Podstawy Automatyki

Politechnika Poznańska
Katedra Sterowania i Inżynierii Systemów

ĆWICZENIE 2

Wprowadzenie do środowiska MATLAB 7.1 (R14SP3): Część 2.

Celem ćwiczenia jest kontynuacja zapoznawania się z podstawowymi cechami środowiska symulacyjno–programistycznego Matlab w wersji 7.1 (Release 14 with Service Pack 3). Podczas przebiegu ćwiczenia zaprezentowane zostaną sposoby tworzenia skryptów i funkcji za pomocą języka Matlab. Omówione zostaną także sposoby tworzenia wykresów przedstawiających wartości zmiennych zdefiniowanych w przestrzeni pracy.

1 Programowanie w Matlabie

Jak wspomniano we wstępie do opracowania Ćwiczenia 1, środowisko *Matlab* to nie tylko bardzo wygodne narzędzie obliczeniowe i symulacyjne, ale również język programowania wysokiego poziomu. W niniejszym punkcie przedstawione są najprostsze sposoby tworzenia programów użytkownika z wykorzystaniem funkcji udostępnianych przez *Matlaba*.

Pliki zawierające kod w języku *Matlaba* posiadają rozszerzenie .m, dlatego nazywane są m-plikami. *Matlab* rozgranicza dwa typy m-plików:

- **skrypty**, które nie pobierają w momencie swego wywołania żadnych argumentów wejściowych oraz nie zwracają danych wyjściowych, a operują jedynie na zmiennych zdefiniowanych w przestrzeni roboczej (czyli widocznych w obrębie bieżącej sesji *Matlaba* w Przeglądarce). Skrypty są ciągami poleceń umieszczonymi kolejno w pliku, dokładnie w taki sam sposób, w jaki są wpisywane w Oknie Poleceń *Matlaba*;
- **funkcje**, które pozwalają przekazać w momencie ich wywołania jakieś argumenty, na których operują, a po zakończeniu swego działania zwracają wynik; zmienne występujące wewnątrz funkcji są domyślnie lokalne dla danej funkcji i nie są widoczne poza nią.

Biorąc pod uwagę właściwości i różnice między skryptami a funkcjami można powiedzieć, że skrypty stosuje się aby uprościć lub „zautomatyzować” czynności wykonywane wielokrotnie. Funkcje zaś pozwalają na rozszerzenie możliwości *Matlaba* o te procedury i algorytmy, które nie zostały w nim wcześniej zdefiniowane.

Do tworzenia m-pliku najwygodniej posłużyć się Edytorem/Debuggerem dostarczanym wraz z pakietem *Matlab*. W celu utworzenia nowego pliku wystarczy z głównego menu wybrać File → New → M-file, posłużyć się klawiszami skrótu *Ctrl+N* lub wybrać pierwszy po lewej przycisk na pasku narzędzi. Jeszcze innym sposobem wywołania Edytora i rozpoczęcia edycji nowego m-pliku jest wpisanie w Oknie Poleceń komendy:

```
edit nazwa_nowego_pliku
```

1.1 Pisanie skryptów

Skrypt *Matlaba* jest m-plikiem zawierającym ciąg poleceń. Po utworzeniu takiego pliku, wywołanie skryptu, czyli rozpoczęcie sekwencyjnego wykonywania poleceń w nim zawartych, polega na wpisaniu w Oknie Poleceń *Matlaba* nazwy pliku ze skryptem, z pominięciem kropki i rozszerzenia. Innym sposobem jest również wykorzystanie ikony *Save and Run*. Ważne, aby bieżący katalog (Current Directory) w Przeglądarce zawierał dany skrypt. Skrypty mogą operować na danych istniejących w przestrzeni roboczej *Matlaba* lub też tworzyć nowe, własne struktury danych. Chociaż skrypty pozbawione są możliwości zwracania jakichkolwiek wartości, to utworzone w trakcie ich działania zmienne pozostają widoczne w obrębie przestrzeni roboczej, również po zakończeniu wykonywania danego skryptu. Poza tym skrypty mogą w trakcie swego działania generować wykresy czy tworzyć graficzne interfejsy użytkownika.

Naturalnie możliwe jest także umieszczanie w skryptach komentarzy – każdy znak umieszczony w tej samej linii co znak procentu (%), po jego prawej stronie, traktowany jest w skrypcie jak komentarz i jest ignorowany.

Dla przykładu, aby utworzyć skrypt o nazwie *nowy_plik*, należy utworzyć plik o takiej nazwie i rozszerzeniu .m. Do tego celu można użyć dowolnego sposobu opisanego powyżej. Po otwarciu okna Edytora zawierającego zawartość pliku *nowy_plik.m* można zauważyć, że plik jest pusty. Przyczyną jest fakt, że plik ten nie istniał wcześniej w systemie. Jeśli byłoby inaczej, wtedy istniejący plik o podanej nazwie zostałby otwarty do edycji. Po otwarciu okna Edytora/Debuggera należy w nim umieścić polecenia *Matlaba*, które w momencie wywołania skryptu mają być sekwencyjnie wykonywane.

Po wprowadzeniu treści skryptu do pliku i zapisaniu go w katalogu bieżącym, w każdym momencie możliwe jest wywołanie poleceń zawartych w tym skrypcie. Dokonuje się tego wpisując w Oknie Poleceń *Matlaba* nazwę m-pliku zawierającego skrypt. Po zakończeniu wykonywania skryptu, wszystkie zmienne utworzone w trakcie jego wykonywania pozostają w przestrzeni roboczej *Matlaba*. Mogą być dalej dowolnie wykorzystywane w innych skryptach bądź funkcjach. Naturalnie wewnątrz skryptu możliwe jest wywoływanie innych skryptów, pod warunkiem, że pliki je zawierające znajdują się w tym samym katalogu, co plik skryptu wywołującego.

