd:')
%podpisy osi
xlabel(,x')
ylabel(,f(x)')
%tytuł wykresu
title(,Wykres funkcji f(x)=x^2-1')
legend('funkcja f(x)')
grid on %siatka prostokątna
```

```
>> Funkcje2
Podaj początek przedziału tabelaryzacji
a = -5
Podaj koniec przedziału tabelaryzacji
b = 5
Podaj krok tabelaryzacji
dx = 0.5
f =
  Columns 1 through 11
   24.0000   19.2500   15.0000   11.2500    8.0000    5.2500    3.0000
1.2500         0   -0.7500   -1.0000
```

```
  Columns 12 through 21
   -0.7500         0    1.2500    3.0000    5.2500    8.0000   11.2500   15.0000
   19.2500   24.0000
```



4.3. Przypadek 3

Polecenie:

Opracuj program, który na podstawie zadanych przez użytkownika końców przedziału a,b oraz kroku wyświetli wykres funkcji Przy wprowadzaniu końców przedziału oraz kroku zadbaj o poprawność danych tzn. b>a, krok>0.

Rozwiązanie:

Sporządzić m-plik jak poniżej.

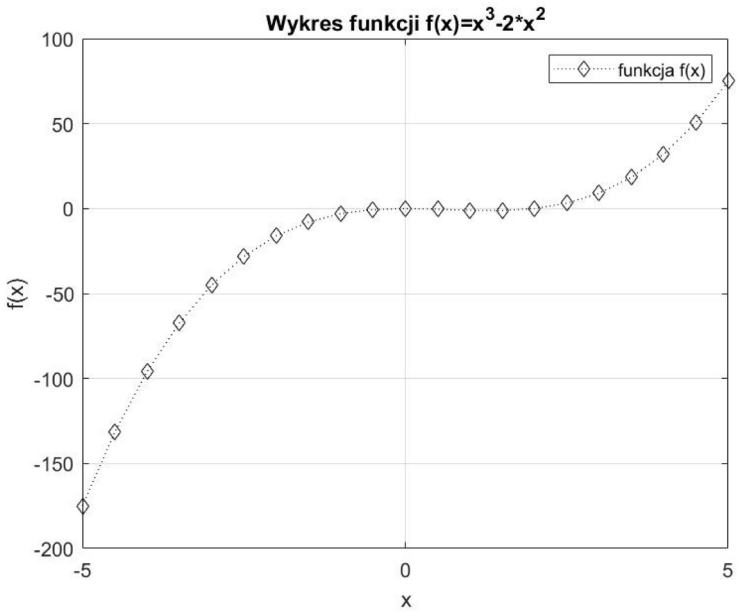
```
%dane: początek i koniec przedziału tabelaryzacji funkcji oraz krok dx
%funkcja oblicza wartość dla ciągu argumentów, a wykonany jest jej wykres
% f(x) = x^3 - 2*x^2
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear x f a b krok
%wczytanie danych
a = input(,Podaj początek przedziału tabelaryzacji\na = ,);
b = input(,Podaj koniec przedziału tabelaryzacji, b>a\nb = ,);

%sprawdzenie poprawności danej b wraz z oczekiwaniem na poprawną wartość b
while b<=a
disp(,niepoprawne b')
b = input(,Podaj koniec przedziału tabelaryzacji, b>a\nb = ,);
end%koniec while

krok = input(,Podaj krok tabelaryzacji krok>0\nkrok = ,);
%sprawdzenie poprawności danej krok wraz z oczekiwaniem na poprawną wartość
%krok
while krok<=0
disp(,niepoprawna wartość kroku')
krok = input(,Podaj krok tabelaryzacji krok>0\nkrok = ,);
end%koniec while

% utworzenie ciągu argumentów x w przedziale [a, b]
x = [a:krok:b];
%obliczenie wartości funkcji dla ciągu argumentów
f = x.^3-2*x.^2;
%wykres
plot(x,f,'kd:')
%podpisy osi
xlabel('x')
ylabel('f(x)')
%tytuł wykresu
title(,Wykres funkcji f(x)=x^3-2*x^2')
legend('funkcja f(x)')
grid on
```

```
Podaj początek przedziału fabularyzacji
a = -5
Podaj koniec przedziału fabularyzacji, b>a
b = -10
niepoprawne b
Podaj koniec przedziału fabularyzacji, b>a
b = 5
Podaj krok fabularyzacji krok>0
krok = 0.5
```



4.4. Przypadek 4

Polecenie:

Opracuj program, który na podstawie zadanych przez użytkownika końców przedziału a,b oraz liczby argumentów w zadanym przedziale wyświetli wykres funkcji

$f(x) = \begin{cases} x^2 + 1, x \leq 0 \\ x - 3, x > 0 \end{cases}$, w zadanym przedziale.

Przy wprowadzaniu przez użytkownika końców przedziału oraz liczby argumentów zadbaj o poprawność danych.

Rozwiązanie:

```
%dane: początek i koniec przedziału fabularyzacji funkcji oraz liczba
%argumentów funkcji
%funkcja oblicza wartość dla ciągu argumentów, a następnie wykonany jest
%jej wykres
% f(x) = x^2+1 dla x<=0
% f(x) = x-3 dla x>0
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear

%wczytanie danych
a = input(,Podaj początek przedziału tabelaryzacji\na = ,);
b = input(,Podaj koniec przedziału tabelaryzacji, b>a\nb = ,);

%sprawdzenie poprawności danej b wraz z oczekiwaniem na poprawną
wartość b
while b<=a
disp(,niepoprawne b')
b = input(,Podaj koniec przedziału tabelaryzacji, b>a\nb = ,);
end

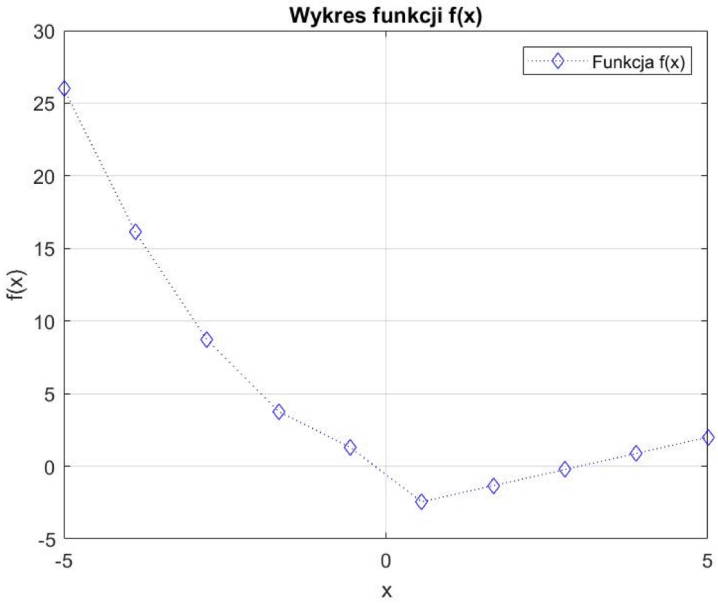
n = input(,Podaj liczbę argumentów funkcji n liczba naturalna\nn = ,);
%sprawdzenie poprawności danej n wraz z oczekiwaniem na poprawną
wartość
%n

while n<=0
disp(,Uwaga: n jest niepoprawne')
n = input(,Podaj liczbę argumentów funkcji n liczba naturalna\nn = ,);
end
%obliczenie dx
krok = (b-a)/(n-1);
%utworzenie ciągu argumentów x w przedziale [a, b]
x = [a:krok:b];
%obliczenie wartości funkcji dla ciągu argumentów
for i = 1:n,
if x(i)>0
f(i) = x(i)-3
else
f(i) = x(i)^2+1
end
end
```

```
%wykres
plot(x,f,'bd:')
%podpisy osi
xlabel(,x')
ylabel(,f(x)')
%tytuł wykresu
title(,Wykres funkcji f(x)')
legend('Funkcja f(x)')
grid on
```

