Podstawy Automatyki

Politechnika Poznańska
Katedra Sterowania i Inżynierii Systemów

ĆWICZENIE 1

Wprowadzenie do środowiska MATLAB 7.1 (R14SP3).

Celem ćwiczenia jest poznanie podstawowych cech środowiska symulacyjno–programistycznego Matlab w wersji 7.1 (Release 14 with Service Pack 3). Matlab jest intuicyjnym językiem obliczeń naukowo–inżynierskich, udostępniającym zestaw podstawowych funkcji matematycznych i zaawansowanych narzędzi graficznych, służących między innymi do analizy i wizualizacji danych, opracowywania i implementacji wszelkiego rodzaju algorytmów. Środowisko Matlab w wersji 7.1 jest wykorzystywane na uczelniach jak i w przemyśle. Na wielu uczelniach wyższych Matlab jest obecnie standardowym narzędziem instruktażowym, wykorzystywanym zarówno we wstępnych, jak i zaawansowanych kursach inżynierskich i naukowych. W przemyśle, Matlab wykorzystywany jest jako wysokowydajne narzędzie wspomagające prace badawcze, odkrywcze oraz analizę.

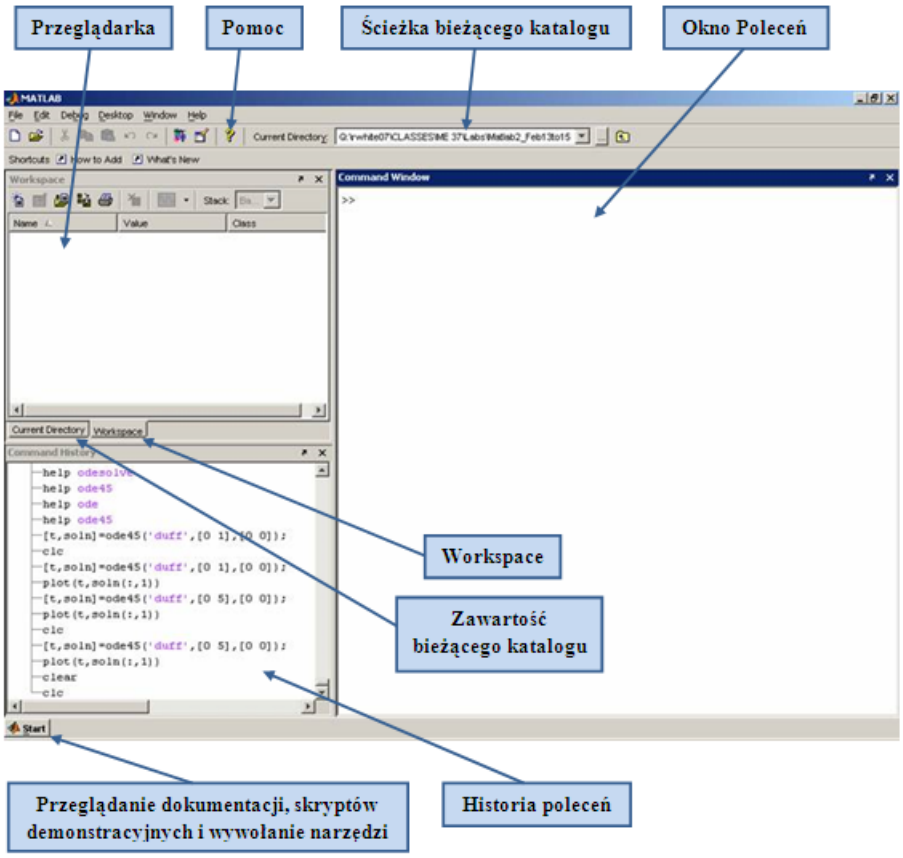
1 Wprowadzenie

Matlab jest interaktywnym systemem, którego podstawowym typem danych jest macierz. Umożliwia rozwiązywanie wielu technicznych problemów, w szczególności tych sformułowanych przy użyciu macierzy i wektorów, w ułamku czasu potrzebnego do rozwiązania tych samych zagadnień przy użyciu nieinteraktywnych języków, takich jak np. C.

Nazwa *Matlab* jest skrótem od angielskich słów ***Mat**rix **Lab**oratory*, czyli laboratorium macierzowe. Wynika to z faktu, że środowisko to zoptymalizowane jest dla obliczeń macierzowych, czyli przeprowadzanych na danych prezentowanych w formie wektorów czy tablic dwu- i wielowymiarowych. Konsekwencją tej optymalizacji jest fakt, że *Matlab* znacznie szybciej wykonuje obliczenia na zmiennych będących macierzami niż pojedynczych zmiennych zmienno- czy stałoprzecinkowych. Podczas korzystania ze środowiska *Matlab* należy więc pamiętać, że poprawę szybkości działania programów tworzonych w tym środowisku, można osiągnąć zastępując pojedyncze zmienne i tablice tych zmiennych wektorami czy macierzami (również wielowymiarowymi) stałych czy zmiennych. Warto też zdawać sobie sprawę, że "konwencjonalne" mechanizmy programowe, taki jak np. pętle, w *Matlabie* również wykonywane są znacznie wolniej niż pojedyncze operacje na zmiennych umieszczonych w wektorach czy macierzach.

System *Matlab* można podzielić na 5 podstawowych części:

- Środowisko Rozwojowe (Development Environment) - jest to zestaw narzędzi i udogodnień pomocnych przy używaniu funkcji i obsłudze plików. Większość z tych narzędzi posiada własny graficzny interfejs użytkownika. Składnikami środowiska rozwojowego są (patrz Rys. 1):
    - Okno Poleceń (Command Window) służące do wpisywania poleceń *Matlaba* (operowania zmiennymi i wywoływania funkcji). W oknie tym zwracane są również wyniki przeprowadzonych obliczeń, wartości zmiennych oraz wyświetlane są komunikaty i informacje o błędach, które występują podczas wykonywania obliczeń.



Rysunek 1: *Wygląd okna głównego środowiska Matlab.*

- Okno Historii Poleceń (Command History), w którym wyświetlana jest lista poleceń wpisanych w Oknie Poleceń od chwili uruchomienia środowiska.
- Edytor z Debuggerem, które przeznaczone są do tworzenia i testowania własnych skryptów oraz funkcji. Edytor wyposażony jest w narzędzia edytorskie takie jak podświetlanie składni, automatyczne wcięcia itp. Natomiast debugger jest niezwykle pożytecznym narzędziem przy testowaniu poprawności działania skryptów i funkcji tworzonych przez użytkownika. Zawiera on wszelkie mechanizmy występujące w typowych środowiskach programistycznych, takie jak możliwość pracy krokowej, ustawiania pułapek, podglądania wartości zmiennych, ustawiania warunków wykonywania, bądź przerywania wykonywania programów itd.
- Przeglądarka (Browser) będąca wygodnym narzędziem służącym do przeglądania zawartości zmiennych zdefiniowanych i widocznych w środowisku *Matlab*. Okno Przeglądarki zawiera 2 zakładki opatrzone etykietkami Workspace i Current Directory. Na pierwszej karcie z tych zakładek wyświetlone są informacje o zdefiniowanych w danej przestrzeni roboczej zmiennych: ich nazwach, rozmiarze, ilości zajmowanej pamięci oraz typie przechowywanych danych. Dwukrotne kliknięcie wiersza odpowiadającego konkretnej zmiennej otwiera okno edytora zmiennych, gdzie możliwy jest podgląd i zmiana wartości składowych danej zmiennej. Z kolei zakładka Current Directory służy do przeglądania zawartości bieżącego katalogu na dysku oraz zmiany tego katalogu. Wybranie pliku z listy w tym okienku i dwukrotne kliknięcie jego nazwy powoduje uruchomienie skojarzonej z danym typem pliku aplikacji. Dla najczęściej wykorzystywanych w