1.1 W katalogu *Matlaba* utworzyć nowy folder o nazwie zawierającej numery indeksów wszystkich studentów z danej grupy laboratoryjnej. Nowo powstały folder ustawić jako folder bieżący.

1.2 Stworzyć nowy m-plik o nazwie *pa_cw2*. Z okna Historii Poleceń skopiować polecenia niezbędne do wykonania ćwiczeń z rozdziału 4 z Ćwiczenia 1. Wkleić komendy do skryptu. Jako pierwsze polecenie wpisać *clear all*, aby przed wykonaniem skryptu usunąć zawartość Workspace. Wywołać działanie skryptu i porównać zawartość Workspace przed i po wykonaniu skryptu.

1.2 Tworzenie funkcji

Jak wspomniano wcześniej, funkcje stanowią bardziej elastyczne i rozbudowane narzędzie rozszerzające możliwości środowiska Matlab. W odróżnieniu od skryptów, pozwalają one operować na zmiennych przekazanych jako parametry w momencie wywołania funkcji, a po zakończeniu działania mogą zwracać wartości. Ponadto funkcje, w przeciwieństwie do skryptów, operują na zmiennych lokalnych chyba, że zostały w miejscu ich tworzenia poprzedzone słowem kluczowym *global*.

Podobnie jak w przypadku skryptów, również funkcje umieszcza się w plikach tekstowych, posiadających rozszerzenie .m. Pierwsza linijka pliku definiującego nową funkcję jest ściśle określona i powinna mieć postać zbliżoną do wzorca przedstawionego poniżej:

```
function y = nazwa_funkcji(argument_wejsciowy)
```

Tworzenie nagłówka funkcji podlega następującym regułom:

- Nagłówek funkcji rozpoczyna się słowem kluczowym *function*.
- Kolejno następuje nazwa zmiennej, do której będzie zwracany rezultat działania funkcji w momencie jej zakończenia.
- Po nazwie zmiennej umieszcza się znak równości.
- Następnie podaje się nazwę funkcji, która musi być identyczna z nazwą m-pliku, w którym jest zapisana.
- W nawiasach podaje się nazwy zmiennych, które funkcja pobiera jako parametry wejściowe.

Jeżeli funkcja ma zwracać nie pojedynczą wartość, lecz np. macierz wartości, wówczas można ją przypisać do jednej zmiennej macierzowej lub każdy element nazwać osobno. To samo dotyczy sytuacji, gdy funkcja pobiera więcej niż jeden argument:

```
function [x, y, z] = nazwa_funkcji(argument1, argument2, argument3)
```

Jeżeli natomiast definiowana funkcja nie wymaga podania żadnych argumentów lub nie zwraca żadnego wyniku, wtedy odpowiednie pola w nagłówku funkcji można pominąć lub zastąpić nawiasami pustymi.

Kolejną ważną różnicą między m-plikiem zawierającym skrypt a tym, który zawiera definicję funkcji, jest możliwość zdefiniowania jaki komunikat będzie wyświetlany w Oknie Poleceń, gdy użytkownik wyda polecenie wyświetlenia pomocy związanej z dana funkcją:

```
help nazwa_funkcji
```

Wystarczy, w linii następującej po nagłówku funkcji, umieścić komentarz opatrzony znakiem specjalnym *%*. Przydatne skróty klawiszowe przy tworzeniu komentarzy: zaznaczenie odpowiednich linijek kodu i naciśnięcie *CTRL+R* spowoduje ich zakomentowanie, natomiast naciśnięcie *CTRL+T* usunie znak komentarza.

Przykład przedstawiony poniżej powinien wyjaśnić funkcjonowanie opisanego mechanizmu. Dla funkcji zapisanej w pliku *funkcja1.m* i zawartości:

```
function in = funkcja1(out)
% Pierwsza funkcja w Matlabie
in = out + 3;
```

wywołanie w Oknie Poleceń wygląda następująco:

```
>> b = funkcja1(5)
 b =
      8
```

W opisie funkcji, wyświetlanym po wpisaniu przez użytkownika polecenia *help* pojawi się następująca informacja:

```
>> help funkcja1
Pierwsza funkcja w Matlabie
```

Wykonywanie każdej funkcji w *Matlabie* można zatrzymać naciskając kombinacje klawiszy *CTRL+C*.

Jak wspomniano wcześniej zmienne wykorzystywane wewnątrz funkcji są lokalne. Na bazie poprzedniego przykładu: zmienne *in* oraz *out* nie zostają zapisane w Workspace po zakończeniu działania funkcji. Jeśli natomiast zmienne utworzone i modyfikowane wewnątrz funkcji mają być dostępne również po zakończeniu działania funkcji (np. wykorzystane w innych funkcjach), konieczne jest zadeklarowanie tych zmiennych z użyciem słowa kluczowego *global*. Należy użyć tego samego słowa kluczowego przy definiowaniu zmiennych w Oknie Poleceń w przypadku, gdy konieczne jest wykorzystanie danych zmiennych przez funkcję. Przykład funkcji korzystającej ze zmiennej globalnej:

```
function wysokosc = spadanie(t)
global GRAWITACJA
wysokosc = 1/2*GRAWITACJA*t.^2;
```

W takim przypadku należy w Oknie Poleceń zadeklarować zmienną *GRAWITACJA* oraz wywołać funkcję *spadanie*:

```
>> global GRAWITACJA
>> GRAWITACJA = 32
>> y = spadanie(0:0.1:5)
```

Pomimo, że przy tworzeniu zmiennych globalnych nie jest wymaganie stosowanie wielkich liter, ułatwia to rozróżnianie tego typu zmiennych od zmiennych lokalnych. Dzięki takiej deklaracji zmiennej globalnej, można zmieniać jej wartość jedynie w Workspace, bez konieczności zmian we wszystkich plikach funkcji, które ją wykorzystują.

