# Laboratorium Techniki Obliczeniowej i Symulacyjnej

Ćwiczenie 3. Operacje logiczne i struktury sterujące.

Opracował: dr inż. Sebastian Dudzik

## 1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z elementami programowania wysokopoziomowego, a w szczególności operacjami logicznymi i strukturami sterującymi języka MATLAB.

## 2. Wprowadzenie

### 2.1. Relacje i wyrażenia logiczne

#### 2.1.1. Operatory porównania

Języki wysokiego poziomu zazwyczaj posiadają specjalny typ przechowujący wartości typu logicznego (prawda lub fałsz). Język MATLAB nie posiada typu logicznego. W języku MATLAB, logicznej prawdzie odpowiada macierz o wszystkich elementach niezerowych, logicznemu fałszowi — macierz pusta lub zawierająca co najmniej jedno zero. Wyrażeń logicznych używa się w instrukcjach sterujących oraz przy przetwarzaniu macierzy. Operatory porównania zestawione w tab. 1 umożliwiają konstrukcje wyrażeń logicznych.

| **Wyrażenie** | **Relacja** |
|---|---|
| $Op1 == Op2$ | Op1 równe Op2 |
| $Op1 \sim= Op2$ | Op1 różne Op2 |
| $Op1 < Op2$ | Op1 mniejsze niż Op2 |
| $Op1 > Op2$ | Op1 większe niż Op2 |
| $Op1 <= Op2$ | Op1 mniejsze równe Op2 |
| $Op1 >= Op$ | Op1 większe Op2 |

Tab. 1. Operatory porównania w jęz. MATLAB

Operatory porównania (relacyjne) badają czy pomiędzy operandami będącymi elementami macierzy zachodzą określone relacje. Jeśli relacja jest spełniona Operatory porównania należy stosować ostrożnie. Reprezentacja zmiennoprzecinkowa jest obarczona niedokładnością. Z tego powodu, nawet przy niewielkiej różnicy pomiędzy wartością spodziewaną a wartością otrzymaną, może dojść do sytuacji nieprzewidzianych (np. pętla nieskończona).

### 2.1.2. Operatory logiczne

W języku MATLAB istnieją trzy rodzaje operatorów logicznych: operujące na elementach macierzy, będących operandami (operandami są macierze), warunkowe operatory logiczne (ang. *short-circuit*) — operujące na skalarnych wyrażeniach logicznych i operatory bitowe — operujące na poszczególnych bitach macierzy lub wartości całkowitych. Operatory logiczne pierwszego rodzaju zestawiono w tab. 2.

| **Operator** | **Funkcja logiczna** |
|---|---|
| Op1 \| Op2 | Alternatywa |
| Op1 & Op2 | Koniunkcja |
| `xor(Op1,Op2)` | Różnica symetryczna |
| ∼Op1 | Negacja |

Tab. 2. Operatory logiczne operujące na elementach macierzy

Warunkowe operatory logiczne (ang. *short-circuit*) służą do wykonywania operacji na skalarnych wyrażeniach logicznych. Umożliwiają one podjęcie decyzji o wartości wyrażenia na podstawie analizy jedynie pierwszego operandu (jeśli jest to możliwe) — jeżeli nie trzeba nie jest brany pod uwagę drugi operand. Kluczową różnicą pomiędzy operatorami operującymi na elementach macierzy i operatorami typu short-circuit jest to, że w przypadku tych pierwszych, jako operandy muszą wystąpić macierze (lub wektory) a w przypadku drugich — wartości skalarne. Operatory typu short-circuit zestawiono w tab. 3.

| **Operator** | **Opis** |
|---|---|
| Op1 && Op2 | Zwraca logiczną prawdę (1), jeżeli oba operandy mają wartość logicznej prawdy (1). Zwraca logiczny fałsz (0), jeżeli którykolwiek z operandów ma wartość logicznego fałszu (0). |
| Op1 \|\| Op2 | Zwraca logiczną prawdę (1), jeżeli jeden lub oba operandy mają wartość logicznej prawdy (1). Zwraca logiczny fałsz (0), jeżeli oba operandy mają wartość logicznego fałszu (0) |

