Metody obliczeniowe
Laboratorium 1

1 Macierze

1.1 Tworzenie macierzy

Macierze można zdefiniować poprzez:

- wymienienie elementów
- wygenerowanie elementów
- sklejenie innych z macierzy

1.1.1 Wymienienie elementów

A = [1 2 3; 7 8 9]
A = [1,2,3; 7,8,9]
A = [1 2 3
7,8,9]

1.1.2 Generowanie elementów

Jednym ze specyficznych operatorów w programie Matlab/Octave jest dwukropek. Umożliwia on m.in. tworzenie wektorów o równomiernie rozłożonych elementach. Polecenie:

$p : q : r$

utworzy wektor postaci:
$[p, p + q, p+, p + 2q, \ldots, r]$
Jeśli parametr q zostanie pominięty domyślnie zostanie użyty krok $q = 1$.

Przykłady zastosowania:

```
>> 1:7
   ans = 1 2 3 4 5 6 7
>> 1:3:10
   ans = 1 4 7 10
```

Tabela 1: Funkcje wspomagające generowanie macierzy

Funkcja	Opis
eye(n) eye(n,m)	macierz jednostkowa o wymiarach $n \times n$ lub $n \times m$
ones(n) ones(n,m)	macierz wypełniona jedynkami
rand(n) rand(n,m)	macierz wypełniona liczbami pseudolosowymi o rozkładzie jednostajnym na [0,1]
zeros(n) zeros(n,m)	macierz wypełniona zerami

1.1.3 Sklejanie

Często spotykanym sposobem jest również tworzenie macierzy z innych macierzy.

Przykład:

```
>> A = 1:4 ;

>> B = 1:5 ;
>> C = 0 ;
>> D = [A, C ; B ]
   D =1 2 3 4 0
      1 2 3 4 5
```

Należy przy tym pamiętać, aby wymiary łączonych macierzy zgadzały się. W przeciwnym przypadku próba utworzenia macierzy zakończy się komunikatem błędu.

1.2 Usuwanie części macierzy

Polecenie
>> []
definiuje macierz pustą. Przypisanie takiej macierzy do wiersza lub kolumny spowoduje ich usunięcie, np:

```
>> D(:,2:3) = []
   D =1 4 0
      1 4 5
>> D(1,:) = []
   D =1 3 4 5
```

Uwaga: W ten sposób możemy usuwać *całe* wiersze, kolumny bądź ich grupy. Usunięcie pojedynczego elementu macierzy nie miałoby sensu i wygeneruje błąd.

Tabela 2: Operatory arytmetyczne

Operator	macierzowy	tablicowy
dodawanie	+	+
odejmowanie	−	−
mnożenie	*	.*
dzielenie prawostronne	/	./
dzielenie lewostronne	\	.\
potęgowanie	^	.^
transpozycja	'	.'

2 Operacje na macierzach

2.1 Operatory arytmetyczne (macierzowe i tablicowe)

Matlab/Octave umożliwia wykonywanie na macierzach dwóch rodzajów operacji arytmetycznych. Operatory **macierzowe** są określone regułami algebry macierzowej. Natomiast operatory **tablicowe** traktują macierze jako tablice liczb i umożliwiają wykonywanie operacji na odpowiadających sobie elementach macierzy. Każdy operator macierzowy ma swój tablicowy odpowiednik poprzedzony kropką. Wyjątkiem są operatory dodawania i odejmowania, które z definicji operują na elementach macierzy.

Uwaga: Aby wykonać jakąkolwiek operację tablicową macierze *muszą* mieć takie same rozmiary bądź jedna z nich musi być skalarem.

Uwaga: Zgodnie z definicją iloczyn (w sensie Cauchy'ego) dwóch macierzy może być określony tylko wtedy gdy liczba kolumn pierwszej macierzy jest równa liczbie wierszy drugiej macierzy. Mnożenie macierzowe nie jest przemienne!

Jak wiadomo dzielenie jest operacją odwrotną do mnożenia ($\frac{a}{b} = a \cdot b^{-1}$). Matlab umożliwia dwa rodzaje dzielenia. Dzielenie prawostronne dwóch macierzy zdefiniowane jest następująco:

$$A/B = A \cdot B^{-1}$$

Z kolei mnożenie lewostronne:

$$A\backslash B = A^{-1} \cdot B$$

W analogiczny sposób zdefiniowane jest dzielenie tablicowe. Każdy element c_{ij} macierzy $C = A./B$ jest równy $c_{ij} = a_{ij} \cdot \frac{1}{b_{ij}}$. Z kolei dla tablicowego dzielenia prawostronnego, czyli gdy $C = A.\backslash B$ mamy: $c_{ij} = \frac{1}{a_{ij}} \cdot b_{ij}$.

