Wybrane elementy syntaktyki pakietu Matlab

© Przemysław Korohoda
Katedra Elektroniki, AGH

Przedstawiony poniżej opis zawiera informacje konieczne i przydatne do przeprowadzania symulacji z zakresu przetwarzania sygnałów. W ramach samodzielnych studiów można ten zakres wiedzy znacznie poszerzyć.

Uwaga - informacje oznaczone symbolem ↓ powinny stać się zrozumiałe po przeczytaniu późniejszego fragmentu.

1. Sposób zapisu i interpretacji zmiennych; edytowanie komend

Liczby mogą być zapisywane jako dziesiętne - **z kropką w roli przecinka**, natomiast wyświetlane są z narzuconą precyzją, zazwyczaj z dokładnością 4 cyfr znaczących po przecinku. W przypadku, gdy liczba jest mała, stosowany jest mnożnik w postaci „10 do odpowiedniej ujemnej potęgi", np.: **3.5600e-4.**

Format (sposób) wyświetlania liczb można zmieniać za pomocą komendy **format** ↓. Należy jednak pamiętać, iż każdy z formatów ma swoje zalety i wady, zatem należy je dobierać tak, by najlepiej odpowiadały naszym potrzebom - np. dokładne wyświetlanie liczb umożliwia ich lepsze porównanie w przypadku kilku liczb, jednak na ekranie komputera może się wtedy zmieścić mniej liczb jednocześnie, zatem trudniej nam będzie oszacować np. rozmieszczenie zer w jakiejś macierzy.

Zmienne traktowane są jak macierze o określonych wymiarach, czyli ilości wierszy i kolumn.

Pojedyncza liczba - skalar - jest macierzą o wymiarach 1x1.

Każdy element macierzy może być określany za pomocą jego współrzędnych w macierzy, czyli indeksów.

Przykładowo jeżeli mamy macierz o nazwie A i wymiarach 3x4, to jej element w drugim wierszu i trzeciej kolumnie może być wskazany jako: **A(2,3)**.

Bardzo przydatny jest operator „:" (dwukropek), który w przypadku zapisu: **2:6** powoduje, iż zapis ten oznacza ciąg liczb naturalnych od 2 do 6.

Operator „:" może być także stosowany z podaniem kroku, np.: **0.9:-0.2:0** , co odpowiada zapisowi ciągu 0.9,0.7,0.5,0.3,0.1 (celowo pokazano, jak interpretowane jest podanie wartości kończącej ciąg tak, iż nie odpowiada ostatniemu elementowi ciągu) - istnieje także inne zastosowanie tego symbolu, o czym będzie mowa przy opisie macierzy;

Należy pamiętać, że indeksy elementów macierzy zawsze rozpoczynają się od „1" (a nie od „0"), i że muszą być całkowite.

Uwaga (!!!) - nie wolno mylić zapisu **xyz(1:3)** oznaczającego pierwsze trzy elementy wektora **xyz** z zapisem **xyz(1,3)**, który oznacza element z pierwszego wiersza i trzeciej kolumny zmiennej **xyz**, która musi być w tym przypadku macierzą (z co najmniej trzema kolumnami) lub wektorem wierszowym.

Przykłady:

Alfa(2:3,3:-1:1) - oznacza fragment macierzy Alfa zawierający drugi i trzeci wiersz oraz kolumny od 1 do 3 ale **w odwróconej kolejności**; macierz Alfa musi być naturalnie na tyle duża, by takie elementy zawierać.

X(4:-1:1)=Alfa(3,1:4) - oznacza przypisanie ↓ elementom wektora wierszowego (czyli macierzy składającej się z jednego wiersza) w odwrotnej kolejności pierwszych czterech elementów trzeciego wiersza macierzy Alfa.

v(5:-1:1)=v - jeżeli **v** był wektorem 5-elementowym, to zapis taki spowoduje odwrócenie kolejności elementów tego wektora, gdyby wektor miał inną liczbę elementów program zasygnalizuje błąd.

v=v(5:-1:1) - wybieramy pierwsze pięć elementów wektora **v**, które w odwróconej kolejności tworzą nowy wektor **v**. Jeżeli wektor ten był dłuższy, to po podstawieniu pozostałe elementy zostają utracone.

Zmiennych nie definiuje się przed użyciem, powstają one w wyniku użycia operatora przypisania „=" (znak równości). Przykładowo:

```
>> x5=3;
>> Alfa_12=125.64;
>> delta_x=15:0.1:-15;
```

Naturalnie po prawej stronie operatora przypisania mogą znaleźć się jedynie istniejące zmienne - tj. stworzone przez nas w danej sesji, przywołane z pliku lub predefiniowane (jak np. **pi** lub **j**).
Zmienne muszą mieć nazwy rozpoczynające się od liter.