```
>> Funkcja4
Podaj początek przedziału tabelaryzacji
a = -5
Podaj koniec przedziału tabelaryzacji, b>a
b = -10
niepoprawne b
Podaj koniec przedziału tabelaryzacji, b>a
b = 5
Podaj liczbę argumentów funkcji n liczba naturalna
n = -1
Uwaga: n jest niepoprawne
Podaj liczbę argumentów funkcji n liczba naturalna
n = 10
f =
    26
f =
   26.0000   16.1235
f =
   26.0000   16.1235    8.7160
f =
   26.0000   16.1235    8.7160    3.7778
f =
   26.0000   16.1235    8.7160    3.7778    1.3086
f =
   26.0000   16.1235    8.7160    3.7778    1.3086   -2.4444
f =
   26.0000   16.1235    8.7160    3.7778    1.3086   -2.4444   -1.3333
f =
   26.0000   16.1235    8.7160    3.7778    1.3086   -2.4444   -1.3333   -0.2222
```

```
f =
   26.0000   16.1235    8.7160    3.7778    1.3086   -2.4444   -1.3333   -0.2222
0.8889
f =
   26.0000   16.1235    8.7160    3.7778    1.3086   -2.4444   -1.3333   -0.2222
0.8889    2.0000
```



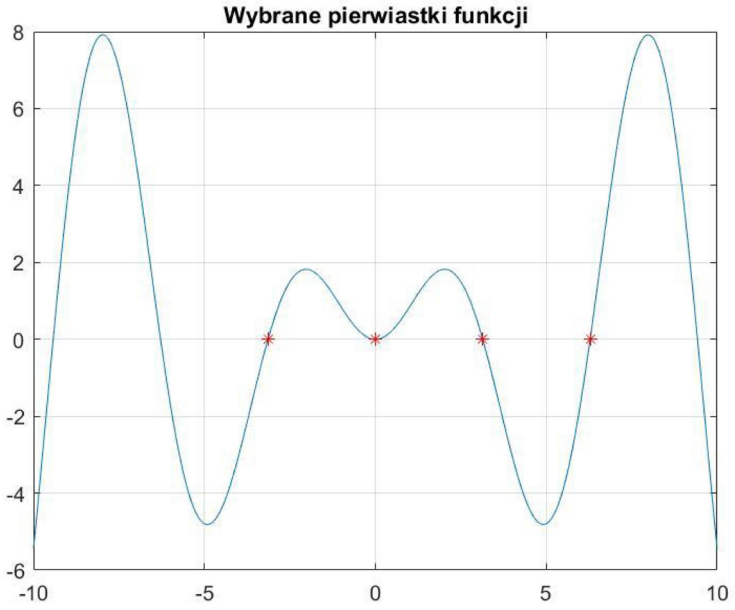
4.5. Przypadek 5

Polecenie:

Opracuj program, który narysuje wykres funkcji oraz zaznaczy na wykresie rozwiązania równania

Rozwiązanie:

```
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear all; clc, % czyści ekran command window
close all;
f4=@(x) x.*sin(x);
r1 = fzero(f4,-3); r2 = fzero(f4,-1); r3 = fzero(f4,0);
r4 = fzero(f4,2); r5 = fzero(f4,3); r6 = fzero(f4,5);
r=[r1 r2 r3 r4 r5 r6];
%prezentacja graficzna wyników
x=-10:.01:10; %przedział zmienności argumentów funkcji
y=f4(x); %wywołanie funkcji
plot(x,y)%wykres funkcji
grid on
hold on
plot(r,f4(r),'r*','MarkerSize',6)%wykres funkcji
title(,Wybrane pierwiastki funkcji')
```





4.6. Zadania

Zadanie 1.
Rozkład temperatury w pewnym układzie opisany jest funkcją:

$$T(x) = x^4 + 2x^3 - x + 3$$

Ustal, w jakim punkcie wartość temperatury jest równa 5.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Zadanie 2.
Napisz program, który rysuje wykres wielomianu $3x^2 + 2x - 7$ dla x zakresu od –10 do 10 oraz wyznacz dla tego wielomianu pierwiastki funkcją roots.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Studium przypadku 5.

Optymalizacja

Cel ćwiczenia

- Opanowanie podstawowych metod z zakresu optymalizacji funkcji z wykorzystaniem oprogramowania Matlab.
- Samodzielne rozwiązanie wybranych problemów matematycznych przy użyciu pakietu.

Problem

W rozdziale zajmiemy się problemem znalezienia minimum funkcji zmiennej rzeczywistej, postaramy się odpowiedzieć na poniższe pytania:

- Czy minimum istnieje?
- Ile wynosi wartość minimalna?

Tego typu problemy nazywamy problemami optymalizacyjnymi. Okazuje się, że wiele zastosowań matematyki sprowadza się właśnie do problemów optymalizacyjnych. Zarządzanie procesami produkcyjnymi, logistyka czy decyzje inwestycyjne to typowe problemy optymalizacyjne. Równie ważne zastosowania ma teoria optymalizacji w mechanice, elektronice oraz budownictwie.

Wstęp teoretyczny

Do podstawowych problemów optymalizacyjnych zaliczamy:

1. **Poszukiwanie minimum funkcji jednej zmiennej** postaci $\min_x f(x)$ z ograniczeniami postaci $x_1 < x < x_2$

gdzie:
x, x1 i x2 – skalary,
f(x) – funkcja celu zmiennej *x* zwracająca wartość w postaci skalara.

2. **Poszukiwanie minimum nieliniowej funkcji wielu zmiennych** postaci $\min_x f(x)$ z liniowymi i nieliniowymi oraz z równościowymi

i nierównościowymi ograniczeniamipostaci:

$$c(x) \leq 0$$
$$ceq(x) = 0$$
$$A \cdot x \leq 0$$
$$Aeq \cdot x = beq$$
$$lb \leq x \leq ub$$

gdzie:
x – wektor zmiennych o wymiarze 1 x *m*, gdzie m – liczba zmiennych zadania optymalizacji,
b, beq – wektory odpowiednio o wymiarach *n* x 1 i *k* x 1,
lb, ub – ograniczenia na zmienne w postaci wektorów o wymiarach 1 x *m*,
A, Aeq – macierze ograniczeń liniowych nierównościowych i równościowych o wymiarach odpowiednio *m* x *n* oraz *m* x *k*,
c(x), ceq(x) – ograniczenia nieliniowe w postaci układów równań odpowiednio nierównościowych i równościowych w postaci funkcji zwracających wartości ograniczeń w postaci wektorów dla danego argumentu *x*,
f(x) – jest funkcją celu zwracającą wartość w postaci skalara dla danego wektora argumentów *x*.

3. **Poszukiwanie minimum funkcji wielu zmiennych bez ograniczeń** (metoda gradientowa) postaci: $\min_x f(x)$

gdzie:
x – wektor zmiennych, którego wielkość zależy od liczby zmiennych zadania optymalizacji,
f(x) – jest funkcją celu zwracającą wartość w postaci skalara dla danego wektora argumentów *x*.



UWAGA

W środowisku Matlaba dostępne są biblioteki z procedurami ułatwiającymi obliczenia znajdujące różne zastosowania.

Jedną z takich bibliotek jest *Optimization Toolbox*, która dostarcza szereg narzędzi i funkcji do rozwiązywania zagadnień optymalizacyjnych.

Wykorzystywane funkcje

Skorzystanie z wybranych funkcji pakietu *Optimization Toolbox* umożliwia rozwiązanie wielu problemów optymalizacyjnych.

Składnia funkcji	Opis
x = fminbnd(fun,x1,x2) x = fminbnd(fun,x1,x2,options) [x,fval] = fminbnd(...) [x,fval,exitflag] = fminbnd(...) [x,fval,exitflag,output] = fminbnd(...)	Funkcja poszukuje minimum funkcji jednej zmiennej z ograniczeniami x1<x<x2, gdzie: x – szukane rozwiązanie zadania optymalizacji fun – funkcja celu zmiennej, x1, x2 – odpowiednio dolne i górne ograniczenie na zmienną x, options – zmienna w postaci struktury z zapisanymi informacjami o procesie rozwiązywania zadania optymalizacji (strukturę tę modyfikuje się, wykorzystując polecenie optimset), fval – wartość funkcji celu dla rozwiązania zadania optymalizacji, exitflag – komunikat wyjściowy o stanie solver'a, output – struktura zawierająca informacje o wykonaniu zadania optymalizacji

Składnia funkcji	Opis
x = fmincon(fun,x0,A,b) x = fmincon(fun,x0,A,b,Aeq,beq) x = fmincon(fun,x0,A,b,A-eq,beq,lb,ub) x = fmincon(fun,x0,A,b,A-eq,beq,lb,ub,nonlcon) x = fmincon(fun,x0,A,b,A-eq,beq,lb,ub,nonlcon,options) [x,fval] = fmincon(...) [x,fval,exitflag] = fmincon(...) [x,fval,exitflag,output] = fmin-con(...)	Funkcja znajduje minimum nieliniowej funkcji wielu zmiennych, gdzie: x – szukane rozwiązanie zadania optymalizacji, fun – nieliniowa funkcja celu wektora zmiennych *x*, x0 – wektor startowy, którego wielkość zależy od liczby zmiennych zadania optymalizacji, b, beq – wektory odpowiednio o wymiarach n x 1 i k x 1, lb, ub – ograniczenia na zmienne w postaci wektorów o wymiarach 1 x m, A, Aeq – macierze ograniczeń liniowych nierównościowych i równościowych o wymiarach odpowiednio m x n oraz m x k, nonlcon – funkcja opisująca ograniczenia nieliniowe równościowe i nierównościowe, options – zmienna w postaci struktury z zapisanymi informacjami o procesie rozwiązywania zadania optymalizacji (strukturę tę modyfikuje się, wykorzystując polecenie optimset), fval – wartość funkcji celu dla rozwiązania zadania optymalizacji, exitflag – komunikat wyjściowy o stanie solver'a, output – struktura zawierająca informacje o wykonaniu zadania optymalizacji

Składnia funkcji	Opis
x = fminsearch(fun,x0) x = fminsearch(fun,x0,options) [x,fval] = fminsearch(...) [x,fval,exitflag] = fminsearch(...) [x,fval,exitflag,output] = fminsearch(...)	Funkcja znajduje minimum funkcji wielu zmiennych bez ograniczeń, gdzie: x – szukane rozwiązanie zadania optymalizacji, fun – nieliniowa funkcja wielu zmiennych zależna od *x*, x0 – wektor startowy, którego wielkość zależy od liczby zmiennych zadania optymalizacji, options – zmienna w postaci struktury z zapisanymi opcjami rozwiązania zadania optymalizacji (strukturę tę modyfikuje się, wykorzystując polecenie optimset), fval – wartość funkcji celu dla rozwiązania zadania optymalizacji, exitflag – komunikat wyjściowy o stanie solver'a, output – struktura zawierająca informacje o wykonaniu zadania optymalizacji
clear all	usuwa wszystkie zmienne z pamięci
close all	zamyka wszystkie okna z wykresami
clc	czyszczenie ekranu
disp	wyświetla dany tekst na ekranie
x=input(tekst)	wyświetla łańcuch tekst, oczekuje na wpisanie przez użytkownika danej liczbowej i przypisuje ją zmiennej liczbowej x

5.1. Przypadek 1

Polecenie:
Znajdź minimum funkcji $y = x4 - 2x2 + x - 4$ na przedziale [–10,10].

Rozwiązanie:
Postawiony problem jest zadaniem poszukiwania minimum funkcji jednej zmiennej z ograniczeniami. W tym przypadku rozwiązanie zostanie opracowane w postaci funkcji