2.2 Operatory porównań

W programie Matlab/Octave porównywać możemy całe macierze (tablice). Warunkiem są takie same rozmiary obu operandów (Wyjątkiem jest sytuacja, gdy jeden z nich jest skalarem. Wtedy skalar ten jest porównywany z każdym elementem drugiego operandu). Wynikiem działania operatorów porównań jest

Tabela 3: Operatory porównania

Operator	Relacja
==	równość
~=	różność
>	większe niż
<	mniejsze niż
>=	większe lub równe
<=	mniejsze lub równe

Tabela 4: Operatory i funkcje logiczne

Operator	Funkcja logiczna
~	negacja
&	koniunkcja
|	alternatywa
xor	różnica symetryczna

macierz (tablica) takiej samej wielkości co operandy wypełniona jedynkami gdy odpowiadające sobie elementy spełniają relację i zerami w przeciwnym przypadku.

I tak na przykład, jeśli A = [-2 3 0], B=[3 3 3], wyrażenie $A >= B$ ma wartość [0 1 0], natomiast $A \sim= B$ będzie równe [1 0 1].

2.3 Operatory logiczne

Najnowsze wersje programu udostępniają trzy rodzaje operatorów i funkcji logicznych. Są to operatory:

- „elementowe" (element-wise) - wykonują operacje na elementach macierzy traktując liczbę 0 jako zero logiczne i każdą inną wartość jako jedynkę logiczną.
- bitowe (bit-wise) - operują na liczbach całkowitych lub zbudowanych z nich macierzach o takich samych rozmiarach. Wykonują operacje logiczne na wartościach bitowych odpowiednich liczb.
- „szybkie" (short-circuit) - operatory alternatywy i koniunkcji, które działają na wyrażeniach logicznych zawierających wartości skalarne. W przypadku gdy po sprawdzeniu pierwszego operandu wartość logiczna wyrażenia jest określona ewaluacja kolejnych operandów jest pomijana. (odpowiedniki operatorów || i && w C/C++)

Ponieważ na laboratorium będą używane głównie pierwsze z wymienionych operatorów logicznych, tylko one są podane w tab. 4. Szczegółowy opis wszystkich, zarówno logicznych jak i pozostałych, operatorów można odnaleźć tutaj.

3 Programowanie

3.1 Instrukcja warunkowa `if`

Instrukcja warunkowa `if`

Ogólna postać instrukcji:

`if` *wyrazenie_warunkowe*1
*instrukcje*1
`elseif` *wyrazenie_warunkowe*2
*instrukcje*2
`else`
instrukcje_koncowe
`endif`

Instrukcja wyboru `switch`

`switch` *wyrazenie*
`case` *wartosc*1
    *instrukcje*1
`case` *wartosc*2
    *instrukcje*2
...
`otherwise`
    instrukcjen
`endswitch`

Pętla `for`
`for` *zmienna* = *macierz_wartosci*
    instrukcje
`endfor`

Pętla `while`
`while` *wyrazenie*
    instrukcje
`endwhile`

Pętla `do-until`
`do`
    instrukcje
`until` *wyrazenie*

Instrukcja `break`
Powoduje przerwanie wykonywania danej pętli. Opuszczany jest tylko jeden poziom zagłębienia.

Instrukcja `return`
Powoduje bezwarunkowe przerwanie wykonywania danego skryptu (funkcji) i powrót do miejsca jego (jej) wywołania.

W składni języka Matlab instrukcje `if`, `switch`, `for`, `while` są kończone poleceniem `end`, a pętla `do-until` nie występuje.

4 Funkcje

W programie Matlab/Octave istnieje wielu rodzajów funkcji takich jak: funkcje w m-plikach, funkcje inline, funkcje anonimowe czy funkcje zagnieżdżone. Jednak na zajęciach nie będzie istniała potrzeba korzystania z tych wszystkich możliwości. Dlatego celem uniknięcia podawania zbędnych informacji poniższy opis jest ograniczony jedynie do dwóch rodzajów, których znajmość uznano za użyteczną.

