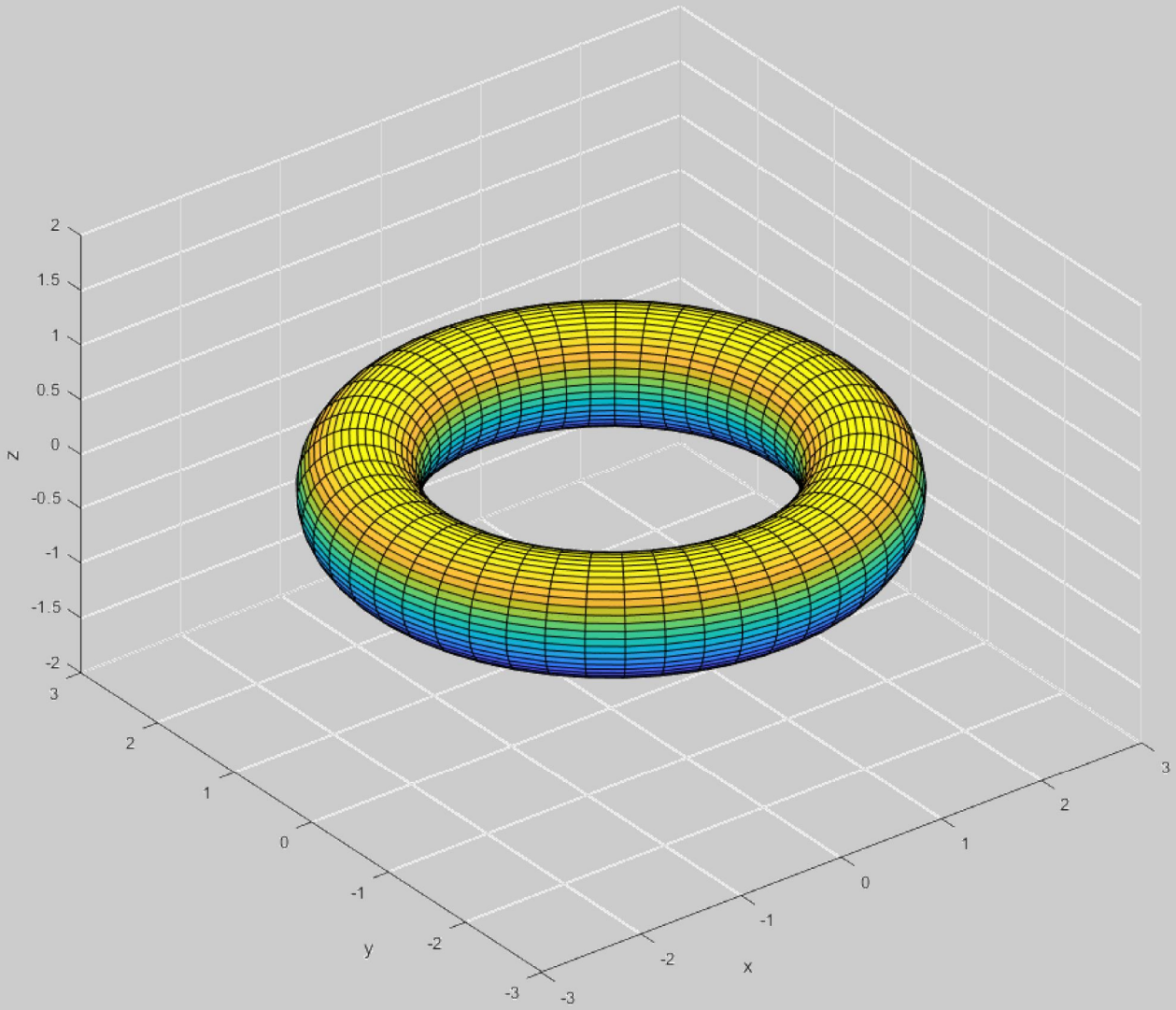
Daniel B. Nycz

MATLAB. Podstawy Użytkowania



Państwowa Wyższa Szkoła Zawodowa
im. Jana Grodka w Sanoku

Daniel B. Nycz

MATLAB
PODSTAWY UŻYTKOWANIA

skrypt dla studentów Instytutu Technicznego
Państwowej Wyższej Szkoły Zawodowej im. Jana Grodka w Sanoku

Recenzenci:
dr hab. Elżbieta Szymczyk, prof. WAT (Wojskowa Akademia Techniczna w Warszawie)
dr Rafał Reizer (Uniwersytet Rzeszowski)

Projekt okładki: Daniel Nycz
Skład i łamanie: Daniel Nycz

Skrypt do przedmiotów *Obliczeniowe Systemy Informatyczne* i *Komputerowe Systemy Pomiarów* prowadzonych na studiach pierwszego stopnia na kierunku Mechanika i Budowa Maszyn, dla studentów Instytutu Technicznego Państwowej Wyższej Szkoły Zawodowej im. Jana Grodka w Sanoku.

Wydanie I
Sanok 2018

Wydawca
Państwowa Wyższa Szkoła Zawodowa im. Jana Grodka w Sanoku

Produkcja: Agencja Wydawnicz PAJ-Press

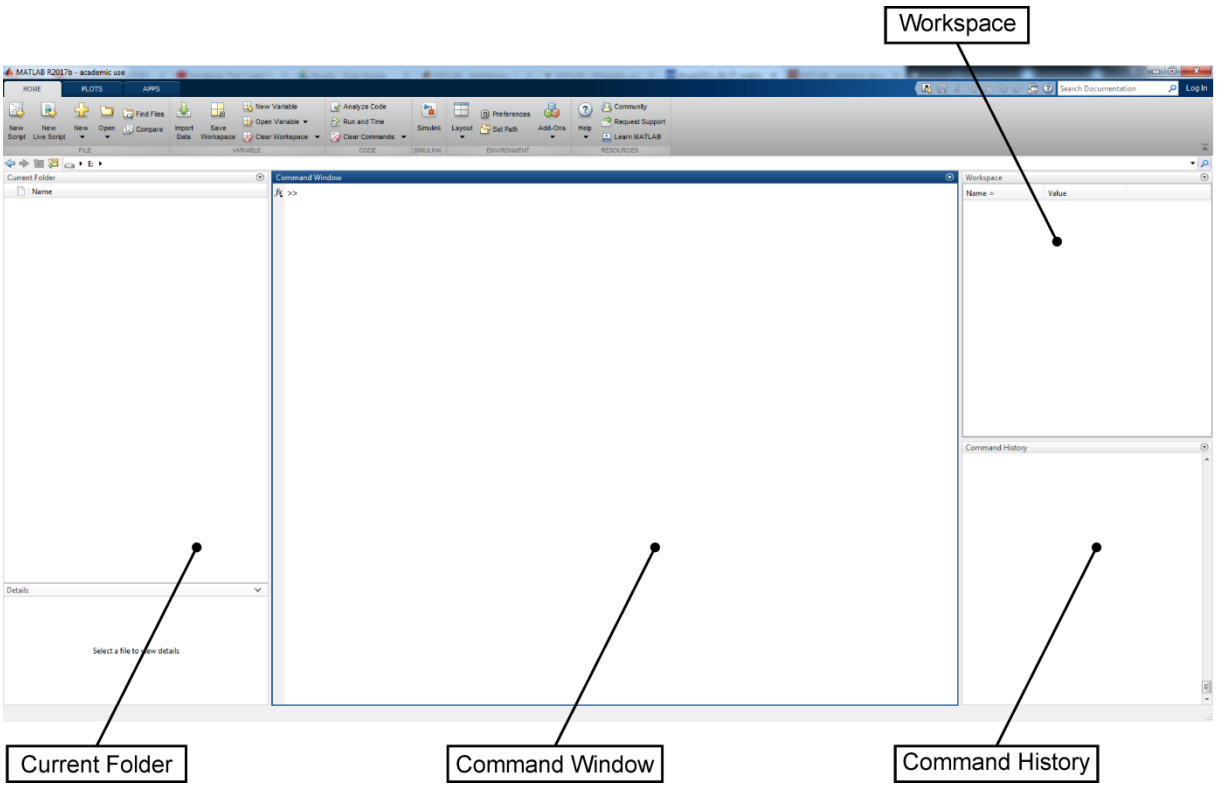
ISBN 978-83-61802-35-8
Nakład: 60

SPIS TREŚCI

1. Wstęp

Środowisko MATLAB jest jednym z najpopularniejszych oraz jednym z najbardziej rozbudowanych funkcjonalnych narzędzi do symulacji i obliczeń naukowych i inżynierskich [1]. Nazwa programu wywodzi się od jego początkowego przeznaczenia do obliczeń macierzowych – *MATrix LABoratory*. Producentem MATLAB jest firma The MathWorks Inc.

Na rys. 1.1 przedstawiono okno główne programu MATLAB (wersja MATLAB R2017b).



Rys. 1.1. Okno główne programu MATLAB (wersja MATLAB R2017b)

Jedną z najważniejszych części programu jest okno poleceń (*Command Window*). Za jego pomocą można bezpośrednio wprowadzać komendy i instrukcje oraz uzyskiwać wyniki obliczeń/poleceń.

Okno przestrzeni roboczej (*Workspace*) umożliwia podejrzenie zawartości wszystkich zmiennych wprowadzonych w bieżącej sesji programu.

Historia poleceń (*Command History*) zawiera zestawienie wprowadzonych przez użytkownika poleceń i instrukcji.

Okno katalogu bieżącego (*Current Folder*) zawiera podgląd zawartości tego folderu.

2. Okno poleceń (*Command Window*)

W oknie poleceń (*Command Window*) użytkownik środowiska MATLAB bezpośrednio komunikuje się z programem wprowadzając komendy/instrukcje, które program wykonuje po ich wprowadzeniu.

Komendy/instrukcje wprowadza się po znaku zachęty „>>”. Efekt operacji wykonanej przez program, domyślnie wyświetlany jest jako zmienna „`ans`” (rys. 2.1):

```
>> 12.54

ans =

   12.5400
```

Rys. 2.1. Przykład bezpośredniej pracy użytkownika w programie MATLAB

Definiując wartości liczbowe, separator dziesiętny wprowadza się w postaci kropki „`.`”.

Format wyświetlania liczb można zmienić za pomocą komend zestawionych w tabeli 2.1 (rys. 2.2 i 2.3).

Tabela 2.1. Formaty wyświetlania liczb w programie MATLAB

symbol	opis
`format short`	format dziesiętny krótki
`format long`	format dziesiętny długi
`format short e`	format naukowy krótki
`format long e`	format naukowy długi
`format rat`	format ułamkowy
`format compact`	pomijanie pustych linii przy wyświetlaniu polecenia i odpowiedzi
`format loose`	oddzielenie pustą linią polecenia i odpowiedzi

```
>> format short
>> 12.54

ans =

   12.5400

>> format long
>> 12.54

ans =

   12.539999999999999

>> format rat
>> 12.54

ans =

   627/50
```

```
>> format short e
>> 12.54

ans =

   1.2540e+01

>> format long e
>> 12.54

ans =

   1.254000000000000e+01
```

Rys. 2.2. Przykład formatów wyświetlania liczb w programie MATLAB

```
>> format compact
>> 12.54
ans =
   12.5400
```

```
>> format loose
>> 12.54

ans =

   12.5400
```

Rys. 2.3. Przykład formatów wyświetlania liczb w programie MATLAB (dla `format short`)

3. Podstawowe działania matematyczne

W tabeli 3.1 zestawiono podstawowe działania matematyczne programu MATLAB.

Tabela 3.1. Podstawowe działania matematyczne programu MATLAB

symbol	opis
^	potęgowanie
*	mnożenie
/	dzielenie
+	dodawanie
−	odejmowanie

MATLAB zachowuje zasady kolejności wykonywania działań. Najwyższy priorytet ma potęgowanie. Niższy priorytet mają jednocześnie mnożenie i dzielenie, które wykonywane są w kolejności w jakiej zostały wprowadzone (od lewej do prawej strony). Najniższy priorytet mają jednocześnie dodawanie i odejmowanie, według tej samej zasady co mnożenie i dzielenie.

Przykład 3.1 [5]

Bezpośrednio w oknie poleceń (*Command Window*) wykonać działania matematyczne (rys. 3.1):

a/ $2 + 2 \cdot 2 = 6$

b/ $\frac{3+6}{3} - 1 = 2$

c/ $5 + \frac{15-3}{2^2} = 8$

```
>> 2+2*2

ans =

     6
```

```
>> (3+6)/3-1

ans =

     2
```

```
>> 5+(15-3)/2^2

ans =

     8
```

Rys. 3.1. Realizacja przykładu 3.1 w oknie poleceń programu MATLAB

Dla zadania a/, MATLAB wykonał w pierwszej kolejności mnożenie, a później dodawanie (zachował kolejność wykonywania działań matematycznych).

Dla zadań b/ i c/, konieczne było wprowadzenie nawiasów, ponieważ przy ich braku MATLAB w pierwszej kolejności wykonałby dzielenie 6 przez 3 dla zadania b/ oraz potęgowanie, a następnie dzielenie 3 przez 2 dla zadania c/.

4. Relacje i operatory logiczne

Wyniki operacji logicznych prawda/fałsz są traktowane przez środowisko MATLAB jak wartości, odpowiednio 1 i 0. W tabeli 4.1 zestawiono operacje i operatory logiczne programu MATLAB.

Tabela 4.1. Operacje relacji i operatory logiczne programu MATLAB

operacje relacji		**operatory logiczne**	
symbol	**opis**	**symbol**	**opis**
= =	równy	~	negacja „not”
~ =	nierówny	&	koniunkcja „and”
<	mniejszy	|	alternatywa „or”
>	większy		
<=	nie większy		
>=	nie mniejszy		

Przykład 4.1 [5]

Bezpośrednio w oknie poleceń (*Command Window*) sprawdzić czy wartość $\frac{1}{2}\pi$ zawiera się w przedziale (0, 1) lub (2, 3) (rys. 4.1):

```
>> (0.5*pi>0) & (0.5*pi<1) | (0.5*pi>2) & (0.5*pi<3)

ans =

     0
```

Rys. 4.1. Realizacja przykładu 4.1 w oknie poleceń środowiska MATLAB

Wynikiem operacji jest „0”, co oznacza, że wartość $\frac{1}{2}\pi$ nie zawiera się w rozpatrywanych przedziałach.

5. Przypisywanie wartości do zmiennych

Przypisywanie wartości do zmiennych odbywa się za pomocą symbolu równości „=”. Raz przypisana wartość do zmiennej jest zapamiętywana przez program do momentu usunięcia/wyczyszczenia zmiennej/zmiennych z pamięci lub przypisania do zmiennej nowej wartości.

W jednym wierszu można wpisać kilka przypisań lub działań rozdzielonych znakami „ `,` ” lub „ `;` ”. Zastosowanie znaku „ `;` ” powoduje wykonanie polecenia (działania lub przypisania) i zapisanie wyniku w pamięci bez jego wyświetlenia. Zastosowanie znaku „ `,` ” powoduje wykonanie polecenia (działania lub przypisania) oraz zapisanie wyniku w pamięci i jego wyświetlenie.

Przykład 5.1

Bezpośrednio w oknie poleceń (*Command Window*) wpisać w jednym wierszu (rys. 5.1):

`>>x=5; x*3, ans*2/3` (enter)

w oknie poleceń wyświetlone zostanie:

```
>> x=5; x*3, ans*2/3

ans =

   15

ans =

   10
```

Rys. 5.1. Realizacja przykładu 5.1 w oknie poleceń programu MATLAB

Zastosowanie średnika powoduje, że w oknie poleceń nie zostaje wyświetlony wynik przypisania `x=5`. Zostaje on zapisany w pamięci tymczasowej programu. Wyświetlone są tylko wyniki operacji `x*3` i `ans*2/3`.

Usuwanie zmiennych z pamięci realizuje się za pomocą poleceń:

`clear` → usuwa wszystkie zmienne z pamięci i przywraca zmienne predefiniowane

`clear` *`nazwa`* → usuwa zmienną *`nazwa`*

UWAGA: MATLAB rozróżnia małe i duże litery.

Przykład 5.2 [5]

Zdefiniować zmienną A według zależności $(2\pi + 2)^2$. Sprawdzić, czy $A \in (75, 100)$. Wyświetlić wartość A (rys. 5.2).

```
>> A=(2*pi+2)^2;
>> (A>75) & (A<100)

ans =

   0

>> A

A =

   68.6112
```

Rys. 5.2. Realizacja przykładu 5.2 w oknie poleceń programu MATLAB

Po przypisaniu zależności $(2\pi + 2)^2$ do zmiennej A nie następuje wyświetlenie jej wartości w oknie poleceń ze względu na zastosowanie średnika. Wynikiem sprawdzenia relacji $A \in (75, 100)$ jest wynik „0”, co oznacza, że wartość zmiennej A nie należy do rozpatrywanego przedziału. Wpisanie zmiennej A powoduje wyświetlenie jej wartości w oknie poleceń.

Przykład 5.3

Bezpośrednio w oknie poleceń (*Command Window*) wyświetlić wartość zmiennej predefiniowanej π. Następnie, do zmiennej „pi” przypisać wartość 9. Usunąć wszystkie zmienne z pamięci, po czym, wyświetlić wartość zmiennej „pi” (rys. 5.3).

```
>> pi

ans =

   3.1416

>> pi=9;
>> pi

pi =

   9

>> clear all
>> pi

ans =

   3.1416
```

Rys. 5.3. Realizacja przykładu 5.3 w oknie polcecń programu MATLAB

Wpisanie zmiennej „pi” w oknie poleceń powoduje wyświetlenie jej wartości. Jest to zmienna predefiniowana. Po przypisaniu do zmiennej „pi” wartości 9, następuje nadpisanie wartości tej zmiennej. Zastosowanie polecenia `clear all` powoduje wyczyszczenie wszystkich zmiennych zapisanych przez użytkownika z pamięci MATLAB. Zmienna „pi” jest ponownie zmienną predefiniowaną.

6. Funkcje

W tabeli 6.1 zestawiono wybrane funkcje matematyczne programu MATLAB.

