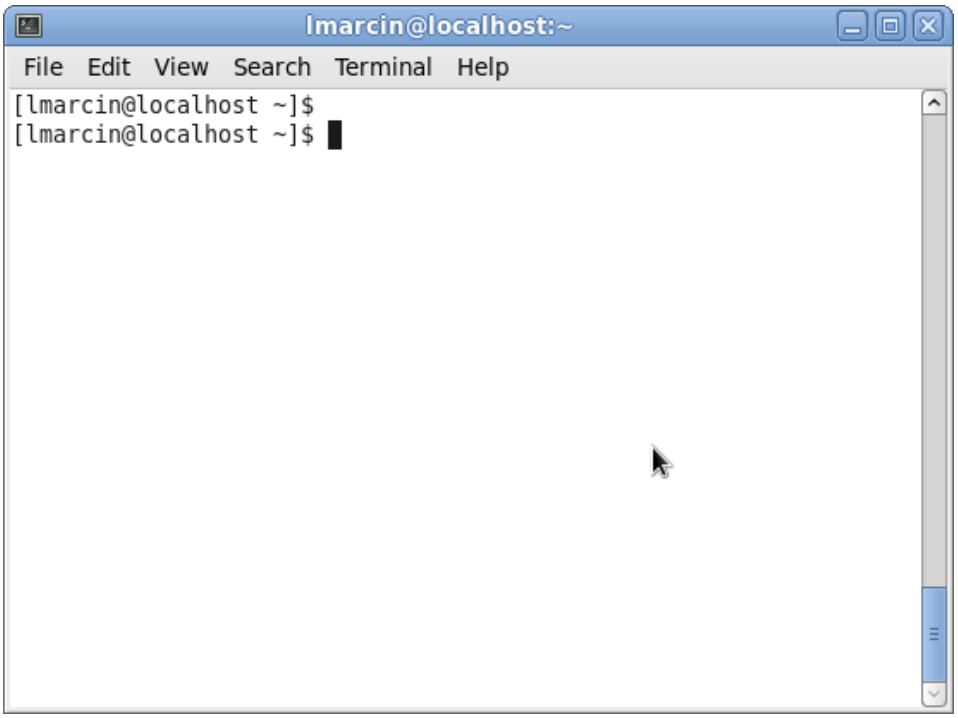
Rozdział 3

Octave - podstawy

3.1 Pierwsza sesja w octave'ie

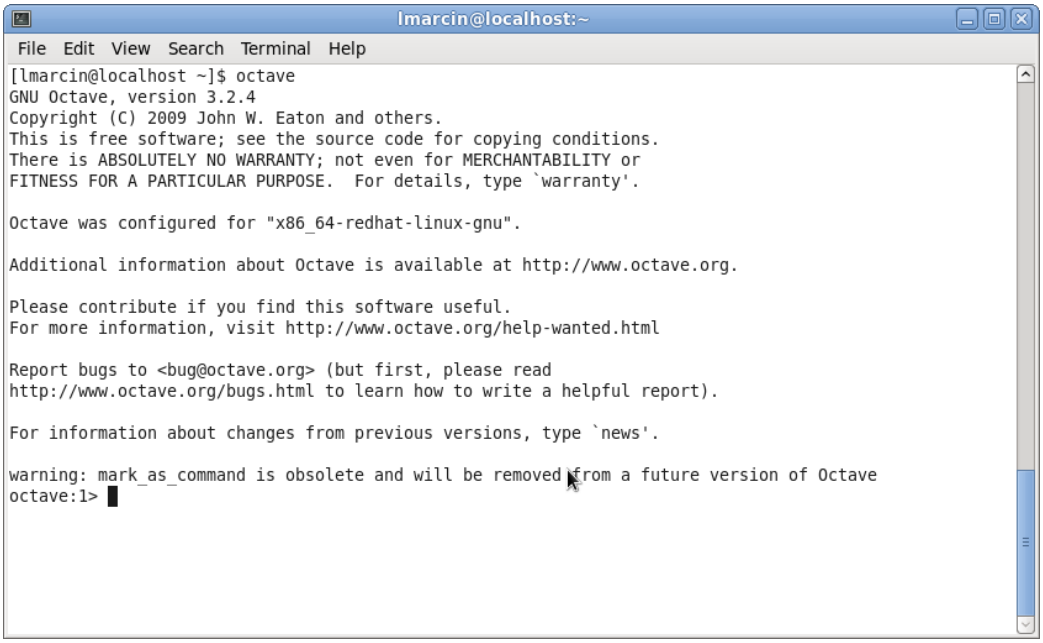
W systemie unix np. w linuxie najpierw musimy uruchomić terminal (np. xterminal). Otworzy się wówczas okienko z linią komend z *promptem* - za-



Rysunek 3.1: Terminal z linią komend.

zwyczaj migającym, w której możemy wpisywać komendy unixa, por. rysunek 3.1.

Wpisujemy komendę octave i potwierdzamy *Enter*. W okienku terminala wyświetli się nagłówek z numerem wersji i z linią komend octave'a, por. rysunek 3.2.



Rysunek 3.2: Terminal z wywołanym octavem.

Dalej, w linii komend octave'a wpisujemy polecenia octave'a.

3.1.1 Octave jako kalkulator, podstawowe zmienne

Najprostsze zastosowanie octave'a to kalkulator naukowy. W linię komend wpisujemy np.: 234+76. Otrzymujemy: **ans**=310 - zmiennej octave'a **ans** zostaje przypisana wartość 310, którą można dalej wykorzystać np. **ans**−1. Wówczas otrzymamy 309.

W octavie są funkcje takie, jak pierwiastek kwadratowy **sqrt**(), sin(), cos(), exp() itp.

Jeśli chcemy zapamiętać wynik - musimy użyć zmiennej np.:

```
> a=12+34;
> sin(a)
ans =  0.90179
```

Typów zmiennych nie deklarujemy - octave domyślnie przyjmuje typ reprezentowany przez daną zmienną.

W octavie są np. zmienne typu zespolonego:

```
> a= sqrt(-1)
a =   0 + 1i
> b=1+4*i
b =   1 + 4i
> c=a*b
c = -4 + 1i
> d=a+b
d =   1 + 5i
```

Zachęcam do samodzielnego testowania.