Tab. 3. Operatory typu short-circuit w języku MATLAB

Logiczne operatory bitowe zebrano w tab. 4. Jako argumentów wymagają one nieujemnych liczb całkowitych. W przykładach zamieszczonych w tab. 4 przyjęto następujące wartości skalarne: $A = 28$, bitowo: 11100, $B = 21$, bitowo: 10101.

### 2.1.3. Funkcje logiczne

Korzystając z funkcji logicznych można w wygodny sposób badać właściwości macierzy lub ich elementów. Nie chodzi tu o właściwości w sensie algebraicznym (rząd, dodatnia określoność itp.). Funkcje logiczne można podzielić na dwie grupy: do badania własności macierzy jako całości (tab. 5) oraz do badania własności elementów macierzy (tab. 6).

| Funkcja | Opis | Przykład |
|---|---|---|
| `bitand(A,B)` | Alternatywa bitowa | `bitand(A,B) = 20 (binary 10100)` |
| `bitor(A,B)` | Koniunkcja bitowa | `bitor(A,B) = 29 (binary 11101)` |
| `bitcmp(A,B)` | Uzupełnienie n-bitowe | `bitcmp(A,5) = 3 (binary 00011)` |
| `bitxor(A,B)` | Bitowa różnica symetryczna | `bitxor(A,B) = 9 (binary 01001)` |

Tab. 4. Logiczne operatory bitowe języka MATLAB

| Funkcja | Działanie |
|---|---|
| `exist('nazwa')` | Zwraca 1 (prawda), jeżeli macierz o podanej w apostrofach nazwie istnieje, w przeciwnym razie zwraca fałsz (0) |
| `isempty(x)` | Przyjmuje 1, jeśli `x` jest macierzą pustą, w przeciwnym razie 0 |
| `issparse(x)` | Zwraca 1, jeśli `x` jest macierzą rzadką, w przeciwnym przypadku 0 |
| `isstr(x)` | Zwraca 1, jeśli x jest łańcuchem tekstowym, w przeciwnym przypadku 0 |
| `isglobal(x)` | Zwraca 1, jeśli x jest zmienną globalną (macierzą lub łańcuchem), w przeciwnym przypadku 0 |

Tab. 5. Funkcje logiczne do badania własności całych macierzy

## 2.2. Instrukcje sterujące

Instrukcje sterujące w języku MATLAB można podzielić na cztery grupy:

- instrukcje warunkowe,
- pętle,
- instrukcje obsługi błędów,
- końca programu — `return`

Instrukcje warunkowe pozwalają na wybranie, który blok kodu zostanie wykonany. Aby dokonać wyboru, w zależności, czy warunek jest czynie jest spełniony, należy użyć instrukcji `if`.Aby wybrać spośród pewnej liczby możliwych opcji, należy użyć instrukcji `switch` i `case`.

Instrukcja `if` oblicza wyrażenie logiczne i wykonuje blok kodu (grupę instrukcji) zależnie od wartości tego wyrażenia. Najprostsza składnia instrukcji `if` jest następująca:

```
if wyrażenie_logiczne
   instrukcje
end
```

| **Funkcja** | **Działanie** |
|---|---|
| `any(x)` | Wektory: 1 — jeśli którykolwiek element jest niezerowy Macierze: tworzy wektor wierszowy (kolumny są zerami lub jedynkami). Jeżeli w kolumnie występuje przynajmniej jeden element niezerowy — wartość elementu: 1. |
| `all(x)` | Podobnie jak `any`, 1 — gdy wszystkie elementy są niezerowe |
| `I=find(x)` | Zwraca indeksy niezerowych elementów wektora `x` |
| `[I,J]=find(x)` | Zwraca indeksy wierszy i kolumn niezerowych elementów macierzy `x` |
| `[I,J,V]=find(x)` | Analogicznie do powyższego, dodatkowo wektor V zawiera elementy macierzy `x` |
| `isnan(x)` | Zwraca macierz z elementami=1 gdy dany element `x` nie jest liczbą |
| `isinf(x)` | Zwraca macierz z elementami=1 gdy dany element `x` jest równy +inf lub -inf |