Tabela 6.1. Wybrane funkcje matematyczne programu MATLAB (*X* – argument funkcji)

symbol	opis	symbol	opis
`sin(X)`	sinus	`log10(X)`	logarytm dziesiętny
`cos(X)`	cosinus	`sqrt(X)`	pierwiastek kwadratowy
`tan(X)`	tangens	`abs(X)`	moduł
`cot(X)`	cotangens	`round(X)`	najbliższa wartość całkowita „zaokrąglenie w górę”
`exp(X)`	eksponens	`floor(X)`	najbliższa wartość całkowita w kierunku ujemnej nieskończoności „zaokrąglenie w dół”
`log(X)`	logarytm naturalny		

Przykład 6.1 [5]

Bezpośrednio w oknie poleceń (*Command Window*) (rys. 6.1):

a/ sprawdzić, czy wartość liczby *e* (liczba Eulera, podstawa logarytmu naturalnego) znajduje się poza przedziałem (2, 3)

b/ policzyć $3\sin^3(45°) + 5\cos^2(30°)$

```
>> ~((exp(1)>2) & (exp(1)<3))

ans =

   0

>> 3*sin(45*pi/180)^3+5*cos(30*pi/180)^2

ans =

   4.8107
```

Rys. 6.1. Realizacja przykładu 6.1 w oknie poleceń programu MATLAB

Wartość liczby *e* jest definiowana w MATLAB, jako funkcja eksponens z wartości 1. Sprawdzenie, czy wartość liczby *e* znajduje się w rozpatrywanym przedziale jest realizowane z wykorzystaniem operatorów logicznych „`and`” i „`not`”. Wynikiem działania jest „`0`”, co oznacza, że wartość liczby *e* znajduje się w przedziale ($e \cong 2.72$).

W przypadku wielkości kątowych MATLAB domyślnie wykonuje obliczenia na radianach, dlatego dla funkcji sinus i cosinus wprowadzono:

$$45° = \frac{45 \cdot \pi}{180} \quad 30° = \frac{30 \cdot \pi}{180} \tag{6.1}$$

Istnieje możliwość wykonywania obliczeń funkcji trygonometrycznych, dla których argument wyrażony jest w stopniach (funkcje `sind`, `cosd`, `tand` i `cotd`).

7. Wektory

Wektor, czyli skończony ciąg liczb zapisany w postaci wiersza lub kolumny [5], może być definiowany w środowisku MATLAB w sposób bezpośredni lub automatyczny.

W metodzie bezpośredniej, definiuje się kolejne elementy wektora w nawiasie kwadratowym, stosując pomiędzy wartościami spacje, przecinki lub średniki (rys. 7.1). W wyniku zastosowania spacji lub przecinka, uzyskuje się wektor w postaci wiersza:

```
>> [2 5 8]

ans =

   2   5   8

>> [2,5,8]

ans =

   2   5   8
```

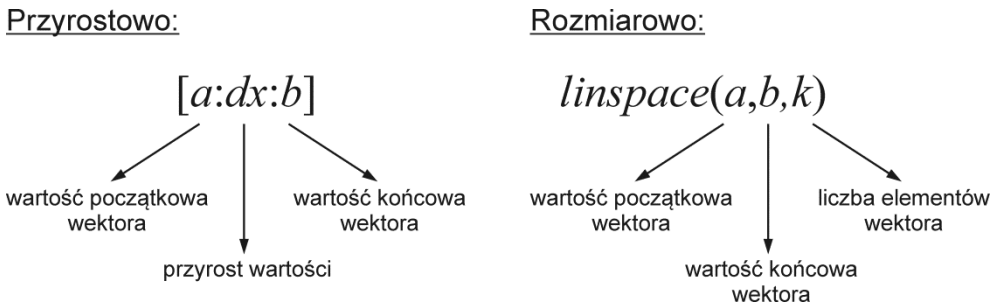
Rys. 7.1. Przykład bezpośredniej definicji wektora w postaci wiersza

W przypadku zastosowania średnika pomiędzy wartościami, uzyskuje się wektor w postaci kolumny (rys. 7.2):

```
>> [2;5;8]

ans =

   2
   5
   8
```

Rys. 7.2. Przykład bezpośredniej definicji wektora w postaci kolumny

Stosując definicję automatyczną można utworzyć wektor metodą przyrostową lub rozmiarową (rys. 7.3).



Rys. 7.3. Automatyczne definiowanie wektorów

Przykład 7.1

Bezpośrednio w oknie poleceń (*Command Window*) zdefiniować przyrostowo i rozmiarowo wektor o wartościach od -6 do 4 o elementach różniących się o 2 (6 elementów) (rys. 7.4).

```
>> [-6:2:4]

ans =

   -6   -4   -2    0    2    4

>> linspace(-6,4,6)

ans =

   -6   -4   -2    0    2    4
```

Rys. 7.4. Realizacja przykładu 7.1 w oknie poleceń programu MATLAB

Wektor zdefiniowany w postaci wiersza można zamienić na wektor w postaci kolumny i odwrotnie, za pomocą operacji matematycznej transponowania, która w środowisku MATLAB jest definiowana za pomocą apostrofa „ ' „.

Przykład 7.2

Bezpośrednio w oknie poleceń (*Command Window*) zdefiniować ciąg *A* w postaci wiersza o składowych od 0 do 2 z przyrostem 0.5, a następnie transponować wektor do postaci kolumnowej (rys. 7.2).

```
>> A=[0:0.5:2]

A =

   0   0.5000   1.0000   1.5000   2.0000

>> A'

ans =

         0
    0.5000
    1.0000
    1.5000
    2.0000
```

Rys. 7.5. Realizacja przykładu 7.2 w oknie poleceń programu MATLAB

8. Operacje arytmetyczne na składowych wektorów

W tabeli 8.1 zestawiono podstawowe operacje arytmetyczne na składowych wektorów (ciągów liczb) w środowisku MATLAB.

Tabela 8.1. Podstawowe operacje matematyczne na składowych wektorów (ciągów liczb) w programie MATLAB

symbol	opis
.^	potęgowanie
.*	mnożenie
./	dzielenie
+	dodawanie
-	odejmowanie

Przykład 8.1

Bezpośrednio w oknie poleceń (*Command Window*) (rys. 8.1):

a/ zdefiniować wektory *X*=[*3 1 5*] i *Y*=[*2 7 3*]

b/ policzyć wektor o składowych równych iloczynom składowych wektorów *X* i *Y*

```
>> X=[3 1 5]; Y=[2 7 3];
>> X*Y
Error using  *
Inner matrix dimensions must agree.
```

Rys. 8.1. Realizacja przykładu 8.1 w oknie poleceń programua MATLAB – błędny zapis operacji mnożenia

Zapis „`X*Y`” oznacza mnożenie macierzowe, które jest możliwe do zrealizowania, jeżeli liczba kolumn pierwszej macierzy jest równa liczbie wierszy drugiej macierzy. W tym zadaniu rozpatrywane są wektory, które są szczególnym przypadkiem macierzy. Wektory *X* i *Y* są macierzami o 3 kolumnach i 1 wierszu. Liczba kolumn macierzy *X* nie odpowiada liczbie wierszy macierzy *Y*. MATLAB nie mógł wykonać obliczeń.

Przykład 8.2 [5]

Bezpośrednio w oknie poleceń (*Command Window*) (rys. 8.2):

a/ zdefiniować wektory *X*=[*3 1 5*] i *Y*=[*2 7 3*]

b/ policzyć sumę i różnicę wektorów *X* i *Y*

c/ policzyć wektor o składowych równych iloczynom i ilorazom składowych wektorów *X* i *Y*

d/ policzyć wektor o składowych równych kwadratom składowych wektora *X*

```
>> A=[3 1 5]; Y=[2 7 3];
>> X+Y

ans =

   5   8   8

>> X-Y

ans =

   1   -6   2

>> X.*Y

ans =

   6   7   15

>> X./Y

ans =

   1.5000   0.1429   1.6667

>> X.^2

ans =

   9   1   25
```

Rys. 8.2. Realizacja przykładu 8.2 w oknie poleceń programu MATLAB

Aby zrealizować działania c/ i d/ z przykładu 8.2 należy zastosować operacje na składowych wektorów *X* i *Y*. Niezastosowanie tych operacji spowoduje wyświetlenie błędu, jak w przykładzie 8.1.

9. Funkcje wektorowe

W tabeli 9.1 zestawiono wybrane funkcje wektorowe programu MATLAB.

Tabela 9.1. Wybrane funkcje wektorowe programu MATLAB (*X* – dowolny wektor)

symbol	opis
`X'`	transponowanie
`length(X)`	rozmiar/ilość elementów wektora
`sum(X)`	suma elementów wektora
`max(X)`	maksymalna składowa wektora
`min(X)`	minimalna składowa wektora
`cross(X,Y)`	iloczyn wektorowy
`roots(X)`	pierwiastki wielomianu o współczynnikach *X*

Przykład 9.1 [5]

Bezpośrednio w oknie poleceń (*Command Window*) (rys. 9.1):

a/ zdefiniować wektory *X*=[*3 1 5*] i *Y*=[*2 7 3*]

b/ policzyć iloczyn wektorowy *W* wektorów *X* i *Y*

c/ policzyć iloczyn skalarny *S* wektorów *X* i *Y*

```
>> X=[3 1 5]; Y=[2 7 3];
>> W=cross(X,Y)

W =

   -32    1   19

>> S=sum(X.*Y)

S =

   28
```

Rys. 9.1. Realizacja przykładu 9.1 w oknie poleceń programu MATLAB

Iloczyn wektorowy obliczany jest za pomocą funkcji `cross`. Iloczyn skalarny dwóch wektorów o trzech składowych jest obliczany analitycznie, jako:

$$\begin{aligned} X &= [x_1 \quad x_2 \quad x_3] \\ Y &= [y_1 \quad y_2 \quad y_3] \\ X \cdot Y &= x_1 y_1 + x_2 y_2 + x_3 y_3 \end{aligned} \tag{9.1}$$

Iloczyn skalarny został zrealizowany jako suma elementów nowego wektora, będącego iloczynem składowych wektorów *X* i *Y*. Suma została obliczona za pomocą funkcji `sum`.

10. Macierze

Macierze są ciągami liczb zapisanymi w prostokątnej tablicy [5].

W środowisku MATLAB, macierze definiuje się w nawiasach kwadratowych poprzez bezpośrednie podanie ich składowych, przy czym składowe w wierszach oddziela cię spacjami lub przecinkami, a wiersze oddziela się średnikiem (rys. 10.1):

```
>> M=[5,3,4;6 2 8]

M =

   5   3   4
   6   2   8
```

Rys. 10.1. Przykład bezpośredniej definicji macierzy

Jeżeli jest to możliwe, poszczególne elementy w wierszach można definiować przyrostowo lub rozmiarowo jak w przypadku wektorów (rys. 10.2):

```
>> M=[5:1:7;3:1:5]

M =

   5   6   7
   3   4   5

>> M=[linspace(5,7,3);linspace(3,5,3)]

M =

   5   6   7
   3   4   5
```

Rys. 10.2. Przykład automatycznej definicji macierzy

W tabeli 10.1 zestawiono funkcje służące do automatycznego generowania wybranych macierzy, gdzie *m* odpowiada liczbie wierszy, a *n* liczbie kolumn.

Tabela 10.1. Wybrane macierze automatyczne programu MATLAB (*m* – liczba wierszy, *n* – liczba kolumn)

symbol	opis
`zeros(m,n)`	macierz zerowa
`ones(m,n)`	macierz jedynkowa
`rand(m,n)`	macierz losowa
`eye(m,n)`	macierz jednostkowa

W tabeli 10.2 zestawiono wybrane funkcje/operacje macierzowe programu MATLAB.

Tabela 10.2. Wybrane funkcje/operacje macierzowe programu MATLAB (*M*, *N* – dowolne macierze)

symbol	opis
`M'`	transponowanie macierzy
`inv(M)*N`	lewostronne dzielenie macierzy *N* przez macierz *M*
`N*inv(M)`	prawostronne dzielenie macierzy *N* przez macierz *M*
`size(M)`	rozmiar macierzy *M*
`det(M)`	wyznacznik macierzy *M*
`inv(M)`	macierz odwrotna macierzy *M*
`eig(M)`	wartości własne macierzy *M*
`[V,D]=eig(M)`	macierz prawych wektorów własnych V oraz macierz diagonalna D z wartościami własnymi na przekątnej

Przykład 10.1 [5]

Bezpośrednio w oknie poleceń (*Command Window*) (rys. 10.3):

a/ zdefiniować wektory *X*=[*3 1 5*] i *Y*=[*2 7 3*]

b/ policzyć iloczyn skalarny *S* wektorów *X* i *Y*

UWAGA: Wektor jest szczególnym przypadkiem macierzy (macierz jednokolumnowa lub jednowierszowa).

```
>> X=[3 1 5]; Y=[2 7 3];
>> S=X*Y'

S =

    28
```

Rys. 10.3. Realizacja przykładu 10.1 w oknie poleceń programu MATLAB

Przykład 10.2

Bezpośrednio w oknie poleceń (*Command Window*) rozwiązać układ równań (rys. 10.4):

$$\begin{cases} 5x + 3y - z = 4 \\ -3x + 4y = 7 \\ 2x - 5y + 3z = -4 \end{cases}$$

Powyższy układ równań można zapisać w postaci macierzowej:

$$A \cdot C = B \tag{10.1}$$

gdzie *A* jest macierzą współczynników:

$$A = \begin{bmatrix} 5 & 3 & -1 \\ -3 & 4 & 0 \\ 2 & -5 & 3 \end{bmatrix} \tag{10.2}$$

B jest wektorem prawych stron układu równań:

$$B = [4 \quad 7 \quad -4] \tag{10.3}$$

C wektorem niewiadomych:

$$C = [x \quad y \quad z] \tag{10.4}$$

Mnożąc lewostronnie obie strony równania macierzowego przez macierz odwrotną macierzy A uzyskuje się:

$$\begin{gathered} A \cdot C = B \quad / \cdot A^{-1} \\ A^{-1} \cdot A \cdot C = A^{-1} \cdot B \\ I \cdot C = A^{-1} \cdot B \end{gathered} \tag{10.5}$$

gdzie I jest macierzą jednostkową. Ostatecznie:

$$C = A^{-1} \cdot B \tag{10.6}$$

```
>> A=[5 3 -1; -3 4 0; 2 -5 3];
>> B=[4 7 -4];
>> C=inv(A)*B'

C =

   0.0500
   1.7875
   1.6125
```

Rys. 10.4. Realizacja przykładu 10.1 w oknie poleceń programu MATLAB

11. Dostęp do elementów wektorów/macierzy

Dostęp do elementów wektorów/macierzy w środowisku MATLAB jest realizowany za pomocą polecenia `nazwa(indeks elementu)`.

Jeżeli odwołanie odnosi się do elementu/elementów wektora, to np. $X(i)$ jest odwołaniem do i-tego elementu wektora X.

Jeżeli odwołanie odnosi się do elementu/elementów macierzy, to np. $X(i,j)$ jest odwołaniem do elementu na przecięciu i-tego wiersza i j-tej kolumny macierzy X. Odwołanie $X(i)$ w przypadku macierzy oznacza odwołanie do i-tego elementu wektora utworzonego z kolejnych kolumn macierzy X.

Przykład 11.1

Bezpośrednio w oknie poleceń (*Command Window*) utworzyć macierz (rys. 11.1 i 11.2):

$$X = \begin{bmatrix} 9 & -8 & 7 \\ -5 & 2 & 8 \\ 11 & 0 & 13 \end{bmatrix}$$

a/ odwołać się do elementów o wartościach -5 i 7 macierzy X

b/ odwołać się do wszystkich elementów drugiego wiersza macierzy X

```
>> X=[9 -8 7;-5 2 8;11 0 13]

X =

   9  -8   7
  -5   2   8
  11   0  13

>> X(2,1)

ans =

  -5

>> X(2)

ans =

  -5

>> X(1,3)

ans =

   7

>> X(7)

ans =

   7
```

Rys. 11.1. Realizacja przykładu 11.1 w oknie poleceń programu MATLAB; część a

```
>> X(2,[1 2 3])

ans =

  -5   2   8

>> X(2,1:3)

ans =

  -5   2   8
```

Rys. 11.2. Realizacja przykładu 11.1 w oknie poleceń programu MATLAB; część b

Przykład 11.2

Bezpośrednio w oknie poleceń (*Command Window*) utworzyć macierze (rys. 11.3 i 11.4):

$$A = \begin{bmatrix} 1 & 9 & 4 \\ 15 & 4 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 4 \\ 12 & 6 \end{bmatrix} \quad C = \begin{bmatrix} 4 & 3 & 6 & 6 & 7 \\ 2 & 8 & 5 & 9 & 1 \end{bmatrix}$$

a/ utworzyć macierz D z macierzy A, B i C:

$$D = \begin{bmatrix} A & B \\ C & \end{bmatrix}$$

b/ podstawić wartość „0” do pozycji w wierszu 3 i kolumnie 2 oraz wierszu 3 i kolumnie 4 macierzy D

c/ do istniejącej macierzy B dodać trzecią kolumnę z wartościami 11 i 5

$$B = \begin{bmatrix} 4 & 4 & \mathbf{11} \\ 12 & 6 & \mathbf{5} \end{bmatrix}$$

```
>> A=[1 9 4;15 4 2]; B=[4 4;12 6]; C=[4 3 6 6 7;2 8 5 9 1];
>> D=[A B;C];

D =

   1   9   4   4   4
  15   4   2  12   6
   4   3   6   6   7
   2   8   5   9   1

>> D(3,2)=0

D =

   1   9   4   4   4
  15   4   2  12   6
   4   0   6   6   7
   2   8   5   9   1
```

Rys. 11.3. Realizacja przykładu 11.2 w oknie poleceń programu MATLAB; część a

```
>> D(3,4)=0

D =

    1    9    4    4    4
   15    4    2   12    6
    4    0    6    0    7
    2    8    5    9    1

>> B(1,3)=11

B =

    4    4   11
   12    6    0

>> B(2,3)=5

B =

    4    4   11
   12    6    5
```

Rys. 11.4. Realizacja przykładu 11.2 w oknie poleceń programu MATLAB; część b i c

12. M-pliki

Praca w środowisku MATLAB przy wykorzystaniu tylko okna komend (*Command Window*), czyli bezpośredniej komunikacji ze środowiskiem, jest w wielu przypadkach niewystarczająca.

MATLAB oferuje tworzenie własnych plików z poleceniami za pomocą tzw. M-plików. Są to pliki tekstowe z rozszerzeniem *.m. Pliki te można tworzyć za pomocą dowolnego edytora tekstowego lub za pomocą programu MATLAB [5].

W tabeli 12.1 zestawiono funkcje przydatne podczas pracy z M-plikami.