Proszę zauważyć, że jeśli wywołamy jakąś funkcję albo operator, który zwraca wartość, np.: 2+3 zwraca wartość 5, i nie przypiszemy tej wartości zmiennej, to octave automatycznie przypisze tę zwracaną wartość do zmiennej **ans**, którą później można wykorzystać.

Wszystkie zmienne, jakie są w pamięci sesji octave'a, możemy wyświetlić poleceniem **who**, czy **whos** - wersją poprzedniej funkcji, zwracającą dokładny opis zmiennych.

Zmienne możemy usuwać z pamięci octave'a poleceniem **clear** np.

```
> a= sqrt(-1);
> b=1+4*i
b =   1 + 4i
> who
Variables in the current scope:

A    B    C    a    ans  b    c    d    f

> clear  a
> who
Variables in the current scope:

A    B    C    ans  b    c    d    f
```

Zmienna *a* zniknęła z sesji octave'a. Podanie średnika po poleceniu spowoduje, że wynik nie zostanie wyświetlony w terminalu.

Pomoc do octave'a wywołujemy poleceniem **help**, pomoc do konkretnej funkcji wywołujemy poleceniem **help** nazwa_funkcji np. **help sin**:

```
> help sin
'sin' is a built-in function

 -- Mapping Function:  sin (X)
      Compute the sine for each element of X in radians.
```

See also: **asin**, sind, **sinh**

Pomoc jest w języku angielskim.

3.1.2 Macierze i operacje na macierzach

Podstawowym typem zmiennym są macierze.

Macierz możemy zdefiniować wprost podając jej wartości:

```
A=[1,2,3,4;2 -4 6 7];
```

- proszę zauważyć, że elementy macierzy w wierszu oddzielamy przecinkiem albo spacją, a kolejne kolumny - średnikiem.

Do elementu macierzy o współrzędnych (i, j) odwołujemy się następująco: A(i,j).

Jeśli zdefiniujemy element macierzy spoza zakresu, np. w tym przypadku

```
A(3,3)=7;
```

octave rozszerzy macierz, przyjmując domyślnie pozostałe niezdefiniowane wartości macierzy za zera.

Ważną rolę pełni operator średnik:

```
a:h:b
```

Generuje on wektor

$$[a, a+h, \ldots, a+k*h]$$

z $a + k * h \leq b$. Wywołanie:

```
a:b
```

działa jak a:1:b, czyli z h równym jeden.

Z macierzy możemy tworzyć inne macierze wycinając podmacierze. Rozpatrzymy macierz A wymiaru $M \times N$. Weźmiemy teraz dwa wektory indeksów $i = [i(1), \ldots, i(K)]$ oraz $j = [j(1), \ldots, j(L)]$ dla $1 \leq i(s) \leq M, 1 \leq j(s) \leq N$. Wtedy polecenie:

```
B=A(i,j);
```

stworzy macierz wymiaru $K \times L$ taką, że B(r,s)=A(i(r),j(s)), np.

```
> A=[3 3 3; 4 5 6]
A =

   3   3   3
   4   5   6
```

```
>   i=[ 2  1]; j=[2  3];
>   B=A(i,j)
B =

    5    6
    3    3
```

W ten sposób można wycinać też podmacierze lub np. kolumny, czy wiersze macierzy:

```
A=[3  3  3; 4  5  6];
B=A(1:2,2);
```

Warto zauważyć, że jeśli wywołamy polecenie x=A(:,3) to jest to równoważne x=A(1:2,3), czyli x staje się trzecią kolumną macierzy A.

Macierze możemy tworzyć z innych macierzy blokowo:

```
B=[A,A];
C=[A;A];
```

Jeśli A miała wymiar $M \times N$, to macierz B ma wymiar $M \times 2N$, a jej odpowiednie podmacierze złożone z pierwszych N kolumn i ostatnich N kolumn są równe macierzy A. Macierz C analogicznie ma wymiar $2M \times N$, a jej podmacierze złożone z pierwszych i ostatnich M wierszy są równe A. Przy takich definicjach musimy operować macierzami o odpowiednich wymiarach.

Popatrzmy na wyniki:

```
octave:6> A=[3  3  3; 4  5  6]
A =

    3    3    3
    4    5    6

octave:7> B=A(1:2,2)
B =

    3
    5

octave:8> B=[A,A]
B =

    3    3    3    3    3    3
    4    5    6    4    5    6
```

```
octave:9> C=[A;A]
C =

   3   3   3
   4   5   6
   3   3   3
   4   5   6

octave:10>
```

Istnieją funkcje definiujących określone macierze - wymienimy teraz podstawowe:

- **zeros**(M,N) - tworzy macierz zerową o M wierszach i N kolumnach.
- **ones**(M,N) - tworzy macierz o wszystkich wartościach równych jeden o M wierszach i N kolumnach
- **eye**(M,N) - tworzy macierz identycznościową o M wierszach i N kolumnach, tzn. o wartościach: jeden - na głównej przekątnej i zero - na pozostałych pozycjach
- **rand**(M,N) - tworzy macierz o M wierszach i N kolumnach o losowych wartościach według rozkładu jednostajnego na odcinku $[0, 1]$
- **hilb**(N) - tworzy macierz Hilberta $N \times N$
- **vander**(x) - tworzy macierz Vandermonde'a dla punktów $\{x_k\}$ z wektora $x = [x(1), \ldots, x(N)]$
- **diag**() - tworzy macierz diagonalną, ale również pozwala otrzymać odpowiednie diagonale macierzy przy odpowiednim wywołaniu

Większość funkcji elementarnych typu sin() może być wywoływana na macierzach. Wtedy: w octave'ie a=**sin**(A) zwraca macierz tego samego wymiaru co A, ale o elementach $(\sin(A(i,j))_{i,j}$. Mówimy, że implementacja takiej funkcji jest wektorowa. Popatrzmy na przykład:

```
octave:6> A=0.5*pi*[0:6;-3:3]'
A =

   0.00000  -4.71239
```

```
    1.57080   -3.14159
    3.14159   -1.57080
    4.71239    0.00000
    6.28319    1.57080
    7.85398    3.14159
    9.42478    4.71239

octave:7> a=sin(A)
a =

    0.00000    1.00000
    1.00000   -0.00000
    0.00000   -1.00000
   -1.00000    0.00000
   -0.00000    1.00000
    1.00000    0.00000
    0.00000   -1.00000

octave:8>
```

