Jakub Możaryn

Programowanie w środowisku Matlab

Materiały dydaktyczne, Ośrodek Kształcenia Na Odległość – OKNO

2009-08-12

Spis Treści

Zawartość

1. Wstęp

Nazwa interaktywnego środowiska programistycznego Matlab pochodzi od angielskich słów MATrix LABoratory, gdyż początkowo program ten był przeznaczony do numerycznych obliczeń macierzowych występujących podczas rozwiązywania problemów inżynierskich. Obecnie zakres zastosowania jest bardzo szeroki m.in. do przetwarzania sygnałów, przetwarzania obrazów, projektowania układów sterowania, przetwarzania danych pomiarowych, telekomunikacji, modelowaniu i analizie finansowej lub biologii obliczeniowej.
Środowisko charakteryzuje się własnym językiem programowania, co umożliwia pisanie w pełni funkcjonalnych programów działających w tym środowisku. Oprócz podstawowych operacji macierzowych cechuje je także duża liczba wyspecjalizowanych funkcji zebranych w dodatkowych bibliotekach oraz duże możliwości rozbudowy przez użytkownika poprzez pisanie własnych funkcji.
Ponadto środowisko pozwala na łatwą i estetyczną wizualizację uzyskanych wyników, co znacznie ułatwia ich prezentacje i analizę.

1.1. Historia

Początki środowiska Matlab sięgają lat 70-te XX wieku, gdy w USA na zlecenie National Science Foundation powstały biblioteki języka Fortran do obliczeń macierzowych: Linpack i Eispack.
Jeden z autorów tych bibliotek, Cleve Moler napisał w 1980 r. program, który umożliwiał korzystanie z tych bibliotek bez potrzeby programowania w Fortranie. Program ten był napisany w formie prostego interaktywnego języka poleceń i rozprowadzany na zasadach public domain. Był on pierwowzorem Matlaba i służył jako pomoc dydaktyczna podczas prowadzenia zajęć ze studentami.
W 1983 C. Moller oraz S. Bangert i J. Little (inżynier z Uniwersytetu Stanford) postanowili rozwinąć powyższy projekt. Zastąpili Fortran językiem C i dodali zintegrowaną grafikę. Założyli oni firmę The MathWorks Inc., która do dziś zajmuje się rozwojem i sprzedażą pakietu Matlab. W 1985 roku pojawiła się pierwsza komercyjna wersja programu.
Obecnie numer najnowszej wersji tego programu to 7.8.

1.2. Podstawowe możliwości środowiska Matlab

Serce pakietu Matlab stanowi interpreter języka umożliwiający implementacje algorytmów numerycznych oraz biblioteki podstawowych działań na macierzach (odwracanie, dodawanie/odejmowanie, wartości własne).
Oprócz pracy na danych, Matlab umożliwia także przeprowadzanie obliczeń symbolicznych (na wzorach)

Podstawowe możliwości środowiska Matlab:

- Obliczenia numeryczne
- Opracowywanie algorytmów
- Przeprowadzanie obliczeń symbolicznych (na wzorach)
- Akwizycja danych
- Modelowanie i symulacja
- Analiza danych
- Wizualizacja
- Budowanie aplikacji wraz z projektowaniem interfejsów graficznych

Cechy szczególne środowiska MATLAB:

- Interfejs graficzny
- Środowisko użytkownika
- Szybka implementacja własnych algorytmów macierzowych
- Optymalizacja obliczeniowa na działania macierzowe
- Dodatkowe wyspecjalizowane pakiety - tzw. Toolbox'y
- Możliwość samodzielnego rozszerzania poprzez dopisywanie nowych funkcji (pakietów)

Standardowe funkcje zawarte w Matlabie, to m.in.

- Algebra liniowa i operacje na macierzach
- Wielomiany i interpolacja
- Filtrowanie i analiza Fouriera
- Analiza danych i statystyka
- Optymalizacja i numeryczne rozwiązywanie równań różniczkowych

1.3. Charakterystyka składni

Pakiet Matlab wykorzystuje język programowania wysokiego poziomu, o składni wzorowanej na języku C. Pozwala on na używanie typowych instrukcji sterujących, pętli, funkcji i struktur, oraz umożliwia pisanie programów zorientowanych obiektowo.
Zaletą środowiska Matlab jest znacznie szybsze pisanie i testowanie aplikacji niż w przypadku innych języków wysokiego poziomu. Nie jest wymagana wykonywanie czasochłonnych zadań jak np. deklarowanie zmiennych, określanie typów danych, alokacja pamięci. Oczywiście możliwe jest stosowanie także bardziej złożonych instrukcji, np. operatorów bitowych. Zaletą nowszych wersji Matlaba jest także możliwość programowania obiektowego.
Matlab pozwala na wykonywanie komend lub grup komend w trakcie pisania, bez kompilacji i przypisywania co pozwala na szybkie poszukiwanie optymalnego rozwiązania.

1.4. Wizualizacja wyników w Matlabie

Matlab ma zaimplementowane funkcje pozwalające na wizualizację macierzy o dwóch lub trzech wymiarach. Może być to wykorzystane do zobrazowania i zrozumienia często złożonych danych wielowymiarowych. Matlab umożliwia określenie własności wykresów np. kąt, perspektywa, efekty świetlne, przeźroczystość.
Ponadto udostępnione są dodatkowe funkcje np. eksport danych do popularnych formatów graficznych, zaznaczanie fragmentów i rysowanie kształtów geometrycznych.
W Matlabie istnieje możliwość projektowania okienkowych interfejsów graficznych do napisanych aplikacji.

1.5. Dodatkowe biblioteki

Biblioteki dodatkowe (z ang. toolboxes) to zbiór dodatkowych funkcji (m-plików) do rozwiązywania specjalistycznych problemów z określonych dziedzin (automatyka, elektronika, telekomunikacja, matematyka etc.). Biblioteki rozszerzają możliwości Matlaba i pisane są zwykle przez oddzielnych producentów oprogramowania. Należy jednak zwrócić uwagę, że każda z dodatkowych bibliotek stanowi tylko uzupełnienie Matlaba i jest sprzedawana oddzielnie od samego środowiska obliczeniowego.

Spośród dużej liczby istniejących bibliotek wymienić można m.in.

- Simulink - pakiet służący do modelowania, symulacji i analizy układów dynamicznych. Simulink dostarcza także graficzny interfejs użytkownika umożliwiający konstruowanie modeli w postaci diagramów blokowych.
- Image Processing Toolbox - programowe narzędzia do przetwarzania obrazów.
- Fuzzy Logic Toolbox - środowisko do projektowania i diagnostyki inteligentnych układów sterowania wykorzystujących metody logiki rozmytej i uczenie adaptacyjne.
- Symbolic Math Toolbox - zestaw funkcji do obliczeń symbolicznych - rozszerza możliwości Matlaba o możliwość wykonywania obliczeń symbolicznych.
- Chemometrix Toolbox - przeznaczony do opracowywania danych chemicznych.
- Financial Toolbox - przeznaczony do analiz i obliczeń finansowych (planowanie stałych przychodów, badanie wydajności obligacji, kalkulacja przepływu gotówki, obliczanie stóp procentowych etc.).
- Mapping Toolbox - przeznaczony do analizy informacji geograficznych i wyświetlania map, z możliwością dostępu do zewnętrznych źródeł geograficznych.
- Partial Differential Equation Toolbox - zestaw funkcji do numerycznego rozwiązywania równań różniczkowych cząstkowych metodą elementów skończonych.

1.6. Publikacja wyników

Matlab udostępnia wiele możliwości dokumentowania i wykorzystywania aplikacji.
Po pierwsze można zintegrować kod z innymi językami programowania i aplikacjami, co pozwala wykorzystać zaprojektowane algorytmy w samodzielnym oprogramowaniu i aplikacjach. Matlab dostarczany jest z funkcjami

pozwalającymi na integrację z językami: C/C++, Fortran, Java i wywoływanie funkcji, które znajdują się w bibliotekach napisanych w ww. językach..
Matlab pozwala także na eksportowanie wyników w postaci wykresów i raportów, pozwalając na współpracę z takimi pakietami jak Microsoft Word, Microsoft Power Point. Korzystając z edytora Matlab'a można automatycznie publikować dane i kod w popularnych formatach: HTML, Word oraz LaTEX.

1.7. Wybór narzędzi do symulacji i szybkiego prototypowania

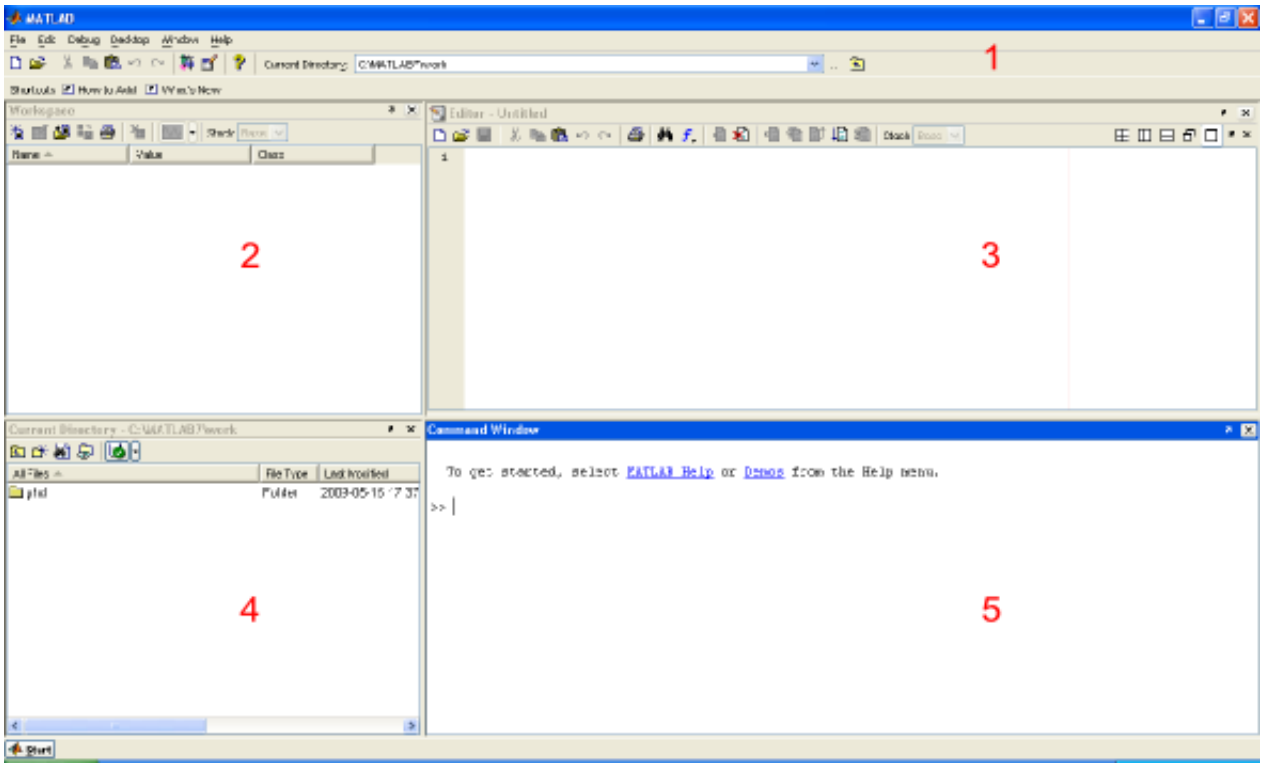
Należy podkreślić, że środowisko Matlab jest stosunkowo kosztowną aplikację o charakterze komercyjnym.
Istotnym elementem, który powinien być brany pod uwagę podczas wyboru narzędzia modelowania, jest cena oprogramowania służącego do symulacji. Decydując się na rozwiązanie komercyjne należy liczyć się ze stosunkowo dużym wydatkiem. Wybierając narzędzie do symulacji należy więc zwykle rozważyć, jakie problemy będą rozwiązywane z jego pomocą. Może to wymagać konsultacji z ekspertami i określenia dokładnej specyfikacji oprogramowania.
Cena programu do symulacji może być bardzo różna. W przypadku komercyjnych narzędzi może wahać się od 25 dol. do ponad 20 tys. dol. Co więcej, wielu elementów nie można otrzymać w wersjach podstawowych, więc trzeba za nie dopłacać. Przykładowo cena pakietu Matlab-Simulink-RealTime Workshop firmy Mathworks do zastosowań komercyjnych wynosi 12 tys. dol, a licencja studencka 89 dol. Ponadto zakup każdej dodatkowej biblioteki funkcji, czyli tzw. toolboxa, to koszt w granicach 1000 do 1500 dolarów. Dlatego na korzystanie z tego typu oprogramowania stać tylko niektóre firmy lub ośrodki naukowe. Zaletą aplikacji komercyjnych jest ciągła współpraca dostawcy oprogramowania z klientem, która polega na szkoleniach, pomocy przy rozwiązywaniu problemów i ciągłym uaktualnianiu produktu.
Istnieje szereg rozwiązań pozwalających zminimalizować koszty przeprowadzania symulacji. Powstało wiele aplikacji, które są ogólnodostępnymi odpowiednikami komercyjnego oprogramowania. Zwykle rozwijanie i stosowane są w środowisku akademickim, jednak coraz częściej mają wsparcie od różnych firm i organizacji zajmujących się badaniami i rozwojem nowych technologii.
Obecnie środowisko do symulacji i szybkiego prototypowania składa się zazwyczaj z trzech podstawowych elementów. Pierwszym z nich jest zintegrowany system służący do obliczeń numerycznych oraz graficznej wizualizacji wyników - np. Matlab, Octave, Scilab. Dodatkowo w system wbudowany jest kompilator języka wysokiego poziomu, zwykle zaprojektowanego specjalnie dla środowiska symulacyjnego. Ułatwieniem w pracy jest szereg zaimplementowanych procedur numerycznych, takich jak operacje macierzowe lub rozwiązywanie układów równań różniczkowych.
Kolejnym elementem jest system do graficznego projektowania i symulacji układów dynamicznych – zarówno z czasem ciągłym, jak i dyskretnym. Zwykle model rzeczywistego układu buduje się w postaci schematów blokowych. W bardziej rozbudowanych systemach dokonać można wizualizacji i symulacji układów na podstawie ich trójwymiarowych modeli (np. w Vissim, 20-sim). Twórcy oprogramowania zwykle udostępniają bogate biblioteki poszczególnych elementów, począwszy od prostych bloków funkcyjnych po złożone modele obiektów rzeczywistych np. napędów i sterowników popularnych firm.
Ostatnim elementem jest system do generowania i uruchamiania na testowej platformie sprzętowej algorytmów zaprojektowanych w środowisku do symulacji. Przykładem mogą być środowiska Real Time Workshop i SynDEx. Zwykle generuje on, na podstawie projektu symulowanego układu, kod w języku C, który może być bezpośrednio zaimplementowany podczas szybkiego prototypowania i testowania aplikacji. Dodatkową zaletą jest wsparcie dla urządzeń, w których obliczenia przeprowadzane są w czasie rzeczywistym.
Współpraca tych trzech systemów pozwala na łatwe projektowanie złożonych modeli rzeczywistych układów oraz ich symulację i ewentualną implementację w prototypach HIL (ang. Hardware in The Loop). Należy też podkreślić, że drogi, komercyjny zestaw np. Matlab-Simulink-Real Time Workshop można w wielu przypadkach zastąpić znacznie tańszym rozwiązaniem Scilab-Scicos-HIL/RTAI/SynDEx.