Tabela 12.1. Wybrane funkcje przydatne podczas pracy z M-plikami (*x* – dowolna zmienna)

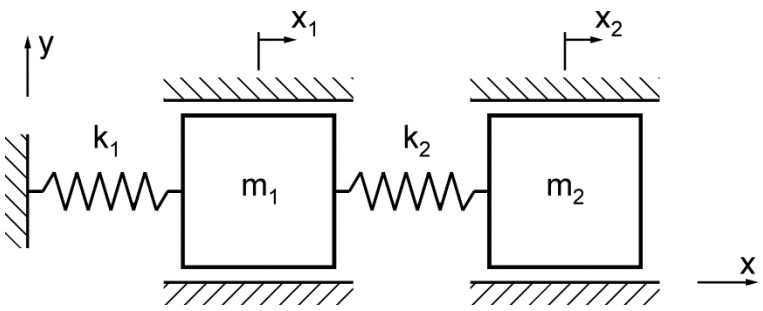
symbol	opis
`clear`	usuwa wszystkie zmienne z pamięci i przywraca zmienne predefiniowane
`clc`	czyszczenie okna komend
`close`	zamknięcie bieżącego okna graficznego
`x=input('tekst')`	wczytanie zmiennej *x*
`disp(x)`	wyświetlenie zmiennej *x*
`tekst=num2str(x)`	zamiana liczby *x* na tekst
`strcat(t1,…,tn)`	łączenie tekstów t1 do tn

Aby uruchomić zapisany M-plik, należy:
- podać lokalizację katalogu, w którym ten plik się znajduje, np. ustawić ten katalog jako katalog bieżący środowiska MATLAB,
- wpisać rdzeń nazwy M-pliku (bez rozszerzenia) bezpośrednio w oknie poleceń lub w innym M-pliku, albo
- wybrać opcję uruchom (*run*) z poziomu edytora M-pliku.

UWAGA: Tekst poprzedzone symbolem „`%`” w danym wierszu M-pliku jest traktowany, jako komentarz.

Przykład 12.1

Wyznaczyć częstotliwości i postacie drgań własnych układu o dwóch stopniach swobody (rys. 12.1 i 12.4)).



Rys. 12.1. Układ o dwóch stopniach swobody

Równanie drgań swobodnych układu w postaci macierzowej przyjmuje postać

$$M\ddot{q} + Kq = 0 \tag{12.1}$$

gdzie:

M – macierz bezwładności

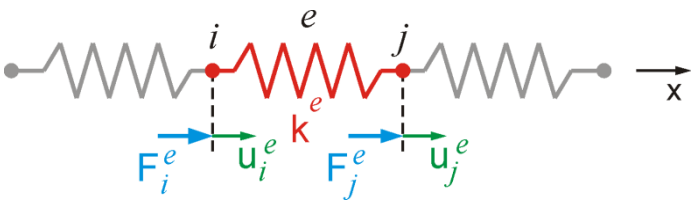
K – macierz sztywności

q – wektor współrzędnych uogólnionych.

Macierz bezwładności ma postać diagonalną i dla układu z rys. 12.1, przyjmuje postać

$$M = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \tag{12.2}$$

Macierz sztywności można wyznaczyć rozpatrując równowagę układu sprężyn połączonych szeregowo [7].



Rys. 12.2. Pojedyncza sprężyna wyodrębniona w układzie szeregowym sprężyn [7]

Dla pojedynczej sprężyny (rys. 12.2), związek siły i przemieszczenia przyjmuje postać

$$F^e = k^e \cdot \Delta u^e \tag{12.3}$$

$$\Delta u^e = u_j^e - u_i^e \tag{12.4}$$

Korzystając z warunku równowagi

$$F_i^e + F_j^e = 0 \quad \Rightarrow \quad F_j^e = -F_i^e = F^e \tag{12.5}$$

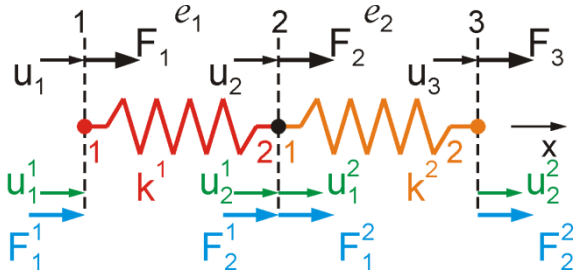
można dla każdego z węzłów napisać

$$F_i^e = -F^e = -k^e\left(u_j^e - u_i^e\right) = k^e \cdot u_i^e - k^e \cdot u_j^e \tag{12.6}$$

$$F_j^e = F^e = k^e\left(u_j^e - u_i^e\right) = -k^e \cdot u_i^e + k^e \cdot u_j^e \tag{12.7}$$

W postaci macierzowej równania (12.6) i (12.7) przyjmują postać

$$\begin{bmatrix} k^e & -k^e \\ -k^e & k^e \end{bmatrix} \begin{bmatrix} u_i^e \\ u_j^e \end{bmatrix} = \begin{bmatrix} F_i^e \\ F_j^e \end{bmatrix} \tag{12.8}$$



Rys. 12.3. Układ dwóch sprężyn połączonych szeregowo [7]

Dla układu dwóch sprężyn połączonych szeregowo, równania równowagi dla każdej sprężyny przyjmują postać

$$\begin{bmatrix} k^1 & -k^1 \\ -k^1 & k^1 \end{bmatrix} \begin{bmatrix} u_1^1 \\ u_2^1 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 \end{bmatrix} \tag{12.9}$$

$$\begin{bmatrix} k^2 & -k^2 \\ -k^2 & k^2 \end{bmatrix} \begin{bmatrix} u_1^2 \\ u_2^2 \end{bmatrix} = \begin{bmatrix} F_1^2 \\ F_2^2 \end{bmatrix} \tag{12.10}$$

Wprowadzając globalne oznaczenia

$$u_1 = u_1^1 \quad u_2 = u_2^1 = u_1^2 \quad u_3 = u_2^2 \tag{12.11}$$

$$F_1 = F_1^1 \quad F_2 = F_2^1 + F_1^2 \quad F_3 = F_2^2 \tag{12.12}$$

układy równań przyjmują postać

$$\begin{bmatrix} k^1 & -k^1 & 0 \\ -k^1 & k^1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 \\ 0 \end{bmatrix} \tag{12.13}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & k^2 & -k^2 \\ 0 & -k^2 & k^2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ F_1^2 \\ F_2^2 \end{bmatrix} \tag{12.14}$$

Łącząc (scalając) równania (12.13) i (12.14), otrzymuje się

$$\begin{bmatrix} k^1 & -k^1 & 0 \\ -k^1 & k^1 + k^2 & -k^2 \\ 0 & -k^2 & k^2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} F_1 = F_1^1 \\ F_2 = F_2^1 + F_1^2 \\ F_3 = F_2^2 \end{bmatrix} \tag{12.15}$$

Równanie (12.15) można zapisać

$$[K][u] = [F] \tag{12.16}$$

gdzie [K] jest macierzą sztywności układu dwóch sprężyn połączonych szeregowo (rys. 12.3).

Uwzględnienie warunków brzegowych (utwierdzeń) układu prowadzi do poniższej postaci macierzy sztywności

$$\begin{bmatrix} k^1 + k^2 & -k^2 \\ -k^2 & k^2 \end{bmatrix} \tag{12.17}$$

Rozwiązanie równania (12.1) przyjmuje się w postaci

$$q = q_0 \sin(\omega t) \tag{12.18}$$

Po podstawieniu równania (12.18) i jego drugiej pochodnej do równania (12.1) otrzymuje się

$$(K - \lambda M) q_0 = 0 \tag{12.19}$$

gdzie $\lambda = \omega^2$.

Równanie (12.19) jest liniowym problemem własnym. Wartości λ_i nazywane są wartościami własnymi, a odpowiadające im wektory q_i – wektorami własnymi układu (12.19). Wartości ω_i oznaczają częstości kołowe, a wektory q_i – postaci drgań układu opisanego równaniem (12.1).

Równanie (12.19) można przedstawić w innej postaci, rozkładając macierz *M* na macierze trójkątne [9]

$$M = L^T \cdot L \tag{12.20}$$

gdzie *L* jest górną macierzą trójkątną.

Podstawiając [9]

$$x_0 = L \cdot q_0 \tag{12.21}$$

oraz

$$A = L^{-T} \cdot K \cdot L^{-1} \tag{12.22}$$

równanie (12.19) można zapisać w postaci [9]

$$(A - I\lambda) x_0 = 0 \tag{12.23}$$

gdzie$I$ jest macierzą jednostkową.

Dodatkowo, gdy macierz M jest macierzą diagonalną, to [9]

$$L = M^{\frac{1}{2}} \tag{12.24}$$

$$A = M^{-\frac{1}{2}} \cdot K \cdot M^{-\frac{1}{2}} \tag{12.25}$$

Częstotliwość drgań określona jest zależnością:

$$f_i = \frac{\omega_i}{2\pi} \tag{12.26}$$

```
clear; clc; close all;

m1=input('Podaj masę m1 = ');
m2=input('Podaj masę m2 = ');
k1=input('Podaj sztywność k1 = ');
k2=input('Podaj sztywność k2 = ');

M=[m1 0;0 m2]
K=[k1+k2 -k2;-k2 k2]

B=sqrt(M);
D=inv(B)*K*inv(B);

[x,lambda]=eig(D)
f=(sqrt(lambda))/(2*pi)
q=inv(B)*x
```

Rys. 12.4. Realizacja przykładu 12.1 – treść M-pliku

Przykładowo, dla m_1=0.02 tony i m_2=0.01 tony oraz k_1=100 N/mm i k_2=200 N/mm, otrzymuje się

$$f_1 = 8.9164Hz \quad q = \begin{bmatrix} -5.4177 \\ -6.4262 \end{bmatrix}$$

$$f_2 = 28.4088Hz \quad q = \begin{bmatrix} -4.5440 \\ 7.6618 \end{bmatrix}$$

13. Wykresy (grafika 2D i 3D)

Do tworzenia wykresów 2D w środowisku MATLAB służy wiele funkcji, m.in. `plot`, `bar`, `bar3`, `stairs`, `stem` i inne. Argumentami tych funkcji są wektory danych.

Przykład 13.1

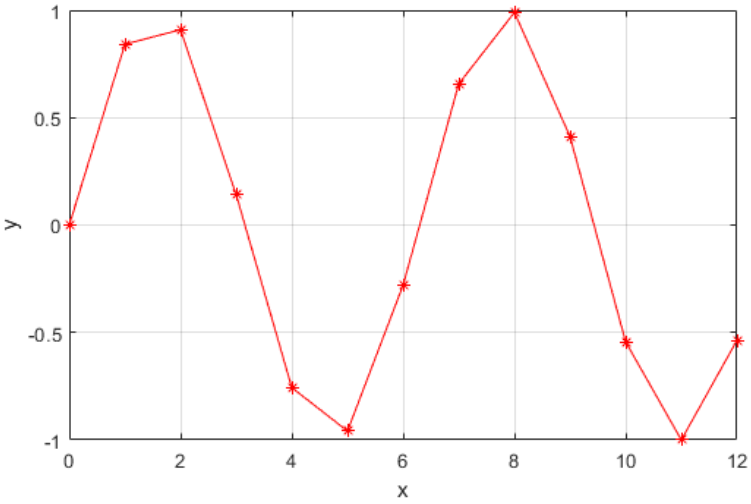
Narysować wykres funkcji *sin*(*x*) w przedziale (0, 4π) (rys. 13.1 i 13.2):

a)

```
1  clear; clc; close all;
2  x=[0:1:4*pi];
3  y=sin(x);
4
5  plot(x,y,'R*-')
6  grid on
7  xlabel('x')
8  ylabel('y')
```

wektor od 0 do 4π
z przyrostem wartości co 1;
13 elementów

b)

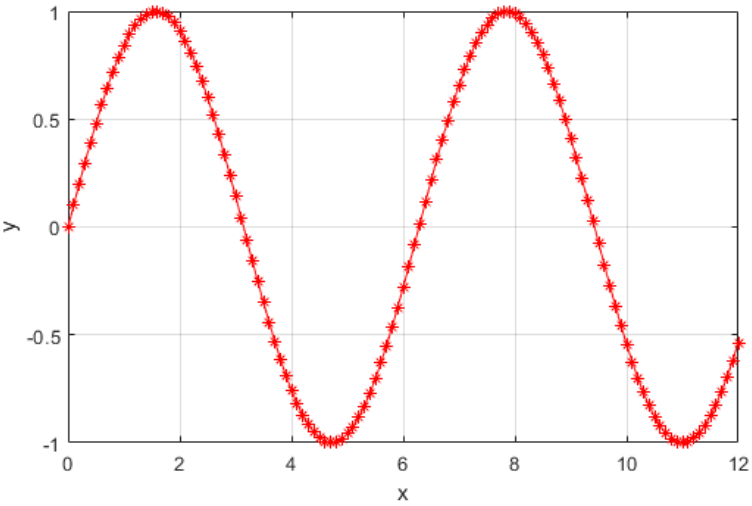


c)

```
1  clear; clc; close all;
2  x=[0:0.1:4*pi];
3  y=sin(x);
4
5  plot(x,y,'R*-')
6  grid on
7  xlabel('x')
8  ylabel('y')
```

wektor od 0 do 4π
z przyrostem wartości co 0.1;
126 elementów

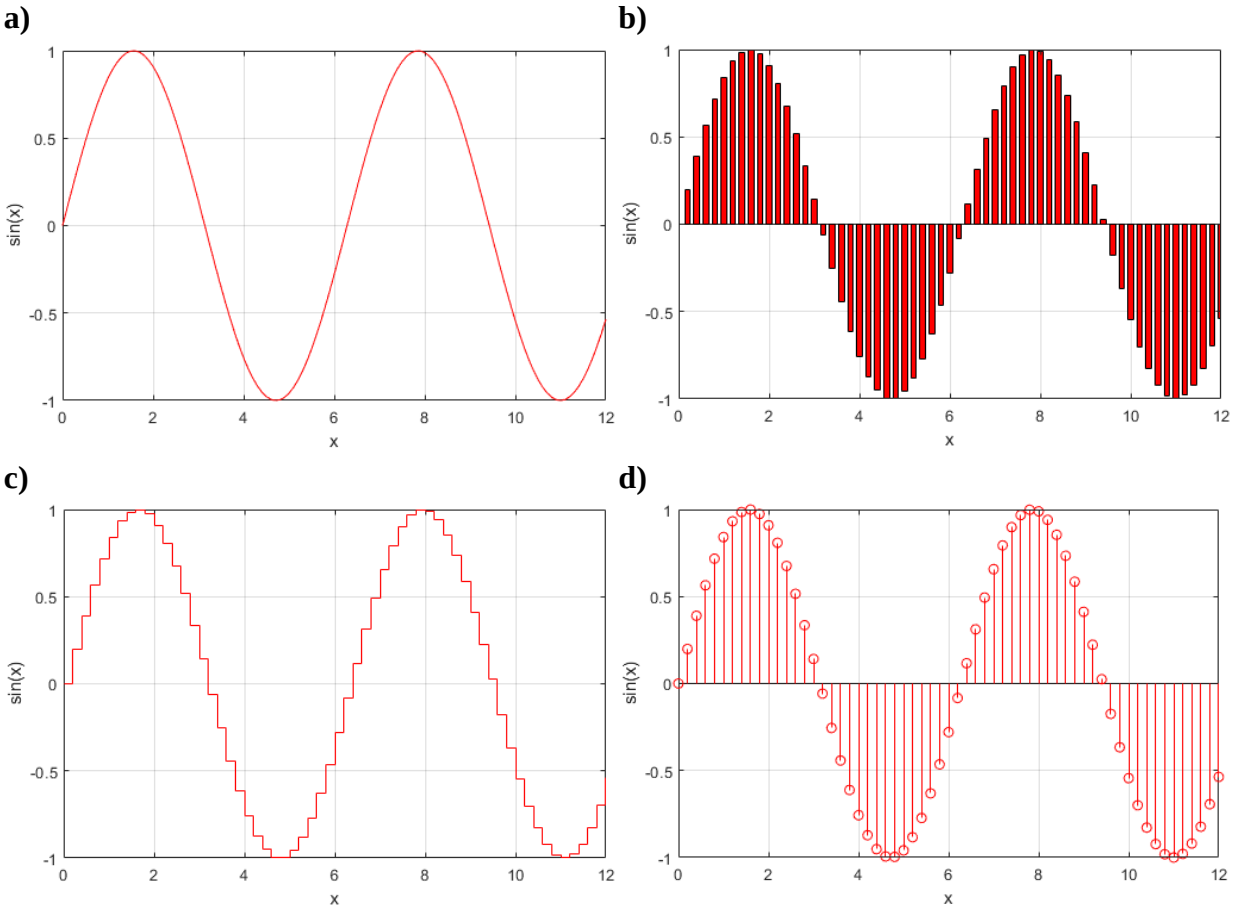
d)



Rys. 13.1. Realizacja przykładu 13.1: a,b) dla argumentu funkcji *sin*(*x*) w przedziale (0, 4π) z przyrostem wartości co 1 (13 elementów); c,d) dla argumentu funkcji *sin*(*x*) w przedziale (0, 4π) z przyrostem wartości co 0.1 (126 elementów)

W pierwszym przypadku wektor *x* (wektor argumentów funkcji sinus) składa się z 13 elementów. Otrzymany wykres funkcji *sin*(*x*) jest niedokładny. Zwiększenie liczby elementów wektora *x* (126 elementów) powoduje udokładnienie otrzymanego wykresu funkcji.

Na rys. 13.2 przedstawiono wykresy funkcji *sin*(*x*) z przykładu 13.1 wygenerowane za pomocą różnych funkcji (`plot, bar, , stairs, stem`).



Rys. 13.2. Przykłady wykresów realizowanych za pomocą: a) `plot`; b) `bar`; c) `stairs`; d) `stem`

Styl wykresów można formatować definiując kolor i rodzaj linii oraz typ znacznika punktowego (tabela 13.1), np. `plot(x,y, 'R*-')` definiuje wykres *y(x)* jako czerwoną linię ciągłą ze znacznikami w postaci gwiazdy.

Tabela 13.1. Oznaczenia formatowania stylu wykresów programu MATLAB

kolor linii		**rodzaj linii**		**rodzaj znacznika punktowego**	
symbol	**opis**	**symbol**	**opis**	**symbol**	**opis**
B	niebieski	-	linia ciągla	.	punkt
G	zielony	:	linia punktowa	O	okrąg
R	czerwony	-.	linia „kropka-kreska”	X	znacznik „iks”
C	cyjan	--	linia przerywana	+	plus
M	magenta			*	gwiazda
Y	żółty			S	kwadrat
K	czarny			D	romb
W	biały			V	trójkąt „dół”
				^	trójkąt „góra”
				<	trójkąt „lewy”
				>	trójkąt „prawy”
				P	pięciokąt
				H	sześciokąt

W tabeli 13.2 zestawiono funkcje formatujące styl i wspomagające tworzenie wykresów.