Macierze możemy dodawać i mnożyć, o ile zgadzają się wymiary:

```
octave:15> A=2+eye(2,2)
A =

    3    2
    2    3

octave:16> B=ones(2,2)+2*A
B =

    7    5
    5    7

octave:17> C=A*B
C =

    31    29
    29    31

octave:18>
```

Operacje arytmetyczne możemy też wywołać w wersji wektorowej, tzn. jeśli np. dwie macierze A, B mają ten sam wymiar, to wywołanie C=A.*B zwróci macierz C tego samego wymiaru taką, że $C(k,l) = A(k,l) * B(k,l)$. To samo dotyczy innych działań:

- dzielenia C=A.\B
- odejmowania C=A.−B
- dodawania C=A.+B
- podnoszenia do potęgi C=A.^B

W przypadku dodawania i odejmowania operatory bez kropek działają tak samo.

Macierz rzeczywistą możemy transponować operatorem C=A'. W przypadku macierzy zespolonej operator ten zwróci sprzężenie hermitowskie, a operator z kropką A.' zwróci zwykłe transponowanie. Najlepiej zobaczyć to na przykładzie:

```
octave:37> A=[1+i  -3+4i;-8+2i  5+7i]
A =

   1 + 1i  -3 + 4i
  -8 + 2i   5 + 7i

octave:38> A'
ans =

   1 - 1i  -8 - 2i
  -3 - 4i   5 - 7i

octave:39> A.'
ans =

   1 + 1i  -8 + 2i
  -3 + 4i   5 + 7i

octave:40>
```

Istnieje wiele funkcji pozwalających na manipulowanie macierzami np.:

- **fliplr** () - odwraca kolejność kolumn
- **flipud**() - odwraca kolejność wierszy

- **reshape**() - tworzy nową macierz o określonych wymiarach z elementów macierzy wyjściowej
- **rot90**() - obraca macierz o wielokrotność 90 stopni w kierunku przeciwnym obrotowi wskazówek zegara

Wektory traktowane są jako macierze o jednej kolumnie, albo jednym wierszu, jakkolwiek istnieją specjalne funkcje działające tylko na wektorach, np. funkcja **length**(x) zwraca długość wektora x pionowego lub poziomego.

Polecenie [m,n]=**size**(A) zwraca wymiar macierzy jako dwie wartości w wektorze.

```
octave:40> x=[1 2 3 4]
x =

   1   2   3   4

octave:41> length(x)
ans = 4
octave:42> size(x)
ans =

   1   4

octave:43>
```

3.2 Octave jako język programowania - skryptu i m-pliki

Octave może być również traktowany jako język programowania.

Istnieje możliwość pisania skryptów, a następnie ich wywoływania z linii komend octave'a.

Skrypty to po prostu pliki tekstowe o rozszerzeniu *.m*, tzn. np. skryptem może być plik o nazwie *nazwaskryptu.m*, zawierające w kolejnych liniach komendy octave'a utworzone przez dowolny edytor tekstowy, np. vi w unixie, czy wordpad w systemie windows.

Skrypt wywołujemy podając nazwę bez rozszerzenia, czyli skrypt zawarty w pliku tekstowym *skrypt.m* wywołujemy z linii komend komendą:

```
skrypt
```

Po wywołaniu np. następującego skryptu:

```
#skrypt.m
#Prosty skrypt - ta linia to komentarz
 a=2+3
```

octave wykona wszystkie polecenia po kolei - o ile nie będzie w nich jakiegoś błędu. W tym przypadku zwróci a=5. Tak więc octave jest interpreterem - nie ma etapu kompilacji skryptu przed wykonaniem.

Pisząc skrypty w octavie najlepiej mieć na pulpicie komputera co najmniej dwa okna - jedno z sesją octave'a, a drugie z edytorem, w którym edytujemy plik tekstowy ze skryptem.

Tu warto dodać, że komentarze w octavie wstawiamy po znakach # lub %.

Octave jako język programowania dostarcza również większość konstrukcji imperatywnych, m.in.

- instrukcja warunkowa **if** - przykład:

```
#instrukcja warunkowa
if(x>0)
 x=1;
else
 x=0;
endif;
```

- instrukcja **switch** wielokrotnego wyboru zastępująca **if** jeśli k przyjmuje określoną ilość wartości

```
switch (taknie)
    case {"T" "t"}
          x = 1;
     case {"N" "n" }
          x = 0;
      otherwise
          error ("tylko t lub n");
endswitch
```

- pętla **while**( )

```
#sumowanie
k=0;
a=0;
while(k<=10)
do
```

```
  a+=k;
  k++;
endwhile;!
```

- pętla z warunkiem zatrzymania na końcu: **do** intrukcje **until**( );

```
k=0;
do
  k++;
until(k>10)
```

- pętla **for**:

```
a=0;
for k=1:10,
  a+=k;
endfor
```

- funkcje octave'a **function** []=f()

```
function [s]=f(a,b)
  s=a+b;
endfunction
```

  Więcej informacji o funkcjach podamy w następnym rozdziale.

W tym miejscu omówimy operacje logiczne octave'a. Ogólnie ujmując - wartość różna od zera traktowana jest jako prawda, 0 to fałsz. Operatory logiczne octave'a są takie same jak w języku C tzn:

- a && b - operator logiczny dwuargumentowy $a\ I\ b$ zwraca wartość jeden (prawda), jak a i b są różne od zera, w przeciwnym przypadku operator zwraca zero (fałsz)

- a || b - operator logiczny dwuargumentowy $a\ LUB\ b$ zwraca jeden, jeśli któreś z a lub b jest równe jeden, w przeciwnym przypadku zwraca zero

- !a - operator logiczny jednoargumentowy negacji $NIEPRAWDA\ a$, jeśli a jest różne od zera - to operator zwraca zero, w przeciwnym przypadku operator zwraca jeden

- a==b - operator dwuargumentowy równości, jeśli a ma tę samą wartość co b, to operator zwraca jeden, w przeciwnym razie operator zwraca zero

- a!=b - operator dwuargumentowy bycia różnym, jeśli a jest różne od b to zwraca jeden, w przeciwnym razie operator zwraca zero
- a < b - operator dwuargumentowy porównania, zwraca jeden gdy a jest mniejsze od b, w przeciwnym razie operator zwraca zero. Pozostałe operatory porównania (a > b, a<=b, a>= b) są zdefiniowane analogicznie

3.2.1 Funkcje

Funkcje w octave'ie możemy definiować z linii komend, czy jako część kodu w skrypcie:

```
function [w,s]=f(a,b)
#pomoc do danej funkcji
  s=a+b;
  w=a-b;
endfunction
```

Wywołujemy je z linii komend (lub w linii skryptu, czy w innej funkcji)

```
w=f(1,2)
[w,s]=f(1,2)
```

Pierwsze wywołanie spowoduje, że funkcja zwróci tylko pierwszą wartość tj. w.