```
function wyj=MZ1(x)
wyj=x.^4-2*x.^2+x-4;
```

Następnie wywołujemy opracowną funkcję

```
>> x=fminbnd(@MZ1,-10,10)
x =
  -1.1072
```

5.2. Przypadek 2

Polecenie:

Znajdź minimum funkcji $f(x_1,x_2,x_3) = -x_1 \cdot x_2 + x_3$ spełniającej ograniczenia $0 < x_1 + 2x_2 + 2x_3 < 100$. Jako punkt startowy należy przyjąć [1,1,1].

Rozwiązanie:

Postawiony problem jest zadaniem poszukiwania minimum nieliniowej funkcji wielu zmiennych z ograniczeniami. Zaczniemy od opracowania funkcji MZ2.

```
function f = MZ2(x)
f = -x(1) * x(2) + x(3);
end %koniec funkcji
```

Wywołując poniższą komendę, otrzymamy rozwiązanie zadania optymalizacji:

```
>> A=[-1,-2,-2;1,2,2];
>> b=[0,100];
>> x0=[1;1;1];
>> [x,fval] = fmincon(@MZ2,x0,A,b)
Problem appears unbounded.
fmincon stopped because the objective function value is less than
the value of the objective function limit and constraints
are satisfied to within the value of the constraint tolerance.
<stopping criteria details>
x =
   1.0e+10 *
    2.6518
    1.8878
   -3.2137
fval =
  -5.0061e+20
```

5.3. Przypadek 3

Polecenie:
Znajdź takie x, dla którego poniższa funkcja osiągnie minimum.

$$f(x_1, x_2) = (x_2 - x_1^2)^2 + (2 - x_1)^2$$

Wczytaj dowolny punkt startowy.

Rozwiązanie:
Zadany problem jest przykładem zagadnienia poszukiwania minimum funkcji wielu zmiennych bez ograniczeń. Przykładowy m-plik może być postaci:

```
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear all; close all;
clc  % czyści ekran command window
disp(,OBLICZANIE MINIMUM FUNKCJI')
%wczytanie danych
a = input(,Podaj pierwszą współrzędną punktu startowego\na = ,);
b = input(,Podaj drugą współrzędną punktu startowego \nb = ,);
y = ,10*(x(2)-x(1)^2)^2+(2-x(1))^2'; %wzór zadanej funkcji
[x,fval] = fminsearch(y,[a b]) %użycie odpowiedniej funkcji optymalizacyjnej
```