Często użytkownik tworząc własne funkcje staje przed koniecznością określenia ilości argumentów pobranych i zwracanych przy wywołaniu funkcji. Służą do tego polecenia *nargin* oraz *nargout* – pierwsze z nich zwraca informację o ilości argumentów podanych przy wywołaniu funkcji, drugi natomiast pozwala określić ile wartości powinna funkcja zwrócić. Z kolei polecenia takie jak *varargin* oraz *varargout* stwarzają możliwość pobierania lub zwracania różnej liczby argumentów/wartości przez tą samą funkcję. Dokładny opis funkcji znajduje się w Tab. 1.

W każdym m-pliku możliwe jest umieszczenie tylko jednej funkcji głównej, której nazwa musi być identyczną z nazwą tego pliku, oraz nieograniczoną ilość tzw. podfunkcji (ang. subfunctions). Podfunkcje umieszcza się zawsze po treści funkcji głównej, a ich nazwy muszą różnić się od nazwy funkcji głównej. Sama struktura podfunkcji jest identyczna ze strukturą funkcji głównej, czyli zaczyna się od nagłówka, po którym następuje ciało funkcji (w tym wypadku podfunkcji). Różnicą między funkcją główną a podfunkcją jest to, że w przypadku tych drugich nie działa mechanizm uzyskiwania pomocy za pośrednictwem polecenia *help*.

1.3 Stworzyć nowy m-plik o nazwie *funkcja5_2* i zapisać go.

1.4 W Oknie Poleceń utworzyć nową zmienną globalną *DZIELNIK* o wartości 2. Zadeklarować funkcję o tej samej nazwie co m-plik, która przyjmuje dwa argumenty wejściowe, dodaje je do siebie i dzieli przez *DZIELNIK*.

1.5 Do funkcji dopisać komentarz wyjaśniający jej działanie tak, aby wyświetlał się on po wpisaniu komendy `help`.

1.6 Wywołać funkcję i przetestować jej działanie.

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
nargin, nargout	Y = nargin; Y = nargout; wewnątrz funkcji w m-pliku dostarcza informacji o ilości argumentów wejściowych/wyjściowych sprecyzowanych dla danej funkcji; Y = nargin(funkcja1); Y= nargout(funkcja1); poza m-plikiem funkcji dostarcza informacji o ilości argumentów wejściowych/ wyjściowych zadeklarowanych dla funkcji funkcja1; jeżeli funkcja1 ma zadeklarowaną zmienną ilość argumentów wejściowych/ wyjściowych wówczas funkcja zwraca wartość ujemną.
varargin, varargout	function y = funkcja1(varargin); tak stworzony nagłówek funkcji zapewni, że będzie mogła ona przyjmować zmienną liczbę argumentów wejściowych; zmienna varargin jest wykorzystywane tylko wewnątrz funkcji; function varargout = funkcja2(n); tak stworzony nagłówek funkcji zapewni, że będzie mogła ona zwracać zmienną liczbę argumentów wyjściowych; zmienna varargout jest wykorzystywane tylko wewnątrz funkcji.

Tablica 1: *Funkcje i zmienne pozwalające określić liczbę argumentów wejściowych i wyjściowych deklarowanych funkcji.*

1.3 Operatory arytmetyczne, logiczne i relacyjne

Operatory udostępniane w środowisku Matlab można, podobnie jak w przypadku operatorów w innych językach programowania wysokiego poziomu, podzielić na następujące grupy: operatory arytmetyczne, operatory relacyjne oraz operatory logiczne.

Operatory arytmetyczne zostały już dokładnie omówione w rozdziale opisującym operacje na macierzach (patrz opracowanie Ćwiczenia 1), dlatego ich opis nie będzie umieszczany w tej części opracowania.

Operatory relacyjne służą do porównywania odpowiadających sobie elementów macierzy o identycznych wymiarach chyba, że operacja dotyczy skalaru i zmiennej macierzowej. Operatory te zawsze działają element po elemencie. W miejscu, gdzie operatory zwracają wartość prawdziwą w macierzy wynikowej umieszczane są wartości 1, natomiast tam, gdzie relacja nie jest prawdziwa umieszczane są zera. W przypadku gdzie operator relacyjny użyty jest w odniesieniu do skalaru i macierzy, relacja stosowana jest między skalarem a każdym elementem składowym macierzy. Operatory zostały zebrane w Tab. 2.

Operatory logiczne w *Matlabie* mogą działać na elementach macierzy, skalarach oraz bitach elementów macierzy. Pełna lista operatorów jest dostępna po wpisaniu polecenia `help relop`. W niniejszym opracowaniu opisane zostaną jedynie dwa najważniejsze operatory logicznego AND oraz OR. Dokładny opis przedstawiono w Tab. 3. Przykładowo, zastosowanie operatorów logicznych może zabezpieczyć przed dzieleniem przez zero:

```
x = (b ~= 0) && (a/b > 18.5)
```

1.7 Stworzyć nowy m-plik o nazwie *funkcja5_3* i zapisać go.

1.8 Zadeklarować funkcję, która przyjmuje dwa argumenty wejściowe i sprawdza czy są sobie równe. Funkcja zwraca wartość 1 w przypadku, gdy są sobie równe oraz 0, gdy są różne. Wywołać funkcję i zweryfikować jej działanie.

Operator	Opis	Operator	Opis
A < B	Sprawdza czy elementy macierzy A są mniejsze od odpowiadających im elementów macierzy B.	A > B	Sprawdza czy elementy macierzy A są większe od odpowiadających im elementów macierzy B.
A <= B	Sprawdza czy elementy macierzy A są mniejsze bądź równe odpowiadającym im elementom macierzy B.	A >= B	Sprawdza czy elementy macierzy A są większe bądź równe odpowiadającym im elementom macierzy B.
A == B	Sprawdza czy elementy macierzy A są równe odpowiadającym im elementom macierzy B.	A ~= B	Sprawdza czy elementy macierzy A są różne od odpowiadających im elementów macierzy B.