2. Podstawy pracy w środowisku Matlab

2.2. Interfejs programisty – Matlab

Interfejs użytkownika środowiska Matlab składa się z okien, które można otwierać lub zamykać korzystając z funkcji Window w menu głównym. Typowe okna przedstawiono na rysunku 2.1. Są to (numeracja zgodnie z rysunkiem):

1. *MainMenu* – Podstawowe menu do zarządzania środowiskiem Matlab
2. *Workspace* - informacje o zmiennych w przestrzeni roboczej, które są aktualnie dostępne.
3. *Editor* – edytor skryptów
4. *Current Directory* - okno katalogu roboczego, z wyświetleniem nazw plików znajdujących się w tym katalogu
5. *Command Window* – okno poleceń
6. *Command History* – okno poprzednich poleceń (nie widoczne na rysunku). Znajduje się w nim historia poleceń wydawanych w oknie poleceń.



Rysunek 2.1. Standardowy interfejs użytkownika w systemie Matlab.

2.1. Przygotowanie środowiska Matlab do pracy

Na początku pracy ze środowiskiem Matlab należy określić aktualny katalog gdzie znajdują się pliki z którym aktualnie pracujemy. Standardowo, po uruchomieniu aplikacji, aktualnym katalogiem jest '(miejsce instalacji Matlaba)/MATLAB/work'.

Po uruchomieniu Matlaba, można zmienić domyślny katalog do pracy na katalog gdzie znajdują się pliki, z którymi aktualnie chcemy pracować. Wystarczy zmienić nazwę katalogu wprowadzając ją w pole edycyjne Current Directory (rysunek 2.2).

Rysunek 2.2. Główne menu Matlaba, (1) strzałka wskazuje na pole edycyjne zmiany aktualnego katalogu

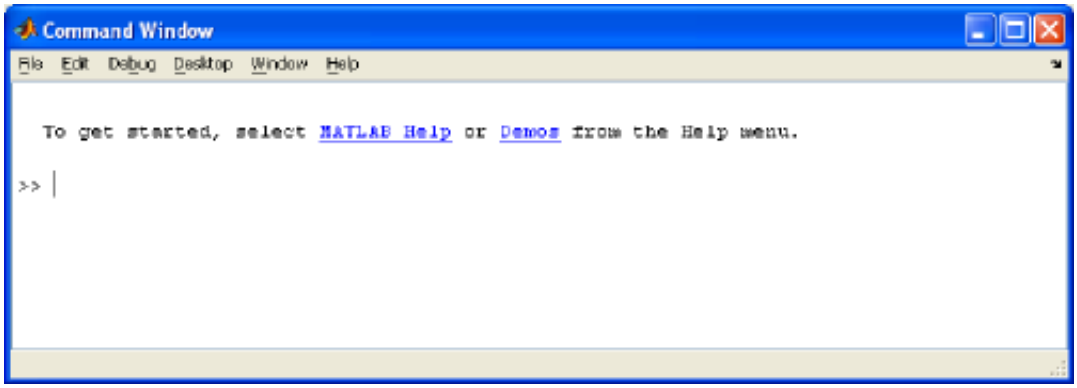
W środowisku Matlab możliwe jest pracowanie w wielu katalogach. W tym celu korzysta się z opcji z menu głównego *File > Set Path… >Add Folder*… . Należy jednak zwrócić wtedy uwagę na nazwy funkcji, które powinny być charakterystyczne i nie pojawiać się w kilku katalogach. Jeśli wywoływana funkcja nie istnieje Matalb zwróci informację o błędzie 'undefined function'.

3. Podstawy programowania w środowisku Matlab

3.1. Pisanie kodu - okno poleceń

Interesującą, bardzo często wykorzystywaną, własnością Matlaba jest dynamiczne kompilowanie kolejnych linijek kodu, wpisywanych w oknie poleceń (rysunek 3.1.). Nie ma więc potrzeby pisania kodu w postaci skryptu i kompilowania go do postaci pliku wykonawczego a następnie uruchomiania go. Każda prawidłowo skonstruowana komenda wpisana w oknie poleceń jest automatycznie kompilowana przez środowisko Matlab od razu po wprowadzeniu. Ułatwia to znacznie pisanie i testowanie kodu, oraz korzystanie z Matlaba jak z zaawansowanego kalkulatora.

Podczas pisania instrukcji w Matlabie, jeśli istnieje wynik działania instrukcji to wyświetli się on i zostanie zapisany do przestrzeni roboczej, jeśli na jej końcu linijki nie będzie żadnego znaku. Postawienie średnika spowoduje, że utworzona zmienna zostanie zapisana do przestrzeni roboczej.



Rysunek 3.1. Okno poleceń

3.2. Zmienne

Podstawowym typem danych w Matlabie jest macierz prostokątna, o *n* rzędach i *m* kolumnach (*n* może być różne od *m*, lub równe *m*) o elementach zespolonych typu rzeczywistego. Wszystkie zmienne interpretowane są przez środowisko Matlab jako macierze. W szczególnych przypadkach macierz 1x1 jest skalarem, natomiast macierz o jednym rzędzie lub jednej kolumnie jest interpretowana jako wektor.

Macierze mogą być podawane

a) w postaci jawnej – kolejne rzędy macierzy są oddzielane średnikiem, kolejne elementy w rzędzie spacją lun przecinkiem. Należy zwrócić uwagę, żeby liczba elementów w każdym rzędzie powinna być równa.

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

b) mogą być wynikiem działania lub funkcji

```
>>b=2*A;
>>b=sin(A);
```

c) mogą być wczytane ze źródła zewnętrznego (np. pliku tekstowego)

```
>>load plik.mat A;
```

3.3. Operatory macierzowe

W Matlabie istnieje zbiór podstawowych operacji macierzowych. Zwykle wykorzystywane są operatory macierzowe znajdujące się w **Tabeli 3.1**. Podczas korzystania z operatorów, należy zwrócić uwagę na rozmiar macierzy.

Ponieważ przy każdym operatorze podany jest przykład, należy wcześniej zdefiniować macierz A i macierz B w nim występujące. Większość instrukcji podanych w tym i następnych rozdziałach działa na 1 lub 2 macierzach. W niektórych przypadkach występują 3 macierze.

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

Tabela 3.1. Operatory macierzowe.

Oznaczenie	Opis	Przykład
+	dodawanie	>>C=A+B
−	odejmowanie	>>C=A-B
*	mnożenie	>>C=A*B
/	dzielenie prawostronne (wynik {x: xA=B})	>>C=A/B
./	dzielenie prawostronne wyraz po wyrazie	>>C=A./B
\	dzielenie lewostronne (wynik {x: Ax=B})	>>C=A\B
.\	dzielenie lewostronne wyraz po wyrazie	>>C=A.\B
'	transpozycja macierzy	>>C=A'
^	potęgowanie macierzy	>>C=B^3
.^	potęgowanie macierzy wyraz po wyrazie	>>C=B.^3
=	przypisane	>>C=[1 2 3;4, 5, 6]
[]	tworzenie macierzy	jw.
()	i. odwołanie do elementu macierzy ii. argumenty funkcji.	>>C(1,1) >>inv(A)
,	separator elementów wiersza, separator indeksów.	Jw.
.	separator dziesiętny	>>D=1.1
'	ogranicznik tekstu	>>S='jakis tekst';
;	i. separator kolumn ii. koniec linii	>>C=A.\B
:	operator zakresu	>> C(1,2:3)

3.4. Operatory relacji

W Matlabie występują operatory relacji, służące do porównania ze sobą macierzy. Wynikiem takiego działania jest wartość logiczna prawda (`true`) lub fałsz (`false`). Wykonywane są one element po elemencie porównywanych macierzy, dlatego dotyczą macierzy o identycznych wymiarach. Operatory relacji zostały zebrane w tabeli 3.2.

Tabela 3.2. Operatory relacji.

Oznaczenie	Opis	Przykład
==	równe	>>A==B
~=	nierówne	>>A~=B
>	większe	>>A>B
<	mniejsze	>>A<B
>=	większe równe	>>A>=B
<=	mniejsze równe	>>A<=B

3.5. Operatory logiczne

W Matlabie występują także operatory logiczne, stosowane do porównywania wartości w macierzach element po elemencie lub wartości logicznych uzyskanych np. w wyniku porównania macierzy z wykorzystaniem operatorów relacji. Zostały one zebrane w tabeli 3.3. Każdy element macierzy o wartości 0 ma logiczną wartość `false`, natomiast o innej wartości ma logiczną wartość `true`.

Tabela 3.3. Operatory logiczne

Oznaczenie	Opis	Przykład
`&`	iloczyn logiczny (and), wykonywany element po elemencie	`>>A&B`
`|`	suma logiczna (or), wykonywana element po elemencie	`>>A|B`
`~`	zaprzeczenie logiczne (not), wykonywane element po elemencie	`>>~(A)`
`xor`	alternatywa wykluczająca	`>>xor(A,B)`
`any`	sprawdza, czy którykolwiek z elementów macierzy nie jest zerem i jeśli taki element istnieje zwraca wartość `true`	`>>B = any(A)`
`all`	sprawdza, czy którykolwiek z elementów macierzy jest zerem i jeśli taki element istnieje zwraca wartość `true`	`>>B = all(A)`
`&&`	iloczyn logiczny (and), wykonywany na wyrażeniach zwracających wartość logiczną	`>>(A==B)&&(A==C)`
`||`	iloczyn logiczny (or), wykonywany na wyrażeniach zwracających wartość logiczną	`>>(A==B)||(A==C)`

3.6. Zmienne i stałe o specjalnym przeznaczeniu

W środowisku Matlab zostały zdefiniowane zmienne i stałe o specjalnym przeznaczeniu. Najważniejsze z nich zebrano w tabeli 3.4. Należy starać się nie nazywać swoich zmiennych jedną z tych nazw, natomiast można je wykorzystywać bez wcześniejszego zdefiniowania w pisanych instrukcjach.

Tabela 3.4. Zmienne i stałe o specjalnym przeznaczeniu

Oznaczenie	Opis	Przykład
`ans`	zmienna robocza przechowująca wynik ostatniego polecenia	
`inf`	nieskończoność (może występować np. jako wynik działania)	
`NaN`	wartość nieokreślona (może występować np. jako wynik działania)	
`i lub j`	jednostka urojona	`>> j` `ans = 0 + 1.0000i`
`eps`	eps = 2e−52	`>>eps` `ans = 2.2204e-016`
`pi`	liczba π =3.14...	`>> pi` `ans = 3.1416`
`realmax`	największa dostępna liczba rzeczywista	`>>realmax` `ans = 1.7977e+308`
`realmin`	najmniejsza dostępna dodatnia liczba rzeczywista	`>>realmin` `ans = 2.2251e-308`

3.7. Instrukcje warunkowe

Język programowania środowiska Matlab, podobnie jak większość znanych języków programowania, posiada podstawowe instrukcje sterujące, służące do kontrolowania przepływu informacji w programie.

3.7.1. Instrukcje warunkowe – `if, elseif, else`

składnia

```
if wyrażenie_logiczne_bloku_if
    instrukcje_bloku_if
elseif wyrażenie_logiczne_bloku_elseif
    instrukcje_bloku_elseif
else
   instrukcje_bloku_else
end
```

Instrukcja `if` (*jeżeli...*) sprawdza wyrażenie logiczne i wywołuje grupę instrukcji bazując na obliczonej wartości wyrażenia logicznego. Wyrażenia `else` (*w przeciwnym przypadku...*) i `elseif` (*w przeciwnym przypadku jeżeli...*) wprowadzają dodatkowe warunki do instrukcji warunkowej `if`.

- Wyrażenie `else` nie posiada warunku i związane z nim instrukcje są wywoływane jeśli wyrażenie logiczne poprzedzającego go bloku `if` lub `elseif` zwraca fałsz `(false (0))`
- Wyrażenie `elseif` posiada logiczny warunek i wywoływane jest jeśli wyrażenie logiczne poprzedzającego go bloku `if` lub `elseif` zwraca fałsz `(false (0))`. Instrukcje znajdujące się w bloku `elseif` są wywoływane jeśli spełniony jest warunek logiczny związany z tym blokiem. Wewnątrz jednego bloku `if` można wywołać wiele bloków `elseif`.
- Na końcu grupy `if ... elseif ... else` musi znaleźć się instrukcja `end` kończąca tę grupę.

