Operacje logiczne i struktury sterujące.

(wspomaganie obliczeń inżynierskich)

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z elementami programowania wysokopoziomowego, a szczególności operacjami logicznymi i strukturami języka GNU octave.

Wprowadzenie

Relacje i wyrażenia logiczne

Operatory porównania

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

x `<` y	`true`, jeśli x mniejsze od y
x `<=` y	`true`, jeśli x mniejsze lub równe y
x `==` y	`true`, jeśli x równe y
x `>=` y	`true`, jeśli x większe lub równe y
x `>` y	`true`, jeśli x większe od y
x `!=` y	`true`, jeśli x różne od y

Tab. 1 . Operatory porównania w języku GNU octave.

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).

Operatory logiczne

W języku GNU octave 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.

x `&` y	`true`, jeśli jednocześnie x i y są `true` (logiczne „AND”)
x `|` y	`true`, jeśli przynajmniej jedno: x lub y jest `true` (logiczne „OR”)
`!` *bool*	`true`, jeśli *bool* jest `false` (logiczne „NOT”)

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 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.

x `&&` y	`true`, jeśli jednocześnie x i y są `true`
x `||` y	`true`, jeśli przynajmniej jedno: x lub y jest `true`

Wartość operatora jest wyznaczana od lewej do prawej, a wartości operandów są obliczane tylko do momentu, gdy wartość całego wyrażenia może zostać prawidłowo określona. Operandy są uprzednio przekształcane do skalarów funkcją `all`.

Tab. 3. Operatory typu *short-circuit* w języku GNU octave.

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.

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 GNU octave.

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	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.

Instrukcje sterujące

Instrukcje sterujące w języku GNU octave można podzielić na cztery grupy:

- instrukcje warunkowe,
- pętle,
- instrukcje obsługi błędów,
- końca programu – `return`, `quit`, `exit`.

Instrukcje warunkowe pozwalają na wybranie, który blok kodu zostanie wykonany. Aby dokonać wyboru, w zależności, czy warunek jest czy nie jest spełniony, należy użyć instrukcji `if`. Aby wybrać spośród pewnej liczby możliwych operacji, należy użyć instrukcji `switch` i `case`.
Instrukcja `if` oblicza logicznie 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 jest prawdą (to znaczy równa się jeden) GNU octave wykonuje wszystkie instrukcje pomiędzy `if` i `end`. Po linii zawierającej `end` wykonanie programu jest wznawiane. Instrukcję `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 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. 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 (scalar or string)
     case wartość1
          instrukcje       %Wykonywane gdy wyrażenie jest równe wartość1
     case wartość2
          instrukcje       %Wykonywane gdy wyrażenie jest 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 następujących po nim możliwej wartości wyrażenie, umieszczonych w jednej linii. W następnych liniach znajduje się dowolna ilość instrukcji (włączając w to instrukcje `switch`),
- opcjonalnej grupy `otherwise`. Składa się ona ze sowa 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 zagnieżdżone. 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 GNU octave). 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 wykonywania 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 wykonywanie pętli `for` lub `while`.
Do kontroli, czy określona komenda w kodzie źródłowym powoduje powstanie błędu, służy instrukcja `try`. Jeżeli błąd wystąpił wewnątrz bloku `try`, GNU octave 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, bloki 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`.

Zadania

1. Uruchomić program GNU octave.
2. Uruchomić program Word (lub inny edytor tekstu).
3. 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)|(y<x)` oraz `(x>y)&(y<x)`
   e) Skopiować zawartość okna poleceń programu GNU octave do programu Word.
   f) Wyczyścić zawartość okna poleceń programu GNU octave poleceniem `clc`.
4. 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 GNU octave do programu Word.
e) Wyczyścić zawartość okna poleceń programu GNU octave poleceniem `clc`.

5. 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 GNU octave do programu Word.
   h) Wyczyścić zawartość okna poleceń programu GNU octave poleceniem `clc`.
6. Analiza fragmentów kodu języka GNU octave (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 GNU octave zapisać go 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. Następnie wprowadzić:

```
n=7
```

w celu uruchomienia skryptu wprowadzić:

```
s1
```

Powtórzyć powyższe 2 operacje dla pozostałych wartości zmiennej n

b) Skopiować zawartość okna poleceń programu GNU octave do programu Word.
c) Wyczyścić zawartość okna poleceń programu GNU octave poleceniem `clc`.
d) Przewidzieć wyniki działania następującego fragmentu kodu (postępować jak w punkcie 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 GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi.

e) Skopiować zawartość okna poleceń programu GNU octave do programu Word.
f) Wyczyścić zawartość okna poleceń programu GNU octave poleceniem `clc`.

7. Analiza fragmentów kodu języka GNU octave (instrukcja `switch… case`).
   a) Przewidzieć wyniki działania następującego fragmentu kosu (jak w punkcie 6.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 GNU octave, zapisać na dysku i uruchomić go dla powyższych wartości zmiennej in1. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

b) Przewidzieć wyniki działania następującego fragmentu kodu:

```
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 GNU octave, zapisać na dysku i uruchomić go dla powyższych wartości zmiennej in1. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

8. Analiza fragmentów kodu jezyka GNU octave (pętla `for`):
   a) Przewidzieć wyniki działania następującego fragmentu kodu:

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

Napisać skrypt w języku GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

b) Przewidzieć wyniki działania następującego fragmentu kodu:

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

Napisać skrypt w języku GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

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 GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

9. Analiza fragmentów kodu (pętla `while`)
   a) Przewidzieć wyniki działania następującego fragmentu kodu:

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

Napisać skrypt w języku GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

UWAGA!!! Powyższa pętla to pętla nieskończona. Najczęściej powstaje przez błąd w zapisie algorytmu. Przerwanie działania 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 GNU octave, zapisać na dysku i uruchomić. Porównać wyniki działania skryptu z przewidywanymi. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.