Tab. 6. Funkcje logiczne do badania własności elementów macierzy

Jeżeli wartość wyrażenia logicznego jest prawdą (to znaczy równa jest jeden) MATLAB wykonuje wszystkie instrukcje pomiędzy `if` i `end`. Po linii zawierającej `end` wykonanie programu jest wznawiane. Instrukcje if można zagnieżdżać dowolną ilość razy. Jeżeli w wyniku obliczenia wyrażenia logicznego powstaje macierz lub wektor, aby było ono spełnione, wszystkie elementy muszą być niezerowe. `else` i `elseif` dodatkowo warunkują wykonanie instrukcji `if`. Instrukcja `else` nie posiada warunku logicznego. Instrukcje z nią związane są wykonywane jeżeli poprzedzające `else` wyrażenie po `if` zwróci wartość 0 (warunek po najbliższym, poprzedzającym `if` nie jest spełniony). Instrukcja `elseif` posiada warunek logiczny, który jest obliczany, jeżeli poprzedzający warunek `if` (wyrażenie) nie jest spełniony. Elementy (polecenia) związane z tą instrukcją są wykonywane, jeżeli warunek po `elseif` jest spełniony. Wewnątrz instrukcji `if` można wiele razy użyć instrukcji `elseif`. Jeżeli wartością wyrażenia warunkowego jest macierz pusta, warunek nie jest spełniony.

Inna grupa instrukcji logicznych to: `switch-case-otherwise`. Podstawowa forma instrukcji `switch-case-otherwise`.

```
switch expression (scalar or string)
   case wartość1
      instrukcje  % Wykonywane gdy wyrażenie równe wartość1
   case wartość2
      instrukcje  % Wykonywane gdy wyrażenie równe wartość2
    .
    .
    .
```

```
   otherwise
      instrukcje       % Wykonywane gdy żadna z wartości case
                       % nie jest równa wartości wyrażenia
end
```

Konstrukcja `switch` składa się z:

- słowa kluczowego `switch` i następującego po nim warunku logicznego,
- pewna liczba bloków `case`. Grupy składają się ze słowa kluczowego `case` i następującej po nim możliwej wartości wyrażenia, umieszczonych w jednej linii. W następnych liniach znajduje się dowolna ilość instrukcji (włączając w to instrukcje `switch`),
- opcjonalnej grupy `otherwise`. Składa się ona ze słowa kluczowego `otherwise` za którym znajdują się instrukcje wykonywane w przypadku gdy wyrażenie wyspecyfikowane po `switch` nie przybiera żadnej z wartości wymienionych w blokach `case`,
- instrukcji `end` kończącej działanie bloku `switch-case-otherwise`

Za pomocą pętli możliwe jest powtarzalne wykonywanie bloków kodu. Jedno powtórzenie zwane jest iteracją. Jeśli znana jest wymagana liczba iteracji, wykorzystuje się pętlę `for`. Instrukcja `while` jest bardziej odpowiednia jeżeli liczba wykonań pętli zależy od tego jak długo spełniony bądź nie jest określony warunek. Instrukcje `continue` i `break` dają większą kontrolę nad opuszczeniem (wyjściem z) pętli.