Niektóre ostatnie parametry funkcji mogą przyjmować domyślną wartość tzn.

```
function [w,s]=f(a,b=0)
#pomoc do danej funkcji
  s=a+b;
  w=a-b;
endfunction
```

Wtedy wywołanie funkcji

```
[w,s]=f(1)
```

zwróci wartości funkcji dla parametru $b = 0$.

Funkcje można przekazywać jako parametry do innych funkcji poprzez przekazanie nazwy funkcji jako napisu - ciągu znaków lub jako tzw. uchwyty do funkcji - tu przykład przekazania nazwy funkcji *sin*() do funkcji **quad**() obliczającej całki:

```
> f=@sin;
> quad("sin",0,1)
```

```
ans =  0.45970
> quad(f,0,1)
ans =  0.45970
> quad(@sin,0,1)
ans =  0.45970
```

Oba sposoby są równoważne.

Proste funkcje można definiować jako tzw. funkcje anonimowe:

```
> f=@(x) + sin(x); #f(x)=x+sin(x)
> f(1)
ans =  0.84147
```

czyli definiujemy od razu uchwyt do funkcji.

W octavie wprowadzono za matlabem tzw. m-pliki, czy - inaczej - pliki funkcyjne. Są to pliki z rozszerzeniem *m* np. *f.m*, zawierające definicje funkcji o tej samej nazwie co plik (bez rozszerzenia). M-plik o nazwie *test.m* powinien więc zawierać definicję funkcji test(), czyli w pierwszej linii powinien znajdować się nagłówek funkcji.

Poniżej widzimy ogólną strukturę funkcji:

```
function [zwracane wartosci]=test(parametry)
#
#pomoc do funkcji
#
    komendy

    zwracane wartosci powinny byc zdefiniowane

endfunction
```

Wywołanie polecenia **help** test powinno wyświetlić pomoc do funkcji tzn. wykomentowane linie spod nagłówka funkcji.

Funkcje można też definiować w skryptach, które mają te same rozszerzenie *m*.

Ważne, aby w pierwszej niepustej linii skryptu nie było słowa kluczowego **function**, tzn. aby przed definiowaniem funkcji w skrycie podać jakąkolwiek komendę. W przeciwnym przypadku octave potraktuje skrypt jako m-plik.

W m-plikach, pod definicją głównej funkcji o tej samej nazwie co m-plik, mogą się znajdować tzw. pod-funkcje (ang. subfunctions), które mogą być wywołane przez tę główną funkcję z m-pliku i inne pod-funkcje tylko w tym m-pliku. Tutaj podajemy przykład takiego m-pliku z funkcją główną obliczającą przybliżenie normy typu L^2 i pod-funkcjami:

```
#m-plik czyli plik tekstowy normaL2.m

function [nL2]=normaL2(FCN,a,b)
#norma L^2(a,b) z funkcji o uchwycie FCN
# na [a,b]
    sf=uchwytkwad(FCN);
    nL2=sqrt(quad(sf,a,b));
endfunction

function [SFCN]=uchwytkwad(f)
#tworzy uchwyt do funkcji g(x)=f(x)*f(x)
 SFCN=@(x) f(x)*f(x);
endfunction

#koniec m-pliku normaL2.m
```

Oczywiście pod-funkcja jest tutaj wprowadzona sztucznie, nie jest ona konieczna.

W funkcji octave'a można używać automatycznych zmiennych **nargin** i **nargout**, które przy wywołaniu danej funkcji przyjmują wartości:

- **nargin** - ilości parametrów, z jaką funkcja została wywołana.
- **nargout** - ilości zwracanych wartości, z jaką funkcja została wywołana.

Tu podajemy przykład wykorzystania tych zmiennych w konkretnej funkcji:

```
function [y1,y2]=test(a,b)
# test nargin nargout
  if(nargin<2)
   b=a;
  endif
  y1=a+b;
  if(nargout>1)
     y2=a-b;
  endif
endfunction
```

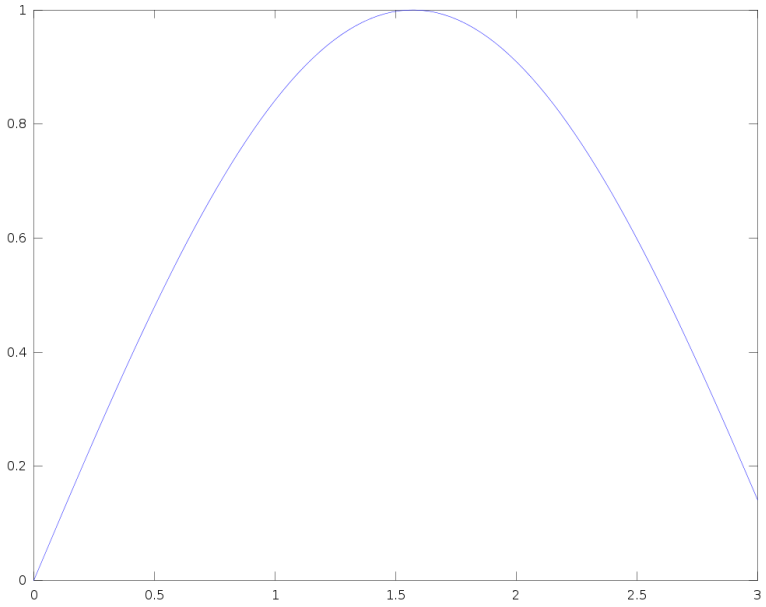
Sprawdźmy, jak taka funkcja działa:

```
octave:46> test(1)
ans =  2
octave:47> test(1,2)
```

```
ans =  3
octave:48> [y1]=test(1,2)
y1 =  3
octave:49> [y1,y2]=test(1,2)
y1 =  3
y2 = -1
octave:50>
```

3.3 Podstawowa grafika

Octave posługuje się zazwyczaj zewnętrznym narzędziem do grafiki w linuxie. Najczęściej jest to program gnuplot, ewentualnie grace lub inny. W przypadku systemu windows w binarnej wersji octave'a zawarta jest też grafika. Podstawową funkcją umożliwiającą rysowanie wykresów jest **plot**().