Tabela 13.2. Funkcje formatujące styl i wspomagające tworzenie wykresów w programie MATLAB

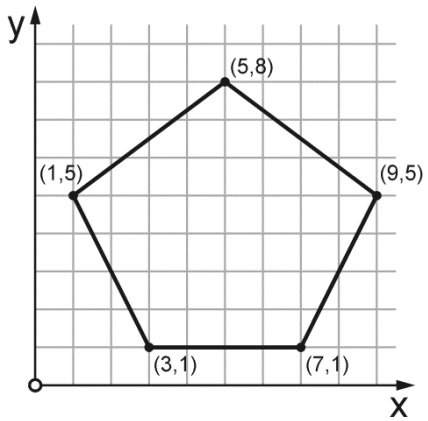
symbol	opis
`xlabel('tekst')`	dodaje opis osi poziomej (odcięta)
`ylabel('tekst')`	dodaje opis osi pionowej (rzędna)
`title('tekst')`	dodaje opis (etykietę) wykresu
`axis on/off`	włącza/wyłącza osie
`axis([xmin xmax ymin ymax])`	definiuje wartości ekstremalne osi
`axis equal`	definiuje jednakową skalę osi
`axis auto`	przywraca domyślny sposób definiowania osi; wartości ekstremalne są określana na podstawie danych wejściowych
`legend('tekst', 'tekst',…)`	definiuje legendę wykresu („tekstów” nie może być więcej niż wykresów)
`text(x,y, 'tekst')`	definiuje pole tekstowe z początkiem w punkcie o współrzędnych (x,y)
`hold on/off`	włącza/wyłącza zachowywanie poprzedniego wykresu w oknie graficznym (domyślnie *hold off*)
`grid on/off`	włącza/wyłącza linie siatki
`figure(n)`	aktywowanie *n*-tego okna graficznego
`close`	usuniecie aktualnego wykresu (usuniecie n-tego wykresu `close(n)`)

Polecenie `line(x,y)` generuje linię łamaną łączącą punkty o współrzędnych określonych przez wektory *x* i *y*.

Polecenie `fill(x,y,'kolor')` generuje wielokąt o wierzchołkach w punktach o współrzędnych określonych przez wektory *x* i *y*, wypełniony określonym kolorem.

Przykład 13.2

Narysować pięciokąt o współrzędnych wierzchołków zgodnych z rys. 13.3 za pomocą polecenia `line` i `fill` (rys. 13.4÷13.6).



Rys. 13.3. Pięciokąt z opisanymi współrzędnymi wierzchołków

```
clear; clc; close all;

x=[3 7 9 5 1 3];
y=[1 1 5 8 5 1];

line(x,y,'Color','R','LineStyle','-','Marker','*')

axis([0 10 0 10])
axis equal
grid on
xlabel('x')
ylabel('y')
```

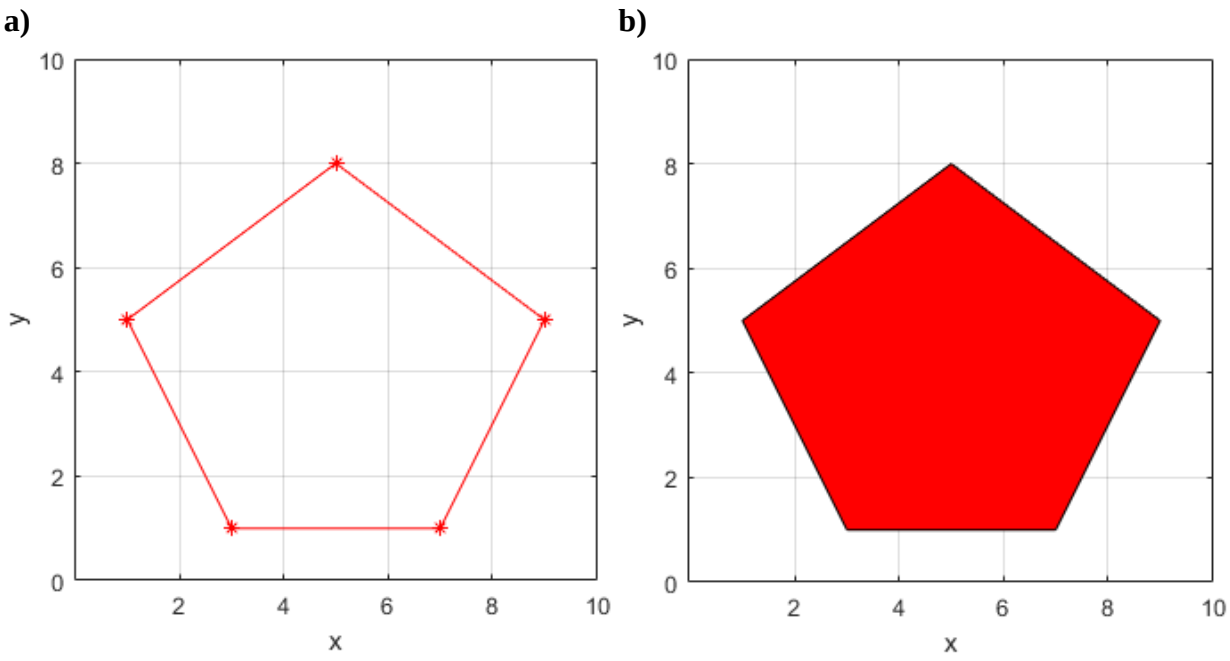
Rys. 13.4. Realizacja przykładu 13.2 przy pomocy funkcji `line`

Polecenie *line* generuje linię łamaną łączącą punkty o współrzędnych określonych przez wektory, dlatego aby uzyskać zamknięty pięciobok należy na początku i końcu wektorów *x* i *y* zdefiniować współrzędne tego samego punktu.

```
clear; clc; close all;

x=[3 7 9 5 1 3];
y=[1 1 5 8 5 1];

fill(x,y,'R')

axis([0 10 0 9])
axis equal
grid on
xlabel('x')
ylabel('y')
```

Rys. 13.5. Realizacja przykładu 13.2 przy pomocy funkcji `fill`

Polecenie `fill` generuje wielokąt o wierzchołkach w punktach o współrzędnych określonych przez wektory, dlatego nie ma potrzeby definicji współrzędnych dodatkowego punktu, takiego samego jak punkt początkowy.



Rys. 13.4. Pięciokąty wygenerowane za pomocą polecenia: a) `line`; b) `fill`

Przykład 13.3

Wygenerować krzywe Lissajous w zależności od amplitud sygnałów wejściowych, ich częstotliwości i przesunięcia fazowego (rys. 13.5).

Krzywe (nazywane także figurami) Lissajous są krzywymi parametrycznymi powstałymi podczas superpozycji drgań harmonicznych w dwóch wzajemnie prostopadłych kierunkach.

Równanie parametryczne krzywych

$$\begin{aligned} x &= A_1 \cos(\omega_1) \\ y &= A_2 \cos(\omega_2 - \varphi) \end{aligned} \tag{13.1}$$

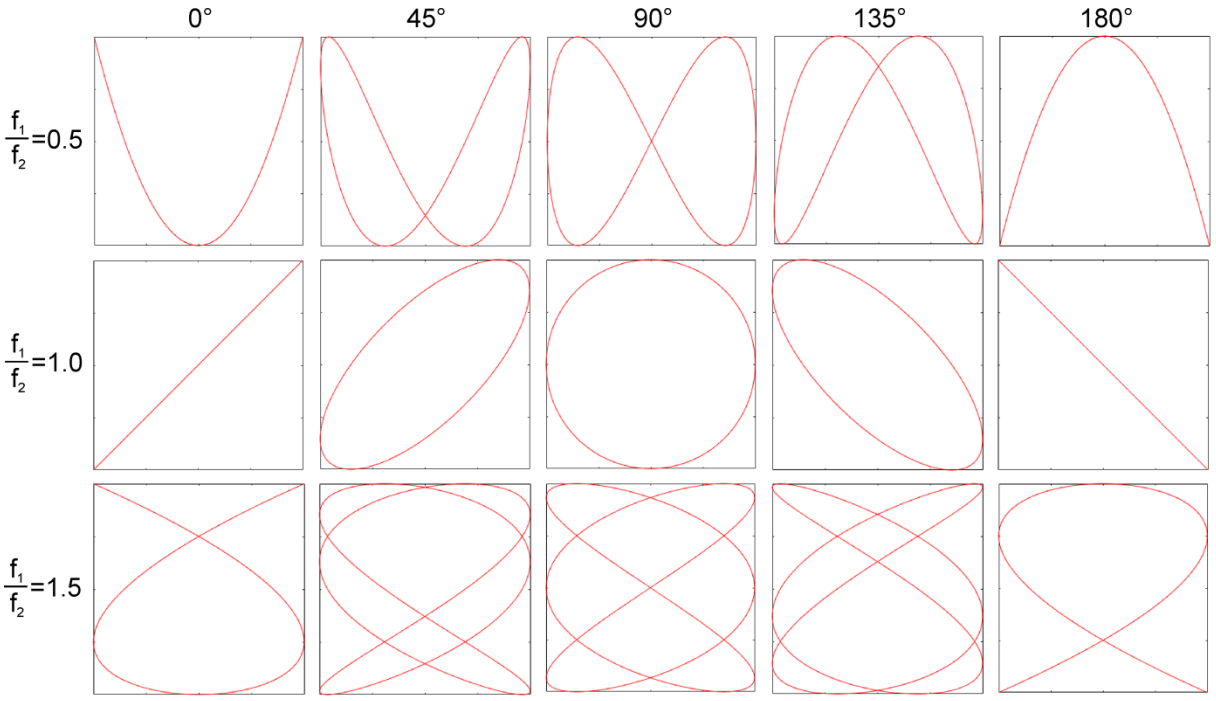
gdzie A_1 i A_2 są amplitudami sygnałów, ω_1 i ω_2 są czestościmi sygnałów, a φ jest przesunięciem fazowym sygnału w kierunku *y* w stosunku do kierunku *x*.

Częstotliwość drgań określona jest zależnością (12.26).

```
clear; clc; close all;

A1=input('Podaj amplitudę pierwszego sygnału A1 = ');
A2=input('Podaj amplitudę drugiego sygnału A2 = ');
f1=input('Podaj częstotliwość pierwszego sygnału f1 = ');
f2=input('Podaj częstotliwość drugiego sygnału f2 = ');
fi=input('Podaj przesunięcie fazowe fi = ');

t=linspace(0,pi,500);

x=A1*cos(2*pi*f1*t);
y=A2*cos(2*pi*f2*t-(fi*pi/180));

comet(x,y)
```

Rys. 13.5. Realizacja przykładu 13.3

Funkcja `comet` rysuje funkcję przy użyciu „głowy komety”, która podążaj od punktu początkowego do punktu końcowego.



Rys. 13.6. Przykładowe krzywe Lissajous uzyskane dla różnych stosunków częstotliwości i przesunięć fazowych (A_1=A_2=1)

Do tworzenia wykresów 3D w programie MATLAB służą m.in. funkcje `line` i `plot3`. W przypadku funkcji `line`, która odnosi się także do grafiki 2D, należy dodać trzeci wektor współrzędnych.

Przykład 13.4

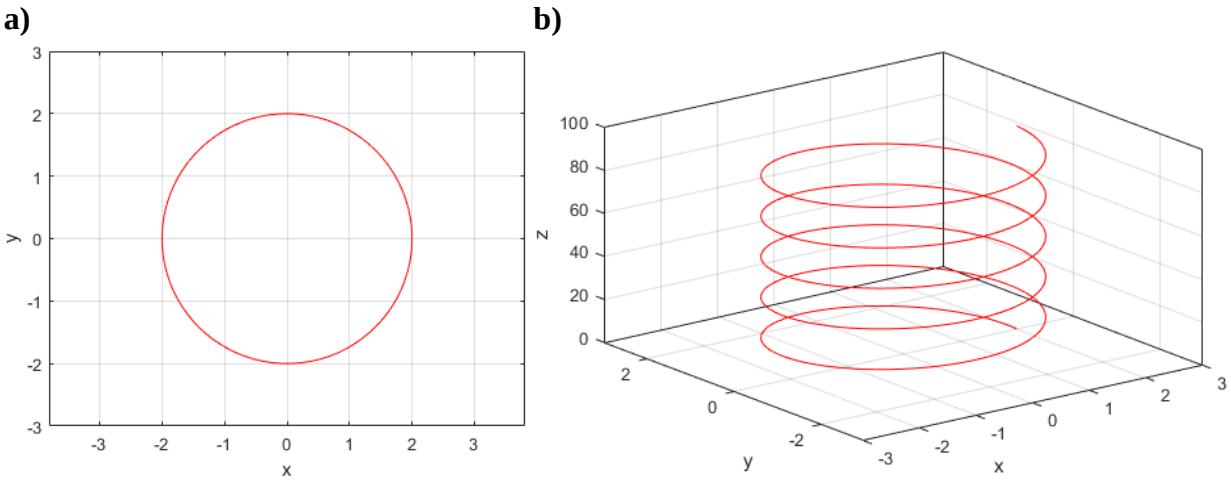
Narysować helisę (spiralę 3D) o promieniu r=2, wznoszącą się o $2\pi b$ co obrót (b=3) (rys. 13.7 i 13.8).

Równanie parametryczne helisy:

$$\begin{aligned} x &= a\cos(t) \\ y &= a\sin(t) \\ z &= bt \end{aligned} \tag{13.2}$$

```
clear; clc; close all;

r=2;
b=3;

t=[0:0.01:10*pi];

x=r*cos(t);
y=r*sin(t);
z=b*t;

line(x,y,z,'Color','R')

axis([-3 3 -3 3])
grid on
xlabel('x')
ylabel('y')
zlabel('z')
view(3)
```

Rys. 13.7. Realizacja przykładu 13.4



Rys. 13.8. Helisa wygenerowana za pomocą polecenia `line` w programie MATLAB: a) bez polecenia `view(3)`; b) z poleceniem `view(3)`

Aby poprawnie wyświetlić helisę (spiralę 3D) utworzoną za pomocą polecenia `line` należy dodać dodatkowo polecenie `view(3)`, które wyświetla krzywą w widoku izometrycznym (rys. 13.8 b). Bez tego polecenia helisa wyświetlana jest w widoku z góry (widok na płaszczyznę *xy*, rys. 13.8 a).

Do przekształcania wektorów w macierze, które można wykorzystać do znajdywania wartości funkcji dwóch zmiennych i generowania wykresów trójwymiarowych (3D) służy funkcja `meshgrid` [4].

Przykład 13.5

Za pomocą polecenia `meshgrid` utworzyć macierze X, Y z wektorów x i y (rys. 13.9):

$$x = [1 \quad 2 \quad 3 \quad 4]$$
$$y = [5 \quad 10 \quad 15]$$

```
>> x=[1 2 3 4];
>> y=[5 10 15];
>> [X,Y]=meshgrid(x,y)

X =

     1     2     3     4
     1     2     3     4
     1     2     3     4

Y =

     5     5     5     5
    10    10    10    10
    15    15    15    15
```

Rys. 13.9. Realizacja przykładu 13.5

Wiersze wynikowej macierzy *X* są kopiami wektora *x*, a kolumny wynikowej macierzy *Y* są kopiami wektora *y*. Para macierzy *X* i *Y* jest iloczynem kartezjańskim wektorów *x* i *y*.

Do generowania trójwymiarowych powierzchni służą m.in. polecenia `mesh`, `surf` i `surfl`.

Przykład 13.6

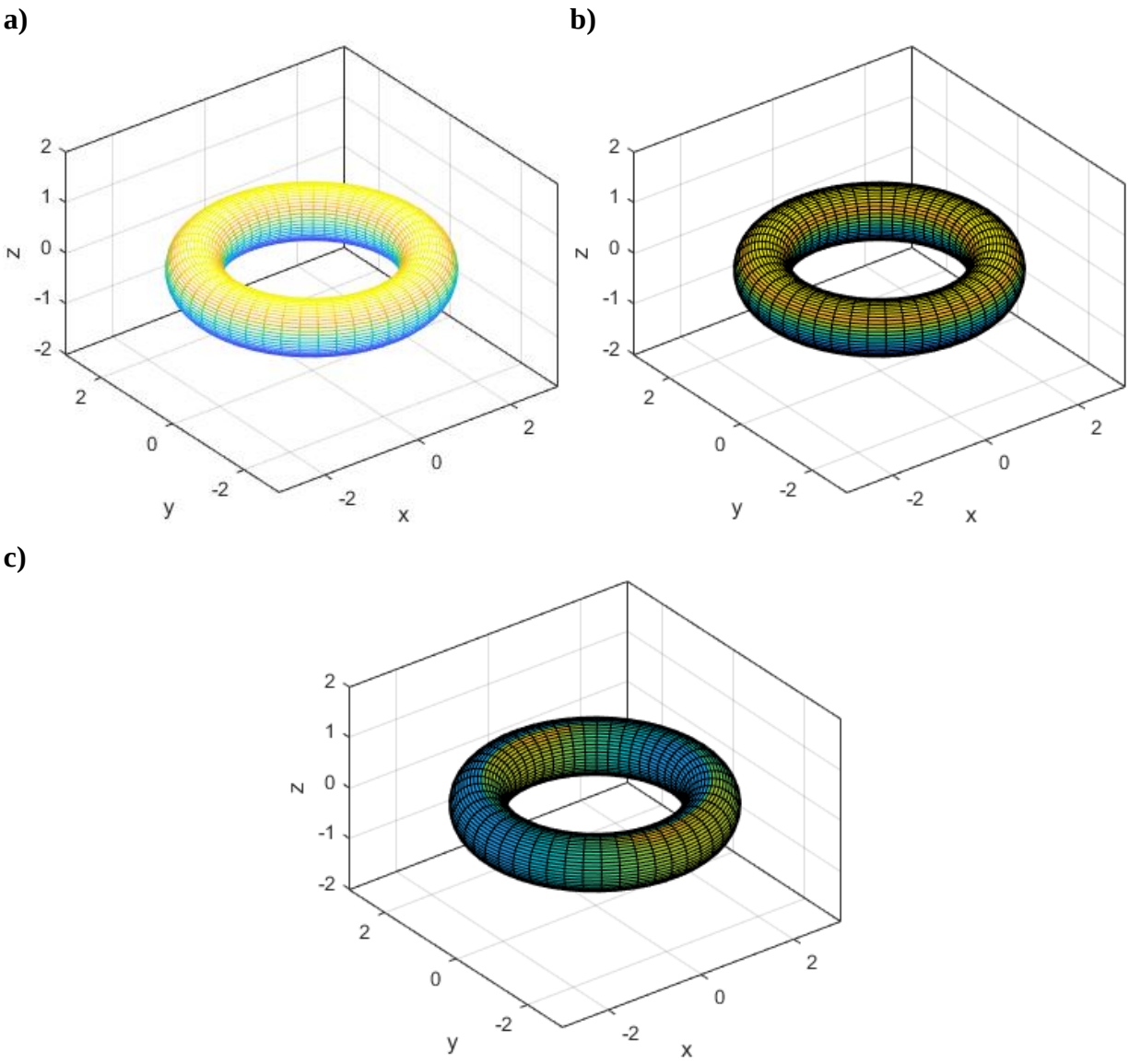
Wygenerować powierzchnię trójwymiarową będącą torusem, którego odległość od środka wynosi *R=2* a promień *r=0.5* (rys. 13.10 i 13.11).