Pętla `for` wykonuje instrukcję lub grupę instrukcji określoną liczbę razy. Pętla `for` ma następującą składnię:

```
for index = start:krok:koniec
    instrukcje
end
```

Domyślną wartością kroku jest 1. Możliwe jest wyspecyfikowanie każdej wartości kroku, włączając w to wartości ujemne. Pętle `for` mogą być zagnieżdżane. Często bardziej wydajną wersję programu obliczeniowego można stworzyć wektoryzując pętlę `for`. Pętle można indeksować za pomocą macierzy

Pętla `while` powtarza instrukcję lub blok instrukcji tak długo jak wyrażenie sterujące przyjmuje wartość logicznej prawdy (w języku MATLAB 1). Pętla `while` ma następującą składnię:

```
while wyrażenie
      instrukcje
end
```

Jeżeli wyrażenie przyjmuje wartość macierzową, każdy element tej macierzy musi być równy 1, aby wykonywanie instrukcji było kontynuowane. Aby zredukować macierz do wartości skalarnej, można wykorzystać funkcję `all` lub `any`.

Opuszczenie pętli while następuje w każdym momencie wykonywanie poprzez użycie instrukcji `break`.

Instrukcja `continue` w miejscu wystąpienia przenosi sterowanie pętli `while` lub `for` do następnej iteracji pomijając część bloku kodu występującego po niej.

Instrukcja `break` kończy wykonanie pętli `for` lub pętli `while`.

Instrukcje obsługi błędów podejmowanie działań w przypadku wystąpienia błędu. Do kontroli, czy określona komenda w kodzie źródłowym powoduje powstanie błędu, służy instrukcja `try`. Jeżeli błąd wystąpił wewnątrz bloku `try`, MATLAB natychmiast przechodzi do odpowiadającego mu bloku `catch`. Blok ten jest odpowiedzialny za obsługę zaistniałego błędu. Ogólna forma instrukcji `try-catch` jest następująca:

```
try
   instrukcja
   ...
   instrukcja
catch
   instrukcja
   ...
   instrukcja
end
```

W powyższym ciągu instrukcji, blok pomiędzy `try` i `catch` są wykonywane dopóki nie wystąpi błąd. Następnie wykonywane są instrukcje pomiędzy `catch` i `end`. Powód błędu może być uzyskany za pomocą `lasterr`.

# 3. Program ćwiczenia

1. Uruchomienie programu MATLAB.
   W ćwiczeniu wykorzystano program MATLAB w wersji 5.3 (R11.1). Uruchomienie programu następuje poprzez skrót na pulpicie (Matlab5.3) lub bezpośrednio z katalogu *C:\MatlabR11\bin*.

2. Uruchomienie programu Wordpad.exe.
   Program można uruchomić poprzez wywołanie: *Start\Programy\Akcesoria\ Wordpad* lub poprzez skrót na pulpicie.

3. Przejście do katalogu roboczego dla grupy laboratoryjnej.
   Domyślnym katalogiem startowym (roboczym) programu MATLAB jest *C:\Matlab R11\work*. Zadanie polega na przejściu do podkatalogu katalogu *work*. Podkatalog (utworzony na pierwszych zajęciach laboratoryjnych) nazwany jest wybranymi 2 nazwiskami studentów, wchodzących w skład grupy laboratoryjnej.

   (a) Wprowadzić:
   `>>pwd`
   W programie MATLAB każde wprowadzone polecenie zatwierdza się klawiszem `<ENTER>`. Zwrócić uwagę na ścieżkę dostępu do katalogu bieżącego.

   (b) Wprowadzić:
   `>>cd` *`nazwa_podkatalogu`*
   Parametr *nazwa_pod-katalogu* powinien składać się z nazwisk 2 wybranych studentów grupy laboratoryjnej (np. `>>cd KowalskiNowak`).

4. Użycie operatorów porównania. Dla: `>>x=[1 5 2 8 9 0 1]` i `>>y=[5 2 2 6 0 0 2]`, wykonać poniższe komendy i zinterpretować ich wyniki.