Przykład 3.1. `(if, elseif, else)`:

```
r=2;
k=1;
if r == k
        A(r,k) = 2;
elseif (abs(r-k) == 1)
        A(r,k) = -1;
else
        A(r,k) = 0;
end;
```

3.7.2. Instrukcje warunkowe - `switch, case, otherwise`

składnia

```
switch wyrażenie %(skalar lub łańcuch znaków)
   case wartość1
      instrukcje       % Wywoływane, jeśli wyrażenie przyjmuje wartość1
   case wartość2
      instrukcje       % Wywoływane, jeśli wyrażenie przyjmuje wartość2
    .
    .
    .
   otherwise
      statements       % Wywoływane, jeśli wyrażenie nie przyjmuje żadnej z
end                    % obsługiwanych wartości
end
```

Instrukcja `switch` wywołuje określone instrukcje korzystając z obliczonej wartości zmiennej lub wyrażenia. Cała pętla składa się z:

- Instrukcji `switch` (*wybierz wyrażenie...*), po której znajduje się wyrażenie do obliczenia/zmienna.
- Dowolnej liczby zgrupowanych bloków `case` (*w przypadku gdy wyrażenie ma wartość..*). Bloki zawierają słowo `case`, po którym w tej samej linijce znajduje się określona wartość, którą może przyjąć wyrażenie do obliczenia/zmiennej występujące po instrukcji `switch`. Kolejne linijki zawierają blok instrukcji dla określonej wartości obliczanego wyrażenia/zmiennej. Wykonywanie instrukcji w bloku `case` kończy się w przypadku, gdy wystąpi kolejne wywołanie `case`, lub wywołanie bloku `otherwise`. W grupie `switch` wywoływany jest tylko jeden blok `case`, dla którego pasuje wartość wyrażenia.
- Jednego opcjonalnego bloku `otherwise` (*w pozostałych przypadkach...*), znajdującego się na końcu. Blok zawiera słowo `otherwise`, po którym w kolejnych linijkach znajdują się instrukcje które zostaną wykonane, jeżeli obliczona wartość wyrażenia/zmiennej nie jest obsługiwana przez poprzednie bloki `case`. Wywołanie instrukcji bloku `otherwise` kończy grupę `switch`.
- Na końcu grupy `switch` musi się znaleźć instrukcja `end`, kończąca tę grupę.

Przykład 3.2. `(switch, case, otherwise)`

```
input_num=10;
switch input_num
    case -1
        disp('ujemna jedynka');
    case 0
        disp('zero');
    case 1
        disp('dodatnia jedynka');
    otherwise
         disp('inna wartość');
end
```

3.8. Pętle

W środowisku Matlab umożliwiono zastosowanie pętli, podobnie jak w przypadku innych znanych języków programowania. Pętla jest wykorzystywana do powtarzania określoną liczbę razy instrukcji w niej zawartych. Pętle stanowią istotny element programowania. Środowisko Matlab pozwala na stosowanie dwóch rodzajów pętli: `for` oraz `while`. Możliwe jest zagnieżdżanie kilku pętli różnego rodzaju..

3.8.1. Pętla - `for`

składnia

```
for zmienna = wartość_poczatkowa:krok:wartość_końcowa
    instrukcje
end
```

Pętla `for` (dla...) jest to pętla sterowana zmienną. Grupa instrukcji w niej zawartych wywoływana jest określoną liczbę razy. Zmienna sterująca przyjmuje wartości od *wartości_poczatkowej* do *wartości_końcowej* z określonym *krokiem*. Zmienna sterująca powinna być typu wyliczeniowego (liczba całkowita, litera). Blok instrukcji musi kończyć się instrukcją końca pętli `end`.

Przykład 3.3. `(for)`:

```
N=10;
for r = 1:N,
             for c = 1:N,
       A(r,c) = 1/(I+J-1);
             end
end
```

3.8.2. Pętla - `while`

składnia

```
while warunek
    instrukcje
end
```

Pętla `while` (dopóki...) jest to pętla sterowana warunkiem. Grupa instrukcji w niej zawartych wywoływana jest do momentu aż *warunek* sterujący pętlą jest spełniony. Ponieważ jest on zmienną logiczną, tak więc pętla kończy pracę jeśli *warunek* zwraca wartość `false (0)`. Należy zwrócić szczególną uwagę na zmienne występujące w warunku. Najlepiej żeby były one modyfikowane wewnątrz pętli `while`. W szczególnym przypadku zmienna występująca w warunku może nigdy nie spowodować zmiany warunku na `false`, a co za tym idzie pętla `while` będzie się wykonywać nieskończoną ilość razy.

Przykład 3.4. `(while)`:

```
while a<0
     a=a+1;
end
```

3.8.3. Instrukcje sterujące wewnątrz pętli

W szczególnych przypadkach może być przydatne wykorzystanie instrukcji sterujących wewnątrz pętli. Matlab dopuszcza stosowanie instrukcji `continue` i instrukcji `break`.

- Instrukcja `continue` wymusza przejście do kolejnego kroku iteracji pętli `for` lub `while`, zaś instrukcje występujące do końca danej grupy są pomijane. W przypadku zagnieżdżonych pętli, instrukcja `continue` dotyczy pętli, w której występuje.
- Instrukcja break wymusza bezwarunkowe wyjście z pętli `for` lub `while`, bez względu na warunek zakończenia pętli. W przypadku wykonania instrukcji `break`, wykonywana jest kolejna instrukcja znajdująca się poza pętla, która ją zawierała. W zagnieżdżonych pętlach instrukcja `break` powoduje wyjścia z pętli, w której się znajduje.

3.9. Formatowanie komunikatów

W środowisku Matlab stosowana jest notacja dziesiętna, z kropka dziesiętną. W przypadku notacji naukowej litera `e` określa wykładnik potęgi 10. Wszystkie liczby są zachowywane w tzw. długim formacie (ang. long). Na komputerach wykorzystujących 32-bitowy procesor odpowiada to dokładności do 16 cyfr znaczących po kropce dziesiętnej – zakres od 10^{-308} do 10^{+308}.
W przypadku wyświetlania wyników, przydaje się odpowiednie formatowanie uzyskanych wartości. Ułatwia to ich czytelność i analizę danych. W Matlabie do zmiany formatowania wyświetlanych wartości stosuje się instrukcję `format`. Odpowiednie typy formatowania wartości liczbowych wraz z ich opisem zebrano w tabeli 3.5.

Składnia

```
format typ_formatowania
format('typ_formatowania')
```

Tabela 3.5. Format dla danych numerycznych

Oznaczenie (`typ_formatowania`**)**	**Opis**
`short`	5 cyfr, reprezentacja stałoprzecinkowa
`long`	15 cyfr, reprezentacja stałoprzecinkowa
`short e`	5 cyfr, reprezentacja zmiennoprzecinkowa
`long e`	15 cyfr, reprezentacja zmiennoprzecinkowa
`short g`	maksymalnie 5 cyfr znaczących
`long g`	maksymalnie 15 cyfr znaczących
`hex`	liczba w układzie szesnastkowym
`bank`	2 cyfry dziesiętne
`rat`	postać ułamka zwykłego (przybliżona)

Często może występować potrzeba dokonania opisu danych na ekranie. W Matlabie można wykorzystać różne polecenia wyświetlające opisy.
Najprostszym sposobem jest wykorzystanie funkcji `disp(X)`. Służy ona do wyświetlania łańcucha znaków lub wartości w danej macierzy. W tym przypadku X może być odpowiednim komunikatem

Przykład 3.5 (`disp`**)**

```
disp('przykładowy opis')
```

Bardziej złożone opisy można przygotować stosując funkcję `sprintf()` służącą do formatowania łańcuchów znaków. Jest ona kopią analogicznej funkcji występującej w języku C. Pozwala ona na łączenia łańcuchów znaków oraz zmiennych zapisanych w różnym formacie. Jest to funkcja o zaawansowanych możliwościach, których dokładny opis można znaleźć w pliku pomocy.

Przykład 3.6 (`sprintf`)

```
komunikat=sprintf('The array is %dx%d.',2,3);
```

3.10. Funkcje specjalizowane

Matlab posiada zbiór wbudowanych funkcji stosowanych do przetwarzania wartości skalarnych lub macierzy. Ich zastosowanie znacznie ułatwia szybkie projektowanie aplikacji. Zestawienie tematyczne tych funkcji można otrzymać, wpisując w linii komend instrukcje zebrane w tabeli 3.6.

Tabela 3.6. Komendy do wypisania zestawień tematycznych funkcji

Komenda	Opis
`help elfun`	Podstawowe funkcje matematyczne.
`help elmat`	Podstawowe funkcje macierzowe.
`help matfun`	Funkcje macierzowe, algebra liniowa.
`help specfun`	Specjalistyczne funkcje matematyczne.
`help polyfun`	Funkcje interpolacyjne i wielomianowe.
`help datafun`	Funkcje analizy danych i analizy fourierowskiej.

3.10.1. Funkcje skalarne

Wybrane funkcje Matlaba wykonują działania na wartościach skalarnych. W przypadku zastosowania ich do macierzy, wykonują zadana operację na każdym z elementów macierzy. Wybrane funkcje skalarne zebrano w tabeli 3.7.

Tabela 3.7. Wybrane funkcje skalarne

Funkcja	Opis
`sin(X)`	Sinus: Zwraca macierz o elementach równych sin(X(i,j)).
`asin(X)`	Arcus sinus: Zwraca macierz o elementach równych asin(X(i,j)).
`exp(X)`	Zwraca macierz o elementach równych $e^{X(i,j)}$.
`abs(X)`	Zwraca macierz modułów elementów macierzy X.
`round(X)`	Zaokrągla elementy macierzy X do najbliższej liczby całkowitej.
`cos(X)`	Cosinus: Zwraca macierz o elementach równych cos(X(i,j)).
`acos(X)`	Arcus cosinus: Zwraca macierz o elementach równych acos(X(i,j)).
`log(X)`	Logarytm naturalny elementów macierzy X.
`sqrt(X)`	Zwraca macierz pierwiastków kwadratowych poszczególnych elementów macierzy X.
`floor(X)`	Zaokrągla elementy macierzy X w kierunku -∞.
`tan(X)`	Tangens: Zwraca macierz o elementach równych tan(X(i,j)).
`atan(X)`	Arcus tangens: Zwraca macierz o elementach równych atan(X(i,j)).
`rem(X,Y)`	Oblicza resztę z dzielenia odpowiadających sobie elementów macierzy X i Y.
`sign(X)`	Tworzy macierz o wymiarach identycznych z X, o elementach równych 1 – jeśli element był dodatni, -1 – jeśli element był ujemny, 0 – gdy był równy 0 .
`ceil(X)`	Zaokrągla elementy macierzy X w kierunku +∞.

3.10.2. Funkcje wektorowe

Wybrane funkcje Matlaba wykonują działania na wektorach. W przypadku zastosowania ich do macierzy, wykonują zadana operację na każdej z kolumn macierzy i zwracają w wyniku wektor o liczbie elementów równej liczbie kolumn. Najpopularniejsze funkcje wektorowe zebrano w tabeli 3.8.

Tabela 3.8. Wybrane funkcje wektorowe

Funkcja	Opis
`max(X)`	Jeśli `X` jest wektorem, zwraca największy element. Jeśli `X` jest macierzą zwraca wektor największych elementów obliczonych dla poszczególnych kolumn.
`sum(X)`	Jeśli `X` jest wektorem, zwraca sumę wszystkich elementów. Jeśli `X` jest macierzą zwraca wektor sum elementów obliczonych dla poszczególnych kolumn.
`median(X)`	Jeśli `X` jest wektorem, zwraca medianę wszystkich elementów. Jeśli `X` jest macierzą zwraca wektor median elementów obliczonych dla poszczególnych kolumn.
`min(X)`	Jeśli `X` jest wektorem, zwraca najmniejszy element. Jeśli `X` jest macierzą zwraca wektor najmniejszych elementów obliczonych dla poszczególnych kolumn.
`prod(X)`	Jeśli `X` jest wektorem, zwraca wartość średnią wszystkich elementów. Jeśli `X` jest macierzą zwraca wektor wartości średnich elementów obliczonych dla poszczególnych kolumn.
`mean(X)`	Jeśli `X` jest wektorem, zwraca iloczyn wszystkich elementów. Jeśli `X` jest macierzą zwraca wektor iloczynów elementów obliczonych dla poszczególnych kolumn.
`sort(X)`	Sortuje kolumny macierzy `X` wg. wzrastających wartości elementów.
`std(X)`	Jeśli `X` jest wektorem, zwraca odchylenie standardowe wszystkich elementów. Jeśli `X` jest macierzą zwraca wektor odchyleń standardowych elementów obliczonych dla poszczególnych kolumn. Odchylenia obliczane są z wykorzystaniem nieobciążonego estymatora wariancji.

3.10.3. Funkcje macierzowe

Najważniejsze funkcje, decydujące o popularności Matlaba, to funkcje wykonujące operacje na macierzach. Wybrane funkcje macierzowe zebrano w tabeli 3.9.