Równanie parametryczne torusa:

$$\left.\begin{array}{c} x = \cos(a)\,(R + r\cos(b)) \\ y = \sin(a)\,(R + r\cos(b)) \\ z = r\sin(b) \end{array}\right\} \quad \begin{array}{l} a \in [0,2\pi) \\ b \in [0,2\pi) \end{array} \tag{13.3}$$

```
clear; clc; close all;

R=2;
r=0.5;

a=linspace(0,2*pi,50);
b=linspace(0,2*pi,50);

[a,b]=meshgrid(a,b);

x=cos(a).*(R+r*cos(b));
y=sin(a).*(R+r*cos(b));
z=r*sin(b);

mesh(x,y,z)

axis equal
grid on
axis([-3 3 -3 3 -2 2])
xlabel('x')
ylabel('y')
zlabel('z')
```

Rys. 13.10. Realizacja przykładu 13.6

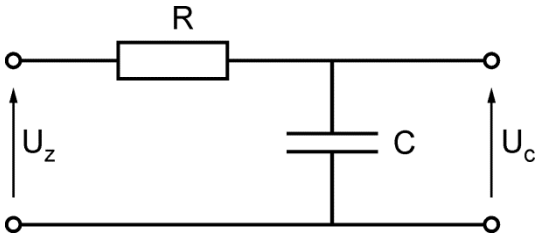


Rys. 13.11. Torus wygenerowany za pomocą poleceń: a) `mesh`; b) `surf`; c) `surfl`

Za pomocą polecenia `subplot(m,n,p)` można wygenerować dowolną liczbę wykresów rozmieszczonych w wierszach i kolumnach. W poleceniu `subplot`, *m* oznacza liczbę wykresów w poziomie (wierszu), *n* oznacza liczbę wykresów w pionie (kolumnie), *p* oznacza numer wykresu.

Przykład 13.7

Dla układu RC (rys. 13.12) wygenerować wykresy ładunku zgormadzonego na okładkach kondensatora, napięcia i natężenie prądu ładowania w funkcji czasu (rys. 13.13 i 13.14).



Rys. 13.12. Układ RC

Ładunek zgromadzony na okładkach kondensatora w funkcji czasu, podczas ładowania określony jest zależnością:

$$Q(t) = C \cdot U_z \cdot \left(1 - e^{-\frac{t}{RC}}\right) \tag{13.4}$$

Napięcie na okładkach kondensatora w funkcji czasu, podczas ładowania określone jest zależnością:

$$U_c(t) = \frac{Q(t)}{C} = U_z \cdot \left(1 - e^{-\frac{t}{RC}}\right) \tag{13.5}$$

Natężenie prądu ładowania w funkcji czasu określone jest zależnością:

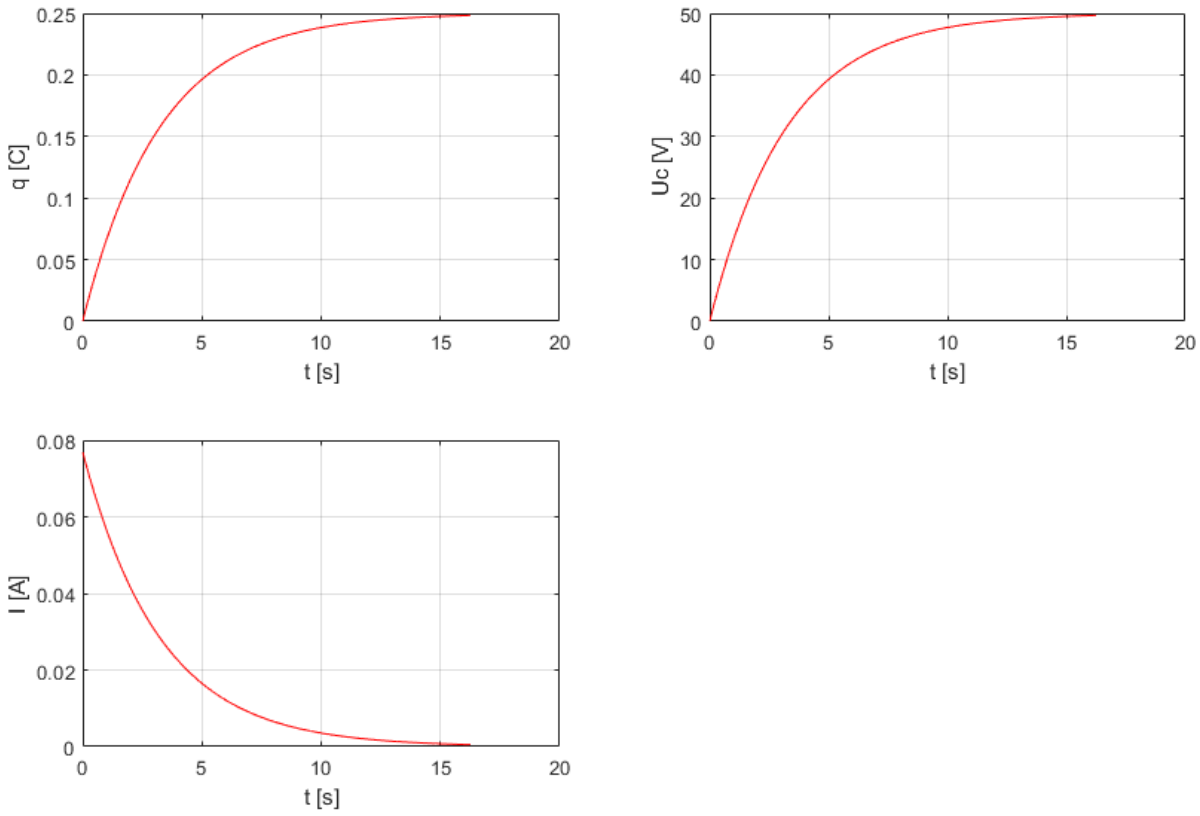
$$I(t) = \frac{U_z}{R} e^{-\frac{t}{RC}} \tag{13.6}$$

```
clear; clc; close all;

R=input('Podaj wartość rezystancji R [ohm] :');
C=input('Podaj wartość pojemności C [farad] :');
Uz=input('Podaj wartość napięcia zasilania Uz [volt] :');

t=[0:0.001:5*R*C];

q=C*Uz*(1-exp(-t/(R*C)));
Uc=q/C;
I=(Uz/R)*exp(-t/(R*C));

subplot(2,2,1)
plot(t,q,'r')
grid on
xlabel('t [s]')
ylabel('q [C]')

subplot(2,2,2)
plot(t,Uc,'r');
grid on
xlabel('t [s]')
ylabel('Uc [V]')
```

Rys. 13.13. Realizacja przykładu 13.7; część a

```
24  subplot(2,2,3)
25  plot(t,I,'r');
26  grid on
27  xlabel('t [s]')
28  ylabel('I [A]')
```

Rys. 13.14. Realizacja przykładu 13.7; część b (kontynuacja)



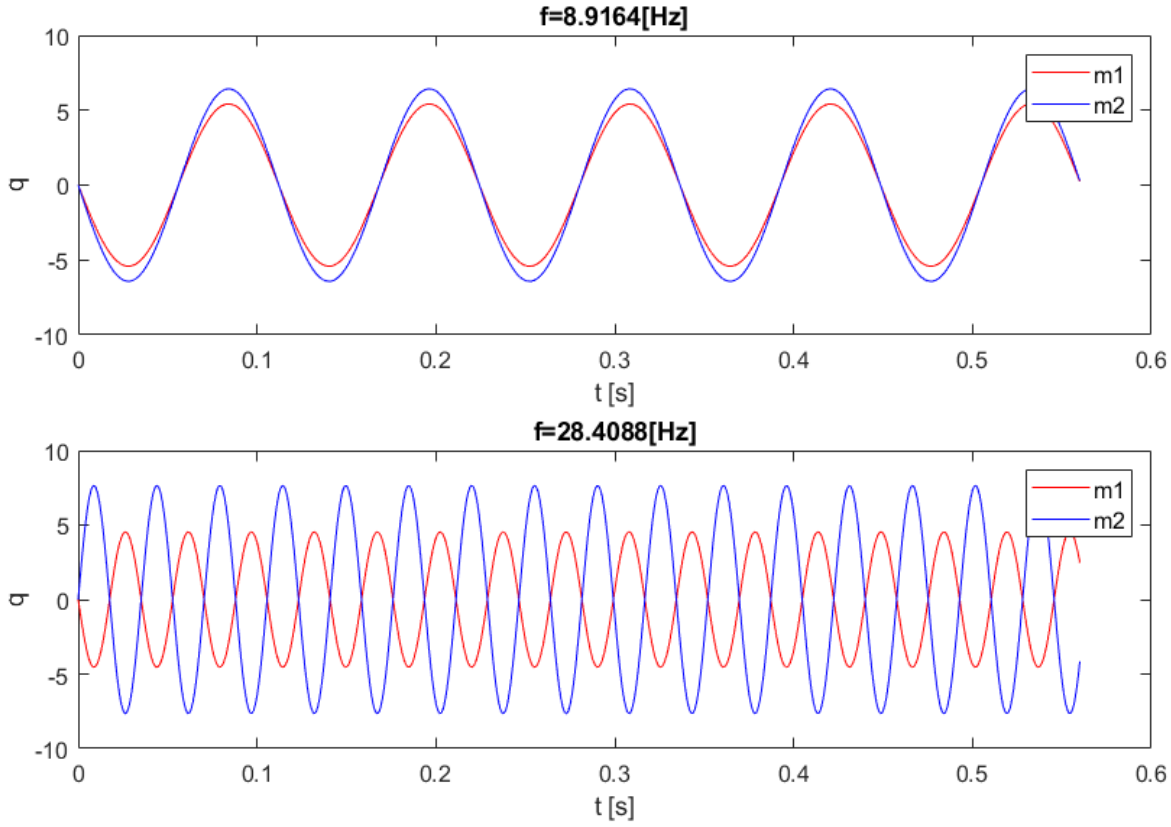
Rys. 13.15. Wykresy ładowania kondensatora dla R=650 Ω, C=0.005 F i U_z=50 V

Przykład 13.8

Rozbudować skrypt z przykładu 12.1 o tworzenie wykresów postaci drgań (rys. 13.16).

```
t=[0:0.001:5*(2*pi)/(sqrt(lambda(1,1)))];

q11=q(1,1)*sin((sqrt(lambda(1,1))*t));
q21=q(2,1)*sin((sqrt(lambda(1,1))*t));
q12=q(1,2)*sin((sqrt(lambda(2,2))*t));
q22=q(2,2)*sin((sqrt(lambda(2,2))*t));

subplot(2,1,1),plot(t,q11,'r')
hold on
plot(t,q21,'b')

xlabel('t [s]')
ylabel('q')
title(['f=',num2str(f(1,1)),'[Hz]'])
legend('m1','m2')

subplot(2,1,2),plot(t,q12,'r')
hold on
plot(t,q22,'b')

xlabel('t [s]')
ylabel('q')
title(['f=',num2str(f(2,2)),'[Hz]'])
legend('m1','m2')
```

Rys. 13.16. Kontynuacja skryptu z rys. 12.4 generująca wykresy przedstawione na rys. 13.17



Rys. 13.17. Postacie drgań układu z rys. 12.1 dla m_1=0.02 tony i m_2=0.01 tony oraz k_1=100 N/mm i k_2=200 N/mm

14. Instrukcje złożone

Przy tworzeniu programów przeznaczonych do wykonywania obliczeń lub złożonych poleceń wykorzystywane są instrukcje warunkowe i powtarzania.

Instrukcja warunkowa `if` stosowana jest, gdy wykonywane działanie uzależnione jest od spełnienia określonego warunku. Składnia instrukcji warunkowej `if` jest następująca:

```
if    warunek
instrukcja
end
```

W przypadku, gdy niespełnienie warunku ma skutkować wykonaniem innej instrukcji, składnia jest następująca:

```
if    warunek
instrukcja
else
      instrukcja
end
```

Przykład 14.1 [5]

Napisać skrypt, który oblicza odwrotność wczytanej liczby (rys. 14.1).

Odwrotność liczb jest możliwa do policzenia w przypadku liczb dodatnich i ujemnych. Dla „0” otrzymuje się symbol nieokreślony. Należy to uwzględnić w programie.

```
clear; clc;
disp('Obliczenie odwrotności liczby');
x=input('Podaj wartość liczby x = ');

%początek instrukcji warunkowej if
if (x<0) | (x>0)
   y=1/x;
   disp(y);

else disp('Błąd…wyrażenie nieokreślone')
end
```

Rys. 14.1. Realizacja przykładu 14.1

Składnia instrukcji warunkowej `if` w przypadku wielokrotnego warunku jest następująca:

```
if      warunek 1
instrukcja 1
elseif      warunek 2
        instrukcja 2
else  warunek 3
        instrukcja 3
end
```

Warunek wielokrotny `switch`, `case` i `otherwise` ma następującą składnię:

```
switch      zmienna
  case  wartość_1
    instrukcje
  case  wartość_2
    instrukcje
.
.
.
  otherwise
    instrukcje
end
```

Dla `case` instrukcje są wykonywane, gdy zmienna jest równa wartości. Dla `otherwise` instrukcja jest wykonywana, gdy zmienna nie pasuje do żadnego warunku.

Przykład 14.2

Napisać skrypt, który po podaniu nazwy planety Układu Słonecznego zwraca informację którą w kolejności jest planetą od Słońca (rys. 14.2).

```
clear;

planeta=input('Podaj nazwę planety Układu Słonecznego : ',’s’);
switch planeta
    case{'Merkury'}
        disp('Merkury jest pierwszą planetą Układu Słonecznego')
    case{'Wenus'}
        disp('Wenus jest drugą planetą Układu Słonecznego')
    case{'Ziemia'}
        disp('Ziemia jest trzecią planetą Układu Słonecznego')
    case{'Mars'}
        disp('Mars jest czwartą planetą Układu Słonecznego')
    case{'Jowisz'}
        disp('Jowisz jest piątą planetą Układu Słonecznego')
    case{'Saturn'}
        disp('Saturn jest szóstą planetą Układu Słonecznego')
    case{'Uran'}
        disp('Uran jest siódmą planetą Układu Słonecznego')
    case{'Neptun'}
        disp('Neptun jest ósmą planetą Układu Słonecznego')
    otherwise
        disp(' ');
        disp('Planeta nie należy do Układu Słonecznego')
        disp(' ');
        run('planeta.m');
end
```

Rys. 14.2. Realizacja przykładu 14.2

```
Podaj nazwę planety Układu Słonecznego : 'GJ1214b'

Planeta nie należy do Układu Słonecznego

Podaj nazwę planety Układu Słonecznego : 'Wenus'
Wenus jest drugą planetą Układu Słonecznego
```

Rys. 14.3. Przykładowy wynik działania skryptu z przykładu 14.2

W realizacji przykładu 14.2 zastosowano wczytanie zmiennej (funkcja `input`) jako tekst (znacznik `'s'`).

Instrukcja powtarzania `for` wykonuje działanie dla określonych wartości zmiennej. Wartości są kolejnymi elementami wektora, będącego warunkiem instrukcji (lub kolejnymi wektorami kolumnowymi pobranymi z macierzy będącej warunkiem instrukcji). Składnia instrukcji powtarzania `for` jest następująca:

```
for   zmienna=warunek
instrukcja
end
```

Przykład 14.3 [3, 6]

Napisać skrypt generujący wektor *A* o wymiarze 1*xn*, którego elementy spełniają zależność (rys. 14.4):

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

```
clear; clc;
n=input('Podaj n = ');

%początek pętli for
for i=1:n
   A(i)=sqrt(1+i);
end

disp(A)
```

Rys. 14.4. Realizacja przykładu 14.3

Przykład 14.4 [3, 6]

Napisać skrypt generujący wektor *A* o wymiarze *mxn*, którego elementy spełniają zależność (rys. 14.5):

$$A_{i,j} = \sqrt{1+\frac{i}{j}}$$

```
clear; clc;
m=input('Podaj m = ');
n=input('Podaj n = ');

%początek pętli for
for i=1:m
   for j=1:n
      A(i,j)=sqrt(1+i/j);
   end
end

disp(A)
```

Rys. 14.5. Realizacja przykładu 14.4

Przykład 14.5 [5]

Zdefiniować wektor losowy *X* złożony z *n* liczb o wartościach z przedziału <0,100>. Policzyć, ile jest wartości większych od *m* (rys. 14.6).

```
clear; clc;
n=input('Podaj n = ');
m=input('Podaj m = ');
X=rand(1,n)*100;
d=0;

for i=1:n
   if X(i)>m
      d=d+1;
   end
end

disp('Ilość liczb większych od 50 w rozpatrywanym przedziale')
disp(d)
```

Rys. 14.6. Realizacja przykładu 14.5

Przykład 14.6 [5]

Napisać skrypt obliczający charakterystyki geometryczne figury płaskiej (rys. 14.7) składającej się z trzech prostokątów [5]. W programie uwzględnić możliwość obliczania charakterystyk dla figur płaskich składających się z dowolnej liczby prostokątów (rys. 14.8).

Charakterystyki geometryczne przekrojów poprzecznych określają wpływ przekrojów poprzecznych obiektu na jego wytrzymałość. Podstawową charakterystyką jest pole przekroju. Reprezentuje ono wytrzymałość obiektu jedynie dla niektórych przypadków obciążeń, takich jak rozciąganie/ściskanie i ścinanie.

Redukcja sił wewnętrznych działających w przekroju odbywa się do tzw. bieguna redukcji, którym jest geometryczny środek przekroju. Wyznaczenie środka geometrycznego figury płaskiej (przekroju) wymaga znajomości momentów geometrycznych pierwszego stopnia (tzw. momentów statycznych).

Przy zginaniu, ważne jest zorientowanie kształtu przekroju poprzecznego względem kierunku obciążenia (momentu zginającego). Cechę tę określają momenty geometryczne drugiego stopnia (tzw. momenty bezwładności).