Tablica 2: *Operatory relacyjne.*

Operator	Opis
A && B	Zwraca wartość 1 (prawdę) jeśli operandy A i B są prawdą lub nie są zerowe i 0 (fałsz) jeśli choć jeden z operandów jest fałszem lub zerem. Matlab sprawdza wartość B tylko w przypadku, gdy A nie decyduje w pełni o wyniku operacji, np. gdy A jest równe 0 wynik operacji jest równy 0, niezależnie od wartości B.
A || B	Zwraca wartość 1 (prawdę) jeśli choć jeden z operandów A i B jest prawdą lub nie jest zerowy i 0 (fałsz) jeśli oba operandy są fałszem lub zerem. Matlab sprawdza wartość B tylko w przypadku, gdy A nie decyduje w pełni o wyniku operacji, np. gdy A jest równe 1 wynik operacji jest równy 1, niezależnie od wartości B.

Tablica 3: *Operatory logiczne.*

1.9 Funkcję z poprzedniego podpunktu zmodyfikować tak, aby zwracała wartość 1 tylko w przypadku, gdy oba argumenty wejściowe są równe 3. Wywołać funkcję i zweryfikować jej działanie.

1.4 Sterowanie przepływem programu

W każdym języku programowania występują mechanizmy pozwalające na przerwanie sekwencyjnego wykonywania instrukcji i przenoszenie miejsca, z którego będą pobierane kolejne polecenia do wykonania. Poniżej zawarte zostały mechanizmy sterowania przepływem dostępne w środowisku programowania *Matlab*.

1.4.1 Instrukcja warunkowa if, else, elseif

```
if      A > B             % Jeśli A jest większe od B
        disp('Wieksze')   % Wyświetl informację dla użytkownika
elseif  A < B             % Jeśli A jest mniejsze od B
        disp('Mniejsze')
elseif  A == B            % Jeśli A jest równe B
        disp('Rowne')
else                      % W przeciwnym wypadku
        disp('Blad!')
end                       % Zakończ warunek
```

1.4.2 Instrukcja wyboru switch - case

```
switch     zmienna_ster
case       -1                   % Jeśli zmienna_ster jest równa -1
           A = -1*zmienna_ster  % Wykonaj polecenie
case       1                    % Jeśli zmienna_ster jest równa 1
           A = zmienna_ster     % Wykonaj polecenie
otherwise                       % W przeciwnym wypadku
           disp('Blad!')        % Wyświetl informację dla użytkownika
end                             % Zakończ instrukcję wyboru
```

1.4.3 Pętla for

```
for  i = 1:  1:  10  % Wykorzystanie operatora :
     A(i) = i;       % Przypisuj wartości do kolejnych elementów A
end                  % Zakończ pętlę
                     % Nie ma konieczności wcześniejszego
                     % deklarowania rozmiaru wektora A
```

1.4.4 Pętla while

```
n = 1
while  n < 4       % Dopóki n jest mniejsze od 4
       n = n + 1;  % Zwiększaj n o 1
end                % Zakończ pętlę
                   % Po wykonaniu pętli n = 4
```

1.4.5 Instrukcje continue i break

Instrukcje te służą do przerywania wykonywania instrukcji występujących w pętli po instrukcjach *continue* lub *break* i przeniesienia sterowania do:

- pierwszej instrukcji w pętli w przypadku napotkania instrukcji *continue*;
- pierwszej instrukcji za pętlą (słowem *end* kończącym daną pętlę) w przypadku wystąpienia instrukcji *break*.

```
n = 1
for    i=1:1:5            % Wywołaj pętlę 5 razy
       if i>=3            % Jeżeli i jest większe lub równe 3
                continue  % Powoduje wywołanie kolejnej iteracji
       end                % pętli bez zwiększania n
       n=n+1;
end                       % i = 5; n = 3;
```

```
n = 1
for    i=1:1:5         % Wywołaj pętlę 5 razy
       if i>=3         % Jeżeli i jest większe lub równe 3
                break  % Powoduje wywołanie przerwanie wykonywania
       end             % pętli bez zwiększania n
       n=n+1;
end                    % i = 3; n = 3;
```

1.4.6 Instrukcja return

Wystąpienie tej instrukcji w kodzie funkcji powoduje zakończenie wykonywania tej funkcji i powrót, albo do funkcji wywołującej daną funkcję zawierającą instrukcję *return*, albo do konsoli w sytuacji, gdy instrukcja *return* wystąpiła w funkcji nadrzędnej, wywołanej bezpośrednio z Okna Poleceń *Matlaba*.

```
function d = det(A)          % Funkcja obliczająca wyznacznik macierzy
if isempty(A)                % Jeżeli macierz jest pusta
                     d = 1   % Ustaw wartość d na 1
                     return  % Zakończ wykonywanie funkcji det
else
                     ...
end
```

1.4.7 Instrukcja pause

Wykorzystanie instrukcji *pause* w kodzie funkcji spowoduje chwilowe zatrzymanie wykonywania m-pliku do momentu, w którym użytkownik przyciśnie dowolny klawisz. Wywołanie funkcji z argumentem wejściowym:

```
pause(n)
```

zatrzymuje wykonywanie programu na n sekund, przy czym n może przyjmować jedynie wartości dodatnie.

1.10 Zadeklarować funkcję, która pobiera jako argument wejściowy macierz dwuwymiarową 4×5. Funkcja powinna zwracać dwa argumenty wyjściowe:

- pierwszy zależny od tego czy macierz wejściowa jest odpowiednich rozmiarów; argument powinien być równy 1 dla macierzy rozmiaru 4×5, 0 w przeciwnym wypadku;
- drugi zależny od wartości elementów macierzy wejściowej; argument powinien być równy 1 jeżeli wszystkie elementy macierzy wejściowej są większe niż 5, 0 w przeciwnym wypadku.

Przy tworzeniu funkcji należy wykorzystać instrukcję *if – else* oraz nie należy używać instrukcji *for*.