Tabela 3.9. Wybrane funkcje macierzowe

Funkcja	Opis
`eig(X)`	Zwraca wektor wartości własnych macierzy kwadratowej `X`.
`L=chol(X)`	Realizuje rozkład *Cholesky'ego* macierzy `X`. Macierz musi być dodatnio określona, w przeciwnym przypadku zwracany jest komunikat o błędzie.
`[U,S,V] = svd(X)`	Realizuje rozkład SVD (ang. *Singular Value Decomposition*) macierzy `X`.
`inv(X)`	Wyznacza odwrotność macierzy `X`.
`[L,U]=lu(X)`	Realizuje rozkład LU macierzy `X`. W wyniku uzyskuje się macierze `L` (permutacja macierzy dolnej trójkątnej), `U` (macierz górna trójkątna) .
`[Q,R]=qr(X)`	Realizuje rozkład `QR` macierzy `X`. W wyniku uzyskuje się macierz `Q` (macierz ortogonalną) i `R` (macierz górną trójkątną).
`H=hess(X)`	Przekształca macierz `X` do postaci *Hessenberga*.
`S=schur(X)`	Przekształca macierz `X` do postaci *Shura*.
`M=expm(X)`	Oblicza macierz `M` funkcji wykładniczej dla macierzy `X`.
`M-sqrtm(X)`	Oblicza macierz `M` równą pierwiastkowi kwadratowemu macierzy `X`.
`v=poly(X)`	Zwraca wektor `v` zawierający współczynniki wielomianu charakterystycznego macierzy `X` uszeregowane w kolejności od najbardziej znaczącego do wyrazu wolnego.
`det(X)`	Oblicza wyznacznik macierzy kwadratowej `X`.
`[r,c]=size(X)`	Zwraca liczbę rzędów (`r`) i liczbę kolumn (`c`) macierzy `X`.
`norm(X,p)`	Oblicza wartość normy macierzy `X`. Parametr p określa rodzaj liczonej normy, np. `p='fro'` – norma *Frobeniusa*.
`cond(X)`	Oblicza współczynnik uwarunkowania macierzy `X`.
`rank(X)`	Oblicza rząd macierzy `X`.

Wymienione funkcje macierzowe mogą być wywoływane z różnymi parametrami, oferując znacznie więcej możliwości niż opisane w tabeli 9. W celu zapoznania się z nimi należy sięgnąć do pliku pomocy, w którym są one dokładnie opisane.

3.11. Praca z plikami – zapis i odczyt danych

W Matlabie istnieje wiele możliwości pracy z plikami. Dotyczy to odczytu i zapisu danych i wyników pracy w Matlabie.
Najważniejszą funkcją do zapisu danych jest `save`. Standardowo zapisuje ona zmienne występujące w przestrzeni roboczej do pliku o rozszerzeniu .mat, który domyślnie będzie się znajdował w katalogu roboczym. Funkcja ta umożliwia różne sposoby zapisu danych do pliku (patrz plik pomocy).

Składnia

```
save nazwapliku.mat zmienna;
```

Inne przydatne funkcje do zapisu danych zostały zebrane w tabeli 3.10.

Tabela 3.10. Funkcje do zapisu danych.

Funkcja	Opis
`uisave`	Instrukcja otwiera nowe okienko z interfejsem do zapisu danych z przestrzeni roboczej do pliku.
`hgsave('nazwa_pliku')`	Funkcja zapisująca okna z wykresami/rysunkami do pliku o rozszerzeniu .fig.
`diary('nazwa_pliku')`	Funkcja zapisująca poszczególne instrukcje i tekst z okna poleceń do pliku tekstowego. Standardowo, jeśli nie jest podana nazwa pliku, tworzony jest plik o nazwie diary.

Odczytywanie danych zapisanych w pliku .mat lub w plikach tekstowych może być zrealizowane z wykorzystaniem funkcji `load`. Funkcja ta umożliwia różne sposoby odczytu danych z pliku (patrz plik pomocy).

Składnia

```
load nazwapliku.mat zmienna;
```

Zapis, wczytywanie i importowanie danych jest również możliwe z okienka przestrzeni roboczej (*Workspace*).

3.12. Skrypty i funkcje użytkownika

Matlab pozwala użytkownikowi tworzyć własne pliki tzw. M-pliki (M-files), zawierające zbiory instrukcji. Są to pliki tekstowe o rozszerzeniu .m. Wyróżnia się dwa rodzaje M-plików, skrypty i funkcje.

3.12.1. Skrypty

Skrypty są to zestawy wykonywanych po kolei instrukcji. Wszystkie zmienne które powstają w instrukcjach skryptu automatycznie znajdują się w przestrzeni roboczej i są dostępne z okna poleceń. Wywołanie skryptu zapisanego w katalogu roboczym polega na wpisaniu jego nazwy w oknie poleceń.

Przykład 3.7. (skrypt - *skrypt.m* **)**

```
%poczatek skryptu
clear all
clc
disp('Poczatek działania skryptu.')
x = [0:pi/10:2*pi];
y = cos(x);
plot(x,y) %wyswietla wykres z punktami x,y
%koniec skryptu
```

Sposób wywołania skryptu z linii komend:

```
>> skrypt
```

3.12.2. Funkcje

Funkcje umożliwiają zamknięcie kilku instrukcji w jednej funkcjonalnej całości i wielokrotne wywołanie jej w programie.

Istotną własnością funkcji jest ich własna przestrzeń robocza, wszystkie zmienne utworzone podczas jej działania i nie przekazane na zewnątrz funkcji są usuwane po zakończeniu działania funkcji i nie są widoczne poza funkcją. Tak więc jeśli istnieją zmienne, które trzeba przekazać dalej, to muszą być zdefiniowane w inny sposób. Najczęściej wykorzystuje się zmienne wyjściowe, podawane w pierwszej linijce definiującej funkcję (patrz: składnia), która zaczyna się od słowa kluczowego `function`. Zmienne przekazywane są do funkcji jako argumenty wejściowe. Wewnątrz funkcji muszą istnieć instrukcje definiujące każdą zmienną wyjściową, w przeciwnym przypadku funkcja zwróci błąd .

Do dobrej praktyki programistycznej należy nazywanie funkcji tak samo jak pliku, w którym jest ona zapisywana.

Składnia

```
function [zmienna_wyjściowa1, zmienna_wyjściowa2, ...]= nazwa_funkcji(argu-
ment_wejściowy1,argument_wejściowy2,...)
%%Komentarz dokumentujący
        instrukcje
```

Komentarz dokumentujący można wyświetlić z wykorzystaniem instrukcji help.

Przykład 3.8. (funkcja - *pierwiastki.m*)

```
function [x1, x2] = pierwiastki(a,b,c)

%%pomoc do funkcji
%%[x1,x2] = pierwiastki(a,b,c)
%%Oblicza pierwiastki równania a*x^2 + b*x + c = 0.

if( a == 0 )
      x1 = -c/b;
      x2 = x1;
else
      delta = b^2 - 4*a*c;
      x1 = (-b+sqrt(delta)) / (2*a);
      x2 = (-b-sqrt(delta)) / (2*a);
end;
```

Sposób wywołania funkcji i pomocy do funkcji w linii komend

```
>> [p1,p2] = pierwiastki(1,0,-1);
>> help pierwiastki; %wyświetlenie pomocy
```

3.13. Dodatkowe polecenia

W środowisku Matlab istnieją także dodatkowe polecenia przydatne podczas pracy., służące do zarządzania i podglądu zmiennych Najważniejsze z nich zebrano w tabeli 11.

Tabela 3.11. Dodatkowe polecenia.

Polecenie	Opis
`clc`	Czyszczenie okna poleceń (*Command Window*).
`who`	Wyświetlenia nazw zmiennych znajdujących się w przestrzeni roboczej (*Workspace*).
`whos`	Wyświetlenia nazw i szczegółów zmiennych znajdujących się w przestrzeni roboczej (*Workspace*).
`clear nazwa_zmiennej`	Usuwa zmienną o podanej nazwie z przestrzenie roboczej.
`clear all`	Usuwa wszystkie zmienne z przestrzeniu roboczej.

4. Grafika w Matlabie

Podstawowym elementem wizualizacyjnym w Matlabie jest okno graficzne (ang. figure). Nowe okno tworzy polecenie: `figure(numer)`, po którego wpisaniu Matlab automatycznie tworzy nowe okno lub korzysta z istniejącego aktywnego okna, w przypadku kiedy wcześniej powstało okno o tym numerze.
Okno o danym `numerze` zamyka się poleceniem `close(numer)`. Polecenie `close all` zamyka wszystkie aktywne okna.
Jeśli w oknie znajduje się wykres, polecenie `cla` czyści ten wykres. Wykorzystywane jest także polecenie `clf`, czyszczące całe aktywne okno.

4.1. Grafika 2D

Do tworzenia dwuwymiarowych wykresów służą przede wszystkim funkcje `plot` i `fplot`. Funkcje rysujące wykresy są bardzo elastyczne i pozwalają na kreślenie różnych, złożonych struktur. Dodatkowo w Matlabie zaimplementowano funkcje rysujące wykresy w skali logarytmicznej: `loglog, semilogx, semilogy`.

Składnia (`plot`)

```
plot(X,Y,LineSpec)
```

Funkcja `plot` wywołana z parametrami `X,Y` rysuje wykres ciągu elementów wektora `Y` (oś rzędnych) względem wektora `X` (oś odciętych). Funkcja plot wywołana tylko z jednym parametrem (wektorem wartości) `Y` rysuje wykres ciągu elementów `Y` względem ich indeksów.
Funkcja `plot` może występować z wieloma zestawami parametrów (parami parametrów) jednocześnie `plot(X1,Y1,LineSpec1,X2,Y2,LineSpec2,...)`. Parametrami funkcji `plot` mogą być także macierze. Parametr `LineSpec` pozwala określić własności wykresu takie jak kolor, grubość, typ linii, znaczniki i ich typ.
Jest on jedno-, dwu-, trój-elementowym ciągiem znaków np. `'r--'` oznacza linię kreskowaną o kolorze czerwonym.
Zbiór kodów oznaczających kolory i rodzaj linii można znaleźć w pliku pomocy.

Składnia (`fplot`)

```
fplot(funkcja, zakres, LineSpec)
```

Funkcja `fplot` powstała do bardziej precyzyjnego rysowania funkcji niż funkcja `plot`. Rysuje ona wykres funkcji zdefiniowanej w oddzielnym pliku. Argument `zakres` jest dwuelementowym wektorem opisującym granice przedziału w jakim ma być rysowany wykres.
Bardzo przydatna jest także funkcja `subplot`, o następującej składni

Składnia (`subplot`)

```
subplot(m,n,p)
```

Funkcja `subplot` dzieli aktywne okno na `m` rzędów i `n` kolumn (równo rozłożonych komórek o identycznym rozmiarze). W każdej z komórek umieszcza miejsce na wykres z nowym układem współrzędnych. Parametr `p` uaktywnia `p`-ty z utworzonych układów (kierunek ruchu od górnego lewego rogu w prawo i w dół, `m*n` utworzonych układów). Parametry `m` i `n` można wybrać z przedziału [1,9].
Wykresy można dostosowywać do własnych potrzeb. Służą do tego specjalne funkcje, z których najważniejsze zebrano w tablicy 4.1. Matlab pozwala opisywać wykresy formułami matematycznymi, które jeśli zapisane są w składni LaTeX To są automatycznie formatowane.

Tabela 4.1. Funkcje do zarządzania wykresami 2D.

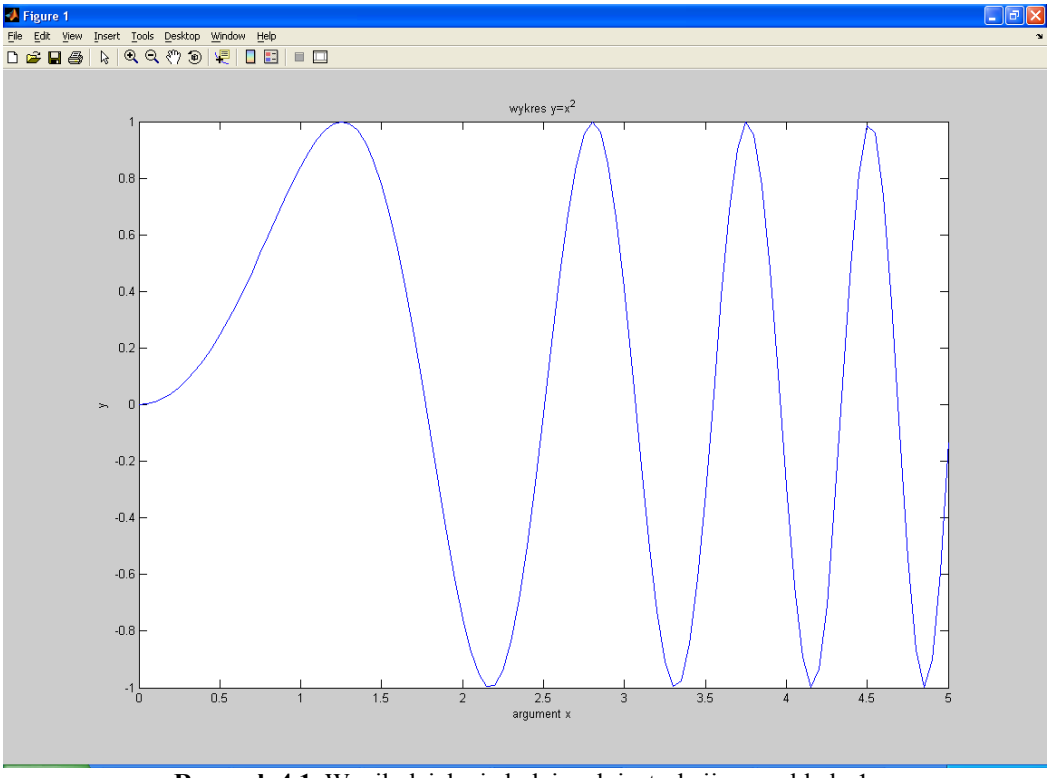
Funkcja	Opis
`hold on`	Blokowanie czyszczenia okna przy nowym wykresie.
`hold off`	Odblokowanie czyszczenia okna przy nowym wykresie.
`title('tytuł wykresu')`	Dodaje tytuł do wykresu w aktywnym oknie .
`xlabel('opis osi x')`	Dodaje opis do osi x.
`ylabel('opis osi y')`	Dodaje opis do osi y.
`text(x,y,'tekst')`	Dodaje tekst do wykresu w danym punkcie o współrzędnych `x, y`.
`legend`	Dodaje legendę do wykresu. Ma dużo możliwości, określonych przez parametry. Funkcja jest dobrze opisana w dokumentacji Matlaba.
`errorbar(x,y,e)`	Rysuje wykres funkcji zaznaczając dodatkowo podane przez użytkownika odchylenia znajdujące się w wektorze `e`.
`grid`	Dodaje na wykresie pomocniczą siatkę współrzędnych