Dla figury z rys. 14.7, pole powierzchni można wyznaczyć:

$$A = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} b_i h_i \tag{14.1}$$

Moment statyczny:

$$S = \sum_{i=1}^{n} y_i A_i = \sum_{i=1}^{n} y_i b_i h_i \tag{14.2}$$

gdzie

$$y_1 = \frac{h_1}{2} \quad ; \quad y_i = \sum_{k=1}^{i-1} y_{i-1} + \frac{(h_{i-1}+h_i)}{2} \tag{14.3}$$

Środek ciężkości:

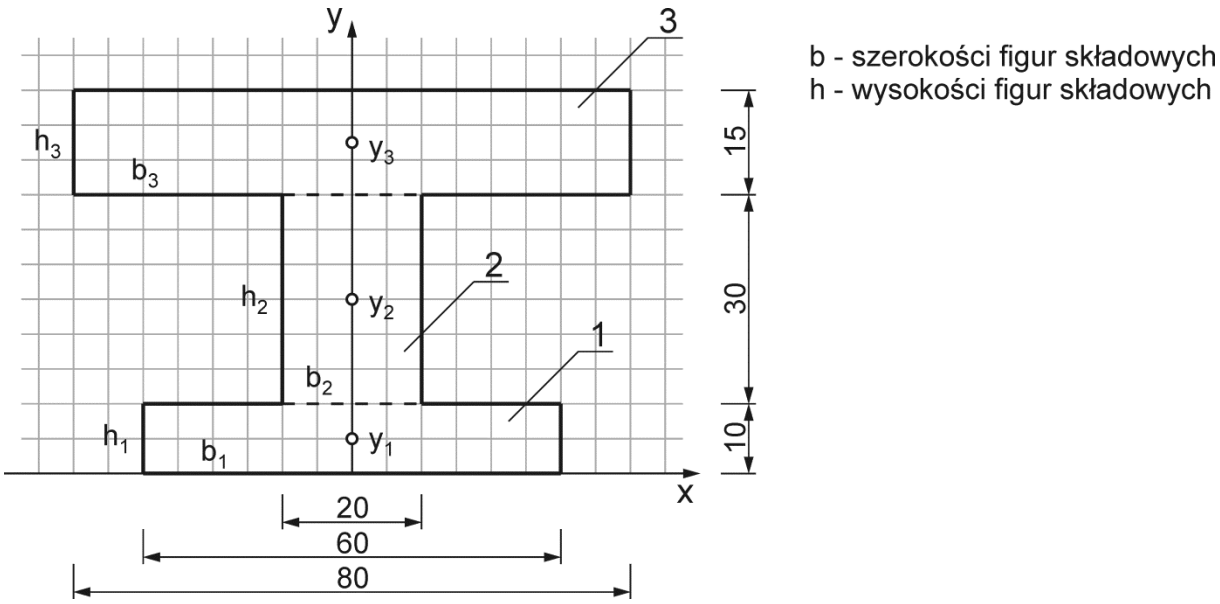
$$y_c = \frac{S}{A} \tag{14.4}$$

Moment bezwładności:

$$I = I_0 + I_{st} = \sum_{i=1}^{n} \frac{b_i h_i^3}{12} + \sum_{i=1}^{n} d_i^2 A_i \tag{14.5}$$

gdzie

$$d_i = y_i - y_c \tag{14.6}$$



Rys. 14.7. Figura płaska składająca się z trzech prostokątów

```
clear; clc;
%wczytanie wymiarów prostokątów
disp('Podaj szerokości prostokątów');
b=input('[b1 b2…bn]:');
disp('Podaj wysokości prostokątów');
h=input('[h1 h2…hn]:');

n=length(b);

%pola powierzchni
Ai=b.*h;
A=sum(Ai);

%środki ciężkości prostokątów składowych
y(1)=h(1)/2;

for i=2:n
   y(i)=y(i-1)+(h(i-1)+h(i))/2;
end

%momenty statyczne
Si=y.*Ai;
S=sum(Si);

%środek ciężkości
yc=S/A;

%momenty bezwładności prostokątów składowych
I0i=b.*h.^3/12;

%odległości środków ciężkości prostokątów od środka ciężkości figury
d=y-yc;

%momenty bezwładności
Ist=Ai.*d.^2;
I=sum(I0i+Ist);

%wyświetlanie wyników
disp('pole powierzchni figury A');
disp(A);

disp('moment statyczny S');
disp(S);

disp('środek ciężkości yc');
disp(yc);

disp('moment bezwładności I');
disp(I);
```

Rys. 14.8. Realizacja przykładu 14.6

Dla figury z rys. 14.7 otrzymuje się:

A=2400 mm^2

S=75000 mm^3

y_c=31.25 mm

I=826250 mm^4

Instrukcja powtarzania `while` wykonuje działanie dopóki spełniony jest warunek. Składnia instrukcji powtarzania `while` jest następująca:

```
while      warunek
instrukcja
end
```

Przykład 14.7 [3, 6]

Napisać skrypt wykonujący dodawanie do kolejnych liczb wartości jeden aż do osiągnięcia wartości 100 (rys. 14.9).

```
1  clear; clc;
2  i=0;
3
4  while i<100
5     i=i+1
6  end
```

Rys. 14.9. Realizacja przykładu 14.7

Przykład 14.8 [5]

Napisać skrypt losujący cyfrę od 0 do 9 i proszący użytkownika o odgadnięcie tej cyfry (rys. 14.10).

```
1  clear; clc;
2
3  x=rand(1)*10;
4  x=floor(x);
5
6  y=input('podaj cyfrę od 0 do 9 : ');
7
8  while y~=x
9      disp('NIE');
10     y=input('podaj cyfrę od 0 do 9 : ');
11 end
12
13 if y==x
14     disp('ZGADZA SIĘ');
15 end
```

Rys. 14.10. Realizacja przykładu 14.8

Przykład 14.9 [5]

Napisać skrypt rozwiązujący dowolne równanie kwadratowe

$$ax^2 + bx + c = 0$$

Dla równania liniowego zastosować funkcję `roots` (rys. 14.11).

```
clear; clc; close all;
disp('Rozwiązanie równania kwadratowego ax^2+bx+c=0')

a=input('Podaj wartość parametru a = ');
b=input('Podaj wartość parametru b = ');
c=input('Podaj wartość parametru c = ');

if a==0
    x1=roots([b c]);
    disp('Rozwiązanie:')
    disp(x1)
else
    delta=b^2-4*a*c;
    if delta>0
        x1=(-b+sqrt(delta))/(2*a);
        x2=(-b-sqrt(delta))/(2*a);
        disp('Rozwiązanie:')
        disp([x1 x2])
    else
        x1=(-b+i*sqrt(-delta))/(2*a);
        x2=(-b-i*sqrt(-delta))/(2*a);
        disp('Rozwiązanie:')
        disp([x1 x2])
    end
end

if a==0
    x=[x1-10:1:x1+10];
    y=(x.^2).*a+x.*b+c;
    plot(x,y,'R')
    hold on
elseif delta>0
    x=[x2-10:1:x1+10];
    y=(x.^2).*a+x.*b+c;
    plot(x,y,'R')
    hold on
else
    disp('Rozwiązanie w dziedzinie liczb zespolonych')
end

if a==0 | delta>0
    y1=x*0;
    plot(x,y1,'B--')
    grid on
    xlabel('x')
    ylabel('y')
end
```

Rys. 14.11. Realizacja przykładu 14.9

15. Definiowanie funkcji zewnętrznych

Funkcje definiowane są w M-pliku o nazwie odpowiadającej nazwie funkcji. Składnia funkcji `nazwa` zdefiniowanej w pliku nazwa.m jest następująca [1, 5]:

```
function  [output]=nazwa(input)
```

gdzie:

`output` – nazwy zmiennych, w których zwracany jest wynik działania funkcji

`input` – nazwy zmiennych wartości wejściowych

UWAGA: Zmienne w funkcjach mają zasięg ograniczony do funkcji, nie są dostępne na zewnątrz funkcji [1].

Przykład 15.1 [1]

Napisać skrypt generujący okrąg za pomocą funkcji zewnętrznej (rys. 15.1 i 15.2).

Równanie parametryczne okręgu o promieniu R:

$$\begin{aligned} x &= R\cos(t) \\ y &= R\sin(t) \end{aligned} \tag{15.1}$$

gdzie $t \in [0,2\pi)$

```
1  function [x,y]=okrag(R,t);
2  t=[0:0.01:2*pi];
3  x=cos(t)*R;
4  y=sin(t)*R;
5  plot(x,y)
6  axis equal
```

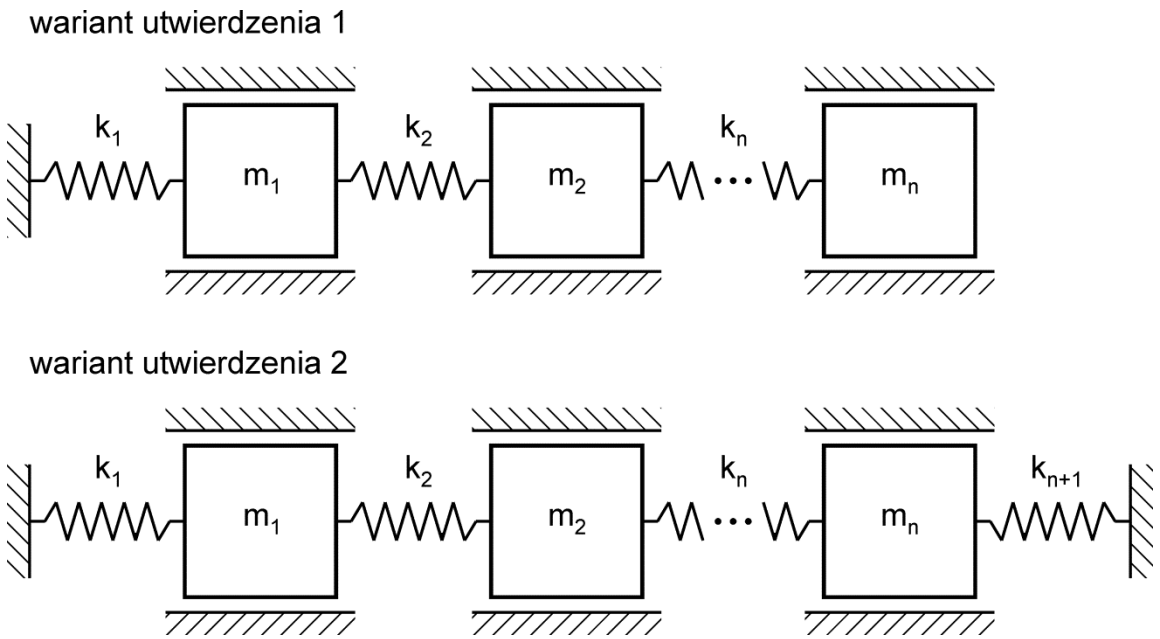
Rys. 15.1. Plik okrag.m

```
1  clear; clc; close;
2
3  R=input('Podaj promień okręgu R = ');
4  okrag(R);
```

Rys. 15.2. Realizacja przykładu 15.1

Przykład 15.2

Rozbudować skrypt do wyznaczania częstotliwości drgań własnych układu (przykład 12.1) na dowolną liczbę stopni swobody i dwa warianty utwierdzenia (rys. 15.3÷15.5).



Rys. 15.3. Układ o *n* stopniach swobody

```
1  function [y]=ke(k)
2  y=ones(2);
3  y(1,2)=-1;
4  y(2,1)=-1;
5  y=y*k;
```

Rys. 15.4. Plik ke.m

```
clear; clc;

mn=input('Podaj liczbę mas =');
for i=1:mn
    m(i)=input('Podaj wartości mas=');
end

kn=input('Podaj liczbę sprężyn =');
for i=1:kn
    k(i)=input('Podaj wartości sztywności=');
end

M=diag(m);

K=zeros(kn+1);
for i=1:kn
    K(i:i+1,i:i+1)=K(i:i+1,i:i+1)+ke(k(i));
end

if mn==kn
    K=K(2:kn+1,2:kn+1);
else
    K=K(2:kn,2:kn);
end

B=sqrt(M);
D=inv(B)*K*inv(B);

[x,lambda]=eig(D);
f=(sqrt(lambda))/(2*pi)
q=inv(B)*x
```

Rys. 15.5. Realizacja przykładu 15.2

16. Analiza funkcji

W tabeli 16.1 zestawiono wybrane operacje na funkcjach środowiska MATLAB [2].

Tabela 16.1. Operacje na funkcjach programu MATLAB [2]

symbol	opis
`fplot(f, [a b])`	tworzenie wykresu funkcji f w przedziale <a, b>
`fminbnd(f, a, b)`	minimum funkcji f w przedziale <a, b>
`fzero(f, a)`	miejsce zerowe funkcji f w otoczeniu a
`fzero(f, [a b])`	miejsce zerowe funkcji f w przedziale <a, b>

Przykład 16.1

Dla funkcji $y = \sin^2(x) + 2\cos^3(x)$ wyznaczyć (rys. 16.1÷16.4):

1/ miejsce zerowe w przedziale <3, 5>

2/ miejsce zerowe w otoczeniu 2

3/ minimum funkcji w przedziale <2, 4>

4/ maksimum funkcji w przedziale <4, 7>

```
1  function y=func(x)
2  y=(sin(x).^2+2*cos(x).^3);
```

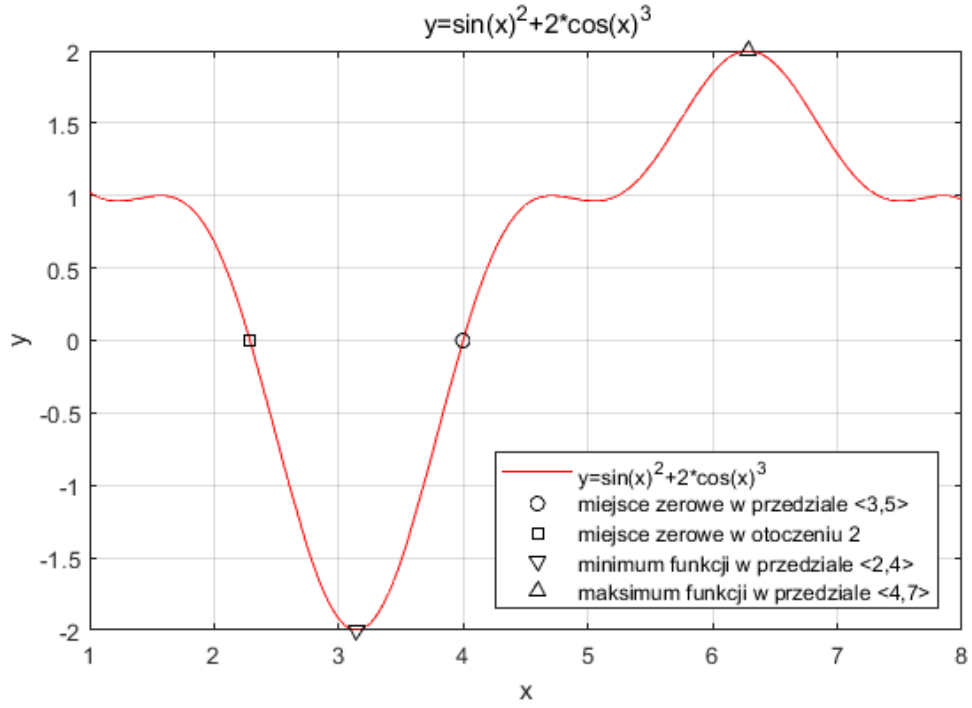
Rys. 16.1. Plik func.m

```
1  function y=mfunc(x)
2  y=-((sin(x).^2+2*cos(x).^3));
```

Rys. 16.2. Plik mfunc.m

```
clear; clc; close all;

zeroA=fzero('func', [3 5])
zeroB=fzero('func',2)
min=fminbnd('func',2,4)
max=fminbnd('mfunc',4,7)

fplot('func', [1 8],'r');

grid on
xlabel('x')
ylabel('y')
title('y=sin(x)^2+2*cos(x)^3')

hold on

plot(zeroA,func(zeroA),'K O')
plot(zeroB,func(zeroB),'K S')
plot(min,func(min),'K V')
plot(max,func(max),'K ^')

legend('y=sin(x)^2+2*cos(x)^3',...
    'miejsce zerowe w przedziale <3,5>',...
    'miejsce zerowe w otoczeniu 2',...
    'minimum funkcji w przedziale <2,4>',...
    'maksimum funkcji w przedziale <4,7>')
```

Rys. 16.3. Realizacja przykładu 16.1



Rys. 16.4. Wynik działania skryptu z przykładu 16.1

Wynikiem działania skryptu jest:

1/ miejsce zerowe analizowanej funkcji w przedziale <3, 5>: zeroA = 3.9952

2/ miejsce zerowe analizowanej funkcji w otoczeniu 2: zeroB = 2.2880

3/ minimum analizowanej funkcji w przedziale <2, 4>: min = 3.1416

4/ maksimum analizowanej funkcji w przedziale <4, 7>: max = 6.2832

17. Analiza sygnałów

Każdy sygnał okresowy można przedstawić w formie sumy sygnałów sinusoidalnych o różnych amplitudach i częstotliwościach. Transformata Fouriera lub jej zoptymalizowana komputerowa wersja szybka transformata Fouriera (FFT – ang. *Fast Fourier Transform*) pozwala na szybkie i dokładne określenie z jakich składowych składa się dany sygnał okresowy.

Program MATLAB umożliwia przeprowadzenie szybkiej transformaty Fouriera FFT za pomocą funkcji `fft`.

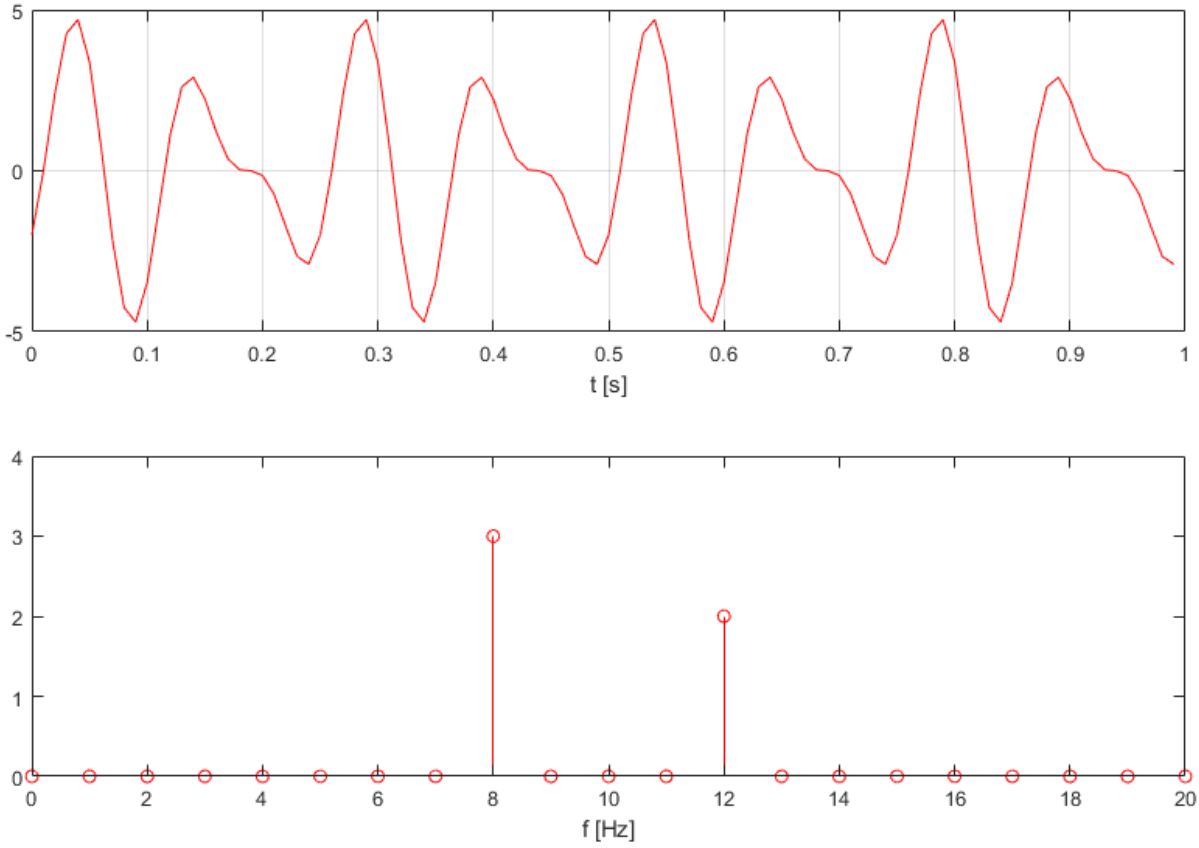
Przykład 17.1 [2, 6]

Przeprowadzić szybką transformatę Fouriera sygnału okresowego określonego wzorem (rys. 17.1 i 17.2):

$$y = A_1 \sin(2\pi f_1 t) - A_2 \cos(2\pi f_2 t)$$

```
clear; clc; close all;

dt=0.01;
t=[0:dt:1-dt];
fp=1/dt;
n=length(t);
df=fp/n;
f=[0:df:fp-df];

A1=input('Podaj amplitudę A1 = ');
f1=input('Podaj częstotliwość f1 = ');
A2=input('Podaj amplitudę A2 = ');
f2=input('Podaj częstotliwość f2 = ');

fun=A1*sin((2*pi*f1)*t)-(A2*cos((2*pi*f2)*t));

subplot(2,1,1);
plot(t,fun,'r');
grid on
xlabel('t [s]');

wid=fft(fun);

widabs=(2/n)*abs(wid);

subplot(2,1,2);
stem(f,widabs,'r');
xlim([0 20]);
xlabel('f [Hz]');
```

Rys. 17.1. Realizacja przykładu 17.1



Rys. 17.2. Przykład działania skryptu z przykładu 17.1 dla funkcji $y = 3\sin(2\pi \cdot 8t) - 2\cos(2\pi \cdot 12t)$

18. Analiza statystyczna

W tabeli 18.1 zestawiono wybrane funkcje statystyczne środowiska MATLAB [2].

