Pakiet Octave

http://www.octave.org

Specjalizowany system programowania

służący do wykonywania obliczeń numerycznych.

- **język programowania**
- **funkcje standardowe**
- **interpreter**

Dla systemu Unix / Linux (są też wersje dla Windows)

Wykonywanie interakcyjne

```
> octave
octave:1 .....
```

zakończenie pracy : `quit exit Ctrl-C`

Wykonywanie wsadowe

- **przygotować plik XXX z programem**

```
octave XXX
```

przygotować plik YYY o budowie:

```
#! /usr/bin/octave -qf
..............
```

i wywołać przez nazwę pliku YYY

Komentarze

```
# Dowolny tekst
```

Polecenia organizacyjne

```
help  hasło
clc
home
```

Zmienne

(nazwa , wartość)

nazwa - ciąg znaków od litery,

duże i małe litery rozróżniane

wartość - liczbowa, tekstowa, macierz, struktura

Typy danych

skalarne : całkowite, rzeczywiste i zespolone (reprezentacja IEEE 754)

struktury danych : macierze, zakresy, teksty, struktury (rekordy)

Typy skalarne

```
105
-22
1.342223
-43323.543
1.34e+4
-234.322e-201

3 + 5i
17.3 - 22.3i
-2.31e-4 + 33.11i
# i j I J
```

Macierze

```
[  ,  ,  ... ;  ,  , ... ; ... ]
```

dowolna liczba wymiarów

elementy : wyrażenia arytmetyczne

```
[ 2 , 4.5 , 44 ; 3 , 11.1 , 2 ]
```

```
  2    4.5    44
  3    11.1   2
```

```
[ 1 , 3 + 4i ; 3 , 7.0 ]
```

```
  1 + 0i    3 + 4i
  3 + 0i    7 + 0i
```

Zakresy

```
1 : 5          [ 1 , 2 , 3 , 4 , 5 ]
1 : 3 : 8      [ 1 , 4 , 7 ]
```

Można użyć do definiowania macierzy

(ale uwaga na liczbę elementów)

```
a = [ 4 : 7 ; 15 : 18 ]
a =
   4   5   6   7
  15  16  17  18

b = [ 1 : 5 ;  2 : 4 ]
error: number of columns must match


c = zeros (2 , 3)
c =
   0  0  0
   0  0  0
c = ones (3 , 2)
c =
   1  1
   1  1
   1  1
c = sqrt(1:5)
c =
    1.0000  1.4142  1.7321  2.0000  2.2361
```

Funkcje standardowe

```
is_matrix(a)           # prawda 1, fałsz 0
is_vector(a)           # prawda 1, fałsz 0
is_scalar(a)           # prawda 1, fałsz 0
is_square(x)           # liczba wymiarów, 0
is_symmetric(x,tol)    # liczba wymiarów, 0
```

Teksty

```
"Dowolny tekst."
'Inny ciekawy tekst.'
```

Znaki sterujące

```
\n    # nowa linia
\t    # tabulacja

"Argunent\tWynik\n----------------------\n"
Argument      Wynik
------------------
```

Sklejanie tekstów

```
[ "aaa" , "bbb" , "ccc" ]
aaabbbccc
```

Funkcje standadowe dla tekstów (liczne)

```
blanks(n)     # tekst złożony z n spacji
isstr(a)      # 0/1
index(s,t)    # pierwsze wystąpienie t w s
split(s,t)    # podział s na części co t
strcmp(s1,s2)    # porównanie
tolower(s)
toupper(s)
```

Konwersje

```
bin2dec   dec2bin   dec2hex   hex2dec
```

Klasyfikacja

```
isalfa  isalnum  isascii
isdigit  islower isupper

islower("abCdEFg")
1101001
```

Struktury danych

```
osoba.nazwisko = "Kowalski"
osoba.pesel = "57020102341"
osoba.pensja = 1500
osoba.pokoje = [ 12 , 34 ; 11 , 56 ]

nowy = osoba
```