Przykład 4.1 (Grafika 2D**) – rysunek 4.1.**

Rysowanie prostej funkcji

$$y = \sin(x^2) \tag{4.1}$$

```
>>x=0:0.05:5;     %utworzenie wektora wartości x
>>y=sin(x.^2);    %utworzenie wektora wartości y (x. - element po elemencie)
>>figure(1);      %utworzenie wykresu o indeksie 1
>>plot(x,y);      %rysowanie wykresu y=f(x)
>>title('wykres y=sin(x^2)');      %dodanie tytułu do wykresu
>>xlabel('argument x');            %dodanie opisu osi x
>>ylabel('y=f(x)');                %dodanie opisu osi y
```



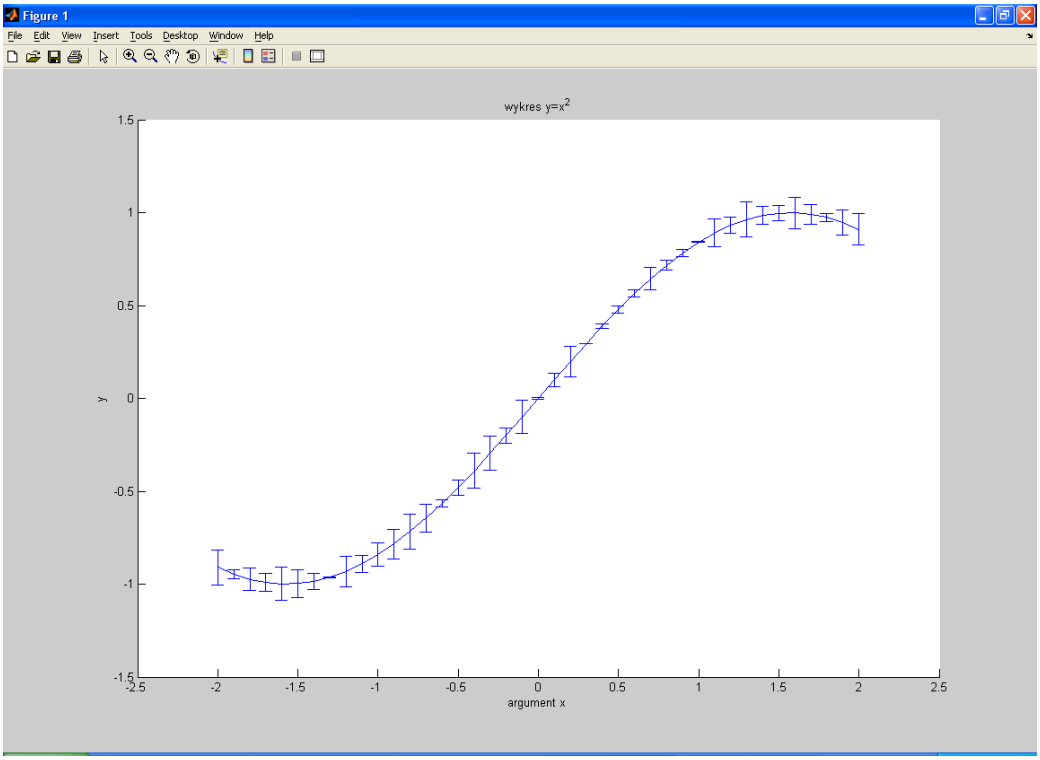
Rysunek 4.1. Wynik działania kolejnych instrukcji z przykładu 1.

Przykład 4.2 (Grafika 2D**) – rysunek 4.2.**

Rysowanie funkcji ze słupkami błędu. Zadana funkcja:

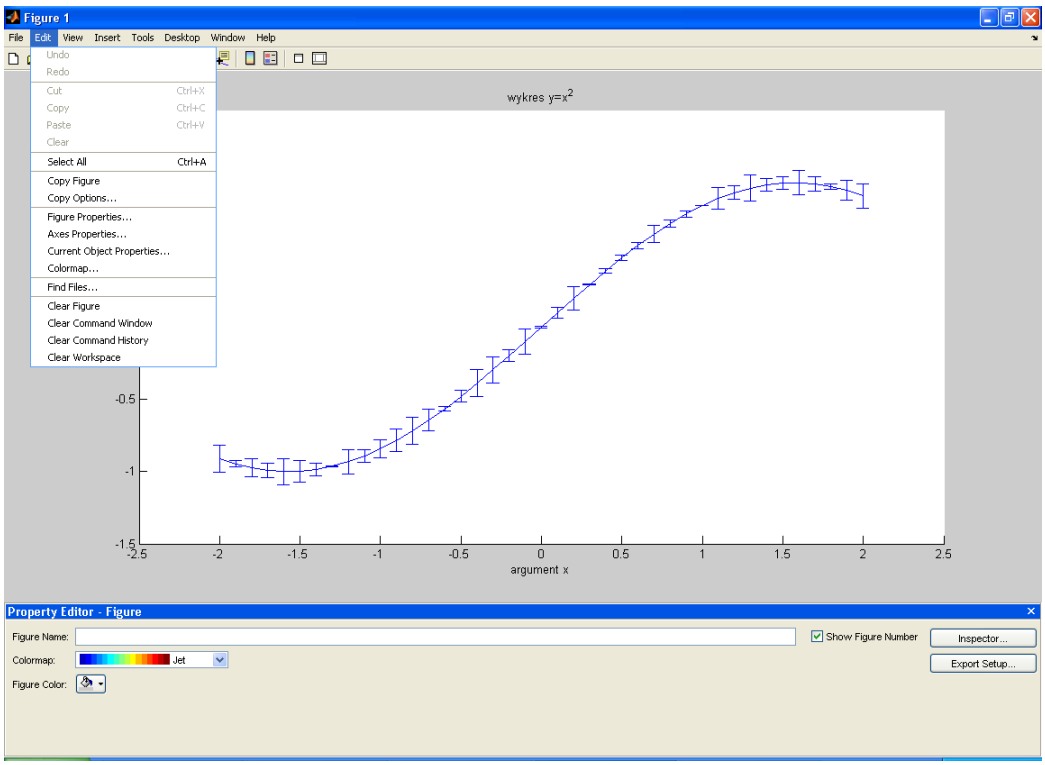
$$y = \sin(x) \tag{4.2}$$

```
>>x=-2:0.1:2;            %utworzenie wektora wartości x
>>y=sin(x);              %wykorzystanie funkcji x
>>e = rand(size(x))/10;       %utworzenie losowych wartości błędów
>>figure(1);             %utworzenie wykresu o indeksie 1
>>errorbar(x,y,e);       %rysowanie wykresu y=f(x) z odchyleniami
>>title('wykres y=x^2'); %dodanie tytułu do wykresu
>>xlabel('argument x');       %dodanie opisu osi x
>>ylabel('y');           %dodanie opisu do osi y
```



Rysunek 4.2. Wynik działania kolejnych instrukcji z przykładu 1.

Oprócz modyfikacji wykresu z okna komend, można dokonać modyfikacji z wykorzystaniem specjalnych zakładek edycyjnych. Zakładkę do modyfikowania własności wykresu (rysunek 4.3.) t.j. rodzaj linii, modyfikowanie punktów otwiera się wybierając z menu głównego opcje: *Edit* > *Figure properties*... Zakładkę do modyfikowania własności osi t.j. gęstość punktów, zakres otwiera się wybierając z menu głównego opcje: *Edit* > *Axes properties*....



Rysunek 4.3. Okno edycyjne do modyfikowania własności wykresu.

4.2. Grafika 3D

Matlab umożliwia też rysowanie funkcji trójwymiarowych, zdefiniowanych przez macierze. Opis przydatnych funkcji znajduje się w tabeli 4.2. Sposób wykorzystania funkcji do grafiki 3D ilustrują dwa przykłady, 4.3 i 4.4.

Tabela 4.2. Funkcje do rysowania wykresów 3D.

Funkcja	Opis
`plot3(X,Y,Z,LineSpec)`	Rysuje łamaną łączącą punkty, których współrzędne stanowią odpowiadające sobie elementy wektorów X, Y, Z. Jeżeli są to macierze o identycznych wymiarach, to rysowana jest jedna linia dla każdej kolumny.
`mesh(X,Y,Z,LineSpec)`	Rysuje powierzchnię opisaną przez macierze `X, Y, Z`, w postaci kolorowej siatki o oczkach wypełnionych kolorem tła. Elementy macierzy `LineSpec` określają kolory obwódek poszczególnych oczek.
`contour(X,Y,Z,LineSpec)`	Rysuje wykres poziomic dla powierzchni
`meshc(X,Y,Z,LineSpec)`	Funkcja łącząca własności funkcji `mesh` i `contour`. Rysuje siatkę identyczną jak funkcja `mesh` i umieszcza pod nią wykres poziomicowy.
`surf(X,Y,Z,LineSpec)`	Rysuje powierzchnię opisaną przez macierze `X, Y, Z`, w postaci kolorowej siatki o oczkach wypełnionych różnymi kolorami. Elementy macierzy `LineSpec` określają kolory wypełnień poszczególnych oczek.
`waterfall(X,Y,Z,LineSpec)`	Rysuje powierzchnię opisaną przez macierze `X, Y, Z`, w postaci kolorowej siatki. W tym wypadku nie są rysowane linie odpowiadające kolumnom macierzy.

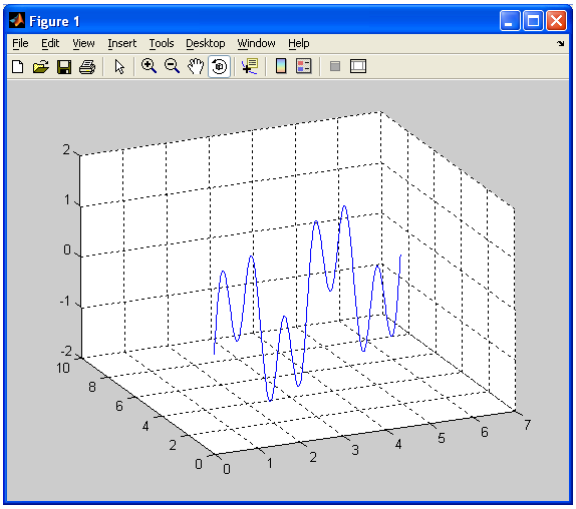
przykład 4.3. (Grafika 3D)

Przykład ilustruje rysowanie dwuargumentowej funkcji w 3 wymiarach. Zadana funkcja ma postać:

$$z = \sin(2x) + \sin(6y) \tag{4.3}$$

```
>>x=0:0.01:2*pi;              %generowanie wektora x
>>y=x;                        %generowanie wektora y
>> z=sin(2*x)+sin(6*y);            %obliczanie wektora z
>>figure(1);                  %pierwszy wykres o indeksie 1
>>plot3(x,y,z)                %przykład działania funkcji plot3
>>grid on;                    %dodanie linii siatki do wykresu
```

Wynik działania poszczególnych instrukcji z przykładu 4.3 przedstawiono na rysunku 4.4.



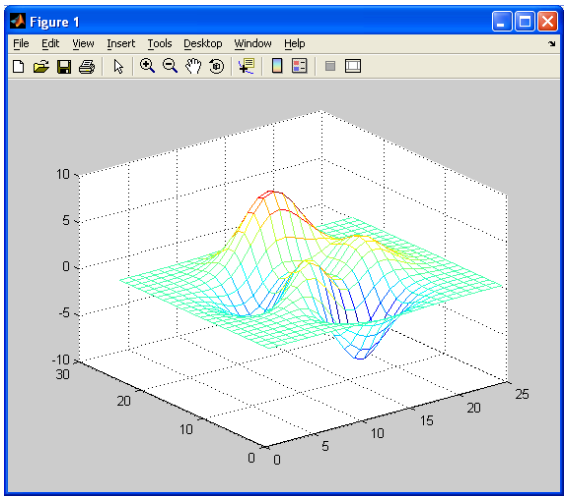
Rysunek 4.4. Wynik działania funkcji `plot3()`.

przykład 4.4. (Grafika 3D)

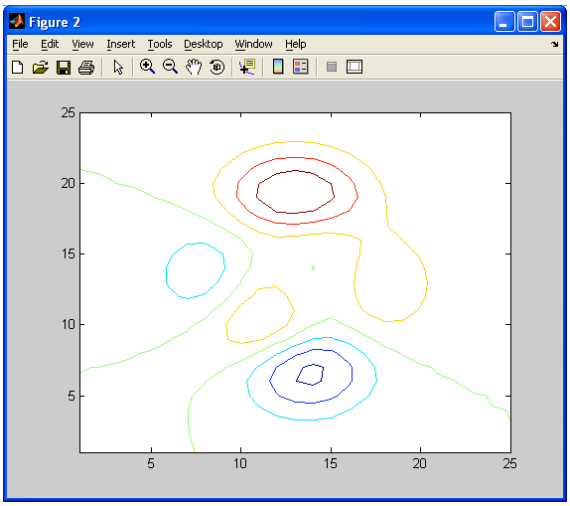
W przykładzie skorzystano z funkcji `peaks(x,y)` będącą funkcją dwóch zmiennych otrzymaną z przekształcenia rozkładów *Gaussa*, która służy za standardowy przykład wykresu 3D.

```
>>z=peaks(25);     %generowanie macierzy 25x25
>>figure(1);       %pierwszy wykres o indeksie 1
>>mesh(z);         %przykład działania funkcji mesh
>>figure(2);       %drugi wykres o indeksie 2
>>contour(z);      %przykład działania funkcji contour
>>figure(3);       %trzeci wykres o indeksie 3
>>meshc(z);        %przykład działania funkcji meshc
>>figure(4);       %czwarty wykres o indeksie 4
>>surf(z);         %przykład działania funkcji surf
>>figure(5);       %piąty wykres o indeksie 5
>>waterfall(z);    %przykład działania funkcji waterfall
```

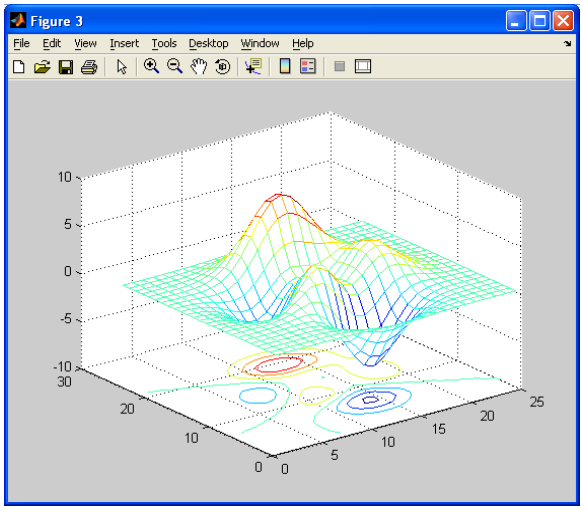
Wyniki działania poszczególnych instrukcji z przykładu 4.4 przedstawiono na rysunkach 4.5-4.9.