4.1 Funkcje w m-plikach

Podstawowym sposobem tworzenia własnej funkcji jest umieszczenie jej treści wraz z definicją w pliku z rozszerzeniem .m. Ogólna postać definicji funkcji wygląda następująco:

`function` $[wartosc1, wartosc2, \ldots] = nazwa\_funkcji(parametr1, parametr2, \ldots)$
instrukcje
`endfunction`

A oto przykładowa funkcja zapisana w pliku *porownaj.m*:

```
% Tutaj mozemy dac opis funkcji, ktory bedzie
% wyswietlany przez program gdy wpiszemy w linii
% polecen: help porownaj

% A ta linia wyswietlona nie zostanie

function k = porownaj(a,b)

    if (a<b)
        k = -1 ;
    elseif (a>b)
```

```
    k = 1 ;
  else
    k = 0 ;
  end
endfunction
```

Zarówno parametry jak i wartości funkcji są opcjonalne. Zmienne utworzone wewnątrz takiej funkcji są lokalne, czyli nie są widoczne w skrypcie, z którego została ona wywołana. Podobnie zmienne utworzone poza funkcją (znajdujące się w przestrzeni roboczej) nie są dostępne w jej wnętrzu. Aby zmienne istniejące we wspólnej pamięci programu były dostępne wewnątrz funkcji muszą być zadeklarowane jako globalne(polecenie `global`).

Charakterystyczną cechą Matlab'a/Octave jest to, że argumenty funkcji są kopiowane jedynie wtedy gdy są modyfikowane w jej ciele.

Argumenty funkcji są przekazywane przez wartość, jednak ich kopiowanie następuje jedynie wtedy gdy są modyfikowane w jej ciele. (W przykładowej funkcji *porownaj* argumenty funkcji nie były kopiowane)

Więcej wiadomości na temat funkcji można odnaleźć tutaj.

UWAGA: Nazwa funkcji z definicji nie musi być taka sama jak nazwa pliku. Co więcej, w pliku definiującym funkcję może być zdefiniowanych więcej niż jedna funkcja. Jednak odwołując się do takiego pliku interpreter jako funkcję główną traktuje pierwszą zdefiniowaną funkcję. Celem uniknięcia niepotrzebnych pomyłek zaleca się w związku z tym, aby pierwszą funkcję w pliku nazywać identycznie jak sam plik wewnątrz którego jest ona zdefiniowana.

4.2 Funkcje anonimowe

W przypadku gdy mamy do napisania "prostą"/krótką funkcję istnieje możliwość jej zdefiniownania bezpośrednio w kodzie (lub w linii poleceń) bez konieczności tworzenia m-pliku. Składnia takiego polecenia wygląda następująco:

nazwa_fun = @(lista_argumentow) cialo_funkcji ;

Niech za przykład posłuży prosta funkcja, która oblicza wartość trójmianu kwadratowego przy czym parametry tego wielomianu są również podawane jako argumenty funkcji:

w = @(x,a,b,c) a*x.^2 + b*x + c ;

Wykorzystując funkcję zdefiniowaną w ten sposób możemy następnie obliczyć np. wartość wielomianu $2x^2 - 3x + 1$ dla punktów z przedziału $< -1, 1 >$ następująco:

y = w(-1:0.01:1, 2, -3, 1) ;