1.11 Zadeklarować funkcję, która pobiera jako argument wejściowy łańcuch znaków i w zależności od jego postaci wyświetla odpowiedni komunikat dla użytkownika:

- dla argumentu wejściowego 'dodawanie' wyświetli +;
- dla argumentu wejściowego 'odejmowanie' wyświetli -;
- dla argumentu wejściowego 'mnożenie' wyświetli *;
- dla argumentu wejściowego 'dzielenie' wyświetli /.

Przy tworzeniu funkcji należy wykorzystać instrukcję *switch – case*.

1.12 Zadeklarować funkcję, która pobiera jako argument wejściowy macierz dwuwymiarową o dowolnym rozmiarze i zlicza ile spośród elementów macierzy znajduje się w przedziale $(4; 7)$. Funkcja powinna zwracać jako argument wyjściowy liczbę elementów znajdujących się w przedziale (należy użyć pętli *for* oraz funkcji *size*).

1.13 Zmodyfikować funkcję z poprzedniego podpunktu tak, aby przerywała zliczanie elementów, gdy doliczy do 5.

2 Graficzna prezentacja danych

Niewątpliwą zaletą środowiska *Matlab*, oprócz olbrzymich możliwości obliczeniowych, jest rozbudowany system graficznej prezentacji danych. Dzięki niemu, wartości składające się na zmienne wektorowe i macierzowe mogą być prezentowane w sposób najbardziej czytelny dla użytkownika. *Matlab* udostępnia szereg technik służących do przedstawiania danych w sposób graficzny. Narzędzia interaktywne oraz dedykowane funkcje pozwalają na modyfikowanie wykresów tak, aby możliwe było odczytanie z nich jak największej ilości informacji. Istnieje również możliwość wprowadzania adnotacji do przedstawionych danych oraz eksportowanie wykresów do wielu popularnych formatów plików graficznych.

2.1 Tworzenie wykresów

Pierwszym krokiem tworzenia wykresu jest utworzenie zmiennych macierzowych, które mają zostać wyświetlone na wykresie, np:

```
>> t = 0 :  0.01 :  10;
>> x1 = sin(t);
>> x2 = cos(t);
```

Poniżej opisano dwa sposoby tworzenia wykresów oraz określania zmiennych, które maja zostać przedstawione w formie graficznej.

2.1.1 Wykorzystanie interaktywnych narzędzi

Aby uruchomić narzędzie do tworzenia wykresów w sposób interaktywny, należy w Oknie Poleceń wpisać polecenie

```
>> plottools
```

Polecenie powoduje otwarcie nowego okna zawierającego wszystkie narzędzia niezbędne do tworzenia i edycji wykresu. Aby zdefiniować zmienne tworzące wykres, należy w nowo otwartym oknie w zakładce *Variable* wybrać zmienną do przedstawienia na wykresie. Można zauważyć pojawienie się nowych opcji w oknie poniżej okna wykresu. Aby określić względem jakiej zmiennej ma być przedstawiony wykres, należy wypełnić prawidłowo pola *X Data Source* i *Y Data Source*. Przykładowo, dla zmiennych zadeklarowanych wyżej, w polu *X Data Source* należy umieścić zmienną t, natomiast w polu *Y Data Source – x1*.

W ten sam sposób można dodawać kolejne zmienne do wykreślenia.

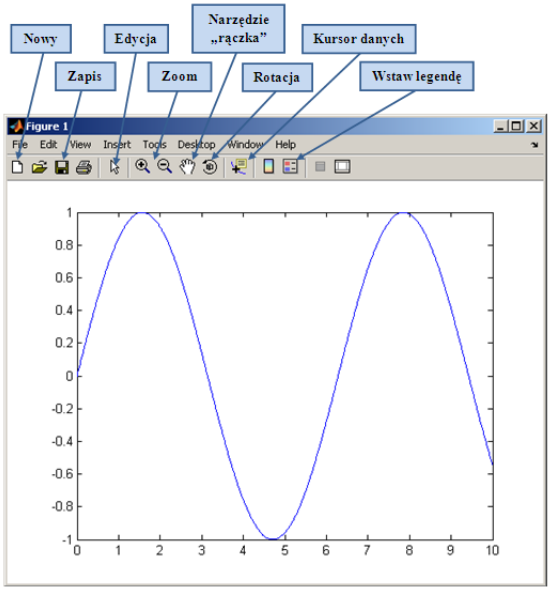
2.1.2 Wpisywanie komend w Oknie Poleceń

Innym sposobem tworzenia wykresu oraz deklarowania wyświetlanych zmiennych jest wpisywanie dedykowanych komend w Oknie Poleceń. Otwarcie nowego okna wykresu (patrz Rys. 1) następuje po wpisaniu polecenia

```
>> figure
```

Następnie, aby wybrać zmienne do zaprezentowania na wykresie, należy powrócić do Okna Poleceń i użyć funkcji `plot`. Przykładowo, dla zmiennych zadeklarowanych wyżej:

```
>> plot(t, x1)
```



Rysunek 1: *Okno wykresu.*

W wyniku opisanych operacji zostanie otwarte okno przedstawione na Rys. 1.

Użycie funkcji `plot`, bez wcześniejszego zastosowania komendy `figure`, w przypadku, gdy nie ma żadnego otwartego okna wykresu, spowoduje otwarcie nowego okna. Z komendy `figure` należy korzystać, gdy wykres ma zostać utworzony w nowym oknie, przy założeniu, że inne okna wykresu są już otwarte. Domyślne ustawienia sprawiają, że każde wywołanie funkcji `plot` dla tego samego okna wykresu, powoduje wyczyszczenie poprzednich przebiegów i wykreślenie w ich miejscu nowych wykresów. Jeśli jednak zachodzi konieczność zachowania poprzedniej zawartości okna graficznego i naniesienie na nim dalszych przebiegów, wtedy wygodne jest posłużenie się komendą:

```
>> hold on
```

Po jej wydaniu wszystkie kolejne wywołania funkcji `plot` spowodują kreślenie w obrębie ostatniego okna graficznego, aktywnego przed wywołaniem funkcji `hold on`.

```
>> plot(t, x2);
```