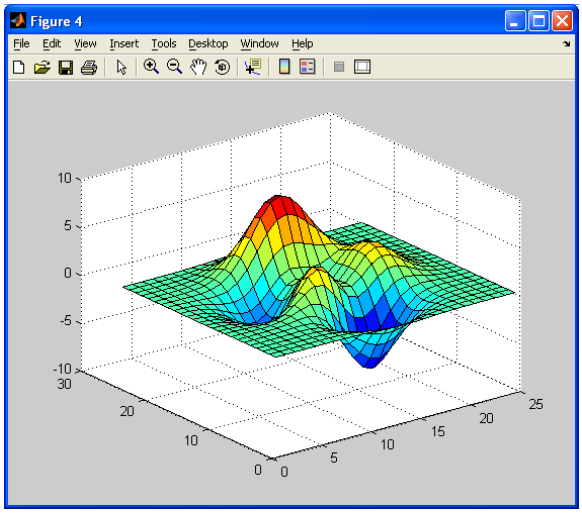
Rysunek 4.5. Wynik działania funkcji `mesh()`.



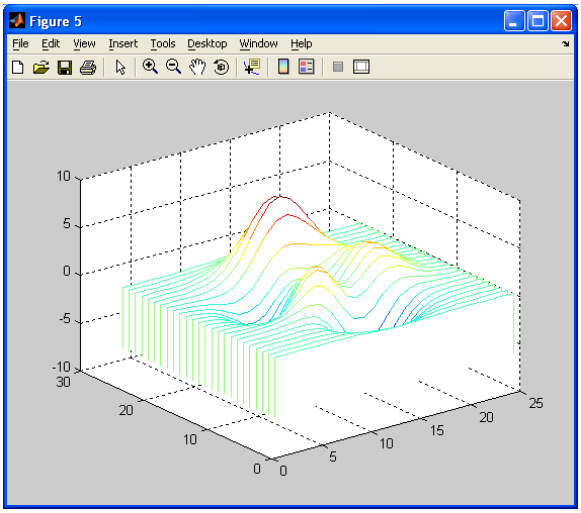
Rysunek 4.6. Wynik działania funkcji `contour()`.



Rysunek 4.7. Wynik działania funkcji `meshc()`.



Rysunek 4.8. Wynik działania funkcji `surf()`.



Rysunek 4.9. Wynik działania funkcji `waterfall()`.

5. Interpolacja i aproksymacja

5.1 Interpolacja

W wielu zagadnieniach inżynierskich postawione jest następujące zadanie. Dane są dwie serie danych x_i, oraz y_i, gdzie $i = 1,...,N$, para $\{x_i, y_i\}$ nazywana jest węzłem. Należy znaleźć funkcję $y = f(x)$ określającą zależność pomiędzy y_i i x_i. Jeżeli wśród serii danych x_i nie ma dwóch takich samych wartości, możliwe jest dopasowanie funkcji w danych węzłach. Interpolacja polega ona na znalezieniu wielomianu przechodzącego przez zadane węzły. Interpolacja spełnia zwykle rolę pomocniczą, np. w metodach optymalizacji.

W Matlabie zadanie interpolacji można rozwiązać na wiele sposobów. Najprostszym jest zastosowanie funkcji `interp1()`.

Składnia

```
yi=interp1(x,y,xi,'metoda')
```

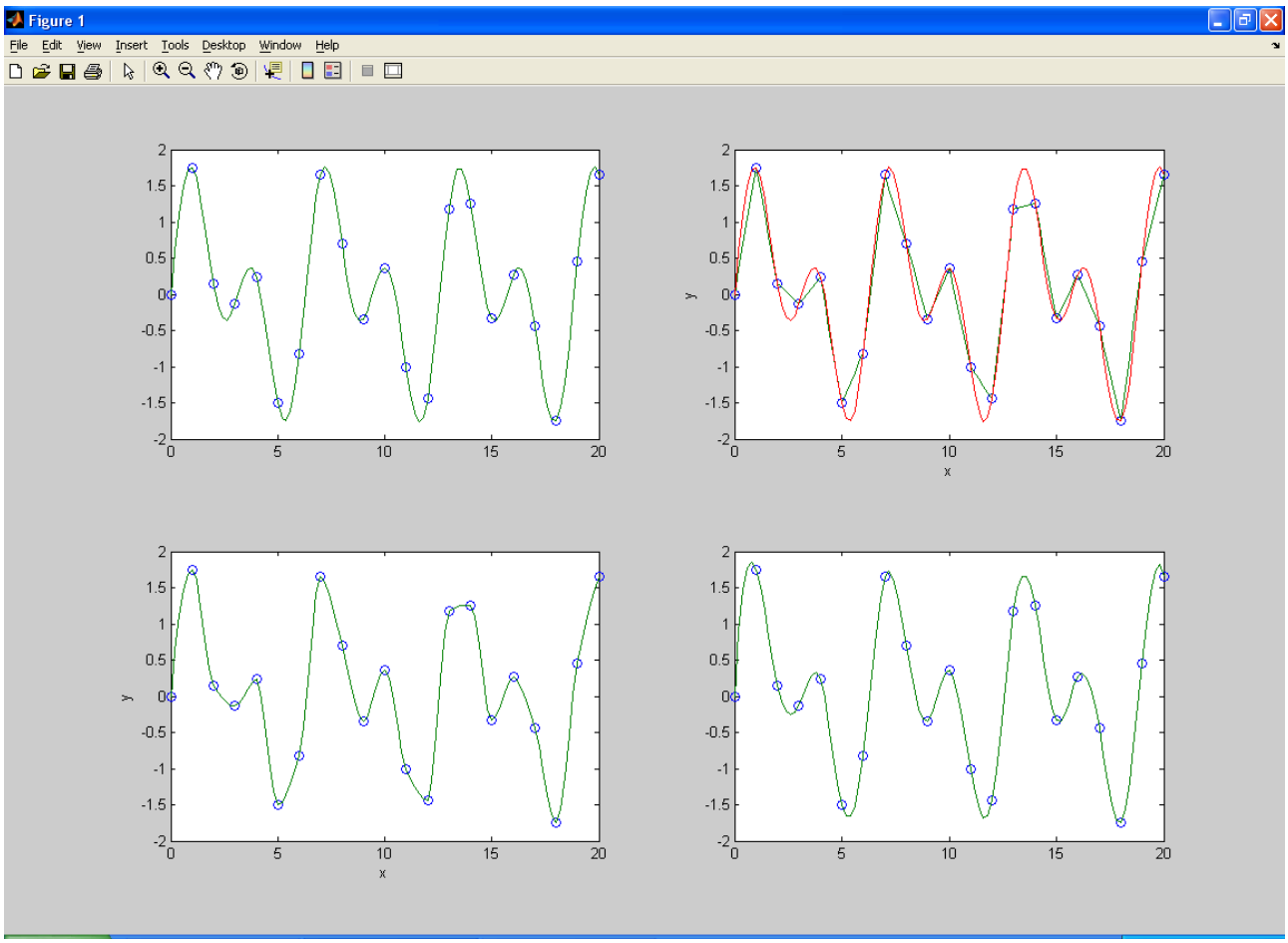
`Funkcja interp1` wykonuje interpolację funkcji jednej zmiennej w punktach określonych wektorem `xi`. Węzły interpolacji określone są parametrami `x` i `y`. W wyniku otrzymujemy wektor `yi` odpowiadający interpolacji w punktach zdefiniowanych przez elementy wektora `xi` (zakres interpolacji) i zadanych węzłach. Opcjonalny parametr '`metoda`' pozwala na wybór metody interpolacji

- `'linear'` – interpolacja funkcją łamaną
- `'cubic'`- interpolacja wielomianami trzeciego stopnia
- `'spline'`- interpolacja funkcjami sklejanymi trzeciego stopnia

Sposób przeprowadzenia interpolacji z wykorzystaniem funkcji interp1 został przedstawiony w przykładzie 5.1.

Przykład 5.1. (Interpolacja)

```
%określenie węzłów
>>x=0:20;
>>y=sin(x)+sin(2*x);
%wektor wartości xi, zakres w którym przeprowadzamy interpolację
>>xi=0:0.2:20;
%interpolacja funkcją łamaną
>>yi=interp1(x,y,xi,'linear');
%wykres
>>figure(1);
>>subplot(2,2,2);
>>plot(x,y,'o',xi,yi,xi,sin(xi)+sin(2*xi));
>>xlabel('x');
>>ylabel('y');
%interpolacja wielomianami 3-go stopnia
>>yi=interp1(x,y,xi,'cubic');
>>subplot(2,2,3);
>>plot(x,y,'o',xi,yi);
>>xlabel('x');
>>ylabel('y');
%interpolacja funkcjami sklejanymi 3-go stopnia
>>yi=interp1(x,y,xi,'spline');
>>subplot(2,2,4);
>>plot(x,y,'o',xi,yi);
>>subplot(2,2,1);
>>plot(x,y,'o',xi,sin(xi)+sin(2*xi));
```



Rysunek 5.1. Wynik interpolacji z wykorzystaniem funkcji `interp1`.

5.2. Aproksymacja

Aproksymacja jest zadaniem przybliżania jednych wartości innymi wartościami, wygodniejszymi z określonych względów. Na przykład w niektórych zagadnieniach dana jest seria danych, lecz nie jest dana zależność funkcyjna pomiędzy tymi danymi. Metody aproksymacji pozwalają na dokonanie próby znalezienia zależności funkcyjnej pomiędzy danymi, co ułatwia ich analizę.

Zadanie aproksymacji można postawić następująco. Dana jest seria danych $x = [x_1, x_2, ..., x_N], x_i \in R$, oraz odpowiadająca jej seria skalarnych wartości $y = [y_1, y_2, ..., y_N], y_i \in R$. Należy znaleźć funkcję $f(x)$ przybliżającą zależność pomiędzy x i y.

Zwykle zakłada się na początku postać funkcji aproksymującej $f(x)$. Na ogół nie dokonuje ona dokładnego odwzorowania serii danych i powstają błędy pomiędzy wartością funkcji aproksymującej w punkcie x_i, $f(x_i)$, a wartością y_i w tym punkcie. W zadaniu aproksymacji dokonywana jest próba najlepszego dopasowania funkcji $f(x)$ do serii danych y, tak żeby zminimalizować otrzymane błędy.

W celu oceny jakości przybliżenia należy zastosować odpowiedni wskaźnik. W podstawowych metodach aproksymacji, stosowany wskaźnik jakości ma postać odchylenia średniokwadratowego o postaci:

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \qquad (5.1)$$

Zaproponowany wskaźnik jakości stosowany jest w procedurach aproksymacyjnych.

Matlab pozwala na wykorzystanie różnych procedur aproksymacyjnych. Na prostych przykładach zostanie pokazana realizacja aproksymacji funkcją liniową tzw. regresja liniowa, oraz aproksymacja wielomianem.

5.2.1. Regresja liniowa

W przypadku regresji liniowej przyjmuje się postać funkcji aproksymującej $f(x) = xa_1 + a_0$, gdzie a jest współczynnikiem funkcji liniowej. Wskaźnik jakości aproksymacji przyjmuje postać:

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - (a_1 x_i + a_0))^2 \qquad (5.2)$$

Wartość optymalnych współczynników funkcji liniowej a_1, a_0 są możliwe do wyznaczenia, z różniczkowania wskaźnika jakości lub twierdzenia o rzucie ortogonalnym.
W Matlabie regresję liniową można zrealizować, stosując proste przekształcenia macierzowe lub funkcję `polyfit`. Znajduje ona współczynniki a_1, a_0 i zwraca w postaci wektora `a=[a1 a0]`.