```
OBLICZANIE MINIMUM FUNKCJI
Podaj pierwszą współrzędną punktu startowego
a = -1
Podaj drugą współrzędną punktu startowego
b = 1
x =
    2.0000    4.0000
fval =
   1.0269e-10
```

5.4. Zadania



Zadanie 1.

Znajdź minimum funkcji $y = 3x^3 - 2x^2 + 5x - 4$ na zadanym przez użytkownika przedziale

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Zadanie 2.

Znajdź minimum funkcji $f(x_1, x_2, x_3) = -x_1 \cdot x_2 + 2x_3$ spełniającej ograniczenia $0 < x_1 + 3x_2 + 2x_3 < 300$. Jako punkt startowy należy przyjąć punkt podany przez użytkownika.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Studium przypadku 6.

Interpolacja funkcji

Cel ćwiczenia

- Zaprogramowanie podstawowych metod interpolacji funkcji.
- Samodzielne rozwiązanie wybranych problemów matematycznych przy użyciu pakietu.

Problem

Wiele zjawisk fizycznych opisywanych jest przez funkcje, których postaci nie znamy, ale potrafimy obliczyć lub zmierzyć wartości tych funkcji oraz ich pochodnych dla określonych wartości argumentu. Na przykład, w określonych chwilach czasowych możemy mierzyć niektóre parametry sygnałów elektrycznych, pneumatycznych czy mechanicznych. Innymi typowymi przykładami zastosowania interpolacji jest obliczanie całek oraz pochodnych funkcji dyskretnych w czasie.

Wstęp teoretyczny

Interpolacją nazywamy postępowanie prowadzące do znalezienia wartości pewnej funkcji w dowolnym punkcie przedziału $(x_o; x_n)$ na podstawie znanych wartości tej funkcji w punktach $x_o; x_{1;\ldots} x_n$ zwanych węzłami interpolacji ($x_o < x_1 < \ldots < x_n$). Zadanie interpolacji odnosi się do działań zmierzających do przedstawienia funkcji w postaci ciągłej, gdy znana jest ona w postaci dyskretnej. Jest to zatem zadanie odwrotne do dyskretyzacji lub próbkowania wielkości ciągłej.

Funkcji interpolującej poszukuje się zwykle w pewnej określonej postaci. Najczęściej zakłada się, że jest to wielomian lub funkcja wymierna. Podstawowym celem interpolacji jest określenie wartości funkcji danej w postaci stabelaryzowanej dla zmiennej mieszczącej się pomiędzy danymi zawartymi w tablicy.

Wykorzystywane funkcje

Interpolacja jest zaimplementowana we wbudowanej funkcji interp1 Matlaba – patrz tabela poniżej.

Funkcja	Opis
yi = interp1(x,Y,xi)	dla węzłów x i wartości w nich Y buduje interpolację liniową i zwraca wartości tej interpolacji w punktach xi
yi = interp1(Y,xi)	dla węzłów 1:N, gdzie N=length(Y) i wartości w nich Y buduje interpolację liniową i zwraca wartości tej interpolacji w punktach xi
yi = interp1(x,Y,xi,metoda)	Interpolacja nie musi być liniowa. Zależne to jest od metody, która może przyjąć jedną z następujących wartości: ‚nearest' – interpolacja wartością najbliższą ‚linear' – wartość domyślna – interpolacja liniowa ‚spline' – spline kubiczny ‚pchip' – interpolacja splinem kubicznym Hermite'a
abs	wartość bezwzględna
max	wartość maksymalna
subplot	umieszczanie wielu rysunków w jednym oknie
plot	wykres z dwiema osiami liniowymi
title	wprowadzenie tytułu na wykresie
xlabel	wprowadzenie opisu osi x
ylabel	wprowadzenie opisu osi y
grid on	wprowadzenie siatki głównej
legend	wprowadzanie pola legendy na bieżącym rysunku
clear	usuwa wszystkie zmienne z pamięci
vander	macierz Vandermondea
linspace	wektor o jednostkowych odstępach liniowych między elementami
polyval(p,x)	oblicza wartość wielomianu o współczynnikach zawartych w wektorze p w punktach wyspecyfikowanych w zmiennej x

Inne możliwe funkcje do interpolacji to:

- interp2 – funkcja interpolująca w dwóch wymiarach dla danych określonych na regularnej siatce,
- interp3 – funkcja interpolująca w trzech wymiarach dla danych określonych na regularnej siatce.

6.1. Przypadek 1

Polecenie:

Dokonaj interpolacji liniowej funkcji $y = x^2\cos(\pi x - 2)$ w przedziale [–4, 4] z krokiem 0.5. Następnie narysuj wykres danej funkcji i funkcji przybliżającej w jednym układzie współrzędnych, natomiast wykres błędu interpolacji w drugim. Oblicz maksymalną wartość bezwzględnego błędu interpolacji w rozpatrywanym przedziale.

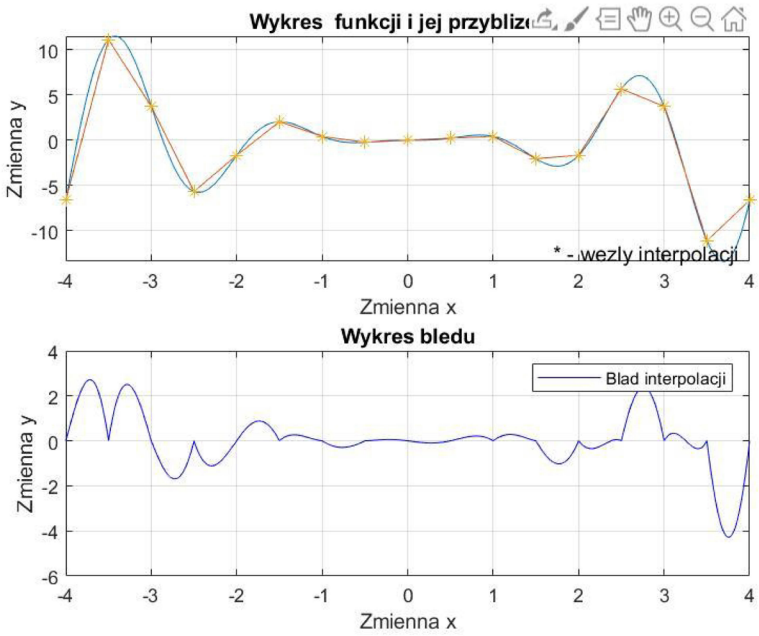
Rozwiązanie:

```
x=-4:0.01:4 %przedział, na którym funkcja będzie interpolowana
y=(x.^2).*cos(pi*x-2) %wzór funkcji

z=-4:0.5:4
%(z,y1) – współrzędne węzłów interpolacji
y1=(z.^2).*cos(pi*z-2)
yi=interp1(z,y1,x) %yi – wartości funkcji przybliżającej w przedziale interpolacji

blad=y-yi %błąd interpolacji
blm=max(abs(blad)) %max wartość błędu interpolacji
```

```
subplot(2,1,1) %dwa wykresy na jednym rysunku
plot(x,y,x,yi,z,y1,'*')
grid on
title(,Wykres  funkcji i jej przybliżenia')
xlabel(,Zmienna x')
ylabel(,Zmienna y')
text(1.7,-12.5,'* - węzły interpolacji')
subplot(2,1,2)
plot(x,blad,'b')
grid on
title('Wykres błędu')
xlabel(,Zmienna x') %oznaczenie osi X
ylabel(,Zmienna y') %oznaczenie osi Y
legend(,Błąd interpolacji ,)
```



6.2. Przypadek 2

Polecenie:

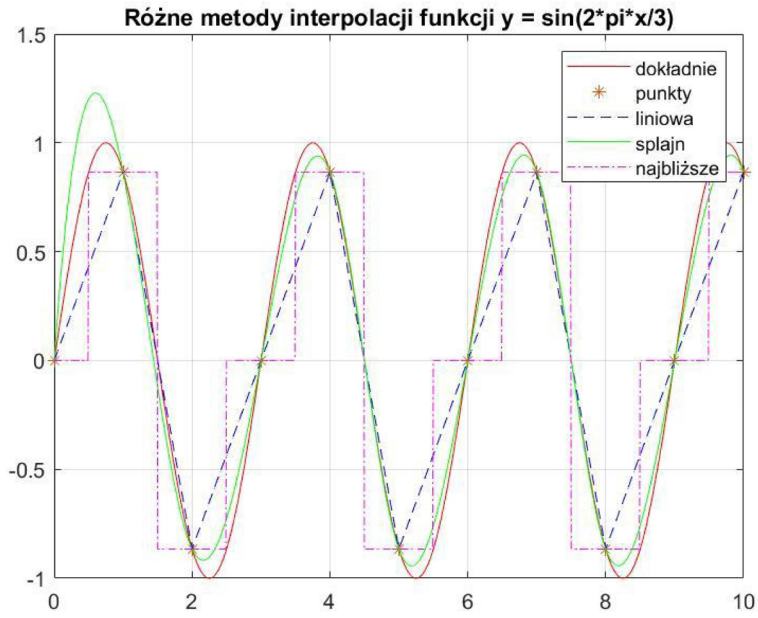
Opracuj skrypt, który dokona interpolacji funkcji $f(x) = sin\frac{2\pi x}{3}, x \in [0,10]$. Dokonaj interpolacji różnymi dostępnymi metodami w Matlabie.

Rozwiązanie:

W rozwiązaniu zagadnienia skorzystamy z dostępnej funkcji yi = interp1(x,Y,xi,metoda), dla której w zależności od wybranej metody można przeanalizować różne metody interpolacji:
‚nearest' – interpolacja wartością najbliższą
‚linear' – interpolacja liniowa, jest to wartość domyślna funkcji
‚spline' – spline kubiczny
‚pchip' – interpolacja splinem kubicznym

Sporządzić m-plik jak poniżej.