środowisku *Matlab* plików z rozszerzeniem .m (tzw. m–plików) otwierany jest program Edytora-Debuggera. W przypadku pozostałych typów plików uruchamiane są albo odpowiednie filtry importujące dane do środowiska *Matlab* albo, w ostateczności, aplikacje zewnętrzne.

- Biblioteka Funkcji Matematycznych *Matlaba* (The *Matlab* Mathematical Function Library) - obszerny zbiór algorytmów obliczeniowych, od funkcji elementarnych takich jak suma, sinus, cosinus, operacje na liczbach zespolonych, aż po zaawansowane algorytmy macierzowe jak inwersja macierzy, obliczanie wartości własnych, funkcji Bessela, szybkich transformat Fouriera i wielu, wielu innych.
- Język Programowania *Matlaba* (The *Matlab* Language) - jest językiem programowania wysokiego poziomu zoptymalizowanym pod kątem obliczeń na macierzach/tablicach. Język ten obsługuje mechanizmy sterowania przepływem, funkcje, struktury danych, operacje wejścia/wyjścia i programowanie zorientowane obiektowo. Korzystając z tego języka możliwe jest tworzenie zarówno prostych, jednorazowych skryptów, zawierających te same polecenia, jakie wpisywane są w Oknie Poleceń jak i rozbudowanych aplikacji i bibliotek funkcji, łącznie z graficznym interfejsem użytkownika.
- Grafika (Graphics) - *Matlab* posiada bogate mechanizmy wyświetlania wektorów i macierzy w postaci wykresów oraz opatrzenia tych wykresów opisami, prostymi obiektami graficznymi jak również drukowania tych wykresów i zapisywania ich w najpopularniejszych formatach graficznych, zarówno rastrowych jak i wektorowych. Narzędzia związane z graficzną prezentacją danych składają się z wysokopoziomowych funkcji pozwalających na dwu- i trójwymiarową wizualizację danych, przetwarzanie obrazów oraz animacje. Dostępne są również funkcje niskopoziomowe, dające możliwość dowolnego dostosowania wyglądu wykresów do potrzeb użytkownika, jak również tworzenie pełnych interfejsów graficznych dla aplikacji stworzonych w środowisku *Matlab*.
- Interfejs programistyczny *Matlaba* (The *Matlab* Application Program Interface (API)) stwarza możliwość pisania własnych programów współpracujących z *Matlabem*, przy czym programy te mogą być tworzone w popularnych językach programowania, takich jak C czy Fortran. API zawiera mechanizmy do wywoływania procedur *Matlaba* (łączenie dynamiczne - dynamic linking), wywoływania *Matlaba* w charakterze mechanizmu obliczeniowego oraz odczyt i zapis plików danych z rozszerzeniem .mat (tzw. mat-pliki).

Matlab zawiera rodzinę dodatkowych bibliotek, zwanych Toolboxami. Dla większości użytkowników *Matlaba*, bardzo ważny jest fakt, że Toolboxy umożliwiają naukę i stosowanie zaawansowanych technologii. Toolboxy są zestawem funkcji *Matlaba* (m-plików), rozszerzającym środowisko o możliwość rozwiązywania określonych, specjalistycznych problemów z dziedziny przetwarzania sygnałów, systemów kontroli, sieci neuronowych, logiki rozmytej, telekomunikacji, symulacji i wielu innych.

Rysunek 2: *Ikona środowiska Matlab.*

1.1 Odnaleźć na pulpicie ikonę środowiska *Matlab* (przedstawioną na Rys. 2) i uruchomić program.

1.2 Zapoznać się z oknami interfejsu użytkownika. Sprawdzić i zapoznać się z właściwościami opisanymi powyżej, w tym punkcie opracowania.

2 Operacje na zmiennych

Tak jak w przypadku wielu języków programowania, język *Matlab* umożliwia wykorzystanie zmiennych. Jednak w przeciwieństwie do większości języków, *Matlab* nie wymaga stosowania deklaracji zmiennych ani określania ich rozmiaru. Gdy napotka na nazwę nowej, nieznanej zmiennej, automatycznie tworzy zmienną i przypisuje jej odpowiedni rozmiar w pamięci. Jeżeli zmienna już istnieje, *Matlab* potrafi dokonywać na niej zmian oraz, o ile to konieczne, alokuje nowe zasoby pamięci.

2.1 Tworzenie nowych zmiennych

Aby stworzyć nową zmienną oraz nadać jej wartość wystarczy wpisać w Oknie Poleceń (patrz Rys. 1) polecenie:

```
zmienna = wartość
```

i zatwierdzić przyciskiem *Enter*. Np. wpisanie poniższego polecenia, spowoduje wytworzenie w *Matlabie* skalaru (macierzy 1×1) o nazwie *num_student* i przypisanie jej wartości 15:

```
>> num_student = 15;
```
[1]

Tworzenie nowej zmiennej podlega następującym regułom:

- Pierwszym elementem, który należy podać, jest nazwa zmiennej. Nazwa tworzonej zmiennej (niezależnie od jej typu) powinna składać się z litery, po której następuje dowolna ilość liter, cyfr i/lub znaków podkreślenia. Nie powinna zawierać znaków specjalnych, takich jak np. polskie litery. *Matlab* rozróżnia wielkość liter, oznacza to, że np. zmienne o nazwach A oraz a są postrzegane jako dwie zmienne. Nazwy mogą być dowolnej długości, jednak *Matlab* wykorzystuje jedynie pierwsze 63 znaki nazwy, a resztę ignoruje. Dlatego, ważnym jest, aby pierwsze 63 znaki nazwy każdej zmiennej były unikatowe. Zabronione jest również nadawanie zmiennym nazw z puli zarezerwowanych wyrazów *Matlaba*, np. *variable*. Listę tych wyrazów można uzyskać przez wpisanie komendy `ssc_reserved` oraz `iskeyword`.
- Po nazwie zmiennej umieszcza się znak równości.
- Za znakiem równości należy podać wartość zmiennej. *Matlab* do zapisu liczb używa konwencjonalnej notacji dziesiętnej z opcjonalnym przecinkiem[2] oraz znakami plus i minus. Notacja naukowa wykorzystuje literę e jako mnożnik „dziesięć do potęgi". Przy korzystaniu z liczb urojonych używa się przedrostka i lub j. Oto przykłady poprawnie zapisanych liczb:

```
      3          -99       0.00001   9.6397438
 1.60210e-20  6.02252e23  -3.14159j     3e5i
```