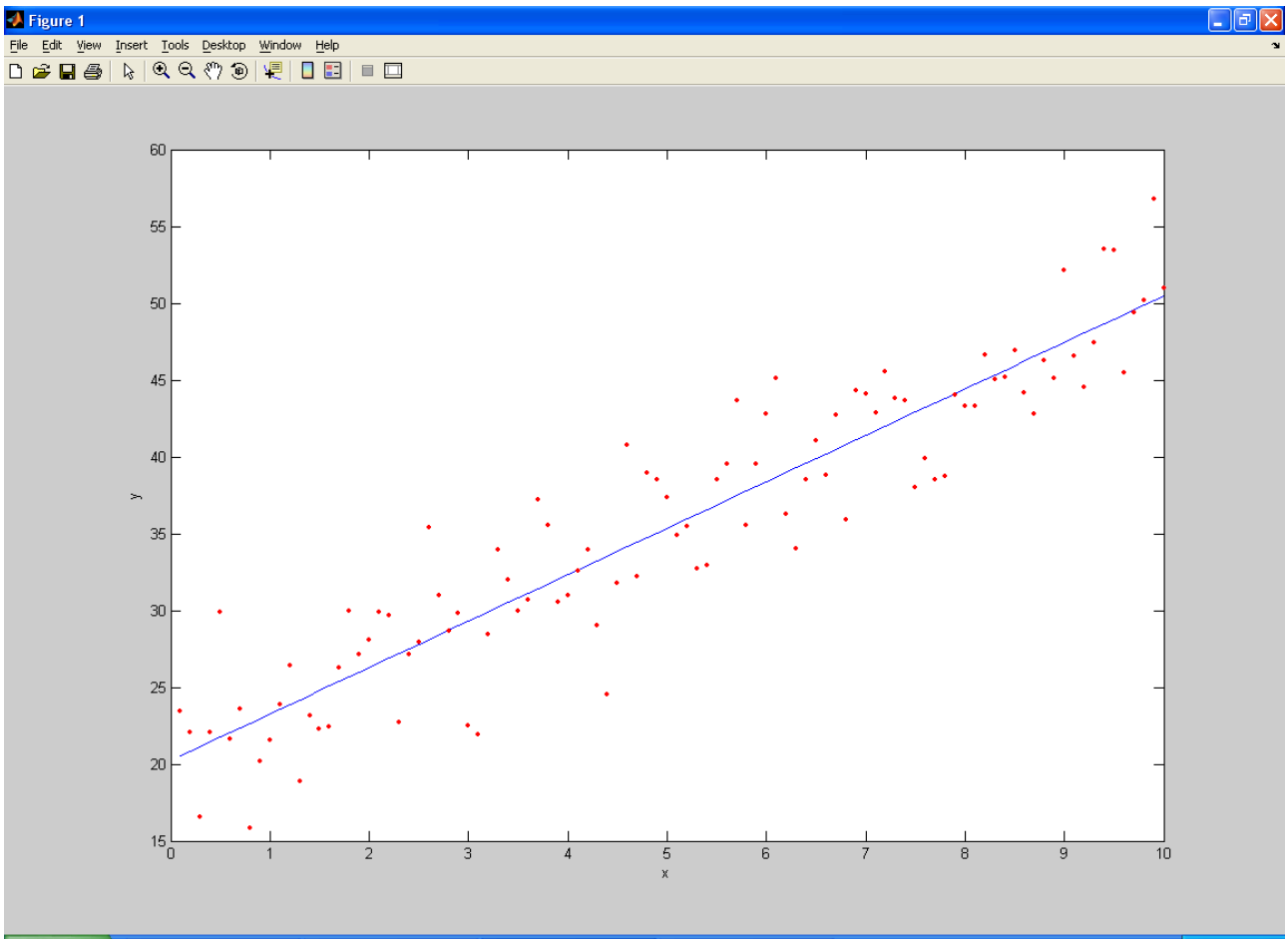
Składnia

```
a=polyfit(x,y,1);
```

Sposób realizacji regresji liniowej obrazuje przykład 5.2.

Przykład 5.2. (regresja liniowa)

```
>>x=[0.1:0.1:10]';
>>[m,n]=size(x);
>>for i=1:m
>>    y(i,1)=20+randn(1,1)*sqrt(10)+0.3*i;
>>end;
>>a=polyfit(x,y,1);
>>for i=1:m
>>   f(i)=a(1)*x(i)+a(2);
>>end;
>>figure(1);
>>plot(x,f,'b',x,y,'r.');
>>xlabel('x');
>>ylabel('y');
```



Rysunek 5.2. Regresja liniowa.

5.2.2. Aproksymacja wielomianem

W przypadku aproksymacji wielomianem przyjmuje się postać funkcji aproksymującej,

$$f(x) = a_r x^r + a_{r-1} x^{r-1} + ... + a_1 x + a_0 \qquad (5.3)$$

Jest to postać wielomianu rzędu r, o współczynnikach $a_r, a_{r-1}, ..., a_1, a_0$.
Zadanie jest to identyczne z zadaniem regresji liniowej. W Matlabie do wyznaczania współczynników wielomianu aproksymującego można zastosować funkcję `polyfit`.

Składnia

```
a=polyfit(x,y,r);
```

Parametr `r` jest rzędem wielomianu. Funkcja zwraca wektor `a` współczynników wielomianu.
Dodatkowo, w Matlabie zaimplementowano funkcję `polyval`, ułatwiającą np. wykreślenie wielomianu aproksymacyjnego.

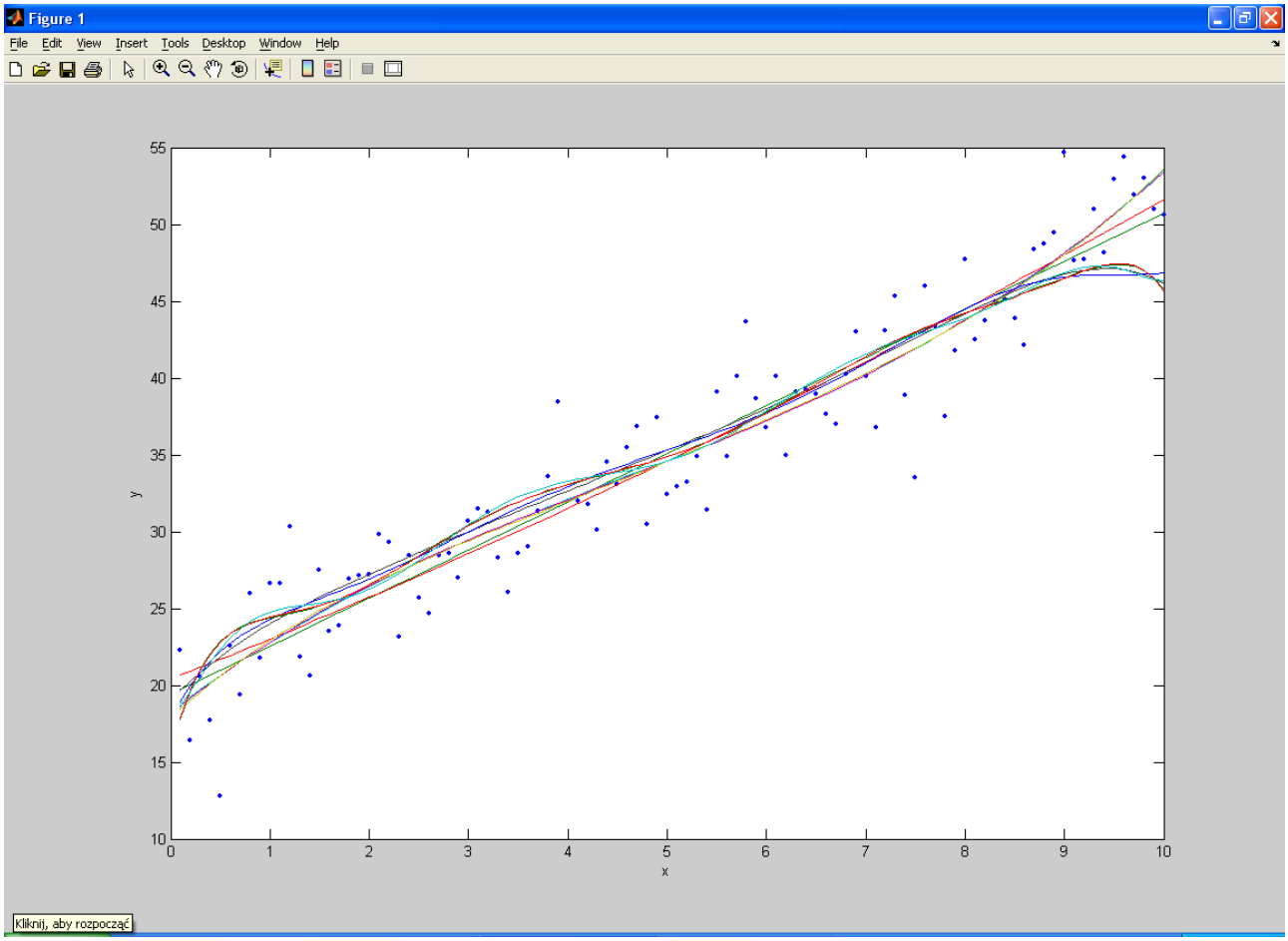
Składnia

```
f=polyval(a,x);
```

Funkcja ta dla danego wektora wartości x i wektora współczynników wielomianu a, zwraca wektor wartości funkcji (5.3). Sposób realizacji aproksymacji wielomianowej obrazuje przykład 5.2.

Przykład 5.2.

```
x=[0.1:0.1:10]';
[m,n]=size(x);
for i=1:m
    y(i)=20+randn(1,1)*sqrt(10)+0.3*i;
end;
for N=1:1:5;
a=polyfit(x,y',N);
f(:,N)=polyval(a,x);
plot(x,y,'.',x,f);
end;
xlabel('x');
ylabel('y');
```



Rysunek 5.3. Aproksymacja wielomianami .

6. Rozwiązywanie równań różniczkowych zwyczajnych

W kolejnej części kursu zajmiemy się rozwiązywaniem równań różniczkowych zwyczajnych pierwszego rzędu, w następującej postaci

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_1}{dt^2} \\ \dots \\ \frac{dx_1}{dt^2} \end{bmatrix} = \begin{bmatrix} F_1(t, x_1, x_2, ..., x_n) \\ F_2(t, x_1, x_2, ..., x_n) \\ \dots \\ F_n(t, x_1, x_2, ..., x_n) \end{bmatrix} \tag{6.1}$$

przy zadanych warunkach początkowych

$$x_i(t_0) = x_{i0} \text{ dla } i = 1,2,..., n \tag{6.2}$$

Równanie (6.1) można zwięźle zapisać jako

$$\frac{dx}{dt} = F(t,x); x(t_o) = x_0; x(t) \in R^n, x(t_0) \in R^n \tag{6.3}$$

gdzie:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}, \; x_0 = \begin{bmatrix} x_1(t_0) \\ x_2(t_0) \\ \dots \\ x_n(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ x_{20} \\ \dots \\ x_{n0} \end{bmatrix}, \; F(t,x) = \begin{bmatrix} F_1(t,x) \\ F_2(t,x) \\ \dots \\ F_n(t,x) \end{bmatrix}.$$

Zadanie postawione jest następująco.
Szukane jest rozwiązanie układu równań (6.3), w przedziale czasu $t \in [t_0, t_k]$, gdzie t_0, t_k to odpowiednio chwila początkowa i chwila końcowa.
W Matlabie zimplementowane są metody do rozwiązywania układów równań różniczkowych w postaci (6.3). Stosowane są metody różnicowe, które rozwiązują zadanie krok po kroku. W uproszczeniu wygląda to następująco, na podstawie warunku początkowego $x(t_0) = x_0$ uzyskuje się rozwiązanie w chwili $t_0 + \Delta t$, gdzie Δt jest okresem próbkowania. Na podstawie otrzymanego rozwiązania oblicza się rozwiązanie w chwili $t_0 + 2\Delta t$, i kontynuuje się te kroki aż do osiągnięcia chwili końcowej.
W większości przypadków funkcja $F(t)$ musi spełniać następujące założenia

1. funkcja $F(t,x)$ jest ciągła w przedziale $t \in [t_0, t_k]$ i $x \in R^n$
2. istnieje taka stała α, że dla każdego $t \in [t_0, t_k]$ i $x, \tilde{x} \in R^n$ zachodzi nierówność,

$$\left\| F(t,x) - F(t,\tilde{x}) \right\| \le \alpha \left\| x - \tilde{x} \right\| \tag{6.4}$$

jest to tzw. warunek Lipschitza względem zmiennej x.

Najczęściej wykorzystywane funkcje do rozwiązywania układu równań różniczkowych mają nazwę `ode(ang. ordinary differential equations)`. Ich składnia jest następująca.

Składnia:

```
[T,X] = odennxx(odefun,t0,tk,x0,tol,tr)
```

`gdzie nn` - rząd metody; `xx` - opcjonalne własności.

parametry:
`odefun`- łańcuch zawierający nazwę zdefiniowanej przez użytkownika funkcji zwracającej wartości $F(t,x)$,
`t0,tk` - granice przedziału czasu, w którym poszukiwane jest rozwiązanie ,
`x0` - warunek początkowy - wartość rozwiązania układu w chwili początkowej,
`tol` - opcjonalny parametr określający wymaganą dokładność,
`tr` - opcjonalny parametr, który jeśli ma wartość różną od zera to powoduje wypisanie kolejnych kroków.

W przypadku gdy równania różniczkowe dane są w postaci jawnej $x_0 = f(t,x)$, do ich rozwiązania w zadanym przedziale stosowane mogą być metody zebrane w tabeli 6.1.

Tabela 6.1. Funkcje stosowane do rozwiązywania układu równań różniczkowych zwyczajnych.

Funkcja	Opis
nie-sztywne	
`ode45`	Metoda *Runge-Kutty*, rzędu 4 i 5, jednokrokowa (*Bogacki-Shampine*).
`ode23`	Metoda *Runge-Kutty,* rzędu 2 i 3, jednokrokowa (*Dormand-Prince*).
`ode113`	Metoda wielokrokowa (*Adams-Bashforth-Moulton*).
sztywne	
`ode15s`	Metoda *Klopfensteina*, wielokrokowa o zmiennym rzędzie, bazująca na formułach różniczkowania numerycznego.
`ode23s`	Metoda *Rosenbrocka*, wielokrokowa, rzędu 2 lub 3.
`ode23t`	Metoda wykorzystująca przybliżenie trapezami.
`ode23tb`	Metoda *Gear'a* ,wielokrokowa, o zmiennym rzędzie i zmiennym kroku całkowania.

W przypadku kiedy równania dane są w postaci niejawnej $f(t,x,x_0) = 0$, stosowana jest metoda `ode15i`

Składnia

```
[T,X] = ode15i(odefun,tspan,x0,xp0)
```

Sposób pracy z funkcjami `ode` jest następujący:

Krok 1. Najpierw należy zdefiniować tzw. m-plik ODE, w którym zapisany jest interesujący użytkownika układ równań różniczkowych pierwszego rzędu. Jeśli dane jest równanie różniczkowe wyższego rzędu, należy zapisać je w postaci układu równań różniczkowych pierwszego rzędu Nazwa m-pliku ODE jest nazwą przekazywaną do funkcji `ode` jako parametr `odefun` .

Krok 2. Określić granice przedziału czasu `t0,tk`, w którym poszukiwane jest rozwiązanie.

Krok 3. Określić warunek początkowy `x0` zawierający wartość rozwiązania układu w chwili początkowej.

Krok 4. Określić dodatkowe parametry funkcji `ode`.

Krok 5. Wywołać funkcję `ode` z zadanymi parametrami.

Przykład 6.1 – Rozwiązanie równań oscylatora liniowego z tłumieniem

Dany jest model matematyczny oscylatora liniowego z tłumieniem w postaci równania drugiego rzędu

$$m\frac{d^2x}{dt^2} = -kx - a\frac{dx}{dt} \tag{6.5}$$

.

Rozwiązać to równanie w przedziale $t = [0,...,20]$, z wykorzystaniem metod `ode45` i `ode23`, dla zadanych parametrów $m = 1, a = 0.5, k = 1$ oraz w przypadku gdy parametr $a = 0.2$. Rozwiązanie przedstawić w formie wykresu.

Sposób rozwiązania zadania z wykorzystaniem środowiska Matlab może być następujący

Krok 1. Ponieważ równanie 6.5 jest równaniem różniczkowym drugiego rzędu, należy sprowadzić je do postaci układu równań różniczkowych pierwszego rzędu. W tym celu należy wprowadzić następujące zmienne

$$x_1 = x \tag{6.6}$$

$$x_2 = \frac{dx}{dt} \tag{6.7}$$

Równanie (6.5) można zapisać korzystając z (6.6) i (6.7) w postaci układu równań różniczkowych pierwszego rzędu

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \dfrac{-kx_1 - ax_2}{m} \end{bmatrix} \tag{6.8}$$

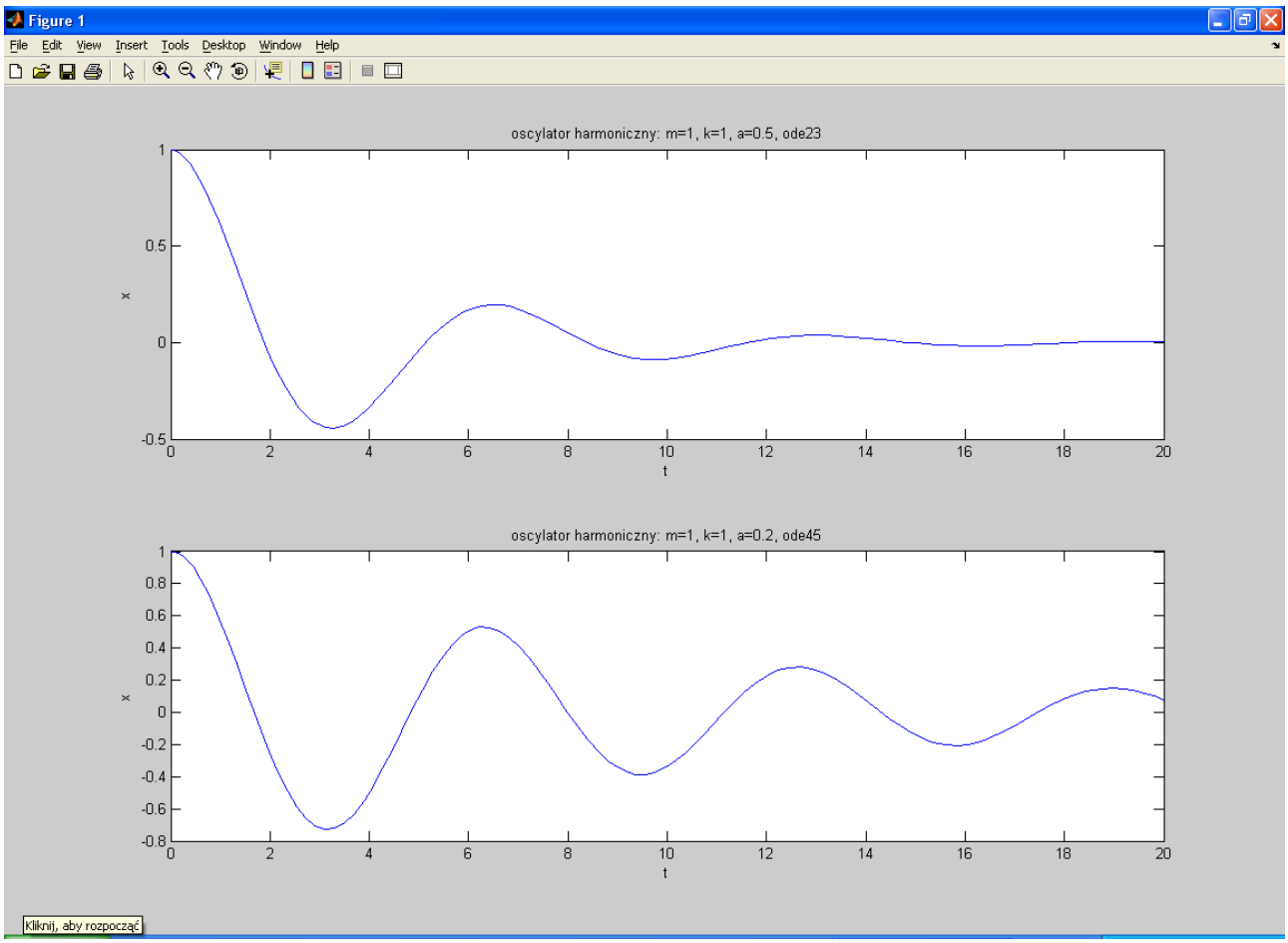
Na podstawie układu równań (6.8) projektuje się m-plik ODE

m-plik ODE - 'osc.m'

```
function [Dx]=osc(t,x)
global M A K
Dx=[x(2);((-K*x(1)-A*x(2))/M)]
```

polecenia w oknie komend (**Kroki 2-5**)

```
>>M=1;
>>A=0.5;
>>K=1;
>>global M A K;
>>[T,X]=ode23('osc',0,20,[1 0]');%krok 2,3,4,5
>>subplot(2,1,1);
>>plot(T,X(:,1));
>>xlabel('t');
>>ylabel('x');
>>title('oscylator harmoniczny: m=1, k=1, a=0.5, ode23');
>>A=0.2;
>>[T,X]=ode45('osc',0,20,[1 0]');
>>subplot(2,1,2);
>>plot(T,X(:,1));
>>xlabel('t');
>>ylabel('x');
>>title('oscylator harmoniczny: m=1, k=1, a=0.2, ode45');
```

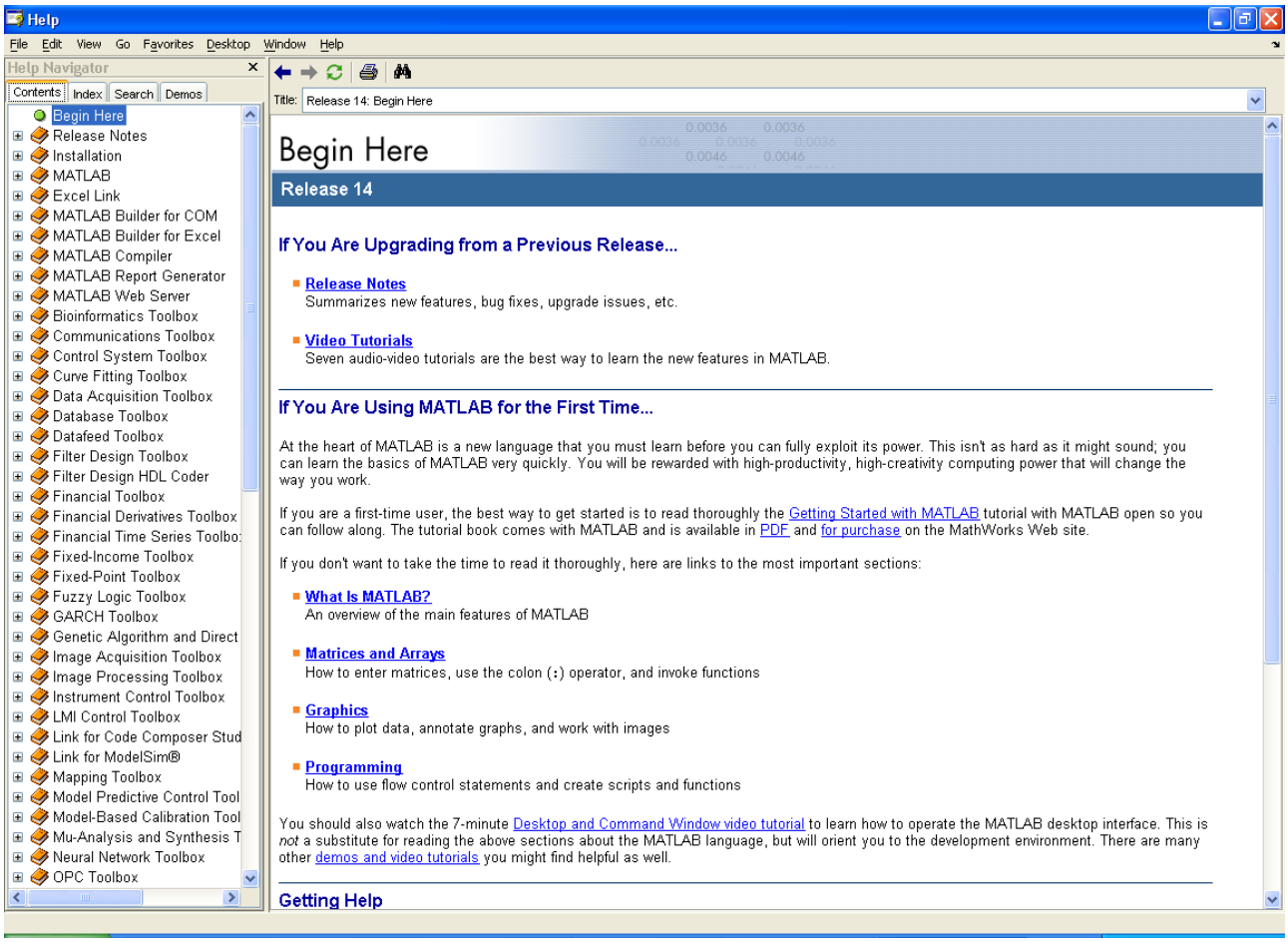


Rysunek 6.1. Wyniki rozwiązania równania oscylatora harmonicznego z tłumieniem przy pomocy funkcji `ode`.

7. Pomoc w środowisku Matlab

Istotnym elementem środowiska Matlab jest bardzo rozbudowana pomoc podręczna dostępna w trybie offline (pliki help oraz pliki PDF). Jest to okno wywoływane przez wybór następujących opcji w menu głównym *Help* > *Full Product Family Help*. Pomoc obejmuje opis podstawowego środowisko oraz zbiorów funkcji zawartych w każdej skrzynce narzędziowej (Toolbox-ach). Ponadto wiele funkcji i dodatkowych narzędzi oprócz opisu ma dobrze przygotowane przykłady ich wykorzystania.
Podczas zapoznawania się z możliwościami środowiska Matlab bardzo przydatne są także programy demonstracyjne. Wszystkie programy demonstracyjne znajdują się w zakładce demos w oknie pomocy, lub przez wpisanie w oknie komend:

```
>>demo
```



Rysunek 7.1. Okno pomocy środowiska Matlab.

7.1. Pomoc dla funkcji z wykorzystaniem instrukcji help

W środowisku Matlab możliwe jest także wyświetlenie informacji na temat funkcji/skryptu w oknie komend. W takim przypadku stosowana jest instrukcja `help`, po której występuje nazwa funkcji na temat której chcemy uzyskać informacje. W przypadku kiedy twórca/twórcy funkcji zawarli w jej kodzie odpowiedni opis, to pojawi się on na ekranie.

Przykład 7.1. – wyświetlenie pomocy dla funkcji `sin`

```
>>help sin
```

Wynik działania instrukcji `help`, wyświetlenie komentarz dodanego wewnątrz funkcji `sin()`

```
SIN    Sine.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded functions or methods (ones with the same name in other
    directories)
       help sym/sin.m

    Reference page in Help browser
       doc sin
```

W przypadku pisania własnych funkcji możliwe jest dodanie opisu w postaci komentarza, który będzie się wyświetlał po wywołaniu go instrukcją `help`. W tym celu kolejne linijki komentarza powinny być poprzedzone znakami `%%`.

Przykład 7.2 - dodanie do funkcji komentarza wyświetlanego z wykorzystaniem instrukcji `help`

```
[out]=function jakas_funkcja(in);
%% komentarz
…
```

Bibliografia

[1] A. Zalewski, R.Cegieła, *Matlab – obliczenia numeryczne i ich zastosowania*, Wydawnictwo Nakom, Poznań, 1998.

[2] B.Mrozek, Z.Mrozek, *Matlab i Simulink, Poradnik użytkownika*, Wydawnictwo Helion, 2004.

[3] S. Osowski, *Modelowanie i symulacja układów i procesów dynamicznych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2007.

[4] R. Pratap, *Matlab 7 dla naukowców i inżynierów*, Wydawnictwo Mikom, 2007.

[5] J. Brzózka, L. Drobczyński, *Matlab. Środowisko obliczeń naukowo-technicznych*, Wydawnictwo Mikom, 2008.