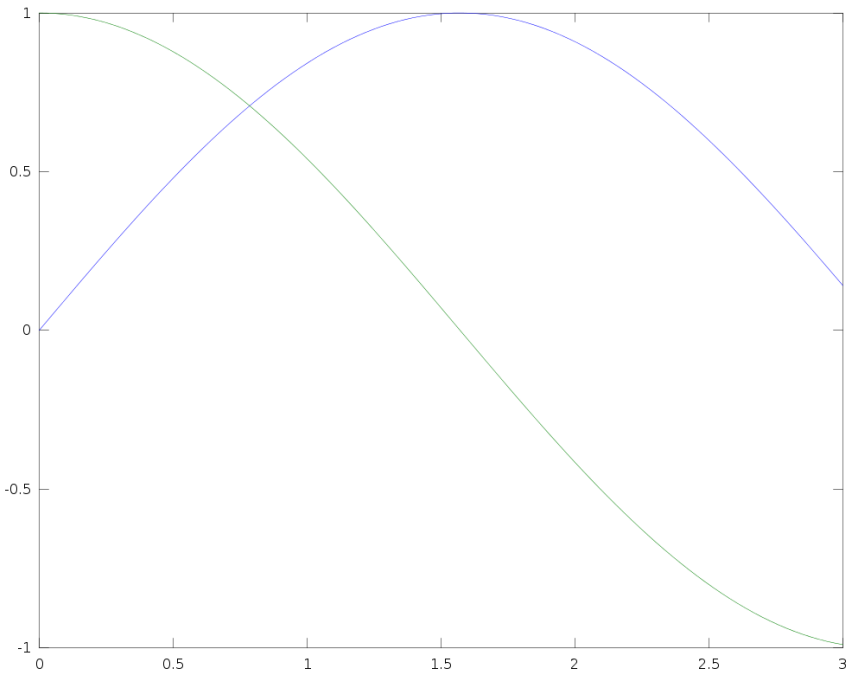
Rysunek 3.3: Możliwie prosty wykres funkcji

Najprostsze wywołanie: **plot**(x) dla wektora x otworzy nowe okno i w nim narysuje punkty $(k, x(k))$. Punkty mogą pozostać połączone prostymi w zależności od ustawienia domyślnych opcji octave'a.

Innym wywołaniem **plot**() pozwalającym np. na rysowanie wykresów jest **plot**(x,y) dla wektorów x, y tej samej długości. W tym przypadku octave narysuje punkty $(x(k), y(k))$.

Tak więc, jeśli stworzymy w wektorze x siatkę stupunktową równomierną na $[0, 3]$ - wygodnym poleceniem jest **linspace**(,a,b,N) - a potem policzymy wartości np. funkcji $y(k) = \sin(x(k))$, to polecenie **plot**(x,y) narysuje wykres sin() na tym odcinku, por. rysunek 3.3.

```
x=linspace(0,3);
y=sin(x); #sin jest wektorowa funkcja
plot(x,y)
```

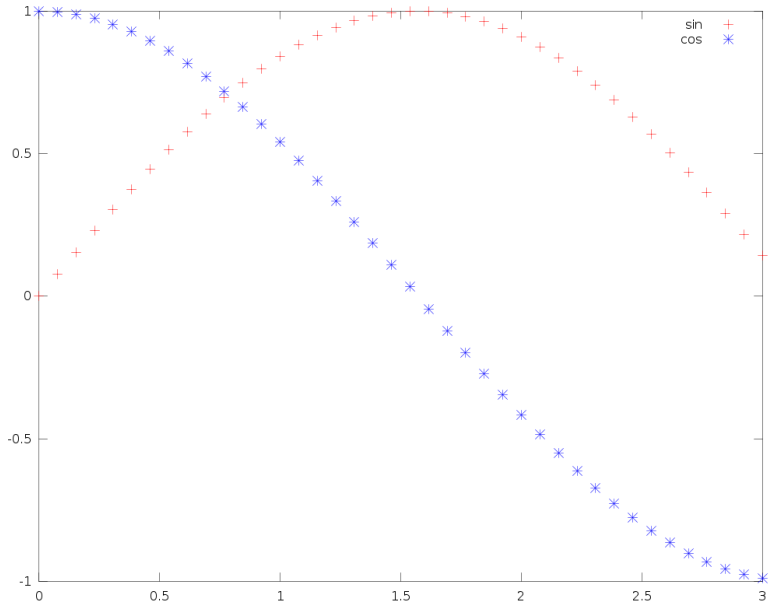


Rysunek 3.4: Dwa wykresy

Istnieje możliwość umieszczania kilku wykresów na jednym obrazku np.

```
x=linspace(0,3);
y=sin(x); #sin jest wektorowa funkcja
z=cos(x);
plot(x,y,x,z)
```

Tu **plot**(x,y,x,z) narysuje punkty $(x(k), y(k)$ i $(x(k), z(k)$, czyli wykresy sin() i cos() na jednym rysunku.



Rysunek 3.5: Dwa wykresy w różnych stylach i podpisane

Inną możliwością jest zastosowanie odpowiednich funkcji dotyczących grafiki; np. **hold** on powoduje, że kolejne wywołania **plot** sprawia rysowanie wykresów na jednym obrazku. Polecenie **hold** off likwiduje tę własność.

Możemy dobrać kolor wykresu, styl wykresu, dodać opis, np. wykres sinusa może być w kolorze czerwonym narysowany plusami, a cosinusa w kolorze niebieskim narysowany gwiazdkami

```
x=linspace(0,3,40);
y=sin(x); #sin jest wektorowa funkcja
z=cos(x);
plot(x,y,"+r;sin;",x,z,"*b;cos;")
#+ - wykres plusami r - red (czerwony)
#* wykres gwiazdkami, b -blue (niebieski)
```

Podamy teraz kilka przydatnych funkcji związanych z grafiką:

- **xlabel** - etykieta osi poziomej
- **ylabel** - etykieta osi pionowej

- **title** - tytuł rysunku
- **figure** - kontroluje okienka z wykresami - możemy ich mieć kilka, czyli

```
x=linspace(0,3,40);
figure(1);
y=sin(x);
plot(x,y)
figure(2)
z=cos(x);
plot(x,z)
```

  stworzy dwa okienka z wykresem sin() w pierwszym i cos() w drugim.
- **semilogx** - wykres w skali półlogarytmicznej - względem zmiennej poziomej
- **semilogy** - wykres w skali półlogarytmicznej - względem zmiennej pionowej
- **loglog** - wykres w skali logarytmicznej
- **bar** - wykres słupkowy.
- **stairs** - wykres schodkowy.

Zachęcam do zapoznania się z tymi funkcjami używając pomocy octave'a.

Istnieje też możliwość używania prostej grafiki trójwymiarowej, ale w przypadku matematyki obliczeniowej nie będziemy tego potrzebowali.

3.4 Przykładowy skrypt

Poniżej zamieszczamy prosty skrypt, w którym przedstawiamy przykłady użycia większości operacji, zmiennych omawianych w tym rozdziale.

```
echo on
#Ostatnia modyfikacje 28-06-2011
##################################################
#
#   Prosty skrypt octave'a mobasic.m z przykladami
#   operacji na macierzach
#   wektorach z podstawowymi
#   strukturami programistycznymi
#
##################################################



##################################################
#
#                       WAZNE
#
#  skrypt w octavie zawiera  dowolna
#  sekwencje komend octave'a
#  tak jakby pisanych z linii komend ale
#  NIE MOZE ZACZYNAC SIE OD SLOWA KLUCZOWEGO
#   function
#
#  (gdyz, wowczas jest to tzw. plik funkcyjny
#   czy m-plik, i pelni role funkcji )
#
##################################################

#Podstawowe obiekty - macierze
#generujemy macierze w roznej postaci
#wprost ; - oddziela wiersze
#spacja albo , kolumny
A=[1,2;4,5;6 7]

#Mozna odwolywac sie do elementu
A(1,2)

#Mozna definiowac macierz inaczej
B=[1 2
4 5
6 7]
```

```
#mozna obliczac wspolrzedne za pomoca wzoru
x=[1+2, sqrt(2), 15/7]

#mozna policzyc funkcje od wektora
sin(x)
#warto dowiedzic sie jak wywolac help
# do danej funkcji np sin
#help sin
#Jak nie znamy nazwy funkcji mozemy sprobowac
#help -i sin


#mozna zmienic element macierzy
A(1,1)=-3

#czy rozszerzyc zakres
x(6)=4.56

A(1,7)=10

#mozna obejrzec jakie obiekty sa w pamieci
who
#ze szczegolami
whos
#obejrzec obiekt B
B
#usunac obiekt
clear B
#usunac wszystko
clear
who

#mozna wygenerowac wektor (macierz)
#przy pomocy operatora a:h:b
#(generuje wektor o elementach
#od a do b z krokiem h tzn.
# x(i)=a +(i-1)*h
#dla   i=1,2,... takie ze a+i*h <=b)
#tu domyslny krok h=1
x=1:5
y=0:0.1:1
```

```
A=[1:4 ; 0 5:7; -3:0]

#Mozna popelnic blad w
#definiowaniu macierzy -
#wykomentowano bo jesli wystapi blad, to wykonanie
#skryptu przez octave'a skryptu konczy sie.
#B=[1:3 ; 1:2]

#mozna wykrawac pod-macierze
A1=A(1:2 , 2:3)
#Jesli chcemy pod-macierz np.
#z dwoma pierwszymi wierszami
A2=A(1:2 ,:)
#Mozna z macierzy budowac macierz
#dodajemy kolumne do A2
[A2 , [1;2]]
#dodajemy wiersz do A2
[A2 ; 1:4 ]
#albo jeszcze inaczej
[A2 ; A2]
[A2,A2]
[A2 ,A2;A2,A2]

#Mozna z macierzy zbudowac
#nowa macierz w innym formacie
#(ale o tych samych elementach)
[m,n]=size(A2)
# dostaniemy wymiary macierzy A2
#Budujemy wektor o elementach z macierzy A2
#- kolumnowo tzn
#w wektorze beda kolejne kolumny z A2
reshape(A2,m*n,1)
#i z powrotem - tu ans wynik ostatniej operacji
reshape(ans,m,n)

#Inne funkcje zmieniajace porzadek w macierzy to
#np.:
#fliplr() odwraca kolejnosc kolumn,
#flipud() odwraca kolejnosc wierszy,
#rot90() obraca macierz o wielokrotnosci 90 stopni
B=[1 2; 3 4]
```

```
fliplr(B)
flipud(B)
rot90(B)


#Jak widac mamy sporo mozliwosci.
#Mozna mnozyc macierze przez siebie
#o ile wymiary sie zgadzaja:
#np. macierz przez wektor
x=1:4

#Trzeba miec wektor pionowy
#(macierz jednokolumnowa)
#wiec transponujemy uzywajac "'":
x=x'
f=A*x

#Mozna mnozyc macierz przez macierz
#(o ile wymiary odpowiednie):
B= [ 1 2 ; 3 4]
C= [ 5 6 ; 1 1 ]
B*C

#Czy kazdy element pierwszej
#macierzy przez odpowiedni
#element drugiej:
B.*C

#Czy macierz przez skalar:
2*A
#Mozna transponowac macierz kwadratowa:
B=A'

#Mozna policzyc norme euklidesowa
#tzn. druga wektora:
norm(x,2)
#Czy pierwsza albo supremum:
norm(x,1)
norm(x,'inf')
#Czasami nie warto wypisywac wszystkiego
#na ekranie - srednik ';' po komendzie
```

```
#to zapewni:
y=A*x;
norm(y,2)


#mozna wygenerowac identycznosc
I=eye(10)
#czy slynne macierze
H=hilb(10); #macierz Hilberta



#mozna przeniesc dlugie wyrazenia
y=[1,2,3, ...
5 6 7]


#I mozna zapisac niektore obiekty w pliku.
#Najpierw zobaczmy co mamy w pamieci
#sesji

who

#Mozna zapisac w pliku binarnym wszystkie
#zmienne
#(nie mozna obejrzec danych poza octavem)
save -binary dane.bin

clear

who

load dane.bin
# Czy pojawil sie znowu y? Sprawdzamy:
who

#Zapisujemy konkretna macierz w pliku tekstowym
#o danej nazwie dane.txt:
save -ascii dane.txt y

#Mozna teraz obejrzec ten plik
```

```
#w jakims edytorze
#lub przy pomocy programu  more
#z linii komend: np.: more dane.dat

#Czyscimy zmienne z pamieci:
clear
#Sprawdzamy teraz zmienne w pamieci:
who
#Wczytujemy zmienne z pliku:
load dane.txt

#Powinnien pojawic sie wektor y ale pod jaka nazwa?
who

#Octave wczytal macierz pod nazwa 'dane',
#ale mozna zmienic nazwe np.:
y=dane
clear dane

#W octavie mozna uzywac podstawowych obiektow
#takich jak  petle:
#while(warunek)
#
#  komendy
#
#endwhile
#Przyklad uzycia:
k=0;
while (k<3)
k++; #k=k+1;
endwhile
k
#Kolejna petla to:
#do
#
#   komendy
#
#until( warunek )

#Petla ta zawsze przynajmniej
#raz sie wykona.
```

```
#Przyklad:
k=0;
do
k++;
until(k>3);
k
#Trzecia petla to:
#for i= wektor wartosci
#
#  komendy
#
#endfor
#Przyklad:
a=0;
for k=1:4,
 a+=k; #a=a+k;
endfor
a
#Istnieja tez inne podstawowe
#struktury programistyczne jak:
#Intrukcja warunkowe
#if
#
# komendy
#
#else (opcja)
#
# komendy
#
#endif
#Przyklad:
k=input("podaj wartosc k=?\n");
#teraz czeka na input
if(k<10)
 disp("k<10"); #disp wyswietla string na ekranie
else
  disp("k >=10");
endif

#Tu warto poznac operatory logiczne octave'a
#ogolnie wartosc rozna od zera jest uznawana za prawde
```

```
#zero za nieprawde
if(2)
 a=3
else
 a=1;
endif
#&& - AND logiczne "i"
i=0;
(i<=10)&&(i>-10)
(i<10)&&(i>1)
(i<-1)&&(i>1)
#||- OR logiczne "lub"
(i<10)||(i>-10)
(i<10)||(i>1)
(i<-1)||(i>1)
#! - negacja (opcjonalnie ~)
!(i>0)
!(i<=0)
#operator rownosci - nie-bycia rownym jak w C
# = ta sam wartosc; != rozna wartosc
(5==4)
(5==(3+2))
(5!=6)
(5!=(3+2))


#Kolejna struktura warta poznania
#to instrukcja switch (troche inna niz w C/C++)
#switch(zmienna)
# case {wartosci zmiennej}
#   komendy
#
# case {wartosci zmiennej}
#   komendy
#
#otherwise
#
#   komendy
#
#endswitch
#Przyklad:
```