   (a) `>>x>y` oraz `>>x<y`

   (b) `>>x==y` oraz `>>x<=y`

   (c) `>>x>=y` oraz `>>x|y`

   (d) `>>x&y` oraz `>>x&(~y)`

   (e) `>>(x>y)|(y<x)` oraz `>>(x>y)&(y<x)`

   (f) Skopiować zawartość okna poleceń programu MATLAB do programu Wordpad.

   (g) Wyczyścić zawartość okna poleceń programu MATLAB poleceniem: `>>clc`

5. Zastosowanie indeksowania logicznego. Mając dane wektory: `>>x=1:10` i `>>y=[3 1 5 6 8 2 9 4 7 0]` wykonać poniższe komendy i zinterpretować ich wyniki:

   (a) `>>(x>3)&(x<8)` oraz `>>x(x>5)`

   (b) `>>y(x<=4)` oraz `>>x((x<2)|(x>=8))`

   (c) `>>y((x<2)|(x>=8))` oraz `>>x(y<0)`

(d) Skopiować zawartość okna poleceń programu MATLAB do programu Wordpad.

(e) Wyczyścić zawartość okna poleceń programu MATLAB poleceniem: `>>clc`

6. Przetwarzanie macierzy. Mając dane `>>x=[3 15 9 12 -1 0 -12 9 6 1]` napisać polecenia wykonujące poniższe operacje.

   (a) Zamiana dodatnich elementów `x` na zera. Wprowadzić:
   ```
   >>x(x>0)=0 % x(warunek) wybiera elementy, dla których
   >> % spełniony jest warunek
   ```

   (b) Zamiana wartości będących wielokrotnościami 3 na 3 (wykorzystać funkcję rem). Wprowadzić:
   ```
   >>x(rem(x,3)==0)=3
   ```

   (c) Mnożenie parzystych elementów x przez 5

   (d) Stworzenie wektora `y` złożonego z wartości `x` większych od 10

   (e) Zamiana wartości x mniejszych od średniej na zera (wykorzystać funkcję `mean`)

   (f) Obliczenie sumy elementów x o wartościach nieparzystych (wykorzystać funkcję sum).

   ! Wskazówka: do znalezienia indeksów `I` elementów wektora spełniających określony warunek logiczny służy funkcja `find` (np. `I=find(x<1)` znajduje indeksy elementów wektora `x` mniejszych od 1.

   (g) Skopiować zawartość okna poleceń programu MATLAB do programu Wordpad.

   (h) Wyczyścić zawartość okna poleceń programu MATLAB poleceniem: `>>clc`

7. Analiza fragmentów kodu języka MATLAB (instrukcja `if...elseif...else...end`).

   (a) Przewidzieć wyniki działania następującego fragmentu kodu:

   ```
   if n>1      %Jeśli n<1
       m=n+1   %Obliczenia jeśli n<1 (spełniony warunek)
   else        %W przeciwnym przypadku (warunek niespełniony)
       m=n-1
   end
   ```

   dla podanych niżej wartości zmiennej n:

   i. $n = 7$, $m =?$

   ii. $n = 0$, $m =?$

   iii. $n = -10$, $m =?$

   Ile wynosi m? Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić go. Porównać wyniki działania skryptu z przewidywaniami. Wprowadzić:
   `>>edit`
   Polecenie `>>edit` uruchamia edytor m-plików. W oknie edytora wprowadzić fragment kodu. Zapisać plik w pliku `s1.m` poprzez wywołanie polecenia *Save* z menu *File*. Następnie, wprowadzić:

`>>n=7`
W celu uruchomienia skryptu wprowadzić:
`>>s1`
Powtórzyć powyższe 2 operacje dla pozostałych wartości zmiennej `n`

(b) Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(c) Wyczyścić zawartość okna poleceń programu MATLAB poleceniem: `>>clc`

(d) Przewidzieć wyniki działania następującego fragmentu kodu (postępować jak w p. (a)):

```
if z<5          %Jeśli z mniejsze niż 5
    w=2*z       %Obliczenia jeżeli spełniony warunek
elseif z<10     %W przeciwnym wypadku, jeśli z<10
    w=9-z       %Obliczenia jeżeli spełniony warunek
elseif z<100
    w=sqrt(z)
else
    w=z
end
```

dla podanych niżej wartości zmiennej z:

i. z=1, w=?
ii. z=9, w=?
iii. z=60, w=?
iv. z=200 w=?