```
% Przygotowanie pamięci Matlaba i ekranu Command Window
clear;
x = 0:0.01:10;
 %Punkty do obliczania wartości funkcji dla argumentu x od 1 do 10 z krokiem 0.01
y = sin(2*pi*x/3);                    %Dokładne wartości zadanej funkcji
xp = 0:10;                            %Węzły interpolacji
yp = sin(2*pi*xp/3);                  %Wartości funkcji w węzłach
y_lin =   interp1(xp, yp, x);         % Interpolacja liniowa
y_spl =  interp1(xp, yp, x, „spline"); % -"- krzywa sklejana trzeciego stopnia
y_najbl = interp1(xp, yp, x, „nearest");% -"- schodkowa (wartość najbliższa)
plot(x,  y, "r-", ...
    xp, yp, "*",...
    x,  y_lin, "b--",...
    x,  y_spl, "g-",...
    x,  y_najbl, „m-.") %zastosowanie różnych oznaczeń na wykresie dla różnych metod
title(‚Różne metody interpolacji funkcji y = sin(2*pi*x/3)')
legend(‚dokładnie','punkty','liniowa','splajn','najbliższe')
osie.isoview="on"; %Tryb isoview
grid on
```



6.3. Przypadek 3

Polecenie:

Wiemy, że liczba studentów (w tys.) na WSEI w kolejnych latach kształtowała się następująco:

Rok	2015	2016	2017	2018	2019	2020
Liczba studentów (w tys.)	13.13	13.32	13.58	14.81	15.27	15.74

Dokonaj interpolacji wielomianowej dla danych.

Rozwiązanie:

Interpolacja wielomianowa ma postać:

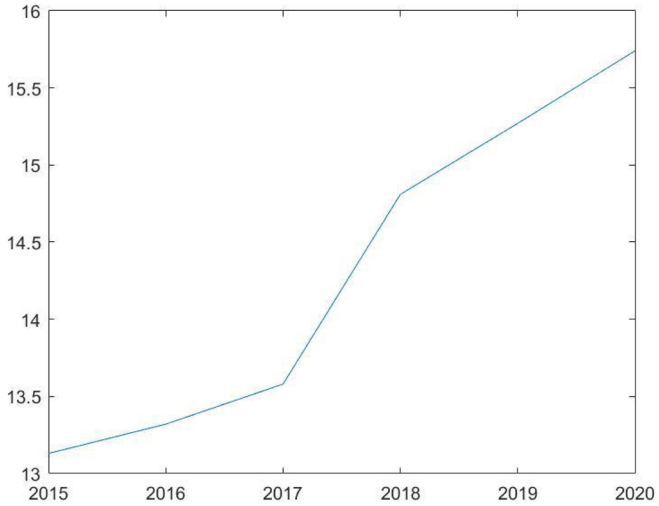
$$P_{n-1}(x) = c_1 x^{n-1} + c_2 x^{n-2} + \cdots . + c_n$$

a do jej obliczenia wykorzystujemy macierz Vandermonde'a postaci:

$$\begin{pmatrix} 1 & \cdots & x_1^{n-1} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_n^{n-1} \end{pmatrix}$$

Sporządzić m-plik jak poniżej.