W nazwach litery duże i małe są rozróżniane - zatem np. **xalfa** oraz **Xalfa** to dwie różne zmienne.

Nazwy mogą zawierać cyfry, a także symbol podkreślenia, jednak **nie na pierwszej pozycji**.

Zmienne są usuwane z pamięci operacyjnej po zamknięciu pakietu Matlab lub po zastosowaniu komendy **clear**. Komenda **clear** bez podania nazwy zmiennej powoduje usunięcie z pamięci wszystkich zmiennych użytkownika powstałych w danej sesji. Chcąc usunąć zmienne o nazwach **x**, **Alfa**, **wektor1** stosujemy komendę podając nazwy zmiennych oddzielone spacją:

```
>> clear x Alfa wektor1      % przykład użycia komendy clear
```

W powyższym przykładzie znak **„>>”** oznacza tzw. znak zachęty, oznaczający gotowość pakietu na przyjęcie kolejnej komendy, natomiast znak **„%”** w danej linii powoduje, iż następujący po nim fragment linii traktowany jest jako komentarz.
Wykonanie danej komendy następuje po podaniu z klawiatury znaku [Enter]. Zatem wykreowanie np. wektora 5 elementowego o wartościach od 3 do 7 nastąpi po następującej komendzie:

```
>> x5=3:7;[Enter]
```

Warto zwrócić uwagę na znak „;” (średnik), który na końcu komendy oznacza, iż nie chcemy, by wynik jej wykonania został wyświetlony na ekranie komputera. Niekiedy chcemy się upewnić, co będzie wynikiem danej komendy i znak ten pomijamy, jednak w przypadku operacji wykonywanej na zmiennej dużych rozmiarów pomijanie znaku „;” jest raczej mało użyteczne.
W dalszych przykładach akceptacja komendy klawiszem [Enter] będzie traktowana domyślnie, zatem nie będzie wypisywana. Dopóki nie podamy znaku akceptacji ([Enter]) linię komendy można edytować korzystając z klawiszy poziomego ruchu kursora, kasowania itp.
Klawisze pionowego ruchu kursora umożliwiają przywoływanie w linii bieżąco edytowanej komendy poprzednich komend zapamiętanych w buforze, które można odpowiednio zmodyfikować (lub pozostawić w stanie pierwotnym) i ponownie zaakceptować do wykonania.
Po uruchomieniu pakietu w pamięci znajdują się wytworzone wstępnie zmienne o nazwach: **pi, i, j**. Pierwsza z nich to liczba pi, a kolejne to dwie identyczne zmienne zawierające jednostkę urojoną - dla liczb zespolonych).
Jeżeli jednak podamy komendę, np.:

```
>> pi=4:-1:1;
```

to zmienna pi stanie się 4-elementowym wektorem zawierającym liczby od 4 do 1 (z krokiem minus 1), a poprzednia zawartość zmiennej zostanie (chwilowo) utracona. W taki sam sposób można zastępować także inne zmienne, tworząc je na nowo za pomocą operatora przypisania.
Zmienna **ans** zawiera zawsze wyniki operacji która może być użyta po prawej stronie operatora przypisania, a której użyjemy bez tego operatora. Każde kolejne zapełnienie tej zmiennej powoduje likwidację jej poprzedniej zawartości. Przykładowo. podając kolejno dwie komendy uzyskamy następujący wynik:

```
>> 3:-0.1:-2;    % zmienna ans będzie zawierała wektor wyspecyfikowanych wartości
>> 3      % zmienna ans stanie się skalarem o wartości „3”
```

Zmienną tą można wykorzystywać w późniejszych operacjach tak samo jak inne zmienne - należy jednak przy tym uważać, czy zmienna **ans** zawiera takie dane, jakich się w niej spodziewamy, ponieważ bardzo łatwo można ją niechcący „nadpisać”.

Niektóre przydatne funkcje

Funkcja	Opis
quit	zakończenie pracy z pakietem Matlab (zamknięcie wszystkich okien pakietu i skasowanie danych)
clear	usuwanie z pamięci wszystkich zmiennych lub wskazanej zmiennej
who	wyświetla nazwy wszystkich zmiennych istniejących w pamięci, tzn. stworzonych w trakcie danej sesji lub przywołanych z pliku
whos	j.w. oraz ponadto pokazuje wymiary każdej ze zmiennych
help	wyświetla opis podanej funkcji lub słowa kluczowego
format	zmienia format wyświetlania liczb (najczęściej używane opcje to: **short** i **long**)
length	zwraca ilość elementów wektora (wierszowego lub kolumnowego)
size	zwraca dwuelementowy wektor (dwie liczby) - liczbę wierszy i liczbę kolumn danej macierzy

Funkcje **who** oraz **whos** umożliwiają np. sprawdzenie, jakie nazwy zmiennych są już wykorzystane. Funkcja **whos** pozwala także na sprawdzenie wymiaru macierzy - np. w celu sprawdzenia dlaczego Matlab nie wykonuje podanego polecenia (przyczyną może być niezgodność wymiarów macierzy). Do podobnych celów może służyć funkcja **size**. Parametry wyliczane przez funkcje **size** oraz **length** mogą być wykorzystane w późniejszych obliczeniach.

Warto podkreślić różnice pomiędzy np. funkcją **help** i **size**. W przypadku funkcji **help** parametr (czyli w tym przypadku nazwę funkcji lub słowa kluczowego) podajemy po spacji, bez nawiasów, np.:

```
>> help function

>> help sin
```

Funkcja ta nie zwraca także żadnego parametru.
Z kolei funkcja **size** zwraca wektor wierszowy, a ponadto parametr w postaci nazwy zmiennej podaje się w nawiasach okrągłych (spacja oddzielająca słowo **size** od lewego nawiasu jest tu wprawdzie możliwa, ale raczej utrudnia odczytywanie takiego zapisu i dlatego nie należy jej stosować). Przykładowo ↓:

```
>> M=size(Atest)    % odczytujemy wektor dwóch liczb: wierszy i kolumn
```

albo

```
>> size(Atest)      % wynik - czyli dwuelementowy wektor wierszowy - w zmiennej ans
```

Zawsze należy zatem zwrócić uwagę, jak należy posługiwać się wywołaniem danej funkcji - przykłady zapewnia funkcja **help**.

3. Operatory arytmetyczne

Operatory arytmetyczne (odnoszą się do macierzy!):

Operator	Opis
*	mnożenie (dla macierzy nie jest przemienne)
/	dzielenie
\	dzielenie lewostronne (ważne w przypadku macierzy)
+	dodawanie
-	odejmowanie
^	potęgowanie
‘	transponowanie (UWAGA: w przypadku macierzy zespolonych nie jest to tylko zamiana wierszy i kolumn, ale dodatkowo sprzężenie)
.* .^ ./	kropka przed operatorem powoduje wykonanie operacji element po elemencie - mnożenia, potęgowania lub dzielenia

W przypadku niezgodności wymiarów macierzy podanych jako operandów dla danego operatora operacja nie jest realizowana, a program sygnalizuje błąd (warto wtedy zrozumieć komunikat o błędzie w języku angielskim...). Poniższy przykład tworzenia macierzy Atest demonstruje, w jaki sposób definiujemy macierze (lub wektory) posługując się nawiasami kwadratowymi oraz separatorami wierszy - przecinkami - i separatorami kolumn - średnikami:

```
>> Atest=[3,3,3;1,2,5;3:-1:1];   % Atest to macierz 3x3, pierwszy wiersz to trzy „3”
>> Btest=Atest^2;                % Btest jest macierzą Atest podniesioną do kwadratu.
>> Ctest=Atest.^2;               % każdy z elementów Ctest jest odpowiednim elementem
                                 % macierzy Atest podniesionym do kwadratu.
```

4. Funkcje do tworzenia macierzy (i wektorów)

W Matlabie można w wygodny i szybki sposób tworzyć macierze (wektory i skalary). Przykładowe funkcje służące do tego celu to:

Funkcja	Opis
ones	tworzy macierz o zadanych wymiarach wypełnioną samymi jedynkami
zeros	tworzy macierz o zadanych wymiarach wypełnioną samymi zerami
eye	tworzy macierz o zadanych wymiarach wypełnioną zerami z wyjątkiem przekątnej głównej, wypełnionej jedynkami
rand	tworzy macierz o zadanych wymiarach wypełnioną liczbami w przedziale od 0 do 1 wylosowanymi za pomocą generatora liczb pseudolosowych o równomiernym rozkładzie

Pomysłowe wykorzystanie powyższych funkcji umożliwia szybkie generowanie odpowiednich danych, np. chcąc otrzymać macierz diagonalną o wymiarach 5x5 z wartościami pseudolosowymi z przedziału od -2 do +2 na przekątnej można to zrobić tak:

```
>> Axy=eye(5).*(rand(5,5)-0.5)*4;
```

Program „domyśli" się, że z pewnością chcieliśmy odjąć od macierzy 5x5 macierz o takich samych wymiarach, ale wypełnioną wartościami 0.5. Nie zawsze jest jednak taki domyślny i dlatego należy zwracać uwagę na komentarze. Gdybyśmy chcieli formalnie zapewnić zgodność wymiarów macierzy, to identyczny wynik otrzymalibyśmy stosując:

```
>> Axy=eye(5).*(rand(5,5)-ones(5,5)*0.5)*4;
```

Inny przykład - generowanie wektora wierszowego, który można zinterpretować jako okresowy (5 okresów) ciąg cyfrowy, przy czym jeden okres zawiera 128 elementów, z czego 32 są „losowe" z przedziału od -1 do +1, 16 to wartości „2", 32 zera i reszta elementów to wartości malejące liniowo od 1 do 0 (warto ten przykład zrozumieć „do ostatniej kropki"):

```
>> sT=[(rand(1,32)-0.5)*2,ones(1,16)*2,zeros(1,32),1:-1/((128-32-16-32)-1):0];
>> length(sT)        % bez średnika, żeby sprawdzić, czy faktycznie wyszło 128
>> s=[sT,sT,sT,sT,sT];
```

W kolejnym podrozdziale zostanie pokazane jak wykreślić powyższy ciąg. W powyższym przykładzie otrzymaliśmy wektor wierszowy. Można było to samo zrobić w wersji kolumnowej (warto zauważyć wszystkie różnice):

```
>> sT=[(rand(32,1)-0.5)*2;ones(16,1)*2;zeros(32,1);[1:-1/((128-32-16-32)-1):0] ' ];
>> s=[sT;sT;sT;sT;sT];
```

Gdybyśmy teraz z jakichś powodów chcieli wyzerować część ciągu, to możemy to zrobić np. tak:

```
>> s(64:128)=0;
```

W tym przypadku program powinien się domyślić, że wektorowi wyspecyfikowanemu po lewej stronie przypisania (65-elementowemu) chcemy przypisać taki sam wektor zerowy. Jeżeli jednak chcemy zapisać to formalnie poprawnie, to należałoby to zrobić tak:

```
>> s(64:128)=zeros(1,65);
```

Powyższy zapis odpowiada wierszowemu wektorowi **s**.

Warto podkreślić, iż nawiasy kwadratowe po prawej stronie znaku przypisania (czyli „=") oznaczają, iż wewnątrz nich musimy precyzyjnie zapewnić zgodność wymiarów podmacierzy. Natomiast nawiasy kwadratowe po lewej stronie znaku przypisania przy wywołaniu funkcji oznaczają, iż odczytujemy więcej niż jedną zmienną zwracaną przez funkcję - wtedy kolejne elementy oddzielone są przecinkami i na kolejnych pozycjach mogą być zmienne o dowolnych wymiarach, określonych w definicji funkcji (patrz **m-pliki** ↓).

Przydatnym niekiedy operatorem może być symbol dwukropka, czyli „:", w swoim drugim zastosowaniu (oprócz definiowania ciągów wartości ze stałym krokiem). Jeżeli np. mamy zmienną macierzową o nazwie **A** i o wymiarach 4x5, to zapis:

```
>> x=A(:);
```

spowoduje, że zmienna **x** będzie wektorem kolumnowym o 20 elementach, zawierającym kolejne kolumny macierzy **A**. Jak łatwo zauważyć, jest to też metoda na zamianę wektora wierszowego na kolumnowy.
Z kolei zapis:

```
>> x=A(:,2);
```

oznacza, iż zmienna **x,** będąca wektorem kolumnowym będzie zawierała drugą kolumnę macierzy **A**, natomiast:

```
>> x=A(3,:);
```

spowoduje, że **x** będzie wektorem wierszowym zawierającym trzeci wiersz macierzy **A**.

Przykładowo zamiast tworzyć ciąg okresowy z jednego okresu (w postaci wektora wierszowego **sT**) za pomocą komendy:

```
>> s=[sT,sT,sT,sT,sT];
```

można to zrobić tak:

```
>> s=sT'*ones(1,5);   % warto nie przegapić operatora transponowania

>>s=s(:)';              % transponowanie, by wynik był wektorem wierszowym
```

Co jest szczególnie przydatne, gdy okresów jest dużo, a nie 5 jak w powyższym przykładzie.

5. Funkcje matematyczne

Wybrane funkcje matematyczne

Funkcja	Opis
sin	sinus - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
cos	kosinus - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
tan	tangens - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
atan	arcus tangens - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy (wynik z przedziału od -pi/2 do +pi/2, zgodnie z def. funkcji)
atan2	arcus tangens tzw. czteroćwiartkowy - kolejno dla każdej pary wartości dwóch macierzy podanych jako parametry wejściowe (wynik z przedziału od -pi do +pi, ponieważ podajemy licznik i mianownik, co określa ćwiartkę układu współrzędnych)
log	logarytm naturalny - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
log10	logarytm dziesiętny - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
exp	eksponenta (czyli liczba **e** do zadanej potęgi) - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy
sqrt	pierwiastek kwadratowy - kolejno dla każdej wartości macierzy podanej jako parametr wejściowy (zamiast tej funkcji można stosować operator potęgowania: ^(0.5))
max	zwraca wartość maksymalną spośród elementów wektora lub (w przypadku macierzy) wektor wierszowy zawierający maksymalne elementy z kolejnych kolumn (dwukrotne zastosowanie pozwala zatem wyznaczyć element maksymalny całej macierzy)
min	j.w. - ale element minimalny
sum	suma elementów wektora - w przypadku macierzy analogicznie jak **max**

W razie potrzeby warto pamiętać o funkcji **help,** która zaprezentuje przykłady zastosować powyższych funkcji.

Przykładowo predefiniowane zmienne typu **pi** lub **j** (jednostka urojona) można w razie potrzeby odtworzyć następująco:

```
>> pi=2*atan(1);
>> j=sqrt(-1);
```

Jak to już było widać w poprzednich przykładach, możliwe jest wykorzystywanie wyników funkcji bez przypisywania im nowej nazwy, ale podając otrzymany wynik bezpośrednio jako daną wejściową dla kolejnej funkcji, a nawet jako element wyrażenia arytmetycznego lub logicznego (↓) np.:

```
>> x_data=sin(2*sqrt(b)*pi-cos(2*pi*t));
```

Należy jednak pamiętać, iż zawsze konieczne jest dopilnowanie zgodności wymiarów macierzy - np. nie można odejmować wektora wierszowego od wektora kolumnowego lub też dodawać wektorów wierszowych, ale o różnej liczbie elementów.

6. Funkcje do operacji na liczbach zespolonych

Do obliczeń na liczbach zespolonych (lub macierzach o elementach zespolonych) można korzystać np. z następujących funkcji:

Funkcja	Opis
real	część rzeczywista z podanej liczby (lub macierzy)
imag	część urojona z podanej liczby (lub macierzy)
abs	moduł z podanej liczby (lub macierzy)
angle	faza dla podanej liczby (lub macierzy)
conj	sprzężona liczba (lub macierz) dla podanej liczby (lub macierzy)
unwrap	„rozwinięcie" ciągu wartości - w tym przypadku oznacza odwołanie stosowanego w funkcji angle „zawijania" wyniku do przedziału od -pi do +pi

Naturalnie funkcja **abs** może być także wykorzystana do wyznaczania modułu z liczby rzeczywistej, podobnie jak funkcja **unwrap** może być stosowana do dowolnego ciągu wartości. Funkcji **unwrap** używa się głównie przy badaniu liniowości fazy za pomocą wykresu.
Przykładowo, chcąc transponować zespolony wektor wierszowy do postaci kolumnowej, można zrobić to tak:

```
>> v_wiersz=rand(1,5)+j*ones(1,5);  % generujemy 5-elementowy wektor zespolony
>> v_kol=conj(v_wiersz');           % i tworzymy z niego wektor kolumnowy
```

lub

```
>> v_kol=conj(v_wiersz)';
```

ponieważ nie ma w tym przypadku znaczenia kolejność wykonywanych operacji.
Kolejny przykład:

```
>> A1=eye(5)+j*rand(5,5);          % tworzymy losowo macierz zespoloną
>> A_MOD=abs(A1);                  % wyznaczamy wartości modułów
>> A_FAZ=angle(A1);                % wyznaczamy wartości faz
>> A2=A_MOD.*exp(j*A_FAZ);         % odtwarzamy macierz zespoloną
>> roznica=max(max(abs(A1-A2)))    % sprawdzamy na ile się udało...(!!!)
```

7. Grafika

Niezwykle pożyteczna jest możliwość graficznego przedstawiania danych. Należy jednak pamiętać, iż nie jest sztuką zrobić to jakkolwiek, ale tak, by informacja graficzna była czytelna i pożyteczna z punktu widzenia rozważanego problemu. Poniższa tabela zawiera najczęściej używane funkcje graficzne, których zastosowanie warto przećwiczyć.

Funkcja	Opis
plot	do tworzenia wykresów dwuwymiarowych przy liniowym opisie obu osi - można dobierać sposób wykreślania (linia ciągła, linia przerywana, za pomocą symboli), kolor itd.
stem	do tworzenia wykresów dwuwymiarowych - specjalnie dla potrzeb cyfrowego przetwarzania sygnałów - można dobierać kolor wykresu
axis	do zawężania osi wykresu już po jego narysowaniu (powiększenie wybranego fragmentu wykresu) - dotyczy także wykresów trójwymiarowych
figure	do tworzenia nowych okien graficznych oraz określania „aktualnego okna graficznego" - tj. takiego, którego mają dotyczyć kolejne komendy
hold	umożliwia wybór, czy kolejny wykres ma zastąpić poprzednią zawartość okna, czy też ma zostać narysowany dodatkowo (bez usuwania zawartości okna graficznego)
grid	wstawianie i usuwanie siatki na wykresie
loglog	podobnie jak plot, jednak skala obu osi jest logarytmiczna
semilogx	podobnie jak plot, jednak skala osi poziomej jest logarytmiczna
semilogy	podobnie jak plot, jednak skala osi pionowej jest logarytmiczna
title	umożliwia umieszczenie nad wykresem tytułu
xlabel	umożliwia umieszczenie pod osią poziomą opisu tekstowego
ylabel	umożliwia umieszczenie obok osi pionowej opisu tekstowego
subplot	do umieszczania w jednym oknie graficznym zestawu osobnych wykresów (w odrębnych zestawach osi)
mesh	do tworzenia wykresów trójwymiarowych
view	do obracania wykresów trójwymiarowych
zlabel	umożliwia umieszczenie obok osi „z" opisu tekstowego

8. Pakiety narzędziowe - *toolbox'y*

Pakiet Matlab składa się z programu podstawowego oraz dodatkowych (zakupionych w zależności od potrzeb) programów pomocniczych, zawierających zestawy specyficznych funkcji. Jednym z takich programów narzędziowych (*ang. Toolbox*) jest *DSP-BlockSet*, który umożliwia projektowanie symulacji z zakresu cyfrowego przetwarzania sygnałów za pomocą symboli graficznych układanych w schemat blokowy. Należy jednak pamiętać, iż użytkownik musi zadbać o podanie odpowiednich parametrów dla wszystkich elementów schematu, ponieważ jest to jedynie rodzaj graficznej „nakładki". Pomiędzy *toolbox'ami* istnieją wzajemne zależności - i tak na przykład *DSP-BlockSet* korzysta z pakietów narzędziowych takich, jak *Simulink* oraz *Signal Processing*.

9. M-pliki

9.1. Informacje ogólne

W wielu sytuacjach posługiwanie się pakietem może być znacznie ułatwione przez zapisanie sekwencji komend do pliku lub utworzenie funkcji. Podanie nazwy tego pliku (bez rozszerzenia) spowoduje automatyczne wykonanie całej zapisanej sekwencji lub funkcji. Muszą być jednak spełnione pewne warunki:

a) plik jest zapisany w kodzie ASCII,
b) nazwa pliku posiada rozszerzenie „m", na przykład: *funkcja1.m*,
c) katalog, w którym znajduje się ten plik, jest wymieniony w odpowiednim wykazie pakietu MATLAB (wykaz ten jest dostępny w odpowiednim okienku dialogowym lub poprzez komendę **path**).

M-plik zawierający sekwencję komend - a nie funkcję - określany jest jako „script", co można przetłumaczyć jako *skrypt*. Jego wywołanie powoduje realizację kolejnych instrukcji, dokładnie tak jak byśmy je po kolei podawali do wykonania w trybie komend.

Zasadnicza różnica pomiędzy funkcją i skryptem polega na tym, iż w przypadku skryptu w trakcie jego realizacji są dla niego dostępne wszystkie obecne w pamięci zmienne i po jego zakończeniu wszystkie utworzone zmienne pozostają w pamięci. Natomiast w przypadku funkcji przesyłanie zmiennych następuje jedynie za pośrednictwem wykazu zmiennych wejściowych i specyfikacji zmiennych wynikowych - pozostałe zmienne są wewnętrzne i znikają po zakończeniu realizacji funkcji.

9.2. Funkcje

Drugim sposobem wykorzystania m-plików oprócz *skryptów* jest tworzenie własnych funkcji. Nazwa funkcji musi (powinna) być zgodna z nazwą pliku (także o rozszerzeniu „m"). W pierwszej linii pliku powinno się znajdować słowo kluczowe **function**, a po nim lista zmiennych wyjściowych, nazwa funkcji i w nawiasie lista zmiennych wejściowych - ich nazwy obowiązują tylko wewnątrz funkcji. Ponadto wszystkie nazwy zmiennych używane w definicji funkcji są lokalne.
Jeżeli pierwsza linia m-pliku o nazwie *suma.m* jest następująca:

function z= suma(x,y)

to funkcja ta może być wywołana jako komenda:

```
>> c=suma(a,b);
```

Wywołanie takie może się znaleźć również w definicji innej funkcji.

Należy uważać, by nową nazwą funkcji (lub zmiennej) nie zasłonić już istniejącej nazwy funkcji (lub zmiennej).

Symbol "%" oznacza, że reszta linii na prawo od niego jest traktowana jako komentarz. Sekwencja linii rozpoczynających się od tego znaku, a następująca po pierwszej linii m-pliku (tej ze słowem kluczowym *function*), jest wyświetlana na ekranie w wyniku podania komendy:

```
>> help nazwafunkcji
```

Korzystanie z tej możliwości jest najlepszym sposobem na uniknięcie pomyłki w interpretacji zmiennych wejściowych i wyjściowych oraz umożliwia przypomnienie, co dana funkcja realizuje. Można też dzięki temu wykryć niepożądane zasłonięcie nazwy funkcji lub zmiennej (w przypadku zmiennej komenda help spowoduje wyświetlenie informacji, że takiego m-pliku nie ma).

Przykładowa zawartość m-pliku:

```
function z=suma(x,y)
% z = suma(x,y)
%
% dodawanie liczb, wektorow lub macierzy

z=x+y;
```

Gdy liczba zmiennych wyjściowych jest większa niż jeden, to lista tych zmiennych umieszczana jest w nawiasach [ ], przykładowo:

```
function [AH,FH]=charakterystyki(H);
% [AH,FH]=charakterystyki(H);
%
% funkcja do wyznaczania charakterystyk: amplitudowej i fazowej
%
% H - zespolona, spróbkowana charakterystyka częstotliwościowa;
%
% AH - spróbkowana ch-ka amplitudowa;
% FH - spróbkowana ch-ka fazowa;

[M,N]=size(H);
if (M-1)*(N-1)~=0,
        disp('H nie jest wektorem ale macierzą');
        AH=[ ];         % zmienna AH będzie „pusta"
        FH=[ ];         % zmienna FH będzie „pusta"
        return
end

AH=abs(H);
AH=AH(:)';              % dla porządku - żeby wektor AH był na pewno wierszowy;
FH=angle(H);
FH=FH(:)';              % dla porządku - żeby wektor FH był na pewno wierszowy;

k=find(AH<=10^(-9));    % k będzie zawierać indeksy tych elementów AH, które są „prawie równe 0";

if length(k)~=0, disp('UWAGA - występują zerowe lub „prawie zerowe" wartości amplitudy!');end;
FH(k)=0;                % ciekawe czemu to może służyć?
% end;
```

Uwaga:
W przypadku, gdy wywołując funkcję nie zaproponujemy zmiennych dla wszystkich zmiennych wynikowych funkcji, to przyporządkowanie zmiennych nastąpi według kolejności na liście w m-pliku (czyli w definicji), np. dla powyższego m-pliku przy wywołaniu:

```
>>amp1=charakterystyki(Hf1);
```

Zmienna „ampl1" stanie się równa wektorowi, który w definicji funkcji nosi nazwę „AH", natomiast wyliczony przez funkcję wektor „FH" nie będzie wykorzystany.

Pełną zawartość danego m-pliku można wyświetlić na ekranie za pomocą polecenia:

```
>> type nazwa_pliku
```

Nie należy nadużywać możliwości tworzenia funkcji do realizacji prostych zadań, gdyż w takich przypadkach zajmuje to więcej czasu niż podanie sekwencji komend i powoduje trudne do usunięcia zaśmiecenie dysku.

9.3. Polecenia, których umieszczenie w m-pliku może ułatwić pracę

Funkcja	Opis
pause	wstrzymanie realizacji funkcji do naciśnięcia dowolnego klawisza
disp	wypisanie na ekranie podanej stałej tekstowej
input	wypisanie na ekranie podanej stałej tekstowej oraz oczekiwanie na wpisanie z klawiatury nowej wartości podanej zmiennej
return	przerwanie wykonywania funkcji i powrót do miejsca jej wywołania

9.4. Uwagi dodatkowe

Do tworzenia m-plików służy specjalny edytor, który umożliwia także *debugg'owanie* plików. Zależy zwrócić uwagę na to, iż użytkownik decyduje gdzie mają zostać zapamiętane nowo utworzone pliki i nie powinien „zaśmiecać" dysku! Plik po to by był dostępny dla Matlaba musi znajdować się w jednej ze ścieżek znajdującej się na liście Matlaba - dodawanie i usuwanie ścieżek z listy realizuje się za pomocą odpowiedniego okna dialogowego. Ponieważ w trakcie zajęć jest wielce prawdopodobne, iż kolejne grupy będą korzystały z takich samych nazw m-plików, zatem by uniknąć zasłaniania nazw, należy po swoich zajęciach usunąć z listy swoją ścieżkę.
Warto też zwrócić uwagę, iż istnieje jedna ścieżka tzw. *default'owa*, którą także można (a niekiedy należy) zmienić na czas trwania zajęć. Ta ścieżka po uruchomieniu pakietu każdorazowo ustawiana jest na tą samą lokację (nie należy tam zapisywać żadnych swoich plików!). Ścieżka *default'owa* to nie to samo, co zestaw ścieżek z listy, choć do ustawienia obu typów ścieżek służy to samo okno dialogowe - warto się zatem dobrze przyjrzeć dostępnym opcjom, by nie popełniać pomyłek.
Po wywołaniu m-pliku program przeszukuje w poszukiwaniu odpowiedniej nazwy ścieżkę *default'ową* oraz całą listę ścieżek do chwili aż napotka pierwszą pasującą nazwę.

10. Elementy programowania

Pakiet Matlab umożliwia także stosowanie typowych konstrukcji sterowania przebiegiem pracy programu. Jednym z nich jest „pętla for", której przykład przedstawiono poniżej (pominięto znaki „>>" ponieważ tego typu struktury stosuje się zazwyczaj w m-plikach):

```
k1=[1,3,-5,7,1:3];    %
for k=k1
  x=sin(2*pi*0.01*k);
end
```

W przykładzie zmienna **k** jest w kolejnych obiegach pętli pobierana z wektora **k1**. Można oczywiście określić kolejne wartości tej zmiennej bezpośrednio, np.:

```
>>for k=2:-1:-2, t(k+3)=k^3; end;
```

Przykład ten pokazuje, iż można w razie potrzeby skonstruować pętlę w linii komend Matlaba. Należy jednak pamiętać, iż jeżeli w linii komend wpiszemy jedynie część struktury pętli (bez słowa kluczowego **end**) i zakończymy klawiszem [Enter], to nawet nie rozpocznie się wykonywanie pętli - Matlab będzie z tym czekał do podania słowa kluczowego **end** [Enter].

Drugi często stosowany element programowania, to instrukcja warunkowa **if**. Zawarta pomiędzy nią i odpowiednim dla niej słowem kluczowym **end** sekwencja instrukcji (komend) jest realizowana w przypadku spełnienia warunku podanego w instrukcji **if**. Przykładowo:

```
x=zeros(1,4);
if k<=5,
    for m=1:4,
      x(m)=(k-m)^2;
    end;
end;
```

lub inny przykład (z linii komend):

```
>> if k~=5, s=sin(2*pi*f*t); else s=cos(2*pi*f*t) end;
```

Możliwe operatory warunku to: == (równe), ~= (różne), <, >, <=, >=. Możliwe jest też budowanie zdań logicznych - dalszych wyjaśnień należy szukać za pomocą funkcji **help**.
Należy podkreślić, iż warunek równości zapisujemy w postaci dwóch znaków przypisania, czyli w postaci „==". Mylenie znaku przypisania i warunku równości jest częstym błędem syntaktycznym.
Pomocnicza konstrukcja ze słowem **else** umożliwia określenie bloku programu, która ma zostać zrealizowany w przypadku, gdy warunek podany w instrukcji **if** nie jest spełniony.
Opisane powyżej instrukcje mogą być zagnieżdżone - dlatego warto stosować w m-plikach wcięcia, by wyraźnie zaznaczyć, który **end** kończy który blok **if** lub **for**.

11. Zapis do pliku i odczyt z pliku

Możliwe jest zapisywanie do plików (tekstowych lub w formacie wewnętrznym Matlaba) wszystkich lub tylko wybranych zmiennych, które potem można na powrót wczytać podczas kolejnej sesji. Służą do tego funkcje **save** (zapisuje do ścieżki *default'owej*) oraz **load** (program przeszukuje w poszukiwaniu odpowiedniej nazwy ścieżkę *default'ową* oraz całą listę ścieżek do chwili aż napotka pierwszą pasującą nazw pliku).
Można także stworzyć dokumentację sesji, zapisując wykonywane operacje i ich wyniki do pliku tekstowego za pomocą funkcji **diary** (zapisuje do ścieżki *default'owej*).
Należy jednak pamiętać, by przy pracy z plikami panować nad lokalizacją tworzonych przez siebie plików (według zaleceń prowadzącego zajęcia), ponieważ „zaśmiecanie" dysku stanowi naruszenie zasad korzystania z laboratorium - patrz też p. 9.4 powyżej.