Ile wynosi w? Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić go. Porównać wyniki działania skryptu z przewidywaniami.

(e) Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(f) Wyczyścić zawartość okna poleceń programu MATLAB poleceniem: `>>clc`

8. Analiza fragmentów kodu języka MATLAB (instrukcja `switch...case`).

(a) Przewidzieć wyniki działania następującego fragmentu kodu ( jak w p. 7(a)):

```
switch in1
    case -1
        k=2.1^(in1) %Wykonywane jeśli in1=-1
    case 0
        k='zero' %Wykonywane jeśli in1=0
    case 1
        k=[1 2 3] %Wykonywane jeśli in1=1
end
```

dla podanych niżej wartości zmiennej in1:

i. in1=-1, k=?
ii. in1=0, k=?
iii. in1=1, k=?

Ile wynosi k i jakiego jest typu? Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić go dla powyższych wartości zmiennej in1. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(b) Przewidzieć wyniki działania następującego fragmentu kodu ( jak w p. 7(a)):

```
clear
switch in2
    case 10
        k=in2.*ones(3)
    case 15
        k='15'
    case 31
        k=in2+30
    otherwise
        k=0
        disp('Inna wartość zmiennej')
end
```

dla podanych niżej wartości zmiennej in2:

i. in2=10, k=?
ii. in2=15, k=?
iii. in2=31, k=?
iv. in2=100 k=?

Ile wynosi k i jakiego jest typu? Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić go dla powyższych wartości zmiennej in2. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

9. Analiza fragmentów kodu języka MATLAB (pętla `for`)

(a) Przewidzieć wyniki działania następującego fragmentu kodu:

```
for i=1:10
   i
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(b) Przewidzieć wyniki działania następującego fragmentu kodu:

```
for i=1:10
   k(i)=i
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(c) Przewidzieć wyniki działania następującego fragmentu kodu:

```
for i=1:5
   for j=1:5
      m(i,j)=j
   end
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

10. Analiza fragmentów kodu języka MATLAB (pętla `while`)

(a) Przewidzieć wyniki działania następującego fragmentu kodu:

```
i=1;
while i<10
   i
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

! Powyższa pętla to pętla nieskończona. Najczęściej powstaje przez błąd w zapisie algorytmu. Przerwanie działanie skryptu (lub funkcji) jest możliwe poprzez *Ctrl+C*

(b) Przewidzieć wyniki działania następującego fragmentu kodu:

```
i=1;
while i<10
   i=i+1
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

(c) Przewidzieć wyniki działania następującego fragmentu kodu:

```
clc;
i=0;
while i == 0
   i=input(['Program MATLAB. Wyświetlić ten napis '...
           'jeszcze raz? 0 - TAK: ']);
end
```

Napisać skrypt w języku MATLAB, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywaniami. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Wordpad.

## 4. Opracowanie sprawozdania

W sprawozdaniu należy umieścić polecenia oraz wyniki ich działania skopiowane w trakcie ćwiczenia z okna środowiska MATLAB. Do Każdej linii kodu oraz do każdego wyniku, należy dodać komentarz objaśniający.

**Przykład.**
. . . `2+round(6/9+3*2)/2-3` — obliczenie wartości wyrażenia. Funkcja `round(6/9+3*2)` zaokrągla wynik działania `6/9+3*2` do najbliższej liczby całkowitej. . .