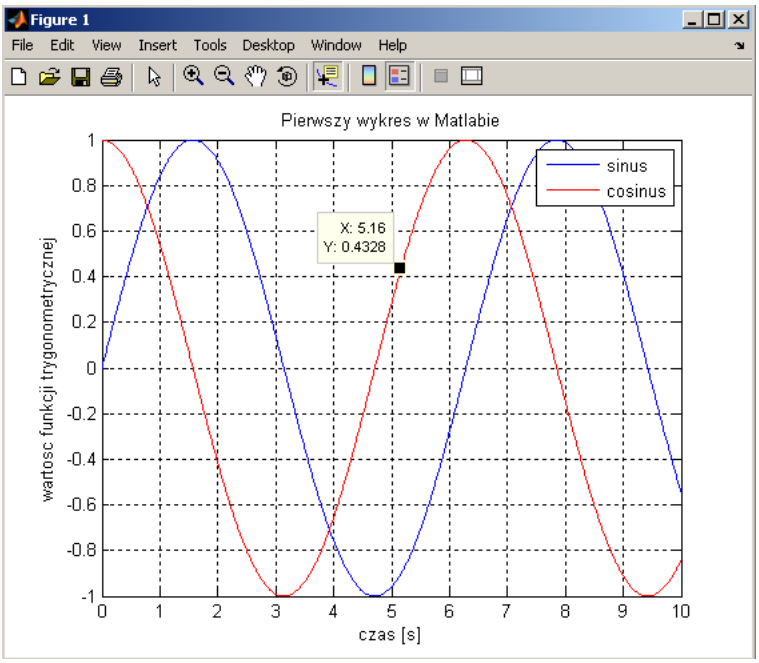
Aby przywrócić poprzednie, domyślne czyszczenie wykresów, należy użyć polecenia `hold off`.

2.2 Edycja wykresów

Po utworzeniu wykresu możliwa jest jego edycja. Modyfikacje obejmują skalowanie i obrót wykresu, umieszczanie elementów graficznych (takich jak linie, strzałki, opisy tekstowe, legenda, skala kolorów, opisy osi) oraz kopiowanie, zapisywanie, drukowanie i eksport utworzonej grafiki.

Modyfikacja może być wykonana przy wykorzystaniu, tych samych co w przypadku tworzenia wykresu, dwóch sposobów. Sposób edycji wykresów można dobierać niezależnie od tego, w jaki sposób wytworzony został wykres, gdyż wynikowe okno jest tej samej postaci. Połączenie dwóch podejść może okazać się niezwykle przydatne. Na przykład, możliwe jest wytworzenie wykresu za pomocą funkcji `plot` oraz późniejsza edycja przy pomocy narzędzi interaktywnych.

Przykład okna wykresu, wykorzystującego dane zadeklarowane w rozdziale 2.1, po procesie modyfikacji przedstawiono na Rys. 2.



Rysunek 2: *Przykład zmodyfikowanego wykresu.*

2.2.1 Wykorzystanie interaktywnych narzędzi

Sposób wykorzystania interaktywnych narzędzi do edycji wykresów zostanie opisany na przykładzie wykresu wytworzonego za pomocą funkcji `plot`. Opisane tu metody mają swoje zastosowanie również w przypadku utworzenia wykresu drugim sposobem.

Do utworzonego wykresu można dodać tytuł, legendę, opisy osi oraz oznaczenia danych (przy użyciu strzałek, podstawowych prymitywów geometrycznych oraz tekstu). Wszystkie opisane elementy znajdują się w zakładce menu *Insert*.

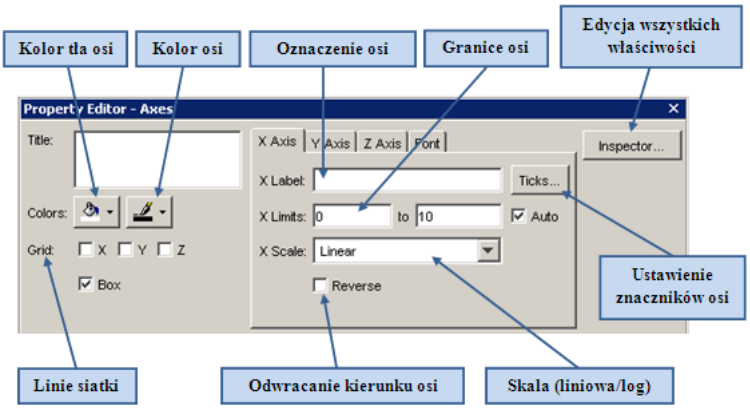
W celu edytowania wykresu, po otwarciu nowego okna, należy wybrać przycisk *Edycji* na pasku narzędzi (Rys. 1). Wszystkie funkcje i opcje związane z właściwościami wykresu dostępne są po kliknięciu wykresu prawym przyciskiem myszy i wybranie z menu podręcznego, które się ukaże, opcji *Properties...*. Operacje te powodują otwarcie okna Edytora Właściwości (Properties Editor...), w którym zmieniać można wszystkie opcje wykresu. W podobny sposób wywołać można Edytor Właściwości dla każdego obiektu graficznego widocznego w oknie graficznym (linii wykresu, osi, legendy, tytułu, itd.) – w tym celu należy kliknąć prawym przyciskiem myszy element wykresu, którego właściwości mają być modyfikowane i z otwartego w ten sposób menu podręcznego wybrać opcję *Properties...*. Przykładowe okno, służące do edycji właściwości osi wykresu przedstawiono na Rys. 3.

2.2.2 Wpisywanie komend w Oknie Poleceń

Już podczas wywoływania odpowiednich funkcji w Oknie Poleceń, służących do tworzenia wykresów, można określić pewne właściwości wykresu. Wystarczy jako trzeci argument funkcji `plot`, podać łańcuch znaków `LineSpec`:

```
>> plot(x, y, 'LineSpec');
```

Łańcuch może określać styl linii, kolor oraz rodzaj znaczników. Znaczenie poszczególnych znaków zebrano w Tab. 4.



Rysunek 3: *Okno Edytora Właściwości osi wykresu.*

LineSpec	Kolor wykresu	LineSpec	Znacznik	LineSpec	Styl linii
b	niebieski	.	punkt	-	ciągła
g	zielony	o	okrąg	:	kropkowana
r	czerwony	x	krzyżyk	-.	kropkowana - kreskowana
c	niebiesko - zielony	+	plus	–	kreskowana
m	purpurowy	*	gwiazdka	brak	brak linii
y	żółty	s	kwadrat		
k	czarny	d	romb		
w	biały	v	trójkąt		

Tablica 4: *Lista funkcji służących do edycji wykresów.*

Podczas podawania właściwości linii wykresu można podać jeden styl linii, jeden kolor oraz jeden ze znaczników. Możliwe jest również podanie tylko jednego z parametrów, np.

```
>> plot(x, y, 'r');
```

tworzy wykres, którego linia ma kolor czerwony, natomiast:

```
>> plot(x, y, 'r+:');
```

tworzy wykres, którego linia jest czerwona i kropkowana, a znaczniki przyjmują postać plusów.

Wywołując funkcję plot można również zmienić wszelkie pozostałe parametry wykresu. Wystarczy podać jako dodatkowe argumenty dowolną liczbę par parametr- wartość:

```
>> plot(x, y, 'NazwaWlasciwosci', WartoscWlasciwosci, ...);
```

Listę nazw właściwości jaką można podać w polu *NazwaWlasciwosci* można odnaleźć po naciśnięciu przycisku *Inspector* w oknie Edytora Właściwości (Rys. 3). Na przykład polecenie:

```
>> plot(x, y, 'LineWidth', 2);
```

utworzy wykres, którego linia ma dwa piksele szerokości. Te same właściwości mogą być edytowane już po wywołaniu funkcji plot:

```
>> set(gcf, 'NazwaWlasciwosci', WartoscWlasciwosci, ...);
```

przy czym skrót *gcf* oznacza uchwyt do aktualnego okna wykresu[1].

[1] Dokładny opis użycia funkcji `set` można znaleźć wpisując komendę `help set` w Oknie Poleceń.

Listę funkcji służących do tworzenia wykresów oraz dodawania elementów graficznych, zebrano w Tab. 5.

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
plot(y)	Wykreśla wartości wektora y, umieszczając na osi x kolejno indeksy tych elementów.
plot(x1, y1, x2, y2, ...)	Wykreśla przebiegi zdefiniowane przez pary elementów z par wektorów (*x1*, *y1*), (*x2*, *y2*) itd.
plot(x, y, LineSpec)	Wykreśla przebieg zmiany parametru x względem parametru y. Łańcuch znaków *Linespec* pozwala na określenie formatu linii oraz jej koloru, przy użyciu których tworzony będzie wykres; plot(x, y, 'r+').
plot (x, y, 'Nazwa', Wartosc, ...)	Ustawia wartość własności o nazwie *Nazwa* na wartość określoną przez parametr *Wartosc;* plot(x, y, 'LineWidth', 2).
set	set(h, 'Nazwa', Wartosc); ustawia wartość własności o nazwie *Nazwa* na wartość określoną przez parametr *Wartosc* dla obiektu o uchwycie h; *gcf* oznacza uchwyt aktualnego okna wykresu; set(gcf, 'LineWidth', 2).
get	V = get(h, 'Nazwa'); zwraca wartość własności o nazwie *Nazwa* dla obiektu o uchwycie h; *gcf* oznacza uchwyt aktualnego okna wykresu; get(gcf, 'LineWidth').
figure	Otwiera nowe okno wykresu.
grid on	Włącza wyświetlanie głównych i pomocniczych linii siatki.
axis	axis([xmin xmax ymin ymax]); ustawia zakresy wartości x oraz y wyświetlane na aktualnym wykresie; axis([0 10 1 2]);
xlabel, ylabel	xlabel(str); ylabel(str); dodaje opis, zapisany w postaci łańcucha znaków *str*, pod osią x/ obok osi y; xlabel('czas'); ylabel('wartosc'). xlabel(str, 'Nazwa', Wartosc, ...); ylabel(str, 'Nazwa', Wartosc, ...); dodaje opis osi x/ y oraz ustawia właściwość *Nazwa* na *Wartosc*.
legend	legend(str1,str2,str3, ...); umieszcza legendę zamieszczonych na wykresie przebiegów; *str1* jest łańcuchem znaków opisującym pierwszy dodany do wykresu przebieg, *str2* – drugi, itd; legend('Pierwszy', 'Drugi', 'Trzeci'); legend(..., 'Location', LOC); umieszcza legendę w określonej pozycji *LOC* na wykresie; dokładny opis definiowania parametru *LOC* można odnaleźć po wpisaniu komendy `help legend` w Oknie Poleceń.
title	title(str); dodaje opis wykresu zawarty w łańcuchu znaków str; title('Tytul'); title(str, 'Nazwa', Wartosc, ...) dodaje opis wykresu oraz ustawia właściwość *Nazwa* na *Wartosc*.
text	text(x, y, str); dodaje tekst *str* na wykresie, umieszczony w pozycji (x, y), gdzie(x, y) są wyrażone w jednostkach aktualnego wykresu; text(0.5, 0.5, 'Wartosc graniczna');

Tablica 5: *Lista funkcji służących do edycji wykresów.*

Jeżeli konieczne jest wytworzenie kilku wykresów z tymi samymi ustawieniami grafiki, przydatne może okazać się utworzenie **skryptu**.

2.1 Na bazie danych zadeklarowanych na początku rozdziału, stworzyć za pomocą narzędzi interaktywnych tworzenia wykresu, wykres zaprezentowany na Rys. 2.

2.2 Napisać skrypt tworzący wykres zaprezentowany na Rys. 2. Zamiast kursora danych, zastosować wyświetlanie napisu (polecenie *text*).

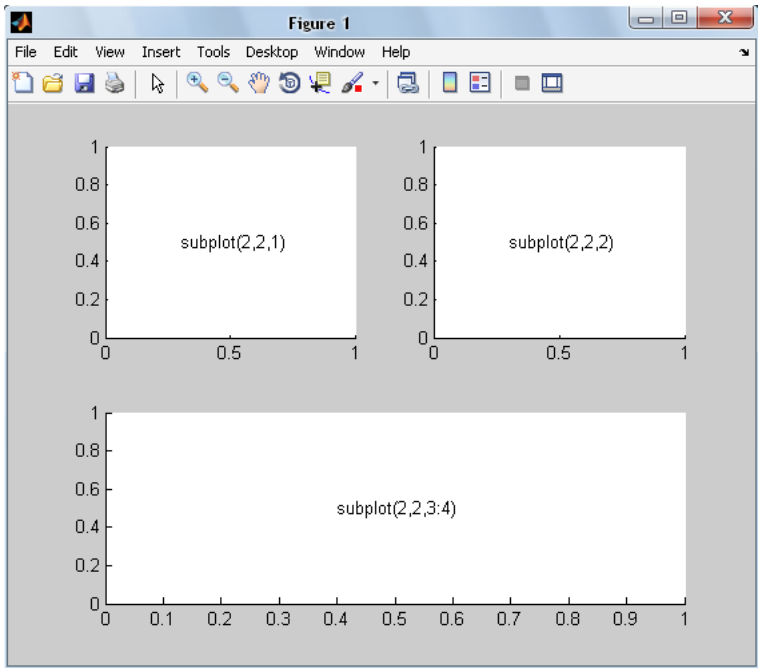
2.3 Wykorzystując polecenie *set*, zmienić kolor tła wykresu na szary oraz ustawić styl linii na kropkowaną.

2.2.3 Umieszczanie kilku przebiegów na wspólnym wykresie

W *Matlabie* możliwe jest również rozmieszczenie wielu wykresów, o różnych osiach wykresu, w jednym oknie graficznym. Wykorzystanie dedykowanej funkcji *subplot* opisane jest następująco:

```
>> subplot(ile_pionowo, ile_poziomo, ktory_wykres)
```

gdzie *ile_pionowo* określa na ile rzędów okno wykresu będzie podzielone (liczba wierszy), natomiast *ile_poziomo* definiuje na ile części okno będzie podzielone poziomo (liczba kolumn). Parametr *ktory_wykres* informuje, do której (do których) części z podzielonego wykresu następuje odwołanie, przy czym numeracja rozpoczyna się od 1 i rośnie w miarę przemieszczania się w prawo oraz w dół po częściach podzielonego wykresu. Przykładowy podział okna wykresu, wraz z odpowiadającymi parametrami funkcji subplot, przedstawiono na Rys. 4.



Rysunek 4: *Umieszczanie wielu wykresów w jednym oknie.*

Po określeniu fragmentu wykresu do jakiego następuje odwołanie, należy określić jaka operacja ma być wykonana, np.:

```
>> subplot(2, 2, 1), plot(x, y)
>> subplot(1, 2, 2), text(0.5, 0.5, 'Wykres 2')
>> subplot(2, 1, 1), set(gca, 'Color', 'r')
```
[2]

Często, w przypadku prezentacji na wspólnym wykresie zbiorów wartości znacznie różniących się zakresem wartości, wygodne jest umieszczenie dwóch osi y, po jednej dla każdej serii danych. W takim przypadku jedna oś umieszczana jest po lewej stronie wykresu, a druga – po jego prawej stronie. W *Matlabie* do tworzenia tego typu wykresów przewidziana jest funkcja *plotyy*, którą można wywołać w następujący sposób:

```
>> plotyy(x1,y1,x2,y2)
```

Powyższa funkcja tworzy wykres zmiennej *y1* względem *x1* z opisem osi y po lewej stronie, oraz *y2* względem *x2* z opisem osi y po prawej stronie.

2.4 Zadeklarować 5 wektorów o długości równej 10 (wartości dowolne).

2.5 Podzielić okno wykresu na 3 rzędy oraz 2 kolumny. Wykreślić na osobnych wykresach (w ramach tego samego okna) wartości wektorów w zależności od ich indeksu. Wykres wartości ostatniego wektora powinien znajdować się w najniższym rzędzie wykresu i zajmować 2 części podzielonego okna wykresu.

2.6 Zmienić kolor linii wykresu wartości wektora drugiego i trzeciego na zielony. Do wszystkich wykresów dodać opis osi y: "Wartosc wektora". Zadanie wykonać przez wpisanie odpowiednich komend w Oknie Poleceń.

2.3 Wykresy 3D

Odpowiednikiem funkcji `plot` dla wykresów trójwymiarowych jest funkcja `plot3`. Sposób wywołania i możliwość edycji właściwości wykresu podlegają tym samym zasadom:

```
>> plot3(x, y, z, 'NazwaWlasciwosci', WartoscWlasciwosci, ...);
```

Jedyną różnicą jest dodatkowy parametr z, określający trzecią zmienną przedstawianą w postaci graficznej.

Istnieje również szereg funkcji służących do prezentacji macierzy w postaci płaszczyzny trójwymiarowej. Najważniejszymi są funkcje *mesh* oraz *surf*.

2.7 Zadeklarować zmienną t postaci:

```
t = 0 :  pi/50 :  10*pi
```

Poleceniem plot3 utworzyć wykres trójwymiarowy *sin(t)* - *cos(t)* - *t* z włączonymi liniami siatki.

2.8 Zapoznać się z zastosowaniem funkcji surf (`help surf`) i przy jej użyciu wytworzyć trójwymiarowy wykres.

[2] *gca* oznacza uchwyt do aktualnych osi wykresu.