Elementem struktury może być inna struktura

Wyrażenia

Indeksowanie wektorów i macierzy

```
a=[11,21,34] ;
a(2)
ans = 21      # element


a(2:3)
ans =
    21   34    # podciąg
```

```
a=[11,12,13 ; 21,22,23 ; 31,32,33] ;
a(2,3)
ans = 23      # element
a(1,:)
ans =
    11   12   13    # pierwszy wiersz
a(2:3 , 1:2)
ans =
    21   22
    31   32

a([2,3],[1,2])
ans =
    21   22
    31   32

a([1,0,1],[0,1,0])  # indeksowanie binarne
ans =
    12
    32
```

```
b = 15
b([1,1,1,1])
ans =
   15
   15
   15
   15
b([1],[1,1,1,1])
ans =
   15  15  15  15

b([1,1],[1,1,1])
ans =
   15  15  15
   15  15  15

a = [1 , 2 ; 3 , 4 ] ; # sklejanie macierzy
x = a
x =
   1  2
   3  4
```

```
x = [ x , a ]
x =
    1   2   1   2
    3   4   3   4
x = a;
x = [x ; a]
x =
    1   2
    3   4
    1   2
    3   4
```

Wywołanie funkcji

```
k = rand()
l = sqrt(2)
m = ones(5,15)
```

Może być więcej niż 1 wynik

Przekazywanie argumentów z kopiowaniem

(przez wartość)

Dopuszczalna rekursja (domyślnie do 256 poziomów)

Operator przypisania

```
a = 1
a = [ 1 ,5 ; 4 , 9 ]
a = b = [ 1 : 3 : 58 ]
```

Operatory arytmetyczne

(dla wartości skalarnych i macierzy)

```
+   .+    -    .-
# dla macierzy i skalarów
a = [ 1 , 2 ; 3 , 4 ]
a =
   1   2
   3   4
b = [ 5 , 6 ; 7 , 8 ]
b =
   5   6
   7   8
a + b
ans =
   6    8
   10  12
```

```
a + 5
ans =
    6    7
    8    9
a - 10
ans =
   -9   -8
   -7   -6
10 - a
ans =
    9    8
    7    6


++  --
# zwiększanie i zmniejszanie,
   prefiksowe i postfiksowe
++x     # x = x + 1  wykonanie przed
                     obliczeniem wyrażenia
--x     # x = x - 1  wykonanie przed
                     obliczeniem wyrażenia
```

```
x++      # x = x + 1  wykonanie po
                     obliczeniu wyrażenia
x--      # x = x - 1  wykonanie po
                     obliczeniu wyrażenia

x = 3
y = 5
s = x++ + y        #  s = 8, x = 4, y = 5

x = 3
y = 5
s = ++x + y        #  s = 9, x = 4, y = 5


*  # mnożenie macierzy, skalarów,
   # macierz * skalar

.* # mnożenie odpowiadających sobie
   # elementów macierzy
```

```
a = [ 1 , 2 ; 3 , 4 ]
a =
    1    2
    3    4

b = [ 5 , 6 ; 7 , 8 ]
b =
    5    6
    7    8
a * b
ans =
    19   22
    43   50
a .* b
ans =
     5   12
    21   32
a * 3
ans =
     3    6
     9   12
```

```
./ i .\   # dzielenie element po elemencie
          # a ./ b = b .\ a
a = [ 1 , 2 ; 3 , 4 ]
a =
   1   2
   3   4
b = [ 5 , 6 ; 7 , 8 ]
b =
   5   6
   7   8
a ./ b
ans =
   0.20000   0.33333
   0.42857   0.50000

a .\ b
ans =
   5.00000   3.00000
   2.33333   2.00000
```

```
/  i  \    # dzielenie skalarów
           # k / l  =  l \  k

\    # dla macierzy
     # x = A \ B = inverse (A) * B
     # czyli rozwiązanie układu
     # równań liniowych
     # Ax = B (B wektor kolumnowy)
A = [ 1 , 2 ; 3 , 4 ]
A =
    1    2
    3    4
B = [ 3 ; 7 ]
B =
    3
    7
x = A \ B
x =
    1
    1
```

```
/    # dla macierzy
     # x = B / A = (inverse (A') * B')'
     # czyli rozwiązanie układu
     # równań liniowych
     # xA = B (B wektor wierszowy)
A = [ 1 , 2 ; 3 , 4 ]
A =
    1    2
    3    4
B = [ 4 , 6 ]
B =
    4    6
x = B / A
x =
    1    1

^   , **    # potęgowanie  skalar, skalar
.^ , .**   # potęgowanie element po
           # elemencie macierz, macierz
```

```
a = [ 1 , 2 ; 3 , 4 ]
a =
  1  2
  3  4
b=[ 2 , 3 ; 4 , 5 ]
b =
  2  3
  4  5
a .** b
ans =
     1     8
    81  1024

.' # transponowanie macierzy
'  # transponowanie: dla macierzy
   # z liczb rzeczywistych to samo co '
   # dla macierzy z liczb zespolonych
   # conj (x.')
   # conj(x) - macierz z elementów
               sprzężonych do x
```

```
a = [ 1 , 2 ; 3 , 4 ]
a =
  1  2
  3  4
a'
ans =
   1  3
   2  4
z = [ 1+2i , 2+3i ; 3-4i , 4-5i ]
z =
  1 + 2i  2 + 3i
  3 - 4i  4 - 5i
z.'
ans =
  1 + 2i  3 - 4i
  2 + 3i  4 - 5i
z'
ans =
  1 - 2i  3 + 4i
  2 - 3i  4 + 5i
```