```
studenci=[13.13 13.32 13.58 14.81 15.27 15.74];
rok[2015 2016 2017 2018 2019 2020];
plot(rok,studenci)
A=vander(rok) %w interpolacji wielomianowej wykorzystujemy macierz
Vandermonde'a
studenci = studenci'; % macierz prawych stron układu
rob=A\ studenci
xnowe=linspace(min(rok),max(rok),100);
ynowe=polyval(rob,xnowe);
plot(rok,cena,xnowe,ynowe);
grid on
title(,Interpolacja liczby studentów WSEI')
xlabel(,Rok')
ylabel(,Liczba studentów')
```



6.4. Zadania



Zadanie 1.

Napisz program przeprowadzający interpolację zbioru punktów z poniższej tabeli z wykorzystaniem funkcji interp1. Program powinien przeprowadzać interpolację następującymi metodami:

a) najbliższego sąsiada (nearest)
b) liniowa (linear)
c) sześcienne funkcje sklejane (spline)
d) wielomian 3-go stopnia (cubic)

Która z metod jest dokładniejsza?

Zmień skrypt tak, żeby interpolował wartości losowe – skorzystaj z funkcji rand.

X	-1	0	2	3	4	7	9	15
Y	-12	-10	0	-2	5	6.5	8.3	14

Studium przypadku 7.

Graficzny Interfejs Użytkownika

Cel ćwiczenia

- Projektowanie z wykorzystaniem Graficznego Interfejsu Użytkownika (GUI).

Problem

GUI (Graphical User Interface) jest systemem ułatwiającym pracę w Matlabie. Umożliwia on użytkownikowi automatyczne wydawanie wielu poleceń, a także ułatwia pracę osobom, które słabo znają Matlaba i jego komendy.



UWAGA

Polecenia w GUI użytkownik będzie wydawał na przykład poprzez naciśnięcie odpowiedniej kontrolki.
Projektowanie GUI jest także pierwszym krokiem do tworzenia aplikacji.

Do większości problemów analizowanych w książce możemy zaprojektować GUI, które zautomatyzuje dane zagadnienie. Opracowywanie interfejsów graficznych przećwiczymy na dwóch zagadnieniach: rysowanie wykresu funkcji oraz prostych obliczeń arytmetycznych.

Wstęp teoretyczny

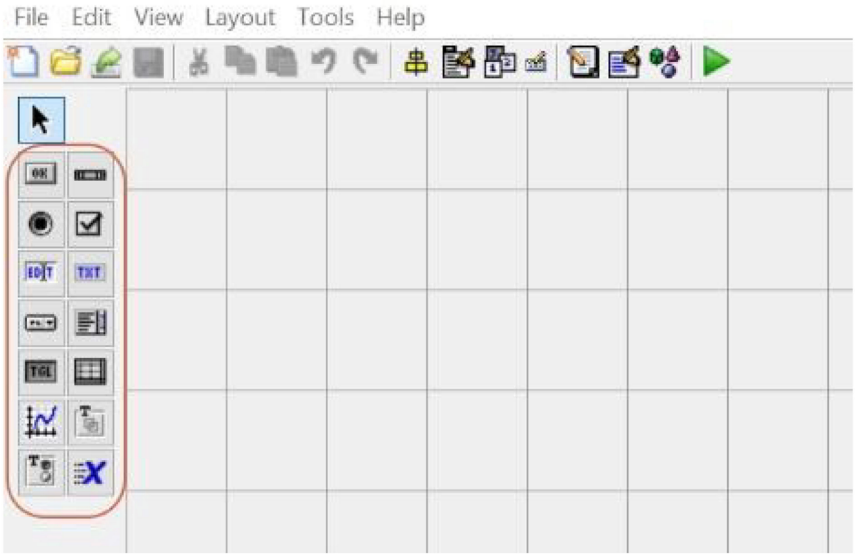
WAŻNE

Do tworzenia oraz edycji graficznego interfejsu użytkownika służy edytor wywoływany poleceniem guide.

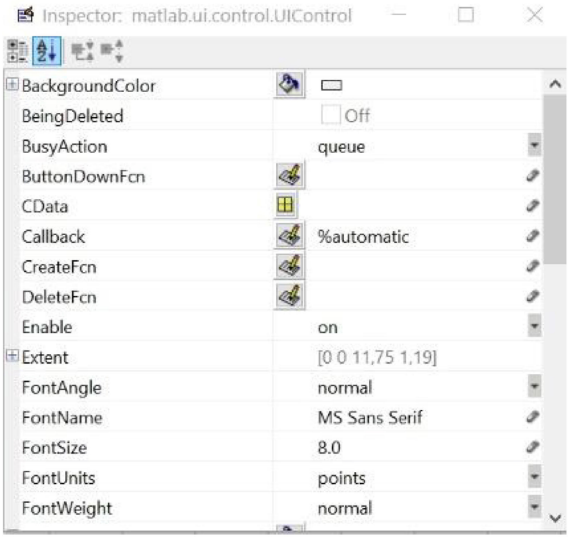
Wynikiem pracy edytora są dwa pliki: plik.fig oraz plik.m zawierające komplet informacji o zaprojektowanym oknie.

Plik o rozszerzeniu .m zawiera procedury obsługi elementów sterujących i służy do uruchomienia okna interfejsu.

Po uruchomieniu GUIDE pojawia się nam okno (patrz Rysunek 4) wraz z dostępnymi kontrolkami do wyboru. Kontrolki wybieramy, klikając na wybraną kontrolkę, a następnie na obszar, w którym ma być umieszczona. Po kliknięciu prawym przyciskiem myszy na daną ikonkę otwieramy okno *Prospery Inspector*, który umożliwi nam zmianę atrybutów danej kontrolki (patrz Rysunek 5).



Rysunek 4. Okno GUIDE z dostępnymi kontrolkami.



Rysunek 5. Prospery Inspector.

Wykorzystywane funkcje

Kontroli/Funkcje	Opis
Push button	standardowy przycisk, wciśnięcie może wywoływać określoną akcję; po naciśnięciu przycisku wywoływana jest funkcja Callback, która obsługuje akcję wykonywaną przez użytkownika
Slider	suwak umożliwiający zmianę wartości, możemy go ustawić w dwóch płaszczyznach, minimalne przesunięcie suwaka określone jest domyślnie na 0.01x maksymalny zakres
Radio button	umożliwia wybór danej opcji
Edit text	okno wprowadzania danych
Static text	wyświetlanie danych
Button group	pozwala grupować przyciski, bez grupowania każda kontrolka istnieje jako oddzielny obiekt
Axes	przeznaczeniem tego jest wyświetlanie danych, głównie wykresów, ale możemy również wykorzystać ją do pokazywania animacji oraz statycznych obrazów
plot	podstawowa funkcja służąca do wyświetlania grafiki dwuwymiarowej
str2num	konwertuje napis do zmiennej liczbowej

Tabela 2. Parametry kontrolek

Parametr	Opis
Max	wartość maksymalna, jaką może przyjąć parametr *Value*
Min	wartość minimalna, jaką może przyjąć parametr *Value*
String	tekst wyświetlany na obiekcie
Tag	właściwa nazwa kontrolki
Title	tekst wyświetlony jako tytuł grupy przycisków

7.1. Przypadek 1.

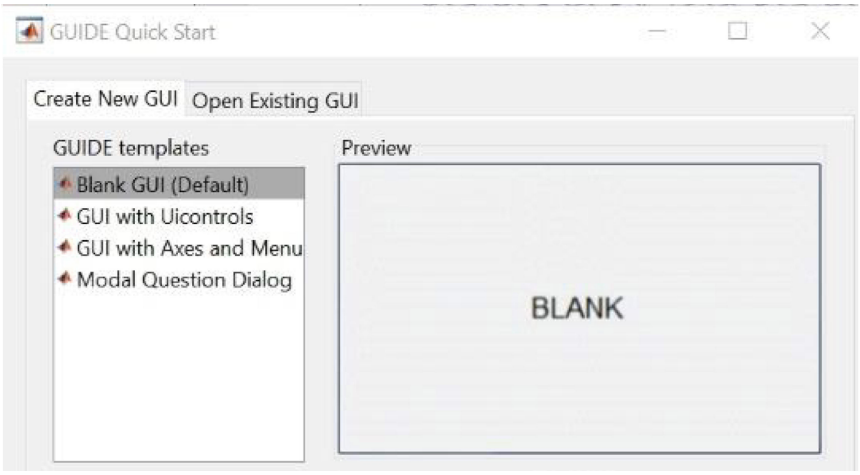
Polecenie:

Zaprojektowanie i opracowanie GUI, które będzie rysowało wykres funkcji cosinus dla , dodatkowo użytkownik będzie mógł zmieniać amplitudę i pulsację.

Rozwiązanie:

Krok 1.

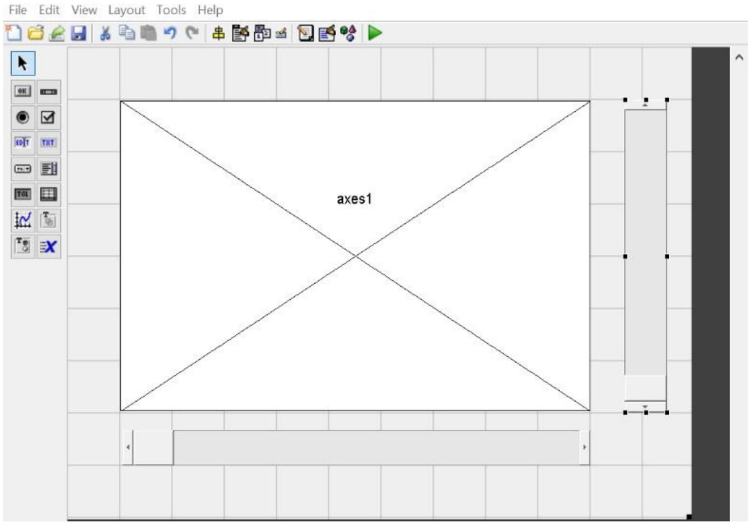
Uruchom aplikację *guide*, wybierając Blank GUI.



Rysunek 6. Uruchomienie GUI.

Krok 2.

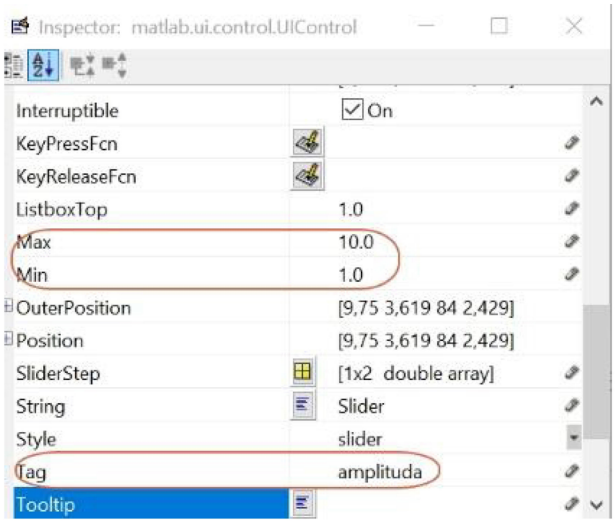
Na oknie rozmieszczamy następujące elementy: jedna kontrolka *axes* oraz dwie kontrolki *slider* – patrz Rysunek 7.



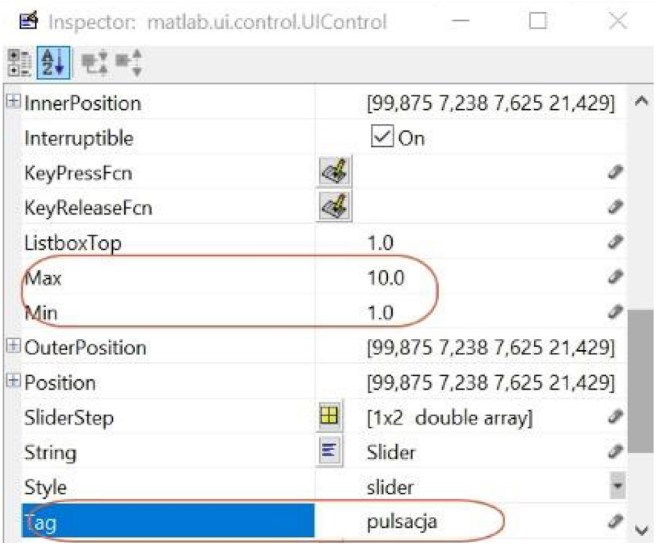
Rysunek 7. GUI – wykres funkcji.

Krok 3.

Otwórz okno *Prospery Inspector* obiektu *Slider1* oraz *Slider 2*. Dla obu kontrolek zmień wartości zgodnie z Rysunkiem 8 oraz Rysunkiem 9, dodatkowo ustaw wartość *Value* na 5 (dla obu kontrolek).



Rysunek 8. *Prospery Inspector* dla konsolki *Slider1*.



Rysunek 9. *Prospery Inspector* dla konsolki *Slider2*.

Krok 4.

W pliku *.m uzupełnij dwie funkcje odpowiedzialne za obsługę zdarzeń zgodnie z poniższym wzorem.

```
function pulsacja_Callback(hObject, eventdata, handles)
% hObject    handle to pulsacja (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range of slider

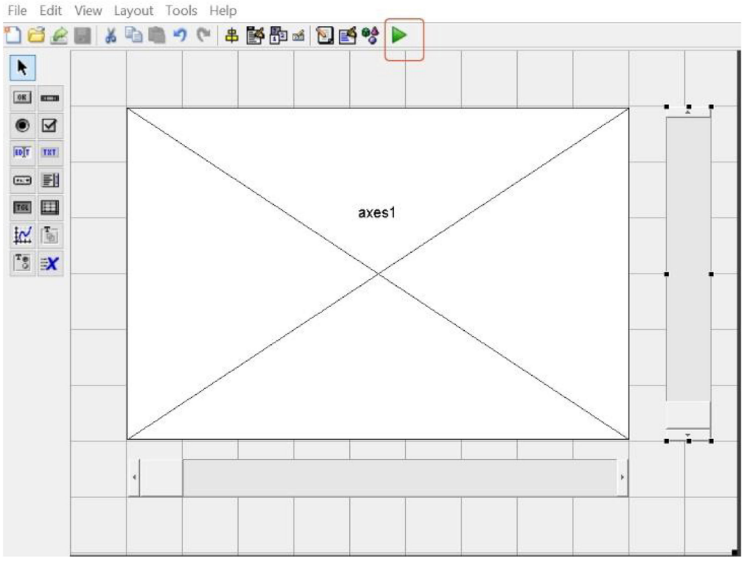
omega = get(hObject,'Value')
 amp = get(handles.amplituda,'Value')
 x = [-2*pi : 0.1 : 2*pi] %zakres argumentu
 handles.wykres = plot( amp * cos( omega * x ) ) %wzór funkcji
% --- Executes during object creation, after setting all properties.
```