Po utworzeniu zmiennej można zaobserwować, że pojawia się ona w oknie Przeglądarki (Workspace), gdzie podawana jest jej nazwa, wymiary, ilość zajmowanej pamięci oraz typ przechowywanej wartości. Po zdefiniowaniu zmiennej można się do niej odwoływać poprzez jej nazwę. Podanie samej nazwy zmiennej w Oknie Poleceń powoduje wyświetlenie jej zawartości. Utworzone zmienne można wykorzystać do dalszych operacji, między innymi do tworzenia nowych zmiennych. W tym celu można wykorzystać podstawowe operacje matematyczne: dodawanie (+), odejmowanie (-), mnożenie (*), dzielenie (/) lub potęgowanie (^), np.:

```
>> num_student_new = num_student + 2;
```

Zmienne można również generować wykorzystując dostępne w *Matlabie* podstawowe funkcje matematyczne, włączając abs, sqrt, exp oraz sin[3] lub inne bardziej kompleksowe (np. funkcję γ lub

[1] Umieszczenie średnika po poleceniu powoduje, że wartość będąca wynikiem operacji nie będzie wyświetlana na ekranie. W przypadku operacji na dużych blokach danych, wyświetlanie ich w Oknie Poleceń Matlaba zajmuje bardzo dużo czasu oraz utrudnia odczytanie istotnych informacji.

[2] Części dziesiętne liczby należy oddzielać kropką.

[3] Odpowiednio: wartość bezwzględna, pierwiastek kwadratowy, funkcja wykładnicza o podstawie e oraz sinus ze zmiennej.

funkcje Bessela). Listę przydatnych funkcji matematycznych zestawiono w Tab. 1. Aby wyświetlić pełną listę funkcji elementarnych należy wpisać `help elfun` w Oknie Poleceń.

Nazwa funkcji lub stałej	Opis funkcji i przykładowy sposób wywołania
asin, acos, atan, atan2	Y = asin(X); Y = acos(X); Y = atan(X); Y = atan2(Y,X); funkcje cyklometryczne; wynik podany w radianach.
sin, cos, tan	Y = sin(X); Y = cos(X); Y = tan(X); funkcje trygonometryczne z argumentu wyrażonego w radianach.
sinh, cosh, tanh	Y = sinh(X); Y = cosh(X); Y = tanh(X); funkcje hiperboliczne z argumentu wyrażonego w radianach.
exp	Y = exp(X); funkcja wykładnicza e^X; exp(1) = 2,71828183.
log, log10	Y = log(X); Y = log10(X); logarytm naturalny z X, logarytm dziesiętny z X.
pow2	Y = pow2(X); funkcja 2^X.
sqrt	Y = sqrt(X); pierwiastek kwadratowy z X.
abs	Y = abs(X); wartość bezwzględna z X.
imag	Y = imag(X); zwraca wartość części urojonej X.
real	Y = real(X); zwraca wartość części rzeczywistej X.
sign	Y =sign(X); funkcja signum.
ceil, floor, round	Y = ceil(X); Y = floor(X); Y = round(X); zaokrąglenie X w górę; zaokrąglenie X w dół; zaokrąglenie X do najbliżej liczby całkowitej.
gcd, lcm	Z= gcd(X, Y); Z= lcm(X, Y); największy wspólny dzielnik kolejnych elementów X i Y; najmniejsza wspólna wielokrotność kolejnych elementów X i Y.
mod, rem	Z = mod(X,Y); Z = rem(X,Y); reszta z dzielenia odpowiadających sobie elementów macierzy X i Y; funkcje nie różnią się w przypadku, gdy X i Y maja ten sam znak; w przypadku gdy X ma inny znak niż Y oraz Y nie jest równe 0 funkcja rem daje w wyniku taki sam znak jak X, a funkcja mod – jak Y.
find	I = find(X); zwraca wektor złożony z numerów indeksów odpowiadających niezerowym wartościom macierzy X; I = find(X>4); zwraca wektor złożony z numerów indeksów macierzy X, dla których element macierzy X jest większy niż 4.
i	Równoważne z sqrt(-1); część urojona.
pi	3,14159265.

Tablica 1: *Przykładowe funkcje matematyczne oraz stałe środowiska Matlab.*

Przykładowo wpisanie kolejno poleceń:

```
>> a = 1.57;
>> b = sin(a);
```

spowoduje przypisanie do zmiennej b wartości 1.

Należy pamiętać, że wszelkie obliczenia związane z kątami wykonywane są na argumentach podanych w radianach, tak więc w niektórych przypadkach konieczne może okazać się przeliczenie kątów wyrażonych w stopniach na radiany, w myśl wzoru:

$$\alpha\,[rad] = \frac{\alpha[^\circ]\cdot\pi}{180}$$

Aby wyświetlić pomoc dotyczącą danej funkcji, uwzględniającą informację o działaniu, parametrach wejściowych, przykładowych sposobach wywołania funkcji, należy wpisać:

```
>> help nazwa_funkcji
```

2.2 Usuwanie i nadpisywanie zmiennych

Podczas wykonywania operacji na zmiennych należy pamiętać, że nazwy zmiennych nie są zarezerwowane. Oznacza to, że istnieje możliwość ich nadpisania przez nową wartość. Wpisanie kolejno poleceń:

```
>> aux = 15;
>> aux = 32;
```

spowoduje, że zmiennej *aux* zostanie przypisana wartość 32, która może być w późniejszym etapie wykorzystana do dalszych operacji. Poszczególne zmienne można usuwać z pamięci *Matlaba* przy pomocy polecenia:

```
>> clear nazwa_zmiennej
```

natomiast wymazanie wszystkich zmiennych znajdujących się w danej chwili w Workspace następuje przez wykorzystanie słowa kluczowego *all*:

```
>> clear all
```

2.3 Wpisywanie poleceń w Oknie Poleceń

W *Matlabie* istnieje możliwość użycia wcześniej wpisanych komend. Wystarczy ustawić kursor w Oknie Poleceń i nacisnąć ↑. Wybierając odpowiednie polecenie przyciskami ↑ oraz ↓, można wywołać je ponownie *Enterem* bez wprowadzania zmian, lub dokonując wcześniejszej edycji.

Jeżeli wpisywane polecenie jest zbyt długie, można przełamać jedną linię polecenia wykorzystując operator "...". Następnie należy przejść do nowej linii wpisywania *Enterem*, np.:

```
>> x = 4;
>> a = 2*(x^2) + 4*x ...
+ 5
 a =
      53
```

Jeśli natomiast wprowadzane jest naraz kilka poleceń, można zapewić, aby nie były one wykonywane natychmiast po zakończeniu każdej linii, ale po wpisaniu całej sekwencji. Należy każdą linię z poleceniem zakończyć wciskając kombinacje klawiszy *Shift* + *Enter*. W takiej sytuacji wszystkie wprowadzone w Oknie Poleceń komendy zostaną wykonane kolejno, dopiero w momencie wciśnięcia klawisza *Enter*.

2.1 Przypisać następujące wartości do zmiennych: *b_dziel* = 10, *m1* = 0.004, *c_aux* =15, *dzielnik* = 7.0064. Wykonać poniższe polecenia, analizując po każdej operacji zawartość Workspace:

- dodać do zmiennej *m1* liczbę 10,45; wynik zaokrąglić w górę;
- wynik poprzedniej operacji podzielić przez zmienną *dzielnik*; wynik wykorzystać jako argument funkcji sinus;
- przypisać zmiennej *c_aux* nową wartość równą liczbie Eulera (*exp(1) = 2,718*); obliczyć logarytm naturalny z *c_aux* do potęgi 3;
- przeanalizować działanie funkcji *mod* w pomocy programu *Matlab*; obliczyć resztę z dzielenia zmiennej *b_dziel* przez zmienną *dzielnik*.

3 Operacje macierzowe

Jak zostało wspomniane w części wprowadzającej niniejszego opracowania, środowisko *Matlab* zoptymalizowane jest pod kątem obliczeń przeprowadzanych na danych przechowywanych w formie wektorów czy macierzy. Stąd też kluczowym zagadnieniem przy rozpoczynaniu pracy w tym środowisku jest poznanie sposobów definiowania takich zmiennych, a następnie manipulowania nimi.

3.1 Definiowanie macierzy

Najprostszym sposobem utworzenia nowej macierzy jest jawne wpisanie jej elementów w Oknie Poleceń. Przykładowo wpisanie następującego ciągu znaków:

```
>> macierz = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```
[4]

i zakończenie go wciśnięciem klawisza *Enter* spowoduje utworzenie nowej zmiennej o nazwie *macierz*. Jak łatwo zauważyć na podstawie powyższego przykładu, tworzenie nowej macierzy podlega tym samym podstawowym regułom co tworzenie nowych zmiennych, z uwzględnieniem dodatkowych zasad:

- Elementy składające się na daną macierzy wyszczególnia się w nawiasach kwadratowych.
- Kolejne elementy w tym samym wierszu macierzy oddziela się spacjami lub przecinkami.
- Kolejne wiersze oddziela się średnikami.
- Należy pamiętać, że w każdym wierszu tworzonej macierzy podana musi być taka sama ilość elementów.

W wyniku wykonania powyższego przykładu zostanie utworzona nowa zmienna macierz o postaci:

```
macierz =
          16   3   2   13
           5  10  11    8
           9   6   7   12
           4  15  14    1
```

Jeżeli elementy nowo tworzonej zmiennej (macierzy lub wektora) mają być takie same lub mają być rozłożone według pewnych specyficznych schematów, do tworzenia takiej zmiennej wygodnie jest posłużyć się jedną z funkcji przewidzianych do tego celu i zebranych w Tab. 2.

[4] Podana macierz jest magicznym kwadratem Durer'a z 1514 roku. Suma liczb w każdym wierszu, kolumnie oraz przekątnej jest równa 34.

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
ones	A = ones(n); tworzy macierz kwadratową *nxn* wypełnioną jedynkami; A = ones(n, m); tworzy macierz *nxm* wypełnioną jedynkami.
zeros	A = zeros(n); tworzy macierz kwadratową *nxn* wypełnioną zerami; A = zeros(n, m); tworzy macierz *nxm* wypełnioną zerami.
eye	A = eye(n); tworzy macierz jednostkową o wymiarze *nxn* (z jedynkami na przekątnej, pozostałe wartości zero); A = eye(n, m); tworzy macierz jednostkową o wymiarze *nxm*.
diag	A = diag(n); tworzy macierz diagonalną; na diagonali umieszcza wartości wektora n; pozostałe wartości są równe zero; A = diag([a b c]); tworzy macierz diagonalną, o wartościach a, b oraz c na diagonali.
magic	A = magic(n); tworzy "magiczny kwadrat" o wymiarze *nxn*.
rand	A = rand(n); tworzy macierz kwadratową *nxn* o losowo wybranych elementach z przedziału (0,1), przy czym rozkład prawdopodobieństwa tych elementów jest jednorodny; A = rand(n,m); tworzy macierz *nxm* o losowo wybranych elementach z przedziału (0,1), przy czym rozkład prawdopodobieństwa tych elementów jest jednorodny;
randn	A = rand(n); tworzy macierz kwadratową *nxn* o losowo wybranych elementach z przedziału (0,1), przy czym rozkład prawdopodobieństwa tych elementów jest normalny; A = rand(n,m); tworzy macierz *nxm* o losowo wybranych elementach z przedziału (0,1), przy czym rozkład prawdopodobieństwa tych elementów jest normalny;
randperm	A = randperm(n); tworzy wektor zawierający permutację elementów całkowitych z przedziału [1,n].

Tablica 2: *Lista funkcji dedykowanych do tworzenia macierzy.*

Przy tworzeniu macierzy i wektorów wykorzystanie sekwencji numerów często jest przydatne, dlatego *Matlab* udostępnia dedykowany operator dwukropka. Operator ten wytwarza macierz (lub wektor) o wymiarach $1 \times n$ złożoną z sekwencji numerów od pierwszej zadeklarowanej wartości do ostatniej (pierwszy:ostatni). Domyślna sekwencja wytworzona jest przez zwiększanie każdej kolejnej wartości o 1, w stosunku do poprzedniej:

```
>> A = 10:15
A = 10 11 12 13 14 15
```

Ten sam mechanizm można też wykorzystać przy tworzeniu macierzy, trzeba wtedy jednak pamiętać, że ilość elementów w każdym wierszu musi być identyczna:

```
>> M = [10:16; 2:8]
 M =
      10   11   12   13   14   15   16
       2    3    4    5    6    7    8
```

Sekwencja nie jest ograniczona tylko do liczb dodatnich. Można wykorzystać liczby mniejsze od zera. Istnieje również możliwość określania skoku pomiędzy kolejnymi wartościami sekwencji (pierwszy:skok:ostatni). Polecenie:

```
>> A = 10:2:20
```

wytworzy następujący wektor:

```
A = 10 12 14 16 18 20
```

Wartość skoku może być zarówno niecałkowita jak i ujemna:

```
>> M = [0:0.2:1; 15:-4:-5]
 M =
       0   0.2   0.4   0.6   0.8   1
      15   11    7     3    -1   -5
```

Innym sposobem tworzenia macierzy w *Matlabie* jest tworzenie macierzy przez połączenie istniejących macierzy. Nawiasy kwadratowe, omówione wcześniej w tym rozdziale, służą również jako operator powiązania macierzy. Wyrażenie `C = [A B]` łączy macierze *A* i *B* poziomo, natomiast `C = [A; B]` łączy je pionowo. Poniższy przykład tworzy nową macierz *M_nowa* przez połączenie macierzy *M_1* oraz wektora *W_1* w kierunku pionowym:

```
>> M_1 = [1 1 1; 2 2 2]
>> W_1 = [7 7 7]
>> M_nowa = [M_1; W_1]
 M_nowa =
           1   1   1
           2   2   2
           7   7   7
```

Przy tworzeniu macierzy przez połączenie, należy pamiętać o tym, że macierz wynikowa musi być prostokątna. Podczas tworzenia macierzy poziomo, każda macierz składowa musi posiadać tą samą liczbę wierszy, przy pionowym – kolumn. *Matlab* udostępnia również funkcje służące do tworzenia macierzy przez połączenie (patrz Tab. 3).

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
cat	C = cat(dim, A, B); połączenie macierzy A i B wzdłuż wymiaru dim; C = cat(1, A, B) jest równoważne z C = [A; B]; C = cat(2, A, B) jest równoważne z C = [A, B].
horzcat	C = horzcat(A1, A2, ...); równoważne z [A1, A2, ...]; poziome połączenie macierzy A1 i A2 itd.; wszystkie łączone macierze muszą posiadać tą samą ilość wierszy.
vertcat	C = vertcat(A1, A2, ...) równoważne z [A1; A2; ...]; pionowe połączenie macierzy A1 i A2 itd.; wszystkie łączone macierze muszą posiadać tą samą ilość kolumn.
repmat	C = repmat(A,m,n); tworzy wielka macierz C, która składa się z *mxn* połączonych kopii macierzy A; rozmiar macierzy C wynosi [size(A,1)*m, (size(A,2)*n]; C = repmat(A, n); tworzy nową macierz przez połączenie *nxn* kopii macierzy A.
blkdiag	C = blkdiag(A1, A2, ...) Tworzy macierz przez ustawienie macierzy A1, A2, itd. wzdłuż diagonali.

Tablica 3: *Lista funkcji dedykowanych do tworzenia macierzy przez połączenie.*

3.1 Stworzyć następującą macierz, przez bezpośrednie wpisanie wartości w Oknie Poleceń:

```
M_A =
        1   30   17   5   10   25
        5    9    4   11    8    3
       13   14    1    7   37   19
```

3.2 Korzystając z następujących funkcji: *eye*, *zeros*, *diag*, *magic*, stworzyć poniższe macierze:

```
M_B =          M_C =          M_D =          M_E =
      1  0  0        0  0  0        7  0  0        8  1  6
      0  1  0        0  0  0        0  8  0        3  5  7
      0  0  1        0  0  0        0  0  9        4  9  2
```

3.3 Korzystając z funkcji *rand* utworzyć macierz M_F o trzech wierszach i trzech kolumnach.

3.4 Korzystając z operatora dwukropka (:) stworzyć poniższą macierz i wektor:

```
M_G =
      1  3  5  7  9  11

M_H =
       0    -3    -6    -9   -12   -15
      4.2   4.3   4.4   4.5   4.6   4.7
      2.5    4    5.5    7    8.5   10
```

3.5 Przy pomocy łączenia macierzy utworzyć:

- macierz dwuwymiarową przez połączenie macierzy *M_A* i *M_B*;
- macierz dwuwymiarową przez pionowe połączenie macierzy *M_C* i *M_D*;
- macierz dwuwymiarową przez połączenie macierzy *M_D*, *M_E* oraz *M_G*;
- macierz trójwymiarową przez połączenie macierzy *M_B*, *M_F*, oraz *M_H*.

3.2 Odwołania do elementów macierzy; informacje o macierzach

Aby pobrać odpowiedni element macierzy, niezbędne jest określenie wiersza oraz kolumny, w której się znajduje. W tym celu, należy posłużyć się odpowiednimi indeksami, umieszczonymi w nawiasach okrągłych występujących zaraz po nazwie zmiennej, przechowującej zawartość macierzy. Zawsze pierwszy indeks określa numer wiersza, natomiast drugi – numer kolumny:

```
macierz(wiersz, kolumna)
```

Jeśli zmienna, z której pobierana jest wartość jest więcej niż dwuwymiarowa, należy również po przecinku podać pozostałe indeksy, określające jednoznacznie położenie elementu pobieranego. Wiersze i kolumny macierzy numerowane są poczynając od lewego górnego narożnika, przy czym numeracja zaczyna się od wartości **1 (a nie zero !)**. Przykładowo, dla macierzy postaci:

```
D =
      1   2   3   4   5
      6   7   8   9   10
     11  12  13  14  15
     16  17  18  19  20
```

wpisanie `D(3,4)` powoduje zwrócenie elementu leżącego w trzecim wierszu i czwartej kolumnie macierzy D, czyli:

```
ans = 14
```

Jeśli natomiast konieczne jest wyłuskanie z macierzy całego wiersza lub całej kolumny, wtedy w miejsce indeksu oznaczającego wiersz (indeks pierwszy) lub kolumnę (indeks drugi) należy wstawić dwukropek, co ilustruje poniższy przykład (wykorzystujący wcześniej zadeklarowaną macierz D):

```
>> D(2, :)
ans = 6 7 8 9 10
>> D(:, 3)
 ans =
        3
        8
        13
        18
```

Jak pokazano w powyższym przykładzie, umieszczenie dwukropka jako pierwszego indeksu powoduje wyłuskanie kolumny o numerze równym wartości podanej jako drugi indeks, natomiast podanie dwukropka w miejscu drugiego indeksu, zwraca wiersz o numerze równym wartości pierwszego indeksu. Naturalnie ten sam mechanizm działa również w przypadku macierzy więcej niż dwuwymiarowych – w takiej sytuacji możliwe jest uzyskanie dostępu do całego wiersza, całej kolumny lub całej tablicy dwuwymiarowej składającej się na daną zmienną wielowymiarową.

W *Matlabie* możliwe jest także odwoływanie się do części macierzy. Tutaj również korzysta się z dwukropka, jednak w tym przypadku rozdziela się nim zakres indeksów wierszy i kolumn, które chcemy wyłuskać z macierzy. Dokładny mechanizm działania operatora dwukropka opisano w poprzednim rozdziale. Przykład wyłuskania części macierzy przedstawiono poniżej:

```
>> D(2:3, 1:4)
 ans =
        6   7   8   9
        11  12  13  14
```

Możliwe są poza tym rozmaite kombinacje opisanych powyżej sposobów dostępu do składowych macierzy, np.:

```
>> D(:, 2:2:4)
 ans =
        2   4
        7   9
        12  14
        17  19
```

Przydatnym słowem kluczowym przy odwołaniu do elementów macierzy, szczególnie w przypadku gdy nie znamy dokładnej liczby kolumn i/lub wierszy danej macierzy, jest słowo *end*. Określa ono ostatni element danego wymiaru macierzy. Komendę z poprzedniego przykładu, można równoznacznie zapisać w postaci:

```
>> D(:, 2:2:end)
```

Specyficznym sposobem odwoływania się do elementów macierzy jest odwołanie za pośrednictwem określonych warunków, jakie muszą spełniać elementy macierzy źródłowej, aby mogły się znaleźć w macierzy docelowej. Należy najpierw stworzyć macierz o tym samym rozmiarze co macierz bazowa (w naszym przypadku macierzą bazową jest macierz D), która jest wynikiem jakiejś operacji logicznej, np.:

```
>> C = D > 13;
 C =
      0   0   0   0   0
      0   0   0   0   0
      0   0   0   1   1
      1   1   1   1   1
```

Wówczas możliwe jest wykonanie operacji tylko na wybranych elementach macierzy bazowej (w naszym przypadku tych, które są większe od 13), np.:

```
>> D(C) = 50;
 D =
       1    2    3    4    5
       6    7    8    9   10
      11   12   13   50   50
      50   50   50   50   50
```

Podstawową informacją o zmiennej typu macierzowego jest jej rozmiar. W *Matlabie* do uzyskania informacji o rozmiarze macierzy służy polecenie *size*, które jako parametr przyjmuje nazwę zmiennej o której chcemy uzyskać informację. Funkcja ta zwraca wektor zawierający informację o rozmiarze zmiennej, co ilustruje poniższy przykład:

```
 macierz =
             8   12   30   24    5   18
             1   21    8    5    0    9
            11    7   34   14   45    3
>> size(macierz);
 ans =
        3   6
>> [nr_wierszy, nr_kolumn] = size(macierz);
 nr_wierszy =
                3
  nr_kolumn=
                6
```

Do pobierania informacji o największym wymiarze wektora bądź macierzy wygodnie jest użyć funkcji:

```
>> length(macierz)
 ans =
        6
```

W przypadku wektorów (zarówno kolumnowych jak i wierszowych, czyli transponowanych) funkcja *length* zwraca zawsze ilość elementów w wektorze, natomiast dla macierzy zwraca tylko ten wymiar, który jest największy.

Dodatkowo istnieje możliwość sprawdzenia liczby elementów macierzy lub wektora za pomocą funkcji *numel* oraz sprawdzenia czy dana macierz jest pusta (jeden w jej wymiarów jest równy

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
length	n = length(A); równoważne z max(size(A)); zwraca wartość największego wymiaru A; jeżeli A jest wektorem, funkcja zwróci wartość długości wektora; jeżeli A jest macierzą, funkcja zwróci wartość największego wymiaru macierzy.
ndims	n = ndims(A); zwraca liczbę wymiarów macierzy A; liczba wymiarów macierzy jest zawsze większa lub równa 2.
numel	n = numel(A); zwraca informację o liczbie elementów macierzy A.
size	n = size(A); zwraca informację o rozmiarach każdego z wymiarów macierzy A, w postaci wektora n o ndims(A) elementach. Jeżeli A jest skalarem, czyli macierzą 1×1, funkcja zwróci wektor [1 1]. [m,n] = size(A); zwraca informację o rozmiarze każdego z wymiarów macierzy A do oddzielnych zmiennych m i n.
isempty	n = isempty(A); zwraca logiczną 1 (prawda) jeżeli macierz A jest pusta; w przeciwnym wypadku zwraca logiczne 0 (fałsz); pusta macierz posiada co najmniej jeden wymiar równy 0.

Tablica 4: *Funkcje służące do pozyskiwania informacji o macierzach.*

zero), przy użyciu funkcji *isempty*. Lista najważniejszych funkcji służących do wyświetlania informacji o macierzach znajduje się w Tab. 4.

3.6 Stworzyć następującą macierz i wektor:

```
M3_2 =
         1   3   5   7   9  11
         2   4   6   8  10  12
        13  14  15  16  17  18

W3_2 =
        20  -3  45  19 -12  25
```

3.7 Odwołać się do następujących elementów macierzy *M3_2*:

- komórki zawierającej liczbę 10 (przez określenie numeru kolumny i wiersza);
- komórki znajdującej się w trzecim wierszu i piątej kolumnie;
- pierwszej i drugiej kolumny (wszystkich wierszy);
- pierwszego i trzeciego wiersza oraz drugiej i trzeciej kolumny;
- kolumny od drugiej do ostatniej (słowo *end*);
- wszystkich elementów, których wartość jest większa niż 9.

3.8 Pobrać następujące informacje o zmiennych:

- jaka jest liczba elementów macierzy *M3_2*;
- jaki jest rozmiar macierzy *M3_2*; liczbę wierszy przypisać do zmiennej *a*, natomiast liczbę kolumn do zmiennej *b*;
- jaka jest długość wektora *W3_2*;
- jaki jest największy wymiar macierzy *M3_2*;
- jaka jest liczba wymiarów macierzy *M3_2*.

3.3 Podstawowe operacje na macierzach

Podstawowe działania arytmetyczne oznacza się w *Matlabie* takimi samymi znakami jak w przypadku zapisu „na papierze", należy jednak pamiętać, że niektóre operacje arytmetyczne (takie jak mnożenie i dzielenie) w odniesieniu do macierzy, zdefiniowane są inaczej niż w przypadku arytmetyki liczb rzeczywistych. Ponadto zawsze wymagane jest, aby wymiary zmiennych podlegających operacjom arytmetycznym były ze sobą w ścisłej relacji. Spis wszystkich operacji zestawiono w Tab. 5.

Warto wspomnieć o możliwości zsumowania elementów każdej z kolumn macierzy, dzięki funkcji *sum*, która jako parametr przyjmuje nazwę zmiennej macierzowej. Domyślnie funkcja oblicza sumę wartości elementów poszczególnych kolumn. Jeśli natomiast wymagane jest uzyskanie sumy wierszy to konieczne jest zdefiniowanie drugiego parametru funkcji *sum*, odpowiadającego za wybór wymiaru, względem którego następuje operacja sumowania. Natomiast suma wszystkich elementów macierzy może być uzyskana dzięki podwójnemu użyciu polecenia *sum* w odniesieniu do tej samej zmiennej. Dla zdefiniowanej w poprzednim rozdziale macierzy *macierz*:

```
>>suma_kolumn = sum(macierz)
 ans =
        20   40   72   43   50   30
>>suma_wierszy = sum(macierz,2)
 ans =
        97   44   114
>>suma_elementów = sum(sum(M))
 ans =
        255
```

Przedstawione powyżej operacje wykonywane są według zasad zgodnych z algebrą macierzy. Jeśli natomiast macierz postrzegana będzie jako tablica liczb, wtedy cenna może okazać się możliwość wykonywania tzw. operacji blokowych, czyli wykonywanie operacji arytmetycznych na elementach o tych samych indeksach. Aby zaznaczyć, że wykonanie operacji ma się odbyć w sposób blokowy, znak operacji nalezy poprzedzić kropką. Spis wszystkich operacji blokowych znajduje się w Tab. 6.

Uwzględniając definicje dodawania i odejmowania macierzy zauważymy, że nie ma różnicy między tymi operacjami a dodawaniem i odejmowaniem blokowym. To samo dotyczy podstawowych operacji matematycznych pomiędzy macierzą a skalarem, np.: pomnożenie macierzy A przez 5 spowoduje, że każdy z elementów macierzy zwiększy się pięciokrotnie.

Większość funkcji *Matlaba* może również operować na macierzach jako tablicach danych, a więc działać na każdym składniku macierzy – w takim przypadku jako argument przy wy-

Nazwa funkcji lub operatora	Opis funkcji i przykładowy sposób wywołania
+	Y = A + B; dodawanie macierzy A i B; macierze A i B muszą być tych samych wymiarów, o ile żadna z nich nie jest skalarem; skalar może być dodany do macierzy dowolnych wymiarów.
-	Y = A - B; odejmowanie macierzy B od macierzy A; macierze A i B muszą być tych samych wymiarów, o ile żadna z nich nie jest skalarem; skalar może być odjęty od macierzy dowolnych wymiarów.
*	Y = A * B; mnożenie macierzy A i B; liczba kolumn macierzy A musi być równa liczbie wierszy macierzy B, o ile żadna z macierzy nie jest skalarem; skalar może być pomnożony przez macierz dowolnych wymiarów.
/	Y = A/B; równoważne z wyrażeniem A*inv(B); dzielenie prawostronne macierzy A przez macierz B.
\	Y = A\B; równoważne z wyrażeniem inv(A)*B; dzielenie lewostronne macierzy A przez macierz B.
^	Y = A^B; podnoszenie macierzy A do potęgi B pod warunkiem, że A jest macierzą kwadratową a B jest skalarem.
'	Y = A'; transpozycja macierzy A.
inv	Y = inv(A); macierz odwrotna do macierzy kwadratowej A.
sum	Y = sum(A); oblicza sumę elementów wektora A; jeżeli A jest macierzą wówczas wynik jest wektorem złożonym z sum poszczególnych kolumn macierzy A; Y = sum(A,n); oblicza sumę elementów macierzy A wzdłuż wymiaru n.
cross	C = cross(A, B); oblicza iloczyn wektorowy wektorów A i B: C = A x B; wektory A oraz B muszą być trójelementowe.
dot	C = dot(A,B); oblicza iloczyn skalarny wektorów A i B; wektory wejściowe funkcji muszą mieć tą samą długość; jeżeli oba wektory A i B są kolumnowe, wówczas C = dot(A, B) oznacza to samo co C = A'*B.

Tablica 5: *Funkcje i operatory macierzowe.*

wołaniu danej funkcji wystarczy, w nawiasach okrągłych, podać zmienną typu macierzowego. Spis podstawowych funkcji matematycznych znajduje się w Tab. 1.

3.9 Stworzyć następujące macierze i wektory:

```
A =                B =
     2   -1   0         1   2   3
    -1    2  -1         2   3   2
     0   -1   2         3   1   3

C =              D =
    -5   3   9         2   6   8
```

Operator	Opis funkcji i przykładowy sposób wywołania
.*	Y = A .* B; mnożenie elementów macierzy A i B o tych samych indeksach; mnożenie odbywa się element po elemencie; A i B muszą mieć te same wymiary, o ile żadna z nich nie jest skalarem; skalar może być mnożony przez macierz dowolnych wymiarów.
./	Y = A ./ B; prawostronne dzielenie blokowe; elementy A są dzielone przez elementy B o tych samych indeksach; A i B muszą mieć te same wymiary, o ile żadna z nich nie jest skalarem; skalar może być dzielony przez macierz dowolnych wymiarów.
.\	Y = A .\ B; lewostronne dzielenie blokowe; elementy B są dzielone przez elementy A o tych samych indeksach; A i B muszą mieć te same wymiary, o ile żadna z nich nie jest skalarem; skalar może być dzielony przez macierz dowolnych wymiarów.
.^	Y = A .^ B; podnoszenie elementów macierzy A do potęgi równej wartościom elementów macierzy B; A i B muszą mieć te same wymiary; wyjątek stanowi sytuacja, w której jedna ze zmiennych jest skalarem; wówczas rozmiar drugiej zmiennej nie ma znaczenia.

Tablica 6: *Operacje blokowe macierzy.*

3.10 Wykonać następujące operacje na macierzach[a]:

- obliczyć sumę macierzy A i B;
- obliczyć różnicę macierzy A i B;
- obliczyć transpozycję macierzy A;
- obliczyć odwrotność macierzy A;
- pomnożyć macierz A przez macierz B;
- obliczyć wynik dzielenia prawostronnego macierzy A przez macierz B;
- obliczyć wynik dzielenia lewostronnego macierzy A przez macierz B;
- podnieść macierz A do drugiej potęgi;

[a] Jedna z operacji zwróci kilka elementów postaci Inf. Oznacza to, że określona wartość jest nieskończona. W opisywanym przypadku jest to wynik dzielenia liczby różnej od zera przez zero. Podobną funkcją informującą o błędnym zdefiniowaniu operacji jest NaN oznaczająca, że wynik operacji nie jest liczbą. NaN można otrzymać w wyniku takich operacji jak 0/0 czy Inf-Inf.

3.11 Wykonać następujące operacje na wektorach:

- uzyskać iloczyn skalarny wektorów C i D;
- uzyskać iloczyn wektorowy wektorów C i D.

3.12 Obliczyć sumę kolumn, sumę wierszy oraz sumę wszystkich elementów macierzy A.

3.13 Wykonać następujące operacje blokowe (na elementach o tych samych indeksach):

- obliczyć iloczyn elementów macierzy A oraz B;
- obliczyć wynik dzielenia prawostronnego elementów macierzy A przez elementy macierzy B; porównać wynik z poprzednim wynikiem dzielenia prawostronnego;
- obliczyć wynik dzielenia lewostronnego elementów macierzy A przez elementy macierzy B; porównać wynik z poprzednim wynikiem dzielenia lewostronnego;
- podnieść wszystkie elementy macierzy A do potęgi 3;
- pomnożyć wszystkie elementy macierzy B przez 10;
- obliczyć wartość funkcji sinus z każdego elementu macierzy A;
- obliczyć wartość funkcji cosinus z każdego elementu macierzy B;
- obliczyć resztę z dzielenia elementów macierzy A przez elementy macierzy B.

4 Ciągi znaków i pobieranie wartości od użytkownika

Łańcuchy, czyli ciągi znaków (litery, cyfry i niektóre znaki specjalne) stanowią osobną klasę typów zmiennych w *Matlabie* i są o tyle ważne, że pozwalają na prostą komunikację z użytkownikiem. Za ich pomocą można przekazywać informacje wyświetlane w Oknie Poleceń oraz pobierać rozmaite wartości od użytkownika. Każdy łańcuch to po prostu wektor liczb zmiennoprzecinkowych, którego długość równa jest ilości znaków w napisie. Kolejne wartości w wektorze to kody ASCII poszczególnych znaków. Zatem wszystkie operacje działające na wektorach działają także na napisach. Można to łatwo sprawdzić definiując napis i dokonując jego transpozycji:

```
>> napis = 'Matlab'
>> napis'
```

Możliwe jest również stworzenie macierzy znaków, w taki sam sposób jak w przypadku tworzenia macierzy liczbowych. Tak samo jak w przypadku liczbowego odpowiednika, każdy wiersz macierzy znaków musi być tej samej długości[5], np.:

```
nazwisko = ['aaa'; 'bbb'; 'ccc'];
```

Przydatne funkcje związane z łańcuchami znaków zebrano w Tab. 7. Sprawdzenie czy zmienna wejściowa jest łańcuchem znaków odbywa się przy użyciu funkcji *ischar*. *Matlab* umożliwia również określenie, jakiego typu jest dany znak w łańcuchu znaków. Funkcja *isletter* odszukuje litery spośród łańcucha znaków, natomiast *isspace* – spacje, np.:

```
>> ciag_znakow = 'Matlab 7.1'
>> ischar(ciag_znakow)
 ans =
        1
>> isletter(ciag_znakow)
 ans =
        1   1   1   1   1   1   0   0   0   0
>> isspace(ciag_znakow)
 ans =
        0   0   0   0   0   0   1   0   0   0
```

[5]Rozmiar łańcucha można uzyskać korzystając ze standardowej funkcji *size*.

Opisane wyżej funkcje mogą okazać się przydatne przy porównywaniu łańcuchów ze sobą. Innym sposobem jest wykorzystanie istniejących funkcji do porównania całych łańcuchów, ich części lub pojedynczych znaków. Funkcja *strcmp* określa czy dwa łańcuchy, które przyjmuje za parametry wejściowe są identyczne. Taką sama funkcjonalność posiada funkcja *strncmp*, przy czym pozwala ona na określenie ile pierwszych n znaków jest porównywanych. Obie funkcje zwracają logiczną wartość 1, gdy łańcuchy są identyczne oraz 0, gdy różnią się, np.:

```
>> str1 = 'hello';
>> str2 = 'help';
>> strcmp(str1, str2)
 ans =
        0
>> strncmp(str1, str2, 2)
 ans =
        1
```

Co więcej, w przypadku gdy łańcuch składa się z samych cyfr, możliwa jest konwersja do wektora liczbowego:

```
>> lancuch = '50 4 13';
>> str2num(lancuch)
 ans =
        50   4   13
```

Najistotniejszym zastosowaniem łańcuchów jest pobieranie informacji od użytkownika. Do tego celu wykorzystuje się polecenie *input*:

```
>> komenda = input('Podaj funkcję do obliczenia:  ');
Podaj funkcję do obliczenia:  sin(pi/4);
```

Należy zwrócić uwagę, że w wyniku tych operacji w Workspace pojawiła się zmienna *komenda* o wartości równej 0.7071. W tym przypadku funkcja *input* pobiera jeden parametr wejściowy jakim jest tekst, który informuje użytkownika jakie dane ma wprowadzić.

Jeżeli oczekujemy, że dane wprowadzone przez użytkownika będą łańcuchem znaków, wówczas funkcja input przyjmie dwa argumenty wejściowe:

```
>> komenda = input('Podaj funkcję do obliczenia:  ', 's');
Podaj funkcję do obliczenia:  sin(pi/4);
```

W takim przypadku, zmienna *komenda* przyjmuje postać łańcucha znaków, a nie obliczonej wartości liczbowej. Aby obliczyć wartość funkcji podanej przez użytkownika, należy wykonać polecenie zapisane w postaci zmiennej łańcuchowej, przy użyciu funkcji *eval*:

```
wynik = eval(komenda)
```

4.1 Stworzyć następujące łańcuchy znaków:

- *napis1* o zawartości *"Koniec rozdzialu 4";*
- *napis2* postaci "*Koniec zajec*";
- *napis3* o zawartości "*10*";
- *napis4* postaci "*log(10)*";

Nazwa funkcji	Opis funkcji i przykładowy sposób wywołania
ischar(A)	tf = ischar(str) zwraca logiczną 1 (prawda) jeżeli str jest znakiem lub łańcuchem znaków; w przeciwnym przypadku zwraca logiczne 0 (fałsz).
isletter(A)	tf = isletter(str) zwraca macierz tych samych wymiarów co str, zawierającą logiczne 1 (prawda) w miejscach, w których w macierzy str znajdują się litery; w pozostałych miejscach zwraca logiczne 0 (fałsz).
isspace(A)	tf = isspace(str) zwraca macierz tych samych wymiarów co str, zawierającą logiczne 1 (prawda) w miejscach, w których w macierzy str znajdują się "białe znaki" (spacje, tabulatory, znaki nowej linii); w pozostałych miejscach zwraca logiczne 0 (fałsz).
strcmp(A,B)	tf = strcmp(str1, str2) porównuje dwa łańcuchy znaków i zwraca logiczną 1 (prawda) gdy str1 i str2 są identyczne (rozmiar i zawartość); w przeciwnym przypadku zwraca logiczne 0 (fałsz).
strncmp(A,B,n)	tf = strncmp(str1, str2, n) porównuje pierwsze n znaków łańcuchów str1 i str2 i zwraca logiczną 1 (prawda) gdy są identyczne; w przeciwnym przypadku zwraca logiczne 0 (fałsz).
str2num(A)	x = str2num(str) dokonuje konwersji łańcucha znaków str, którego każdy element jest zapisany w systemie ASCII, do jego reprezentacji numerycznej. Jeżeli str nie jest liczbą lub macierzą liczb, w wyniku działania funkcji powstanie pusta macierz.
num2str(A)	str = num2str(x) dokonuje konwersji macierzy liczb do łańcucha znaków.
eval(A)	tf = eval(str) wykonuje polecenie zapisane w łańcuchu znaków str o ile jest to poprawne polecenie Matlaba.
input	tf = input('str') wyświetla komunikat w Oknie Poleceń i czeka na odpowiedź użytkownika. Wpisany tekst jest interpretowany jako polecenie, przy wykorzystaniu aktualnie zadeklarowanych zmiennych. Wynik operacji jest zapisany w zmiennej tf. Jeżeli użytkownik naciśnie Enter bez wpisywania tekstu, funkcje input zwróci pustą macierz. tf = input('str','s') wyświetla komunikat w Oknie Poleceń i zapisuje wpisany przez użytkownika tekst jako łańcuch znaków tf.

Tablica 7: *Funkcje dokonujące operacji na łańcuchach znaków.*

4.2 Sprawdzić, które spośród znaków łańcucha *napis1* są literami, a które spacjami.

4.3 Porównać 5 pierwszych znaków zmiennych *napis1* i *napis2*.

4.4 Napisać polecenie, proszące użytkownika o podanie dowolnej liczby w zakresie 1 – 10 i zapisać wynik w postaci łańcucha znaków o nazwie *napis5*. Porównać łańcuchy znaków *napis3* i *napis5*.

4.5 Wymusić wykonanie funkcji wpisanej do zmiennej łańcuchowej *napis4*. Wykorzystać funkcje *str2num* oraz *eval*.