Operatory porównania **1 - prawda, 0 = fałsz**

- skalary

- dwie macierze o równych rozmiarach

(wynik macierz 0 / 1)

- macierz i skalar (wynik macierz 0 / 1)

== < <= > >= != ~= <>

Do porównywania tekstów funkcja strcmp.

Operatory logiczne

```
&    |    !    ~        # pełne
&&   ||                  # skrótowe

a = [ 1 , 2 ; 3 , 4 ] ;
b = [ 2 , 3 ; 4 , 5 ] ;
c = [ 1 , 5 ; 9 , 2 ] ;
a > b | b <= c
ans =
  0  1
  1  0
```

Obowiązują priorytety operatorów. Do zmiany kolejności obliczeń nawiasy okrągłe (i)

Instrukcje

Instrukcja warunkowa

```
if  ( warunek )
  ciąg_instrukcji
endif

# warunek - wyrażenie arytmetyczne
            lub logiczne
            0 - fałsz,  nie 0 - prawda
if  ( warunek )
  ciąg_instrukcji_1
else
  ciąg_instrukcji_2
endif

if  ( warunek_1 )
  ciąg_instrukcji_1
elsif ( warunek_2 )
  ciąg_instrukcji_2
. . . . . .
```

```
else
  ciąg_instrukcji_3
endif

a = 4
if ( a > 7 )
   b = 4 ;
   c = 1 ;
else
   b = 0 ;
   c = 9 ;
endif
b = 0
c = 9

if ( b + 1 )
  s = a + c ;
else
  s = a - c - 1 ;
endif
```

```
if ( k == 8 )
.................
```

Przełącznik

```
switch wyrażenie
  case wartość_1
    ciąg_instrukcji_1
  case wartość_2
    ciąg_instrukcji_2
  ...
  otherwise
    ciąg_instrukcji_n
endswitch
switch a + 3
 case 1
 b = 1 ;
 case 5
 b = 15;
 otherwise
 b = 0;
endswitch
b = 15
```

Instrukcja pętli `while`

```
while  ( warunek )
  ciąg_instrukcji
endwhile
# zakończenie wykonywania
  gdy warunek fałszywy.
```

```
fib = ones ( 1, 10 ) ;
i = 3 ;
while ( i <= 10 )
 fib ( i ) = fib ( i - 1 ) + fib ( i - 2 );
 i++ ;
endwhile
fib =
   1   1   2   3   5   8  13  21  34  55
```

Instrukcja pętli `for`

```
for  zmienna = wyrażenie
  ciąg_instrukcji
endfor
```

```
# w kolejnych wykonaniach zmiennej
  przypisywane są kolejne składniki
  wyrażenia (wektora, macierzy, zakresu)

s = 0 ;
v = [ 1 , 2 , 3 ]
v =
  1  2  3
for x = v
 s = s + x ;
endfor
s = 6

s = 0 ;
for x = 1:2:9
 s = s + x
endfor
s = 25
```

```
s = [ 0 ; 0 ; 0 ]
s =
  0
  0
  0
a = [ 1 , 2 , 3 ; 4 , 5 , 6 ; 7 , 8 , 9 ]
a =
   1   2   3
   4   5   6
   7   8   9
for x = a        # podstawiane są
                   kolejne kolumny
 s = s + x ;
endfor
s =
   6
  15
  24
```