5 Ćwiczenia

1. Utwórz własny skrypt z rozszerzeniem .m.
2. Sprawdź wyniki poniższych poleceń wpisując je do skryptu i wykonując go

```
% TWORZENIE MACIERZY
% Wymienianie elementów
A = [1 2 ...
          3; 7 8 9]

B = [1, 2, 3; 7, 8, 9]

C = [1 2 3
     7,8,9]

% Generowanie
%  Operator :
w1 = 1 : 0.31 : 3
w2 = -3:7
w3 = 10:-2:0

% Wykorzystanie funkcji
X = ones(2)
Y = eye(3)
Z = rand(2,5) % round(10*rand(3))

% Zbudowanie z innych macierzy
U = [A, X ; Z]
%  Mieszanie powyzszych sposobow
V = [ [9:-1:7; 1:3], zeros(2,1); 1 1 1 1]

%-----------------------------------------------------
% ODWOLYWANIE SIE DO ELEMENTOW MACIERZY
V(1,3)
V(3,1)
V(:,2)
V(2,:)
V(2, 1:3)
V(2, [1 3 4])
V([1 3], [1 3 4])
V(7)
V(:)

%-----------------------------------------------------
```

```
% USUWANIE FRAGMENTOW MACIERZY
V(:,4) = []
V(2,:) = []
% V(1,1) = []   % !!!

%------------------------------------------------------
% OPERACJE NA MACIERZACH
A = [1 0 0;
     2 3 -1;
     0 7 2]

B = [1 0 3;
    -1 5 2;
     2 2 2]

A+B
A-B
A+2

2*A
2.*A

disp('Mnozenie macierzowe')
MM1 = A*B
MM2 = B*A

disp('Mnozenie tablicowe')
MT1 = A.*B
MT2 = B.*A

disp(['Dzielenie',' ','macierzowe'])
DM1 = A/B
DM2 = A\B
DM3 = B/A
DM4 = B\A

disp('Dzielenie tablicowe')
DT1 = A./B
DT2 = A.\B
DT3 = B./A
DT4 = B.\A

disp(['Potego' 'wanie'])
PM = A^2
A*A
PT = A.^2
```

```
A.*A

disp(['Transpozycja'])
A.' % transpozycja
A'

C = [1-i 2i    3;
    -1   5+3i  2+3i;
     3+i 2     2]
C.'
C' % hermitowskie sprzezenie macierzy (dla m. zespolonych)

% Operacje porownan i funkcje logiczne
A == B
A ~= B
2 < A
% A > B
% A < B
% A >= B
% A <= B

~A
A & B
% A | B
% xor(A,B)

A = [2 0 7;
     0 0 1]

all(A)
any(A)

%-----------------------------------------------------
%-------------- INSTRUKCJE STERUJACE ---------------%

                  % INSTRUKCJA "if"
a = 2
b = 4

if (a<b)
    k = -1
elseif (a>b)
    k = 1
else
k = 0
endif
```

```
                    % INSTRUKCJA "switch"
c = 2 ;
switch c
   case 0
      zeros(2)
   case 1
      disp('wpisano 1')
   otherwise
      disp(['wpisano ' num2str(c) ' tym razem'])
endswitch

                        % PĘTLA "for"
suma=0;
wekt = 1:10 ;
for k = wekt
   suma=suma+k;
endfor
S=suma;

                         % PĘTLA "while"
suma=0;
k=1;
while k<=10
   suma=suma+k;
   k=k+1;
endwhile
S=suma;

                       % INSTRUKCJA "break"
      % zatrzymuje wykonywanie pętli "for" lub "while" ...
       %   ... i powoduje wyjscie poza petle
suma=0;
i=1;
while i<=10
   suma=suma+i; break
   i=i+1;
endwhile
S=suma;
```

3. Napisz funkcję porownaj.m opisaną w rozdziale 4. Wywołaj ją z linii poleceń oraz z wnętrza powyższego skryptu.

4. Utwórz plik wielomian1.m i zaimplementuj w nim funkcję, która będzie liczyć wartość wielomianu $x^3 + x^2 - 3x - 3$. Narysuj wykres tej funkcji dla $x \in < -5, 5 >$.

```
x = -5:0.01:5 ;
y = wielomian1(x) ;
plot(x,y)
```

Uwaga: Jako parametr funkcja powinna przyjmować nie tylko wartości skalarne, ale również macierze (wektory). Wykorzystaj w tym celu pętle.

5. Utwórz plik wielomian2.m i zaimplementuj w nim funkcję, która będzie liczyć wartość wielomianu x^3+x^2-3x-3 przy wykorzystaniu tablicowych operatorów arytmetycznych.

6. Porównaj czas działania obu funkcji w przypadku, gdy argument jest wektorem o małej/dużej ilości elementów. Test można przeprowadzić przy pomocy następującego kodu:

```
x = linspace(-5,5,1001) ; % x = linspace(-5,5,10001) ;
tic
y = wielomian1(x) ;
toc

tic
y = wielomian2(x) ;
toc
```

7. Zdefiniuj funkcję wielomian3 obliczającą wartość wielomianu z zadania 4 i wywołaj ją dla tych samych argumentów. (Należy oczywiście wykorzystać tablicowe operatory arytmetyczne).

8. Napisz funkcję, która w wyniku zwróci macierz kwadratową o losowych elementach będących liczbami całkowitymi z zakresu $< 0, 10 >$ oraz jej ślad (Ślad macierzy to suma elementów leżących na jej przekątnej głównej).

9. Napisz funkcję silnia korzystając z instrukcji pętli.

10. Napisz funkcję silnia wykorzystując rekurencję.

11. Napisz funkcję realizującą poniższy wzór:

$$y = \begin{cases} \sin(x) & \text{dla} \quad x < 0 \\ x^2 & \text{dla} \quad 0 \leqslant x \leqslant 1 \\ 1 & \text{dla} \quad x > 1 \end{cases}$$

Narysuj wykres tej funkcji w przedziale $< -3, 3 >$ dla argumentów równoodległych z krokiem 0.1.

12. (*) Korzystając z operatorów porównań oraz operatorów tablicowych funkcję z zadania 11 można zapisać jako pojedyńczą instrukcję. Zdefiniuj tę funkcję jako funkcję anonimową.