Tabela 18.1. Wybrane funkcje statystyczne programu MATLAB [2]

symbol	opis
`max`	wartość maksymalna
`min`	wartość minimalna
`mean`	wartość średnia
`median`	wartość środkowa
`std`	odchylenie standardowe
`sort`	uporządkowanie rosnące
`corrcoef`	współczynnik korelacji
`hist`	histogram
`sum`	suma elementów
`prod`	iloczyn elementów
`cov`	kowariancja
`var`	wariancja

Przykład 18.1

Wygenerować ciąg 200 losowych liczb w przedziale od 0 do 100. Wyznaczyć (rys. 18.1÷18.3):

1/ wartość maksymalną i minimalną

2/ wartość średnią i odchylenie standardowe

3/ histogram

Wartość średnia (arytmetyczna) jest miarą tendencji centralnej, czyli pozycji skali pomiarowej, wokół której skupiają się zaobserwowane wartości zmiennej:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_1}{n} \tag{18.1}$$

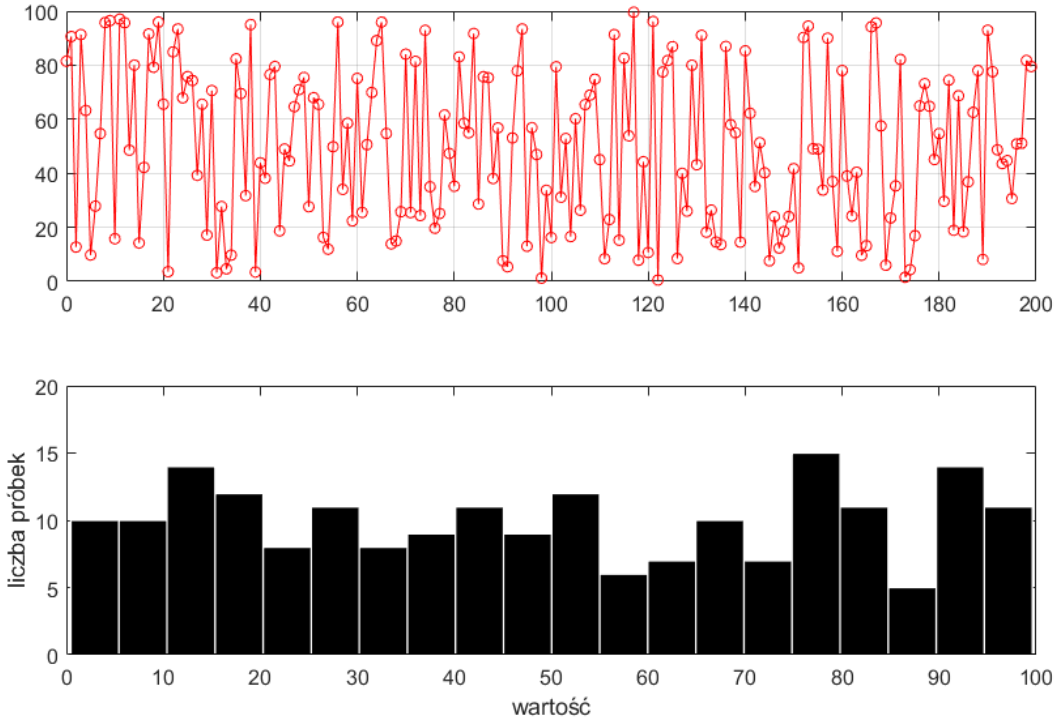
Odchylenie standardowe jest miarą zmienności, czyli określa rozrzut wyników wokół wartości średniej:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} \tag{18.2}$$

Histogram jest graficznym przedstawieniem rozkładu empirycznego cechy, czyli wartości przyjmowanych przez cechy w próbie przy pomocy częstości ich występowania.

```
clear; clc; close all;

n=200;
x=[0:1:n-1];
y=rand(1,n)*100;

subplot(2,1,1)
plot(x,y,'r-O')
grid on

MaxValue=max(y)
MinValue=min(y)
MeanValue=mean(y)
StandardDeviation=std(y)

subplot(2,1,2)
hist(y,20)
xlabel('wartość');
ylabel('liczba próbek');
```

Rys. 18.1. Realizacja przykładu 18.1



Rys. 18.2. Przykład działania skryptu z przykładu 18.1

```
MaxValue =

   99.6135

MinValue =

   0.4634
```

```
MeanValue =

   49.7724

StandardDeviation =

   29.4198
```

Rys. 18.3. Przykładowe wyniki liczbowe uzyskane dla skryptu z przykładu 18.1

19. Równania różniczkowe

Równanie różniczkowe postaci

$$\frac{dy}{dt} = F(t, y)$$
$$y(t = 0) = y_o \tag{19.1}$$

można rozwiązać w programie MATLAB za pomocą siedmiu funkcji (solverów) [2]:

`ode45` – zmodyfikowana jednokrokowa metoda Rungego-Kutty

`ode23` – zmodyfikowana jednokrokowa metoda Rungego-Kutty, rzędu 2 i 3

`ode113` – metoda Adamsa-Bashtortha-Moultona (PECE)

`ode15i` – algorytm rozwiązywania równania różniczkowego w postaci niejawnej $f(t, x, x') = 0$

`ode15s` – metoda wielokrokowa NDFs (opcjonalnie BDFs)

ode23s – zmodyfikowana jednokrokowa metoda Rosenbrocka, rzędu 2

`ode23t` – metoda trapezowa

`ode23tb` – metoda TR-BDF2

Składnia polecenia wywołującego funkcję rozwiązywania równania różniczkowego jest następująca (z pominięciem parametrów dodatkowych) [1, 2]:

```
[tout, yout]=odeXX(odefun, tspan, y0)
```

gdzie:

`[tout, yout]` – wektory czasu i wartości rozwiązań y(t)

`odeXX` – nazwa funkcji rozwiązującej równanie różniczkowe

`odefun` – nazwa M-pliku (lub jego uchwytu `@odefun` w którym zapisane jest równanie lub układ równań różniczkowych pierwszego rzędu

`tspan` – wektor, w którym określa się początkową i końcową wartość zmiennej niezależnej (zwykle czas)

`y0` – wartości początkowe

W pierwszej kolejności zaleca się stosowanie funkcji `ode45`, która daje poprawne rozwiązanie dla dużej grupy równań [1, 2].

Przykład 19.1 [1]

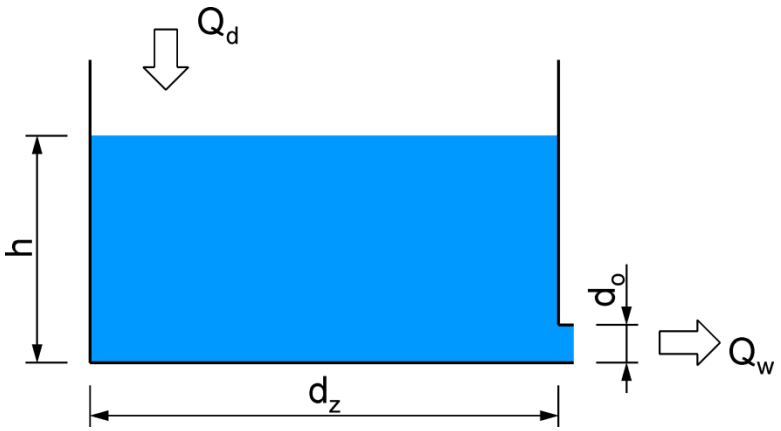
Do zbiornika cylindrycznego o średnicy d_z=2 m napływa ciecz doskonała z natężeniem Q_d i wypływa przez otwór o średnicy d_w=0.1 m. Wysokość początkowa lustra cieczy w zbiorniku wynosi $h(t=0)$=1.5 m. Ciecz napływa do zbiornika:

1/ w czasie od 0 s do 100 s, z natężeniem Q_d=0.06 m^3/s

2/ w czasie od 100 s do 150 s, ciecz nie napływa

3/ w czasie od 150 s do 200 s, z natężeniem Q_d=0.10 m^3/s.

Wyznaczyć wykres wysokości lustra cieczy w zbiorniku w czasie od 0 s do 1500 s (rys. 19.2÷19.4).



Rys. 19.1. Zbiornik z cieczą

Dla rozpatrywanego zbiornika bilans natężenia przepływu wynosi

$$Q = Q_d - Q_w \tag{19.2}$$

Natężenie przyrostu ilości cieczy wynosi

$$Q = A_z \frac{dh}{dt} \tag{19.3}$$

Natężenie wypływu cieczy wynosi

$$Q_w = A_w \sqrt{2gh} \tag{19.4}$$

gdzie A_z i A_w są polami powierzchni, odpowiednio dna zbiornika i otworu wylotowego.

Po podstawieniu wzorów (19.3) i (19.4) do równania (19.2) otrzymuje się

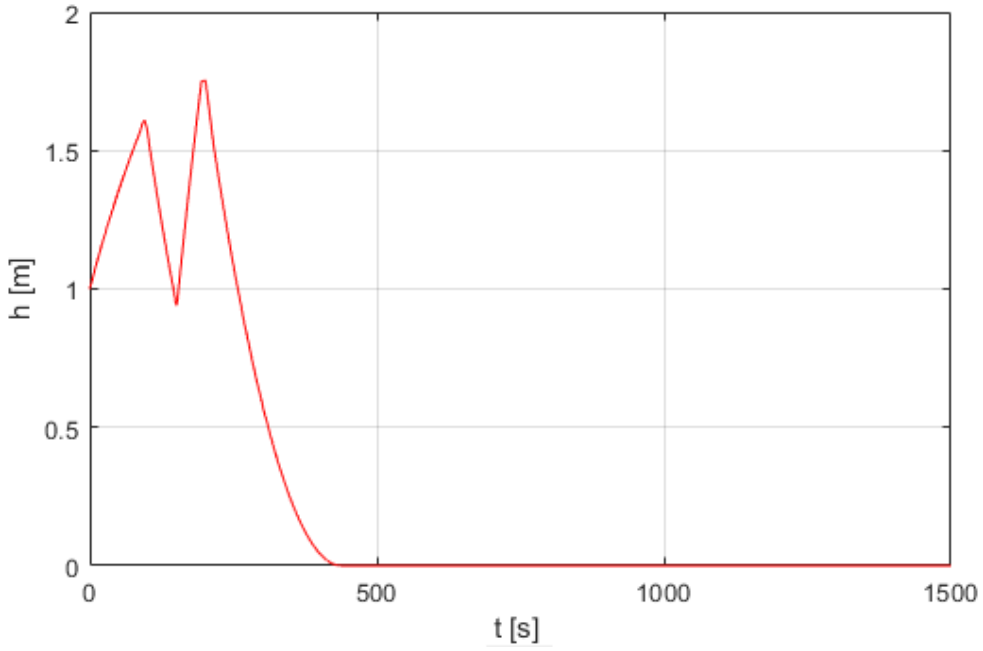
$$\frac{dh}{dt} = \frac{Q_d - A_w\sqrt{2gh}}{A_z} \tag{19.5}$$

```
function dh=ciecz(t,h)
dz=2;
dw=0.1;
g=9.81;
Az=0.25*pi*dz^2;
Aw=0.25*pi*dw^2;
dh=0;

if t<100
    Qd=0.06;
elseif t>=100 & t<150
    Qd=0;
elseif t>=150 & t<200
    Qd=0.1;
else
    Qd=0
end

Qw=(Aw*(2*g*h).^0.5);

if h<0
    Qw=0;
end

dh=(Qd-Qw)/Az;
```

Rys. 19.2. Plik ciecz.m

```
clear; clc; close all;

[t,y]=ode45('ciecz',[0 1500],1);

plot(t,y,'r')
xlabel('t [s]')
ylabel('h [m]')
grid on
```

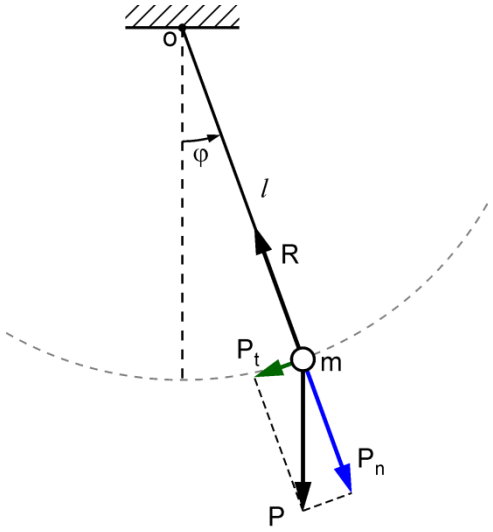
Rys. 19.3. Realizacja przykładu 19.1



Rys. 19.4. Wykres zmiany poziomu cieczy w zbiorniku w funkcji czasu dla danych z przykładu 19.1

Przykład 17.2 [5]

Napisać skrypt umożliwiający analizę ruchu wahadła matematycznego w zakresie małych wychyleń (rys. 19.5÷19.7).



Rys. 19.5. Wahadło matematyczne

Wahadło matematyczne jest punktem materialnym o masie m zawieszonym na cienkiej, wiotkiej i nierozciągliwej linie o pomijalnie małej masie.

Torem ruchu wahadła matematycznego jest łuk koła o promieniu równym długości liny, a położenie wahadła dla dowolnej chwili czasu jest określone za pomocą kąta φ.

Na masę skupioną m działają dwie siły, pionowa siła ciężkości (siła czynna)

$$P = mg \tag{19.6}$$

oraz reakcja R (siła naciągu liny). Siłę czynną P można rozłożyć na dwie składowe, styczną i normalną do toru ruchu:

$$\begin{aligned} P_t &= mg \sin\varphi \\ P_n &= mg \cos\varphi \end{aligned} \tag{19.7}$$

Składowa P_n jest równoważona przez reakcję R.

Zgodnie z zasadą d'Alemberta można zapisać

$$ma_t = -mg \sin\varphi \tag{19.8}$$

gdzie ma_t jest siłą bezwładności. W przypadku ruchu po okręgu, przyspieszenie styczne a_t jest równe

$$a_t = \ddot{\varphi} l \tag{19.9}$$

Po podstawieniu i uproszczeniu

$$\ddot{\varphi} + \frac{g}{l} \sin\varphi = 0 \tag{19.10}$$

Dla małych kątów $\sin\varphi \approx \varphi$, równanie (16.10) można zapisać

$$\ddot{\varphi} + \frac{g}{l}\varphi = 0 \tag{19.11}$$

Warunki początkowe przyjmują postać

$$\varphi(0) = \varphi_0 \quad \dot{\varphi}(0) = 0 \tag{19.12}$$

Rozwiązanie równania różniczkowego drugiego stopnia otrzymuje się zastępując je układem równań pierwszego stopnia

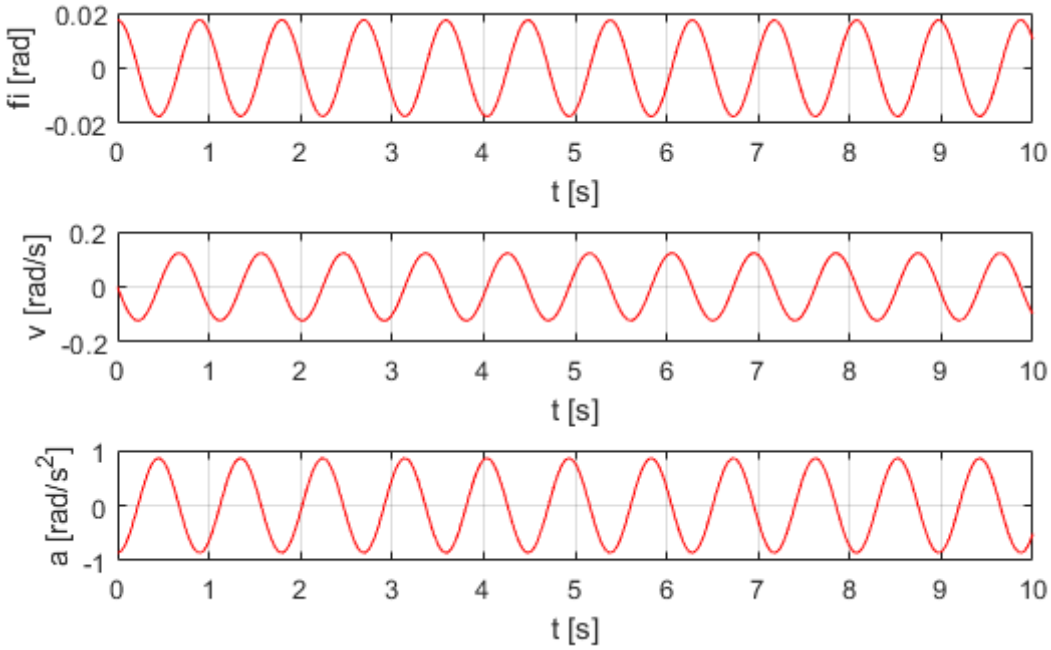
$$\begin{cases} \dot{\varphi} = v \\ \dot{v} = -\frac{g}{l}\varphi \end{cases} \tag{19.13}$$

```
1  function dy=poch(t,y)
2  global g l
3  dy=[y(2);-(g/l)*y(1)];
```

Rys. 19.6. Plik poch.m

```
1  clear; clc; close all;
2  global g l
3
4  tmax=10;
5  t0=0:tmax/1000:tmax;
6  g=9.81;
7  v0=0;
8
9  fi0=input('podaj wychylenie początkowe fi0 [st] = ');
10 l=input('podaj długość wahadła l [m] = ');
11 y0=[fi0*pi/180;v0];
12 [ty,y]=ode45('poch',t0,y0);
13
14 subplot(3,1,1)
15 plot(ty,y(:,1),'r');
16 grid on
17 xlabel('t [s]')
18 ylabel('fi [rad]')
19
20 subplot(3,1,2)
21 plot(ty,y(:,2),'r');
22 grid on
23 xlabel('t [s]')
24 ylabel('v [rad/s]')
25
26 subplot(3,1,3)
27 a=-(g/l)*(y(:,1));
28 plot(ty,a,'r');
29 grid on
30 xlabel('t [s]')
31 ylabel('a [rad/s^2]')
```

Rys. 19.7. Realizacja przykładu 19.2



Rys. 19.8. Przykład wykresów położenia, prędkości i przyspieszenia wahadła o długości 0.2 m wychylonego o 1°

20. Obliczenia symboliczne

W środowisku MATLAB, dzięki *Symbolic Math Toolbox*, można wykonywać obliczenia symboliczne, czyli operacje matematyczne na wyrażeniach matematycznych.

W tabeli 20.1 zestawiono wybrane funkcje symboliczne środowiska MATLAB [5]. Operacje algebraiczne na zmiennych symbolicznych wykonywane są w identyczny sposób jak na liczbach, a większość funkcji analitycznych i algebraicznych ma swoje odpowiedniki symboliczne [5].

Na rys. 20.1 i 20.2 przedstawiono porównanie przykładowych obliczeń numerycznych i symbolicznych.

Tabela 20.1. Wybrane funkcje symboliczne środowiska MATLAB [5]

symbol	opis
`sym('x')`	zdefiniowanie zmiennej symbolicznej x
`syms x y z`	zdefiniowanie zmiennych symbolicznych a b c
`simplyfy(x)`	uproszczenie postaci wyrażenia zmiennej symbolicznej x
`char(x)`	zamiana zmiennej symbolicznej x na tekst
`subs(f,'x',a)`	podstawienie wartości a za symbol x w zmiennej symbolicznej f
`finverse(y,x)`	obliczenie wyrażenia funkcji odwrotnej do y(t), czyli x(y)
`compose(f,g)`	obliczenie wyrażenia symbolicznego funkcji złożonej f(g(x))
`solve`	rozwiązanie równań algebraicznych
`dsolve`	rozwiązanie równań różniczkowych
`diff(f)`	obliczenie wyrażenia symbolicznego pochodnej funkcji f
`int(f)`	obliczenie wyrażenia symbolicznego funkcji pierwotnej f
`ezplot(f,[a,b])`	rysowanie wykresu funkcji wyrażenia symbolicznego f w przedziale <a, b>

```
>> x=2+3

x =

   5


>> x=10/3

x =

   3.3333


>> x=sqrt(3)

x =

   1.7321

>> x=5^3

x =

   125
```

```
>> x=sym(2+3)

x =

5


>> x=sym(10/3)

x =

10/3


>> x=sym(sqrt(3))

x =

3^(1/2)

>> x=sym(5^3)

x =

125
```

Rys. 20.1. Porównanie obliczeń numerycznych i symbolicznych w środowisku MATLAB

```
>> y=a^2+b

Undefined function or
variable 'a'.


>> y='a^2+b'

y =

   'a^2+b'
```

```
>> y=sym(a^2+b)

Undefined function or variable
'a'.


>> y=sym('a^2+b')

y =

a^2+b
```

Rys. 20.2. Porównanie obliczeń numerycznych i symbolicznych w środowisku MATLAB

Przykład 20.1

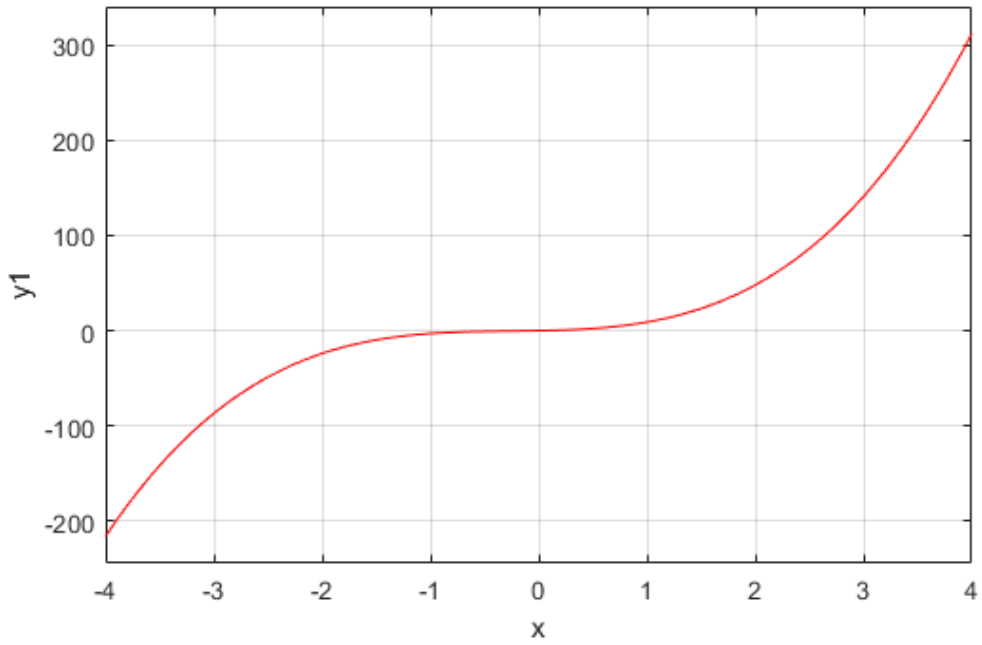
Zapisać w postaci symbolicznej wyrażenie:

$$y = \frac{ax^3+bx^2+cx}{d} \tag{20.1}$$

Obliczyć wyrażenie dla a=8, b=6, c=4 i d=2. Narysować wykres funkcji w przedziale <-4, 4> (rys. 20.3 i 20.4).

```
>> syms a b c d x
>> y=(a*x^3+b*x^2+c*x)/d

y =

(a*x^3 + b*x^2 + c*x)/d

>> y1=subs(y,[a,b,c,d],[8,6,4,2])

y1 =

4*x^3 + 3*x^2 + 2*x

>> ezplot(y1,[-4 4])
>> xlabel('x')
>> ylabel('y1')
>> grid on
```

Rys. 20.3. Realizacja przykładu 20.1



Rys. 20.4. Funkcja wygenerowana dla przykładu 20.1

Przykład 20.2

Do równania z przykładu 20.1 podstawić *a*=12, *b*=8, *c*=4 i *d*=2 i dokonać uproszczenia wyrażania (rys. 20.5).

```
>> syms a b c d x
>> y=(a*x^3+b*x^2+c*x)/d

y =

(a*x^3 + b*x^2 + c*x)/d

>> y1=subs(y,[a,b,c,d],[12,8,4,2])

y1 =

(6*x^3 + 4*x^2 + 2*x)^(1/2)

>> simplify(y1)

ans =

2*x*(3*x^2 + 2*x + 1)
```

Rys. 20.5. Realizacja przykładu 20.2

Przykład 20.3

Rozwiązać symbolicznie układ równań z przykładu 10.2 (rys. 20.6).

```
>> syms x y z
>> [x,y,z]=solve(5*x+3*y-z-4,-3*x+4*y-7,2*x-5*y+3*z+4)

x =

1/20

y =

143/80

z =

129/80
```

Rys. 20.6. Realizacja przykładu 20.3

Przykład 20.4

Obliczyć pochodną pierwszego rzędu funkcji (rys. 20.7)

$$y = 2\frac{x+1}{x-1} \tag{20.2}$$

Pochodna pierwszego rzędu funkcji (20.2) wynosi

$$y' = 2\left(\frac{x+1}{x-1}\right)' = 2\frac{(x+1)'(x-1)-(x+1)(x-1)'}{(x-1)^2} = 2\frac{(x-1)-(x+1)}{(x-1)^2} = \frac{-4}{(x-1)^2}$$

```
>> syms x
>> y=2*((x+1)/(x-1))

y =

(2*(x + 1))/(x - 1)

>> poch=diff(y)

poch =

2/(x - 1) - (2*(x + 1))/(x - 1)^2

>> poch=simplify(poch)

poch =

-4/(x - 1)^2
```

Rys. 20.7. Realizacja przykładu 20.4

Przykład 20.5

Obliczyć pochodną drugiego rzędu funkcji (rys. 20.8)

$$y = \cos(4x) \tag{20.3}$$

```
>> syms x
>> f=cos(4*x)

f =

cos(4*x)

>> poch2=diff(f,2)

poch2 =

-16*cos(4*x)
```

```
>> syms x
>> f=cos(4*x)

f =

cos(4*x)

>> poch2=diff(diff(f))

poch2 =

-16*cos(4*x)
```

Rys. 20.8. Realizacja przykładu 20.5

Przykład 20.6

Obliczyć pochodną drugiego o rzędu funkcji

$$y = A\sin(\omega t + \varphi) \tag{20.4}$$

względem czasu *t* (rys. 20.9).

Pochodna pierwszego rzędu funkcji (20.4) wynosi

$$\dot{y} = -A\omega^2 \sin(\omega t + \varphi) \tag{20.5}$$

```
>> syms A omega t fi
>> y=A*sin(omega*t+fi)

y =

A*sin(fi + omega*t)

>> poch_t=diff(y,t,2)

poch_t =

-A*omega^2*sin(fi + omega*t)
```

Rys. 20.9. Realizacja przykładu 20.6

Przykład 20.7

Obliczyć całkę nieoznaczoną oraz oznaczoną w przedziale <1, 2> funkcji (rys. 20.10)

$$y = \frac{1}{x^2} \tag{20.6}$$

```
>> syms x
>> y=1/x^2

y =

1/x^2

>> cn=int(y)

cn =

-1/x

>> co=int(y,1,2)

co =

1/2
```

Rys. 20.10. Realizacja przykładu 20.7

Przykład 20.8

Rozwiązać symbolicznie zadanie z przykładu 17.2 (rys. 20.11 i 20.12).

```
clear; clc; close all;

n=dsolve('D2y+(g/l)*y=0','y(0)=x','Dy(0)=v');

g=9.81;
l=input('Podaj długość wahadła : ');
alfa=input('Podaj kąt wychylenia początkowego (alfa) : ');
x=(alfa*pi)/180
v=0;

n=subs(n,{'g','l','x','v'},{g,l,x,v'});

ezplot(n,[0,10])
```

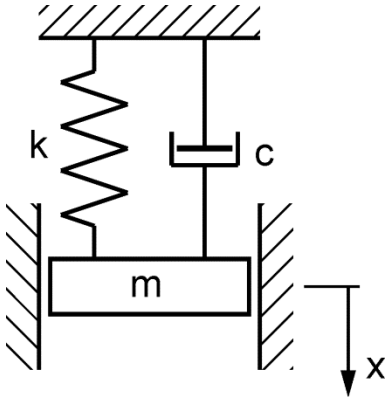
Rys. 20.11. Realizacja przykładu 20.8 (wersja 1)

```
clear; clc; close all;

syms g l x v y(t)

eqn=diff(y,t,2)==-(g/l)*y;
Dy=diff(y,t);
n=dsolve(eqn,[y(0)==x,Dy(0)==v]);

g=9.81;
l=input('Podaj długość wahadał : ');
alfa=input('Podaj kąt wychylenia początkowego (alfa) : ');
x=(alfa*pi)/180;
v=0;

n=subs(n,{'g','l','x','v'},{g,l,x,v'});

ezplot(n,[0,10])
```

Rys. 20.12. Realizacja przykładu 20.8 (wersja 2)

Przykład 20.9

Napisać skrypt umożliwiający analizę ruchu oscylatora harmonicznego z tłumieniem (rys. 20.13÷20.15).



Rys. 20.13. Oscylator harmoniczny z tłumieniem

Równanie różniczkowe ruchu drgającego tłumionego układu o jednym stopniu swobody

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{20.7}$$

można przekształcić do postaci

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = 0 \tag{20.8}$$

gdzie c jest współczynnikiem oporu.

Po podstawieniu

$$\omega_0 = \sqrt{\frac{k}{m}} \tag{20.9}$$

$$\zeta = \frac{c}{2\sqrt{km}} \tag{20.10}$$

otrzymuje się

$$\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_o^2 x = 0 \tag{20.11}$$

gdzie ζ jest współczynnikiem tłumienia wiskotycznego, który można wyznaczyć z zależności

$$\zeta = \frac{\Lambda}{2\pi} \tag{20.12}$$

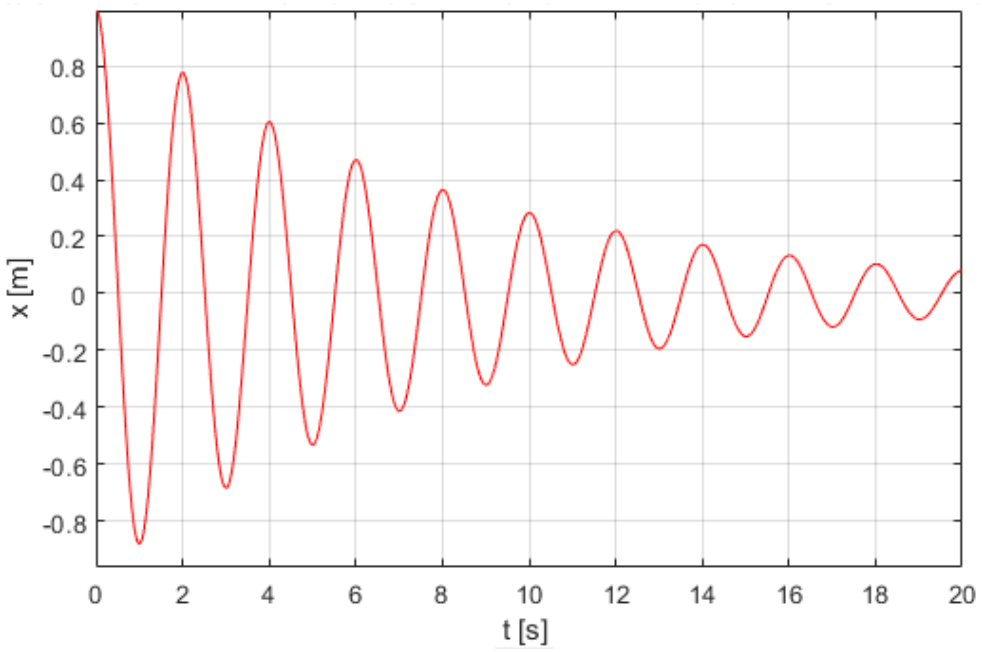
gdzie Λ jest logarytmicznym dekrementem tłumienia.

```
clear; clc; close all;

n=dsolve('D2y+2*dzeta*w*Dy+w^2*y=0','y(0)=x','Dy(0)=v');

dzeta=input('Podaj współczynnik tłumienia wiskotycznego : ');

f=input('Podaj częstotliwość drgań [Hz] : ');
w=2*pi*f;
x=input('Podaj wychylenie początkowe [m] : ');
v=0;

n=subs(n,{'dzeta','w','x','v'},{dzeta,w,x,v});

ezplot(n,[0,20])
xlabel('t [s]')
ylabel('x [m]')
grid on
```

Rys. 20.14. Realizacja przykładu 20.9 (wersja 1)

```
clear;clc; close all;

syms x v y(t) dzeta w

eqn=diff(y,t,2)==-2*dzeta*w*diff(y,t)-w^2*y;
Dy=diff(y,t);
n=dsolve(eqn,[y(0)==x,Dy(0)==v]);

dzeta=input('Podaj współczynnik tłumienia : ');
f=input('Podaj częstotliwość drgań [Hz] : ');
w=2*pi*f;
x=input('Podaj wychylenie początkowe : ');
v=0;

n=subs(n,{'dzeta','w','x','v'},{dzeta,w,x,v});

ezplot(n,[0,20])
xlabel('t [s]')
ylabel('x [m]')
grid on
```

Rys. 20.15. Realizacja przykładu 20.9 (wersja 2)



Rys. 20.16. Drgania tłumione oscylatora harmonicznego dla współczynnika tłumienia wiskotycznego ζ=0.04, częstotliwości drgań własnych f=0.5 Hz i wychyleniu początkowym 1 m

Literatura

1. Sradomski W.: *Matlab. Praktyczny podręcznik modelowania*, Wydawnictwo Helion, Gliwice, 2015.

2. Mrozek B., Mrozek Z.: *Matlab i Simulink. Poradnik użytkownik*a, Wydawnictwo Helion, Gliwice, 2010.

3. Ścięż*o*r T.: *Podstawy programowania w języku Matlab*, Skrypt Politechniki Krakowskiej, Kraków 2015.

4. http://prac.im.pwr.edu.pl/~kajetano/Matlab/Przyklady/Grafika3D.html (z dnia 08.01.2018)

5. Szymczyk E.: *Matlab dla mechaników*, Wydawnictwo WAT, Warszawa 2006.

6. Jankowski R., Lubowiecka I., Witkowski W.: *Matlab – podstawy programowania*, Skrypt Politechniki Gdańskiej, Gdańsk 2001.

7. Koczubiej S., Cichoń C.: *Podstawy mechaniki komputerowej*, wykłady Politechniki Świętokrzyskiej, 2014.

8. System pomocy programu MATLAB, załączony z programem i dostępny na stronach internetowych.

9. Gawroński SW., Kruszewski J., Ostachowicz W. i in.: *Metoda elementów skończonych w dynamice konstrukcji*, Arkady, Warszawa 1984.

Skrypt stanowi pomoc dydaktyczną dla studentów Instytutu Technicznego Państwowej Wyższej Szkoły Zawodowej im. Jana Grodka w Sanoku do realizacji zajęć z przedmiotów Komputerowe Systemy Obliczeniowe oraz Komputerowe Systemy Pomiarów, prowadzonych na studiach inżynierskich pierwszego stopnia na kierunku Mechanika i Budowa Maszyn.

ISBN 978-83-61802-35-8