Instrukcje `break` i `continue`

Funkcje

```
function nazwa      # bez argumentów
                      i bez wyniku
  ciąg_instrkcji
endfunction

function NL
 printf("\n") ;
endfunction

function nazwa (lista_argumentów)
                             # bez wyniku
  ciąg_instrukcji
endfunction

function MSG (tekst)         # jeden argument
  printf("\n%s\n", tekst) ;
endfunction
```

```
function WI ( a , i )      # dwa argumenty
  x = a( i , i );
  if ( x > a( i + 1 , i + 1) )
   printf ("\nJest\n.") ;
  else
        printf("\nNie ma\n.");
  endif
endfunction
K = [ 14 , 21 ; 5 , 8 ]
WI ( K , 1 )
Jest

function wynik = nazwa ( lista_argumentów )
                       # wynik i argumenty
   ciłg_instrukcji
endfunction

# wynik - zmienna, której wartość jest
# wartością funkcji po jej zakończeniu
```

```
function s = SumaElem ( a , n )
  s = 0 ;
  for x = 1:n
    for y = 1:n
      s = s + a(x, y) ;
    endfor
  endfor
endfunction

a = [ 1 , 2 , 3 ; 4 , 5 , 6 ; 7 , 8 , 9 ] ;
h = SumaElem( a , 3 )
45


function [ lista_wyników ] = nazwa
( lista_argumentów )  # wyniki i argumenty
   ciąg_instrukcji
endfunction
```

```
function [ max_val, max_poz ] = vmax ( v )
  max_poz = 1;
  max_val = v ( max_poz );
  for i = 2:length ( v )
    if ( v( i ) > max_val )
      max_val = v ( i );
      max_poz = i;
    endif
  endfor
endfunction

a = [ 11 , 23 , -3 , 54 , -95 , 46 , -17 ,
       8 , -29 ] ;
[m , p] = vmax ( a )
54
4
```

Instrukcja return

WEJŚCIE - WYJŚCIE

1) styl MATLAB

2) styl C

Praca interakcyjna:

PAGER = less , more lub pg

fflush - wysłanie wszystkich danych z bufora na ekran

ad 1)

Wyprowadzanie automatyczne

wartość każdego wyrażenia nie zakończonego ;

```
b = 5 ^ 7 ;
print_answer_id_name =  1 ; # domyślne
a = 5 ^ 7
a = 78125
5 ^ 7
ans = 78125
print_answer_id_name =  0 ;
a = 5 ^ 7
78125
5 ^ 7
78125
```

Funkcja disp (x) : wyprowadza tekst
lub wartość wyrażenia i NL

```
disp ( Wartość pi wynosi : ), disp ( pi )
Wartość pi wynosi :
3,1416
```

Funkcja `format opcje` : ustala postać danych wyprowadzanych przez `disp`

`short` : pola 8 - znakowe, min 3 cyfry znaczące

`long` : pola 24 - znakowe, min 15 cyfr znaczących

`short e , long e` : jak `short i long` ale notacja wykładnicza z e

`short E , long E` : jak `short i long` ale notacja wykładnicza z E

`free , none` : format dowolny dopasowane kropki dziesiętne w macierzach, liczby zespolone jako (0.11 , 0.12) zamiast 0.11 + 0.12i

`bank` : 2 miejsca po kropce dziesiętnej

`+` : wyprowadza + zamiast niezerowych elementów macierzy i SP zamiast zerowych

`hex` : reprezentacja heksadecymalna

`bit` : reprezentacja binarna

```
a = [ 12.3456789 , -12.3456789 ] ;
format short
disp(a)
   12.3  -12.3
format long
disp (a)
   12.3456789000000  -12.3456789000000
format short e
disp (a)
    1.23e+01   -1.23e+01
format long e
disp(a)
1.23456789000000e+01  -1.23456789000000e+01
format short E
disp(a)
    1.23E+01   -1.23E+01
```

```
format long E
disp(a)
1.23456789000000E+01  -1.23456789000000E+01
format bank
disp(a)
   12.35  -12.35
format hex
disp(a)
  4028b0fcd324d5a2  c028b0fcd324d5a2
format bit
disp(a)

 Column 1:
0100000000101000101100001111110011010011001
0010011010110100010

 Column 2:
1100000000101000101100001111110011010011001
0010011010110100010
```

```
A = [ 12 , 0 , 5 , 0 ;
      11 , 4 , 0 , 0 ;
       4 , 0 , 0 , 1 ]  ;
format +
disp(A)
  + +
  ++
  +  +
```

Wprowadzanie danych

Funkcja `input ( zaproszenie )`

```
x = input ( "Daj liczbę .. " )
Daj liczbę .. 27
27

credo  = input ( "Powiedziałbyś coś .. " ,
                 "s" )
Powiedziałbyś coś .. Jutro.
Jutro.
```

Funkcja `menu (tytuł, opcja1, ... )`

```
kolor = "Jaki kolor?" ;
menu ( kolor , "biały" , "czerwony" ,
"niebieski" )
Jaki kolor?

   [ 1] biały
   [ 2] czerwony
   [ 3] niebieski

pick a number, any number: 3
3
```

Funkcja `kbhit( )` **: czyta jeden znak z klawiatury**

Zapisywanie i odczytywanie wartości zmiennych z plików dyskowych

Funkcja

```
save opcje plik zmienna1 zmienna2 ...
```

Opcje:

`-ascii` : format tekstowy

`-binary` : format binarny

`-mat-binary` : format binarny MATHLAB

`-save-buildins` : zmienne standardowe

Uogólnienia `?  *  [ lista ]`

```
a = 15 ;
b = 1.12574 ;
c = [ 12 , 34 ; 56 , 78 ] ;
d = 1.12 + 3.45i ;
e = "Masz wiadomość!" ;
save -ascii dane a b c d e

# Created by Octave 2.0.16, Tue Apr  2
15:22:51 2002 <jkniat@armada>
# name: a
# type: scalar
15
# name: b
# type: scalar
```

```
1.12574
# name: c
# type: matrix
# rows: 2
# columns: 2
 12 34
 56 78
# name: d
# type: complex scalar
(1.12,3.45)
# name: e
# type: string array
# elements: 1
# length: 15
Masz wiadomość!
```

Funkcja

```
load opcje plik zmienna1 zmienna2 ...

load -force -ascii dane a b c d e
                                  # kolejność
```

ad 2)

`stdin` : strumień wejściowy (0)

`stdout` : strumień wyjściowy (1)

`stderr` : strumień diagnostyczny (2)

Otwieranie plików

```
[fid , msg] = fopen ( nazwa , tryb , arch )
```

`fid` : identyfikator pliku, –1 gdy nie otwarto

`msg` : zmienna tekstowa (opis błędu)

`nazwa` : jednoznaczna ścieżka pliku

`tryb` : sposób dostępu

`arch` : sposób pamiętania liczb zmiennopozycyjnych

Tryby:

`r` : odczyt

`w` : zapis

`a` : zapis na końcu pliku

`r+` : zapis i odczyt

`w+` : zapis i odczyt,

poprzednia zawartość jest niszczona

`a+` : zapis i odczyt na końcu pliku

Architektura:

`native` : tak jak w lokalnym komputerze

`ieee-le` : little endian

`ieee-be` : big endian

```
Plik1 = fopen ( "Pomiary_03052002" , "r+" ,
                "ieee-be" )
```

Zamykanie plików

```
fclose ( fid )
fclose ( Plik1 )
```

Wprowadzanie i wyprowadzanie tekstów

```
fputs ( fid , tekst )          # do pliku
puts ( tekst )                 # do stdout
fgetl ( fid , max_długość )    # bez końc. NL
fgets ( fid , max_długość )    # z końc. NL
```

Wyprowadzanie z konwersją i formatowaniem

```
printf ( wzorzec , wyrażenie , ... )
                                    # do stdout
fprintf ( fid , wzorzec , wyrażenie , ... )
                                    # do pliku
sprintf ( wzorzec , wyrażenie , ... )
                          # tekst jest wynikiem

wzorzec:
%[szerokość][precyzja] znak_konwersji . . .
```

Znaki konwersji

znak konw.	wynik konwersji
d	liczba dziesiętna ze znakiem
i	liczba dziesiętna ze znakiem
o	liczba oktalna bez znaku
u	liczba dziesiętna bez znaku
x	liczba heks. bez znaku (a ... f)
X	liczba heks. bez znaku (A ... F)
f	liczba rzeczywista ddd.ddd

e	liczba rzeczywista d.ddde[+/-]ddd
g	konwersja f lub e,
E	jak konwersja e z użyciem E
G	jak konwersja g z użyciem E
s	ciąg znaków
c	pojedynczy znak

```
printf("Wartość zmiennej i wynosi %d", i);
printf("%12.3f\n%12.1f", x, x);
   151.378
   151.4
printf("%E", sin(1 - 1 / x) *
         cos(1 / exp(x) * x));
napis = "raz dwa trzy";
printf("%s", napis);   // wszystkie znaki
```