```
function amplituda_Callback(hObject, eventdata, handles)
% hObject    handle to amplituda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

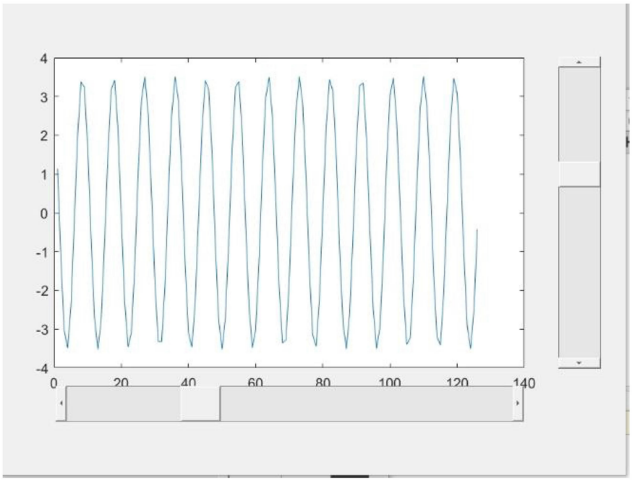
% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range of slider
amp = get(hObject,'Value')
 omega = get(handles.pulsacja,'Value')
 x = [-2*pi : 0.1 : 2*pi] %zakres argumentu
 handles.wykres = plot( amp * cos( omega * x ) ) %wzór funkcji
```

Krok 5.

Zapisz plik w bieżącym katalogu i uruchom aplikację – Rysunek 10. Po uruchomieniu aplikacji aktualnie wygląda ona tak jak na Rysunku 11.



Rysunek 10. Uruchomianie aplikacji.

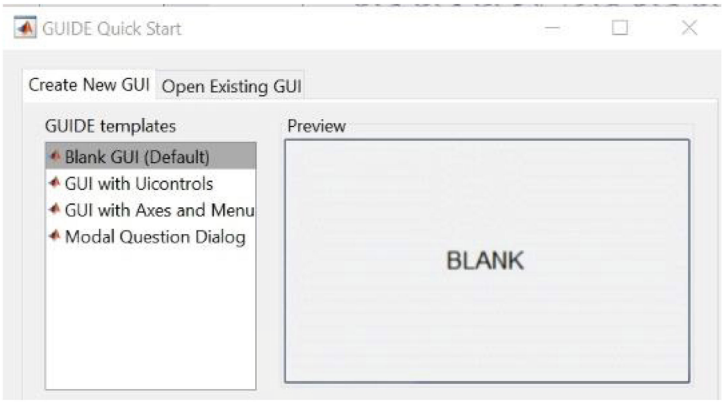


Rysunek 11. Aplikacja GUI – wykres funkcji cosinus.

7.2. Przypadek 2

Polecenie:
Opracuj GUI spełniające funkcje prostego kalkulatora z działaniami: dodawanie, odejmowanie, mnożenie i dzielenie.

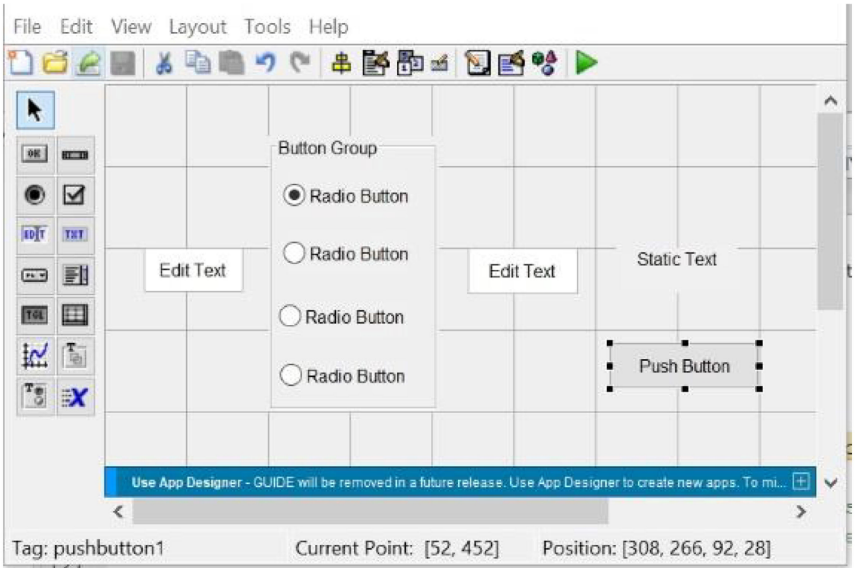
Rozwiązanie:
Krok 1.
Uruchom aplikację *guide*, wybierając Blank GUI.



Rysunek 12. Uruchomienie GUI.

Krok 2.

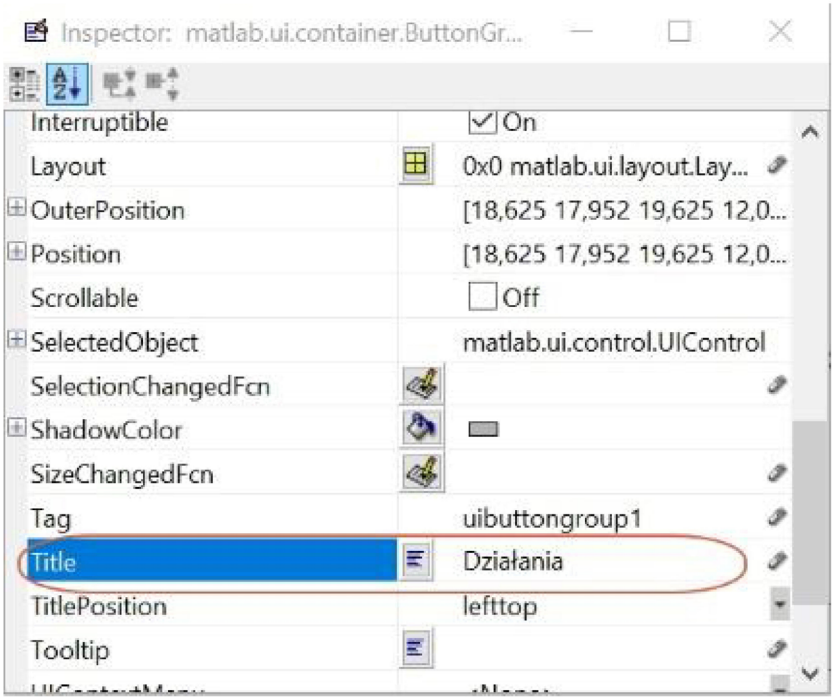
Na oknie rozmieszczamy następujące elementy: dwie kontrolki *edittext*, jedna kontrolka *buttongroup*, jedna kontrolka *statictext*, cztery kontrolki *radiobutton*, jedna kontrolka *pushbutton*. W tym momencie GUI wygląda jak na Rysunku 13.



Rysunek 13. GUI – kalkulator.

Krok 3.

Otwórz *Prospery Inspector* obiektu *Button Group*. Zmień właściwości '*Title*' na 'Działania' – patrz Rysunek 14.



Rysunek 14. Prospery Inspector dla konsolki Button Group.

Krok 4.

Zmień kolejno właściwości konsolek zgodnie z informacjami z Tabeli 2.

Tabela 2. Właściwości *Prospery Inspector* dla konsolek

Konsolka	Właściwość	Zmiana
Radio Button1	*String*	+
	Tag	plus
Radio Button2	*String*	-
	Tag	minus
Radio Button3	*String*	*
	Tag	razy
Radio Button4	*String*	/
	Tag	podziel

Konsolka	Właściwość	Zmiana
Edit Text1	*String*	
	Tag	liczba1
Edit Text2	*String*	
	Tag	liczba2
Static Text1	*String*	
	Tag	wynik
Push button	*String*	Oblicz
	Tag	oblicz

Krok 5.

Klikamy prawym przyciskiem myszy na kontrolkę Pus Button i z menu View Callback wybieramy funkcję Callback – patrz Rysunek 15.



Rysunek 15. Funkcja Callback.

Krok 6.

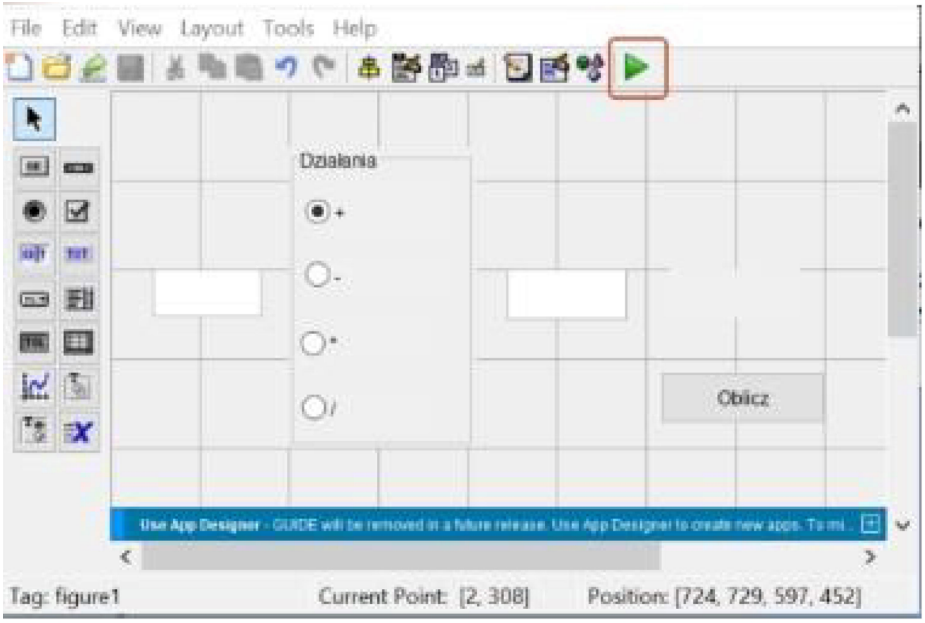
Funkcje Callback uzupełniamy poniższym kodem