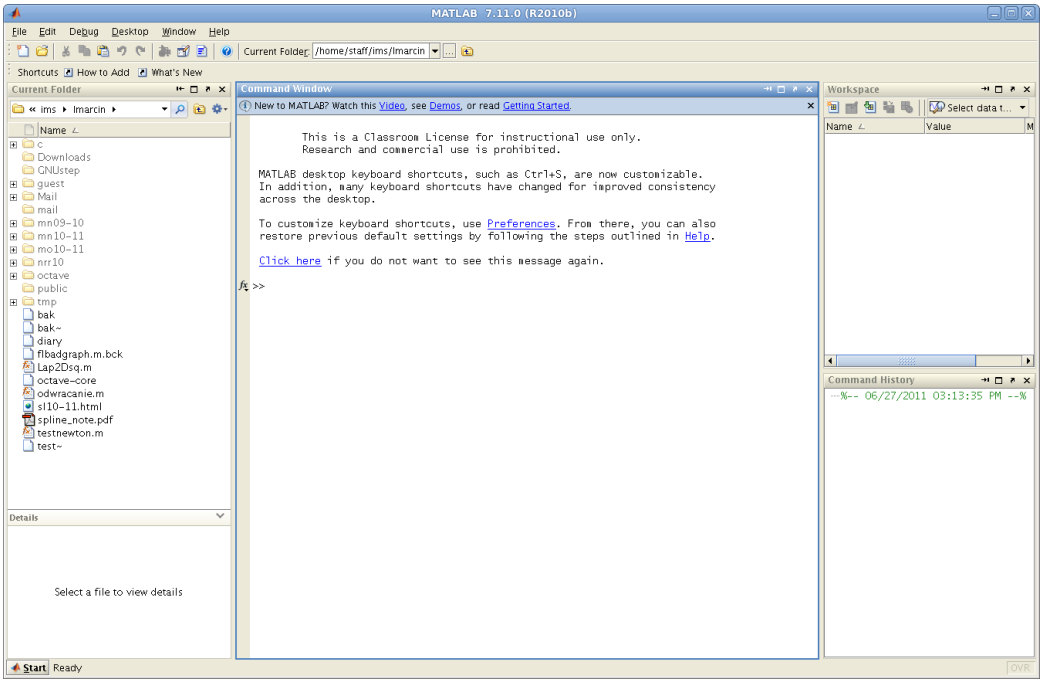
```
taknie=menu("test switch","yes","no","do not know");
# menu wygodna funkcja  gdy oczekujemy wybrania opcji
switch(taknie)
 case {1}
  disp("yes");
 case {2}
  disp("no");
 case {3}
  disp("don't know");
endswitch
#inna instrukcja na wprowadzanie danych z klawiatury to
#kbhit

#funkcje
#najprostsza
function y=f(x)
  y=x+sin(x);
endfunction
#wywolujemy
f(0)
f(1)
#funkcja anonimowa
f=@(x) 1+x+sin(x)
#Formalnie f tu jest
#uchwytem do funkcji ale
#wywolanie jest takie same:
#Wywolujemy:
f(0)

#prosta grafika
#siatka rownomierna na [0,3]
x=linspace(0,3);
#wektorowo wartosci sin na siatce
y=sin(x);
#i cos
z=cos(x);
#najprostszy wykres
plot(y);
pause(1);
#pause - aby nie zniknelo od razu
```

```
#Plot rysuje wykres
#wartosci y wzgledem indeksow
#Lepiej narysowac wykres y
#wzgledem punktow siatki:
plot(x,y);
pause(1);

#Mozemy zamiescic
#dwa wykresy na jednym rysunku
plot(x,y,x,z);
pause(2);

#Mozemy wykres podpisac
plot(x,y,";sin;",x,z,";cos;");
pause(2);

#I mozemy ustalic kolor wykresu:
plot(x,y,"k;sin;",x,z,"g;cos;");
pause(2);
#Oczywiscie kolory sa w jezyku angielskim
#r - red; b - blue; k - black; g - green
#m - magenta; c - cyan; w - white

#Czy zmienic styl wykresy -
#zaznaczymy punkt przeciecia przez czerwony plus:
zx=pi/4;zy=cos(pi/4);
plot(x,y,"k;sin;",x,z,"g;cos;",zx,zy,"r+;pkt przec.;");
pause(2);

#i to tyle podstawowych rzeczy ktore trzeba
#wiedziec o macierzach i octavie
#(i ktore przyszly mi do glowy)
echo off
```

3.5 Przykładowa sesja matlaba

W tym rozdziale przedstawimy przykładową sesję matlaba.

W przeciwieństwie do octave'a, matlab posiada środowisko graficzne. Ogólnie matlaba wywołujemy klikając na odpowiednią ikonę, czy wybierając program matlab w odpowiednim menu systemu. W linuxie możemy też



Rysunek 3.6: Terminal z wywołanym środowiskiem matlaba.

wywołać ten program z linii komend komendą: *matlab*. Wówczas powinno otworzyć się środowisko matlaba, por. rysunek 3.6.

Jak widać, w środku mieści się *Command Window*, czyli okno komend, w którym możemy wpisywać komendy matlaba, z lewej strony widać podgląd bieżącego katalogu pod nazwą *Current Folder*, z prawej strony, z góry pod nazwą *Workspace*, widać okno ze spisem zdefiniowanych zmiennych oraz z dołu, z lewej strony - małe okno historii komend, por. rysunek 3.7.

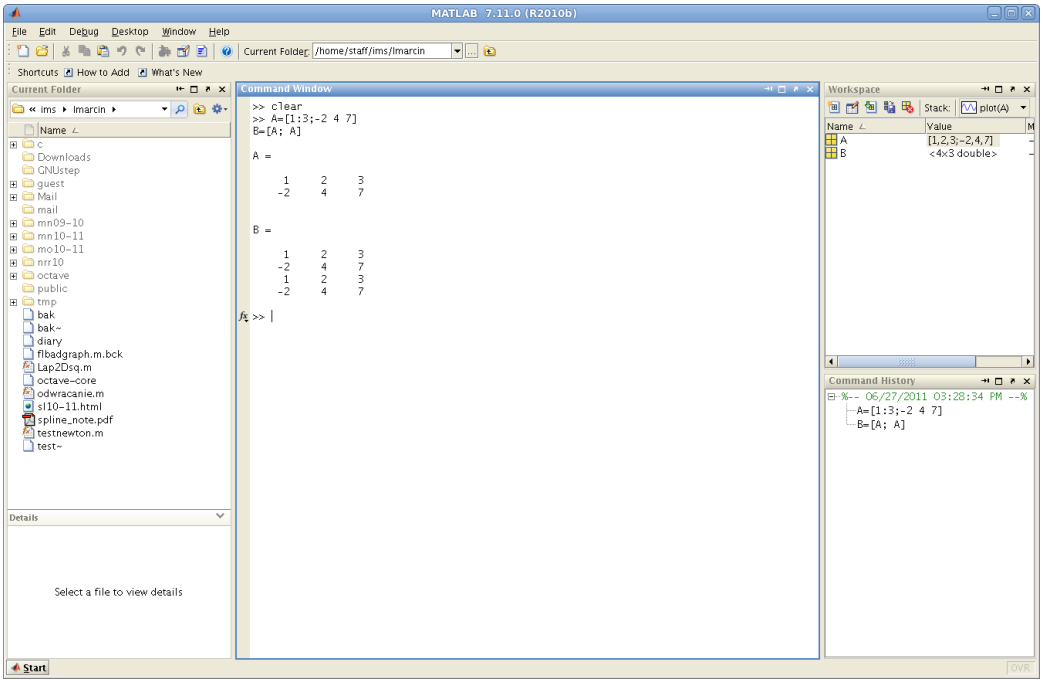
Ustawienia można zmieniać, np. można część okien ukryć, czy odłączyć. Po szczegóły odsyłam do pomocy do matlaba.

Wpiszmy kilka komend takich samych jak w octave'ie w okno komend:

```
A=[1:3;-2  4  7]
B=[A;  A]
```

W oknie komend widać, że wynik jest taki sam jak w octave'ie. Natomiast w oknie *Workspace* widzimy, że w sesji są zdefiniowane dwie zmienne A,B, a w oknie historii widzimy dwie nasze komendy, por rysunek 3.7.

W środowisku matlaba najważniejsze jest okno komend, które pełni tę samą rolę co terminal z wywołanym octavem. Pozostałe okna pełnią rolę pomocniczą ale niewątpliwie środowisko matlaba ułatwia pracę. Dalej nie będziemy się zajmować szczegółowo matlabem, ale zdecydowana większość



Rysunek 3.7: Terminal z wywołanym środowiskiem matlaba po wykonaniu komendy.

kodów z tego skryptu działa zarówno w octave'ie, jak i w matlabie.