Wprowadzanie danych z konwersją

```
[war , licznik] = fscanf ( fid , wzorzec ,
                           rozmiar ) # z pliku
[ zmienna1, ...] = fscanf ( fid , wzorzec ,
                           "C" )     # z pliku
```

```
[ war , licznik ] = sscanf ( tekst ,
            wzorzec , rozmiar ) # z bufora
[ zmienna1, ... ] = sscanf ( tekst ,
            wzorzec , "C" )     # z bufora
[ war , licznik ] = scanf (  wzorzec ,
                       rozmiar ) # z stdin
[ zmienna1, ... ] = scanf (  wzorzec ,
                     "C" )     # z stdin
```

war : macierz, do której zostaną wstawione wyniki

rozmiar : liczba wczytywanych wartości

Inf : cały plik, wektor kolumnowy

nr : maksymalnie nr danych, wektor kolumnowy

[nr , Inf] : cały plik, macierz o nr wierszach (reszta 0)

[nr , nc] : nr * nc danych, macierz nr wierszy i nc kolumn

wzorzec:

%znak_konwersji %znak_konwersji . . .

Znaki konwersji

d	liczba całkowita dziesiętna
D	liczba całkowita dziesiętna
o, O	liczba całkowita oktalna
i, I	liczba całkowita dziesiętna oktalna lub heksadecymalna
u, U	liczba całkowita dziesiętna bez znaku
x, X	liczba całkowita heksadecymalna
e,E	liczba zmiennopozycyjna
f	liczba zmiennopozycyjna
g,G	liczba zmiennopozycyjna
s	ciąg znaków
c	znak

```
[ x , k ] = scanf ( ”%f%d” , ”C” )
```

Wykrywanie końca pliku

```
feof ( fid )  # 1 : był koniec,
                0 : jeszcze nie
```

Dalsze funkcje dla plików binarnych

`fread` : **odczyt bloku danych**

`fwrite` : **zapis bloku danych**

`ftell` : **odczyt położenia głowicy**

`fseek` : **przesunięcie głowicy**

`frewind` : **przewinięcie do początku**

FUNKCJE STANDARDOWE

Przetwarzanie macierzy

`any (wektor)` : **1 dla elementów niezerowych**

`all (wektor)` : **1 gdy wszystkie niezerowe**

`diff (wektor)` : **x(2) - x(1) ... x(n) - x(n-1)**

`isinf (wektor)` : **1 gdy INF**

`isnan (wektor)` : **1 gdy NaN**

`finite (wektor)` : **1 gdy skończony**

`find (macierz)` : wektor indeksów niezerowych elementów

`flipr (macierz)` : odwrotna kolejność kolumn

`flipud (macierz)` : odwrotna kolejność wierszy

`rot90 (macierz)` : obrót o 90 stopni w kierunku przeciwnym do zegara

`reshape (macierz, m, n)` : wybór m * n elementów z macierzy

`shift (wektor, b)` : przesunięcie cykliczne o b pozycji w prawo

`sort (wektor)` : porządek rosnący

`tril (macierz)` : dolny trójkąt (z przekątną)

`triu (macierz)` : górny trójkąt (z przekątną)

`vec (macierz)` : zamiana na wektor kolumnowy

`vech (macierz)` : zamiana na wektor wierszowy

`eye (n)` : macierz n * n, przekątna = 1, reszta 0

`one (n)` : macierz n * n, same 1

`zeros (n)` : macierz n * n, same 0

`rand (n)` : macierz n * n, wartości losowe z przedziału (0,1)

`diag (wektor, k)` : macierz z elementami wektora na przekątnej k

`linspace (baza, granica, liczba)` : wektor z podaną liczbą elementów równomiernie od bazy do granicy

`logspace (baza, granica, liczba)` : wektor z podaną liczbą elementów logarytmicznie od bazy do granicy

`hankel (c , r)` : macierz Hankla

`hilb (n)` : macierz Hilberta

`invhib (n)` : macierz odwrotna macierzy Hilberta

`sylvester_matrix (k)` : macierz Sylvestra

`toeplitz (c, r)` : macierz Toeplitza

`vander (c)` : macierz Vandermode'a

Pozostałe działy:

Arytmetyka

Algebra liniowa

Równania nieliniowe

Kwadratura

Równania różniczkowe

Optymalizacja

Statystyka

Teoria mnogości

Przetwarzanie wielomianów

Teoria sterowania

Teoria sygnałów

Przetwarzanie obrazów

Przetwarzanie dźwięku