```
function oblicz_Callback(hObject, eventdata, handles)
% hObject    handle to oblicz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = str2num( get( handles.liczba1 , ,string' ) );
b = str2num( get( handles.wynik , ,string' ) );

if( get( handles.plus , ,value' ) == 1 ) c = a + b; end;
if( get( handles.minus , ,value' ) == 1 ) c = a - b; end;
if( get( handles.razy , ,value' ) == 1 ) c = a * b; end;
if( get( handles.podziel , ,value' ) == 1 ) c = a / b; end;

set( handles.oblicz, ,string' , num2str(c) );
```

Krok 7.
Uruchamiamy nasze GUI – patrz Rysunek 16 oraz Rysunek 17.



Rysunek 16. Uruchamianie GUI.



Rysunek 17. GUI – kalkulator.

7.3. Zadania



Zadanie 1.

Opracuj GUI, którego zadaniem będzie rysowanie wykresu funkcji sinus z możliwością zmiany amplitudy i pulsacji.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Zadanie 2.

Opracuj GUI, który będzie prostym kalkulatorem z następującymi działaniami: dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie, pierwiastkowanie.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Bibliografia

1. Banasiak, K. (2017). Algorytmizacja i programowanie w MATLABIE. Legionowo: Wydawnictwo BTC.
2. Bjarck, A., Dahlquist, G. (1987). Metody numeryczne. Warszawa: PWN.
3. Brzózka, J., Dorobczyński, L. (2008). MATLAB: środowisko obliczeń naukowo-technicznych. Warszawa: Wydawnictwo Naukowe PWN.
4. Dyka, E., Markiewicz, P., Sikora, R. (2006). Modelowanie w elektrotechnice z wykorzystaniem środowiska MATLAB. Łódź: Wydawnictwo Politechnika Łódzka.
5. Kincaid, D., Cheney, E.W. (2006). Analiza numeryczna. Warszawa: Wydawnictwa Naukowo-Techniczne.
6. Kłosowski, P., Ambroziak, A. (2011). Metody numeryczne w mechanice konstrukcji z przykładami w programie Matlab. Gdańsk: Wydawnictwo Politechniki Gdańskiej.
7. Kula, K. (2009). Zbiór zadań z podstaw automatyki. Rozwiązania analityczne oraz komputerowe w środowisku Matlab. Gdynia: Wydawnictwo Akademii Morskiej w Gdyni.
8. Lubowiecka, I., Ambroziak, A. (2016). MATLAB i jego środowisko. Gdańsk: Wydawnictwo Politechniki Gdańskiej, 2016.
9. Mrozek, B., Mrozek, Z. (2010). MATLAB i Simulink: poradnik użytkownika. Gliwice: Helion.
10. Rudra, P. (2007). Matlab 7 dla naukowców i inżynierów. Warszawa: PWN.
11. Regel, W. (2003). Wykresy i obiekty graficzne w Matlab. Warszawa: Mikom.
12. Stachurski, M. (2003). Metody numeryczne w programie Matlab. Warszawa: Mikom.
13. Szymczyk, E. (2006). Matlab dla mechaników. Warszawa: Wyd. WAT.
14. Wanat, K. (1994). Algorytmy numeryczne. Gliwice: Helion.

Grafika użyta do rozdzielania poszczególnych rozdziałów:

1. Ad. 1 – tungnguyen0905/Pixabay.com
2. Ad. 2 – pressfoto/Freepik.com
3. Ad. 3 – AbsolutVision/Pixabay.com
4. Ad. 4 – pressfoto/Freepik.com
5. Ad. 5 – ijeab/Freepik.com

Z recenzji:

Podręcznik przybliża środowisko i podstawowe procedury programu MATLAB®. Pakiet MATLAB® jest potężnym narzędziem do symulacji i obliczeń naukowych i inżynierskich. Wielkim atutem programu jest jego architektura, która pozwala rozbudowywać program o nowe funkcje i algorytmy. Różnorodność zastosowań, w których można wykorzystać program, powoduje, że specjaliści różnych dziedzin nauki mogą znać zupełnie inne funkcje i narzędzia. MATLAB® jest efektywnym i uniwersalnym językiem programowania wysokiego poziomu. Jest też interaktywnym środowiskiem do obliczeń naukowych i technicznych oraz do tworzenia algorytmów i analizy danych. Ma znakomite narzędzia do dwu i trójwymiarowej wizualizacji danych i wyników obliczeń w postaci różnych wykresów, grafiki biznesowej i animacji. Jest też przyjaznym, interaktywnym środowiskiem integrującym niezawodne algorytmy matematyki stosowanej i liczne moduły rozszerzeń i narzędzi zorientowanych na określone dziedziny zastosowań. MATLAB® umożliwia rozwiązywanie różnorodnych problemów poprzez ułatwienie dostępu do efektywnych algorytmów obliczeniowych i możliwość wizualizacji wyników. W znacznym stopniu wyparł z obliczeń naukowo-technicznych języki uniwersalne (Fortran, C, C++), ograniczając ich zasadniczą rolę do funkcji oprogramowania narzędziowego. Środowisko MATLAB jest otwarte. Daje to możliwość zarówno wykorzystywania oprogramowania istniejącego, jak i budowania oprogramowania własnego, w tym aplikacji App, toolboksów i bibliotek modeli symulacyjnych.

Innowacyjnym i odróżniającym od dostępnej literatury jest również fakt, że ta publikacja zawiera dużo czytelnych przykładów. Przekazane w przystępnej formie informacje dokładnie prezentują zagadnienia skoncentrowane na wektorach, macierzach, tablicach i wizualizacji wyników obliczeń.

dr inż. Paweł Lonkwic

Wyższa Szkoła Ekonomii i Innowacji w Lublinie
20-209 Lublin, ul. Projektowa 4
tel.: +48 81 749 17 77
www.wsei.lublin.pl

ISBN – wersja elektroniczna: 978-83-66159-90-7

WSEI LUBLIN
WYDAWNICTWO NAUKOWE
INNOVATIO PRESS

KOMPETENCJE WIEDZA INNOWACJE
Zintegrowany program rozwoju WSEI III ETAP
Fundusze Europejskie Program Regionalny
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny