Wprowadzenie do MATLAB

Jacek Dajda

15 czerwiec 2001

Praca egzaminacyjna z przedmiotu Metody Obliczeniowe w Nauce i Technice
Informatyka II rok
Źródła dostępne pod adresem:
http://student.uci.agh.edu.pl/ jdajda/mownit/referat

Spis treści

1 Wstęp

1.1 Czym jest MATLAB

MATLAB firmy The MathWorks to jednocześnie interakcyjne środowisko i wysokiego poziomu język programowania przeznaczony przede wszystkim dla obliczeń naukowych i inżynierskich. Pozwala na wykonywanie skomplikowanych obliczeń numerycznych oraz wizualizację wyników. Nazwa pakietu jest skrótem od ,,MATrix LABoratory", czyli ,,Laboratorium Macierzowe". Już sama nazwa wskazuje, że użytkownik operuje tylko na jednym typie danych - macierzach (rzeczywistych bądź zespolonych). Dlatego też nawet pojedyncze liczby reprezentowane są w formie jednowymiarowej macierzy kwadratowej. Do ważniejszych cech Matlaba należy dodać możliwość pracy w trybie interakcyjnym. Wyniki obliczeń są dostępne natychmiast i można je przedstawić w postaci wykresów 2 lub 3 wymiarowych, albo map wielobarwnych.

1.2 Zastosowanie i możliwości

Zakres zastosowań pakietu obejmuje różne dziedziny nauki i techniki, można tu wymienić choćby meteorologię, elektronikę, telekomunikację, biologię, medycynę, czy ekonomię i automatykę. Jedną z wielu zalet pakietu jest dostęp do najnowszych algorytmów numerycznych algebry liniowej, analizy matematycznej i numerycznej, całkowania numerycznego i rachunku macierzowego. Atrakcyjność pakietu podnosi graficzny interfejs dający szerokie możliwość pracy interakcyjnej za pomocą okienek edycyjnych, przycisków, suwaków i menu. Dzięki temu możliwe jest tworzenie własnych wygodnych dla użytkownika aplikacji, pozwalających na uzyskiwanie przejrzystych wyników. Do innych zalet należy zaliczyć wręcz nieograniczoną rozszerzalność pakietu bądź to poprzez importowanie własnych aplikacji napisanych w języku C lub Fortran, definiowanie własnych poleceń i algorytmów obliczeniowych czy też tworzenie wyspecjalizowanych pakietów (tzw. toolboxy) z procedurami i funkcjami specyficznymi dla danej dziedziny nauki.

2 Podstawy programowania

2.1 Informacje wstępne

Zanim przystąpimy do pracy warto zapoznać się z kilkoma użytecznymi cechami MATLABA.

- macierze indeksowane są od 1 np. x(1)
- zmienne inicjalizowane są automatycznie, w trakcie pierwszego przypisania im wartości
- stałe tekstowe podaje się w apostrofach np. 'to jest tekst'
- istnieje zmienna predefiniowana **ans**, która przyjmuje wynik ostatniej operacji, dla której nie określono zmiennej wynikowej
- operatorem przypisania jest ,,=" np. x=2
- listę zdefiniowanych przez użytkownika zmiennych dostarcza komenda **who**
- komenda **clear** ***nazwa zmiennej*** spowoduje usunięcie zmiennej o podanej nazwie, jeśli nie podamy nazwy konkretnej zmiennej usunięte zostaną wszystkie zdefiniowane do tej pory zmienne
- jeśli utworzymy zmienną o nazwie identycznej z nazwa funkcji (np. sin), to funkcja ta przestaje być dostępna do czasu ponownego uruchomienia programu
- nazwy zmiennych składają się z dowolnej litery, po której następuje dowolna ilość liter, cyfr i podkreśleń, przy czym rozróżnianych jest tylko pierwsze 19 znaków
- rozróżniane są duże i małe litery

2.2 Tworzenie macierzy

Jak już wspomniano, macierz jest jedyną strukturą danych. Aby utworzyć najprostszą macierz wystarczy wymienić jej elementy w nawiasach kwadratowych. np. A=[5 1 .2 ; 7.3 10/3 2] oznacza macierz o wymiarach 2x3. Oznaczmy tę macierz zmienną A:

```
>>  A=[5 1 .2 ; 7.3 10/3 2]}
```

Na ekranie zobaczymy wówczas:

```
A =
    5.0000    1.0000    0.2000
    7.3000    3.3333    2.0000
```

Przy większych macierzach niewskazane jest, by kompilator drukował ich zawartość na ekran. Aby tego uniknąć należy postawić znak średnika na końcu linijki z definicją zmiennej. Aby otrzymać później zawartość zmiennej wystarczy podać jej nazwę. Ponadto do zdefiniowanych zmiennych mamy zawsze dostęp dzięki menu Workspace Browser, gdzie możemy dowolnie edytować ich zawartość.
Macierz można także zdefiniować w inny sposób. Do szybkiego generowania macierzy mogą posłużyć następujące funkcje:

zeros(w,k	tworzy macierz o w wierszach i k kolumnach, wypełnioną jedynkami
ones(w,k)	tworzy macierz o w wierszach i k kolumnach, wypełnioną zerami
rand(w,k)	tworzy macierz o w wierszach i k kolumnach, wypełnioną liczbami losowymi z przedziału [0,1)
eye(n)	tworzy n-wymiarową macierz jednostkową

Istnieje jeszcze jeden ciekawy sposób tworzenia wektorów. Na przykład zapis t=0:5 spowoduje, że w miejsce zmiennej t otrzymamy 6 elementowy wektor o wartościach od 0 do 5. Chcąc ustawić swój własny krok należy wstawić dodatkową wartość pomiędzy ustalone granice. Przykładowo zapis t=10:-3:1 wygeneruje wektor: 10 7 4 1.
Elementy macierzy możemy odczytać stosując zapis: A(numer wiersza, numer kolumny) np. A(2,2)=3.333. Jesli podamy tylko jeden parametr np. A(5) to otrzymamy 0.2, gdyż licząc według wierszy 0.2 jest właśnie 5 elementem, w tym przypadku wektora, A. Jeśli zaś spróbujemy podstawienia w miejsce wykraczające poza wymiary macierzy, MATLAB automatycznie dynamicznie przekształci ją na macierz o odpowiednich wymiarach, inicjalizując na 0 nowo utworzone i niezdefiniowane elementy np.

```
>> A(3,1)=100
A =

    5.0000    1.0000    0.2000
    7.3000    3.3333    2.0000
  100.0000         0         0
```

2.3 Proste operacje na macierzach

Mając macierz A z poprzedniego punktu spróbujmy wykorzystać ją do kilku prostych operacji. Utwórzmy na przykład macierz transponowaną:

```
>> A'
ans =

    5.0000    7.3000  100.0000
    1.0000    3.3333         0
    0.2000    2.0000         0
```

Bazując na macierzy A można utworzyc nową macierz:
» B=A(2:3,1) - takie polecenie spowoduje, że powstanie macierz 2x1 utworzona z elementów macierzy A znajdujących się w pierwszej kolumnnie i 2 oraz 3 wierszu czyli

```
B =
    7.3000
  100.0000
```

Poniżej przedstawię kilka przykładów ilustrujących kilka ważniejszych operacji na macierzach:

- A*[2 3 ; 1 2 ; 100 11] - operacja mnożenia macierzy. W przypadku błednych wymiarów program informuje użytkownika o błędzie (analogicznie przebiega dodawanie czy odejmowanie.
- A(:,2) - zwraca całą drugą kolumnę macierzy A, analogicznie A(2,:) zwraca cały drugi wiersz macierzy, a A(:,:) całą macierz
- A^n - potęgowanie macierzy, wynik jest analogiczny do zapisu $\underbrace{A * A \cdots * A}_{n}$
- A.^n - potęgowanie elementów macierzy - każdy element podnosimy do n-potęgi. Generalnie stosując operator „.” (kropki) możemy wykonywać operację na elementach macierzy np. (A.*3, A./2,. . .)
- B=sin(A*2*pi*/32) - powstanie macierz B o takich samych wymiarach jak macierz A, a elementy macierzy B będą wyliczone na podstawie zapisanego wzoru oraz odpowiadającym im elementom macierzy A.
- B=[ones(3,4) ; eye(4) ; rand(1,4)] - powstanie macierz o wymiarach 8x4, zbudowana z kolejnych wierszy podanych macierzy.
- [3 2 ; 1 2].*[2 3 ; 6 3] - da w wyniku macierz 2x2 złożoną z samych szóstek. Zastosowano tu operator kropki, którego znaczenie omówiono już wyżej
- sin([0 pi/2 pi 3/2*pi 2*pi]) da w wyniku wektor [0, 1, 0,-1,0], gdyż większość standardowych funkcji MATLABA (cos, atan, sqrt, exp, . . .) operuje na elementach macierzy
- V2(4:-1:2)=V1(1:3) - elementy od 4 do 2 wektora V2 będą pochodziły od elementów od 1 do 3 wektora V1.

Ciekawym zagadnieniem jest operacja dzielenia macierzy. Mając równanie macierzowe A*X=B macierz X wyliczymy w inny sposób niż w przypadku układu X*A=B. Stąd twórcy MATLABA wprowadzili dwa operatory dzielenia - lewostronny(\ ) i prawostronny (/). Ich działanie jest następujące:

X=A\B jest równoważne zapisowi: X=A'*B
X=A/B jest równoważne zapisowi: X=B*A'

3 Skrypty i funkcje

3.1 Czym są skrypty i funkcje

Skrypty i funkcje są niczym innym jak pewną sekwencją komend MATLAB'a, zapisaną do pliku. Pliki, w których zapisuje się skrypty i funkcje są zwykłymi plikami tekstowymi o rozszerzeniu „m”. Aby MATLAB mógł wykonać/odnaleźć dany plik, katalog, w którym się on znajduje musi znajdować się na liście ścieżek MATLABA. Nową ścieżkę dodaję się w menu Path Browser za pomocą opcji Add Path.

3.2 Skrypty

Skrypty są prostym narzędziem ułatwiajającym pracę z pakietem i oszczędzającym czas. Zamiast wpisywać poszczególne komendy w linii poleceń MATLABA lepiej jest wpisywać je do pliku. Aby wykonać zapisaną sekwencję wystarczy podać nazwę pliku (bez rozszerzenia), w którym zapisaliśmy swoje komendy, ewentualnie można użyć opcji Run Script z okna dialogowego. Dzięki temu po zakończeniu pracy nie utracimy zapisanej sekwencji komend.
W porównaniu do funkcji skrypty nie są zbyt sprawnym narzędziem programowania. Nie można przekazywać do nich parametrów. Są one przydatne w przypadkach, gdy wykonujemy proste i szybkie obliczenia, zajmujące niewiele kodu lub też gdy wykonywane obliczenia nie zależa od żadnych parametrów.

3.3 Funkcje

Nazwa nowo tworzonej funkcji powinna być zgodna z nazwą pliku, w której się znajduje. Jest to wynikiem tego, iż po wykryciu nazwy funkcji pakiet rozpoczyna przeszukiwanie po nazwach dostępnej listy plików. Następnie (po utworzeniu pliku i dodaniu go do listy ścieżek) w pierwszej linii pliku umieszcza się słowo kluczowe **function**, a za nim listę zmiennych wyjściowych, nazwę funkcji i w nawiasach listę zmiennych wejściowych. Oprócz napisania funkcji zawsze powinniśmy pamietąc o jej udokumentowaniu, stąd w następnych linijkach rozpoczętych od znaku komentarza ,,%" podaje się jej opis. Jest on wyświetlany na ekranie w wyniku podania komendy *help nazwa funkcji*. Dla przykładu napiszmy funkcję iloczyn, która pobiera dwa parametry i zwraca ich iloczyn.

```
function z=iloczyn(x,y)
% z = x*y
% mnozy liczby, wektory, macierze, mozliwe przypadki mieszane
z=x*y;
```

Jeżeli funkcja nie zwraca żadnej zmiennej lub nie pobiera żadnych parametrów to po prostu pomijamy tę część specyfikacji funkcji np.

```
function pierwsze
% wywolanie: pierwsze
% funkcja zwraca wszystkie liczby pierwsze mniejsze lub rowne od podanej wartosci
zakres=input('Podaj zakres: ');
liczby_pierwsze=primes(zakres)
```

Jeśli funkcja zwraca więcej niż jedną zmienną listę zmiennych podaje się w formie wektora np.

```
function [x,y]=pierwsze(n)
% wywolanie: [tablicaliczbpierwszych,tablicakwadratowliczpierwszych]=pierwsze(n)
% funkcja zwraca nastepuja zmienne:
% x - wszystkie liczby pierwsze mniejsze lub rowne od podanej wartosci
% y - lista kwadratow liczb pierwszych z listy x
x=primes(n);
y=x.^2;
```

Jeśli przy wywolaniu funkcji podamy tylko jedną zmienną np.

```
>> A = pierwsze(n);
```

to pod zmienną A otrzymamy wektor liczb pierwszych (x). Wektor kwadratów zostanie utracony. Ogólna zasada jest więc taka, że podstawianie zaczyna się od lewej strony i odbywa dopóki jest to możliwe.
Na zakończenie warto dodać, że przekazywanie zmiennych do funkcji odbywa się przez wartość. Wszystkie utworzone zmienne w danej funkcji są zmiennymi lokalnymi. Jeśli zależy nam na utworzeniu zmiennej globalnej należy przez nazwą zmiennych umieścić słowo kluczowe **global**

4 Instrukcje sterujące. Bardziej zaawansowane funkcje

4.1 Operatory

W instrukcjach warunkowych niezbędne jest stosowanie operatorów logicznych i porównania, które zostały przedstawione poniżej:

Operatory logiczne

Operator	Skrót	Opis
and	&	Logiczne **i**
not	~	Logiczne **nie**
or	—	Logiczne **lub**
xor	brak	Logiczne **xor**
any	brak	Wieloargumentowe **or**
all	brak	Wieloargumentowy **and**

Operatory porównania

Operator	Skrót	Opis
eq	==	czy równe
ne	=	czy różne
lt	¡	mniejsze niż
gt	¿	większe niż
le	¡=	mniejsze lub równe
ge	¿=	większe lub równe

Wartościowanie wyrażeń odbywa się identycznie jak ma to miejsce np. w języku C, mianowicie: wartość logiczna **false** reprezentowana jest przez 0, zaś **true** przez pozostałe wartości. Wszystkie podane operatory (z wyjątkiem **any** i **all**) są dwuargumentowe. Jeśli jako argument podamy macierze A i B (muszą być takich samych wymiarów) wynikiem będzie macierz C (załóżmy), której element c_{ij} będzie wynikiem operatora na elementach a_{ij} i b_{ij}. Poniżej zaprezentowano kilka przykładów:

```
>> 1 & -1
ans =
     1
```

```
>> xor([-2 1 ]  [1 0])
ans =
      0 1
```

```
>> [ 2 3 ; 1 1 ] > [ 0 1 ; 3 4]
ans =
       1   1
       0   0
```

Operatory **all** i **any** są jednoargumentowe. Kwestia szacowania wyniku jest identyczna jak wyżej, z tym wyjątkiem, że w tym przypadku operacje logiczne odbywają się nie na liczbach lecz na wektorach (w przypadku podania macierzy jako wektory traktowane są kolumny), jakkolwiek jeżeli podamy liczbę to oba operatory zadziałają poprawnie. Poniżej przedstawiono kilka przykładów:

```
>> all(0)
ans =
     0
```

```
>> any([-2 0 0 0])
ans =
     1
```

```
>> all([ 1 -3 2 3  ; 1 0 3 0 ; 1 2 3 4])
   ans =
          1   0   1   0
```

4.2 Instrukcje warunkowe - if/elseif/else

Najprostsza forma instrukcji **if** ma postać:

if *warunek wyrażenia* **end**

Na przykład:

```
>> A = [2 4 6 8];
>> n = 2;
>> if n ~= 0
     A = A/n;
   end
```

Po tych operacjach macierz A ma postać:

```
A =
     1     2     3     4
```

Jeśli chcemy wyspecyfikować alternatywne bloki czynności używamy słowa kluczowego **elseif** lub **else** jeśli jest to ostatnia alternatywa. Ich użycie zaprezentowane zostanie w następnym punkcie.

4.3 Bloki iteracyjne - instrukcje for/while

Składnia instrukcji **for** jest następująca:

for i= początek: krok: koniec *wyrażenia* **end**

Krok nie jest obowiązkowy. Domyślna wartość to 1.
Składnia instrunkcji **while** ma postać:

while *warunek wyrażenia* **end**

Przykład - napiszmy funkcję, która otrzymuje wektor i zwraca liczbę elementów ujemnych, równych zero i dodatnich.

```
function [u,z,d]=my_sgn(wektor)
%% wszystko wydaje sie jasne
n=length(wektor);u=0; z=0; d=0;
    for i=1:n
    if wektor(i)<0 u=u+1;
    elseif wektor(i)==0 z=z+1;
    else d=d+1;
    end

end
```

```
function [u,z,d]=my_sgn(wektor)
%% wszystko wydaje sie jasne
i=1;u=0; z=0; d=0;
         while i<=length(wektor)
         if wektor(i)<0 u=u+1;
         elseif wektor(i)==0 z=z+1;
         else d=d+1;
         i=i+1;
         end
end
```

Naturalnie możliwe są także bloki zagnieżdżone np.

```
function H=hilb(n)
% macierz Hilberta
H=ones(n,n);
i=1;
while i<=n
   for j=1:n
   H(i,j)=H(i,j)/(i+j-1);
   end
   i=i+1;
end
```

```
function H = hilb(n)
% to samo ale krocej
J = 1:n;
J = J(ones(n,1),:);
I = J';
E = ones(n,n);
H = E./(I+J-1);
end
```

4.4 Instrukcjach switch

Ponieważ składnia instrukcji jest prawie identyczna jak ta znana z języka C ograniczę się do prostego przykądu:

```
switch mod(liczba,10)
    case {0,1,2,3},disp('Przypadek 1')
    case {4,5,6,7},disp('Przypadek 2')
    otherwise ,disp('Przypadek 3')
end
```

4.5 Wyjątki w MATLABIE - instrukcja try-catch

W MATLABIE zaimplementowano znany z języków obiektowych mechanizm wychwytywania wyjątków. Jest on niezwykle użyteczny w miejscach, gdzie np. podanie złego typu parametru spowoduje błąd. Dzieki temu mamy pewność, że błąd nie spowoduje jakichś nieprzewidzianych dla nas wypadków i użytkownik zostanie poinformowany o błedzie (jeśli zatroszczymy się o odpowiedni komunikat w bloku **catch**). Składnia instrukcji jest następujaca:

try, *i*nstr1, ..., instrN ,**catch**, *i*nstr1, ..., instrN,**end**

5 Generowanie wykresów

MATLAB pozwala na szybkie generowanie różnych rodzajów wykresów. Oto najważniejsze z nich:

5.1 Wykresy 2D

5.1.1 Funkcja plot

plot(Y) - generuje wykres, w którym oś OY opisana jest przez wartości wektora Y, natomiast oś OX przez indeksy elementów wektora Y np. plot([1 2 3]) utworzy wykres funkcji y=x.

plot(X,Y) - generuje wykres, w którym oś OY opisana jest przez wartości wektora Y, natomiast oś OX przez wartości wektora X np. plot([1 2 3],[2 4 6]) narysuje wykres funkcji y=2x

plot(X,Y,S) - zachowuje się jak wyżej z tym, że S jest stringiem opisującym atrybuty (w następujacej kolejności: kolor, styl znaczników, styl linii) wykresu. Listę wszystkich możliwych atrybutów można znaleźć w pomocy pakietu.

- Przykład: plot([1 2 3],[3 6 9],'r*:'] - kolor czerwony, punkty oznaczone symbolem gwiazdki, linia kropkowana
- Przykład: plot([2 4 6],'bo–') - kolor niebieski, symbol punktów to okrąg, linia przerywana.

plot(X_1**,**Y_1**,**S_1**,...,**X_N**,**Y_N**,**S_N**)** - to samo jak wyżej z tym, że pozwala na rysowanie wielu wykresów na raz np. plot([1 2 3 4 5],[1 2 3 4 5],'b+-',[1 2 3 4 5],[2 4 6 8 10],'bd–')

5.1.2 Funkcja stem

stem(Y) lub stem(X,Y) lub stem(X,Y,S) - zachowanie podobne jak dla funkcji plot, z tą różnicą, że funkcja stem nie łączy kolejnych punktów liniami. Jej zastosowanie to funkcje dyskretne.

stem(. . . ,'filled') - znaczniki reprezentujące węzły są wypełnione

5.1.3 Funkcja fplot

fplot(FUN,LIM) - generuje wykres funkcji określonej w formie stringu (FUN) w obszarze zadanym parametrem LIM np. fplot('sin(x)',[0,2*pi]) - narysuje cały okres funkcji sinus

fplot(FUN,LIM,TOL) - jak powyżej z tą rórnicą, że można zdefiniować bład maksymalny TOL. Domyślny bład maksymalny ma wartość 2e-3. Przykład: fplot('sin(x)',[0,2*pi],0.5) - otrzymamy bardzo niedokładny wykres funkcji sinus

[X,Y]=fplot(FUN,LIM,. . .) - wartości rzędnych i odciętych punktów wykresu wpisane zostaną do wektorów X i Y. Żaden wykres nie powstanie.

5.1.4 Funkcje pomocnicze

Warto także wymienić kilka funkcji, które odpowiadają za wizualną stronę wykresu:

axis(a,b,c,d) - pozwala na ustawienie zakresów osi - oś OX w przedziale [a,b] natomiast OY w przedziale [c,d]

gtext(string) - pozwala na umieszczenie na wykresie dowolnego ciagu znaków - określenie pozycji odbywa sie za pomoca myszki

grid on/off - pokazuje/chowa siatkę wykresu

xlabel(string), ylabel(string) - naniesienie na bieżące okno graficzne opisu osi OX lub OY.

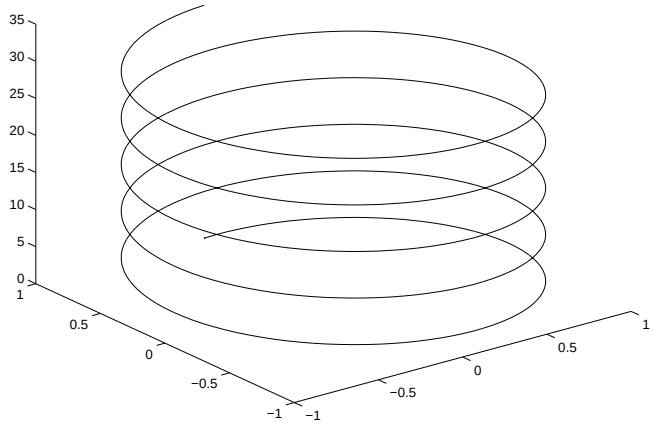
5.2 Wykresy 3D

5.2.1 Funkcja plot3

Właściwie jest to przeniesienie funkcji **plot** z płaszczyzny do układu przestrzennego. Mamy więc:

plot3(X,Y,Z) - X,Y,Z są wektorami o takiej samej długości n, dzięki czemu otrzymujemy n trójwymiarowych punktów np. (poniższy przykład rysuje helisę)

```
>> t=0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t)
```



plot3(X,Y,Z,S) - podobnie jak dla funkcji **plot** S jest maksymalnie trójelementowym stringiem, który określa wygląd wykresu

plot3(X_1,Y_1,Z_1,S_1,X_2,Y_2,Z_2,S_2,**...)** - analogicznie jak dla dwóch wymiarów, możemy otrzymąc wiele wykresów na jednym rysunku

5.2.2 Funkcja surf

Funkcja ta umożliwia rysowanie kolorowych, cieniowanych powierzchni

surf(A) - gdzie A jest macierzą wartości, natomiast osie OX i OY określone są przez wymiary macierzy np. surf([2 3 ; 2 3]) - rysuje równię pochyłą.

surf(x,y,C) - to samo z tym, że teraz sami dobieramy wartości współrzędnej x i y.

surf(x,y,A,C) - macierz C jest macierzą kolorów dla każdego punktu powierzchni, jeśli jej nie podamy to domyślnie przyjmuje się macierz A.

5.2.3 Funkcje specjalne: sphere/cylinder

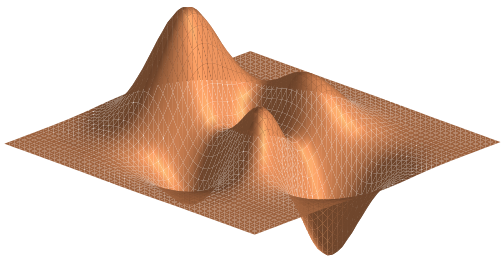
sphere(n) - rysuje sferę (a właściwie kulę) w oparciu o funkcję **surf**, przy czym n (domyślna wartość 20) jest wymiarem macierzy C.

cylinder(n) - zachowuje się podobnie jak funkcja **sphere** z tą różnicą, że wynikiem jej działania jest walec (cylinder)

peaks(arg1,arg2) - (argumenty opcjonalne) bardzo ciekawa funkcja, niezwykle przydatna dla pokazania możliwości funkcji **surfl** (wersja funkcji **surf** z efektami świetlnymi), bazuje na funkcji rozkładu Gaussa. Poniżej przedstawiono przykład jej użycia:

```
>> z=peaks(51);
>> surfl(z)
>> shading interp
>> colormap(copper)
>> axis off
```

co daje w wyniku:



5.2.4 Ciekawsze funkcje interfejsu

rotate3d on/off - umożliwia dynamiczne obracanie wykresem za pomocą myszki

colormap(mapa kolorów) - dobór mapy kolorów, znacznie ułatwia orientację w przebiegu powierzchni

camlight(left/right/headlight) - pozwala na zdefiniowanie kierunku padania światła

6 MATLAB a metody numeryczne

6.1 Funkcje fzero,quad,quad8,fmins,eigen,lu

Jak już we wstępie wspomniano MATLAB jest często wykorzystywany do obliczeń inżynierskich. Jego bogata biblioteka funkcji numerycznych (a także wyspejalizowane pod kątem konkretnych wymagań pakiety) sprawiają, że jest on niezwykle przydatnym narzędziem dla dla takiej dziedziny wiedzy jaką są Metody Numeryczne.
Aby tę użyteczność zaprezentować postanowiłem wybrać kilka funkcji numerycznych i pokazać ich dziaąnie zaprezentować.

fzero(F,T)) - pozwala na znajdywanie miejsc zerowych funkcji F przy czym parametr F może być punktem pocza"tkowym poszukiwań bądź pewnym przedziałem poszukiwań np.

```
>> fzero('sin(x)+log(x)', 6 )
ans =
     0.5787
```

```
>> fzero('sin(x)+log(x)', [0.1 100] )
>> ans =
          0.5787
```

I jeszcze przykład, w którym funkcja sobie nie radzi:

```
>> fzero('sin(x)+log(x)', 9 )
Complex function value encountered during
   search for an interval containing a sign change.
Function value at -2.52 is 0.34193+3.1416i Aborting since no such
interval was found. Check function or try again with a different
starting value.

ans =

   NaN
```

quad('F',A,B,TOL) - całkowanie niskiego stopnia funkcji F, na przedziale [A,B] z dokładnością TOL. Istnieje także funkcja calkująca algorytmem adaptacyjnym (sama dobierająca krok) **quad8** o takich samych parametrach. Przykład:

```
>> Q=quad('exp',0,7,2)
Q =
   1.0989e+003
```

```
>> Q=quad8('exp',0,7,2)
Q =
    1.0956e+003
```

fmins('F',X0) - poszukuje miejsc zerowych metodą Simplex funkcji wielu zmiennych. Jako X0 należy podać punkt początkowy poszukiwań. Dla przykładu weźmy funkcję Schwefela, która ma wiele lokalnych minimów na przedziale [(-500, 500) ; (-500, 500)]:

```
>> fmins('837.9658+x(1)*sin(sqrt(abs(x(1))))+x(2)*sin(sqrt(abs(x(2))))',
[-400 -400])
ans =
    -420.9687 -420.9688

>> fmins('837.9658+x(1)*sin(sqrt(abs(x(1))))+x(2)*sin(sqrt(abs(x(2))))',
[-200 -200])
ans =
    302.5249  302.5249
```

eigen(A) - znajdywanie wartości i wektorów własnych macierzy A np.

```
>> A= [ 1 2 3; 1 1 1 ; 2 3 4];
>> [V,D]=eig(A)
V =
   -0.5459    0.7529   -0.4082
```

```
   -0.2529   -0.6500    0.8165
   -0.7988    0.1029   -0.4082

D =
    6.3166         0         0
         0   -0.3166         0
         0         0    0.0000
```

D to macierz wartościami własymi natomiast kolumny macierzy V to wektory własne odpowiadajaące odpowiednim wartościom własnym.

lu(A) - znajdywanie rozkładu LU macierzy A.

```
>> A= [ 1 2 3; 1 1 1 ; 2 3 4];
>> [L,U]=lu(A)

L =

    0.5000   -1.0000    1.0000
    0.5000    1.0000         0
    1.0000         0         0


U =

    2.0000    3.0000    4.0000
         0   -0.5000   -1.0000
         0         0         0
```

7 Zakończenie

7.1 Zawartość

Jak wspomniano we wstępie, MATLAB zawiera wiele wyspecjalizowanych pakietów tzw. toolboxów. Na zakończenie warto wspomnieć o najciekawszych należą: **The Neural Network Toolbox** (pakiet dotyczący sieci neuronowych, oferuje specyfikację funkcji aktywacji, reguł uczenia i funkcji uczących), **The Signal Processing Toolbox** (pozwala na przetwarzanie sygnałów, projektowanie i analiza filtrów cyfrowych, estymacja widma analiza FFT), **The Symbolic Math Toolbox** (niezwykle przydatny pakiet, umożliwia dokonywanie operacji na symbolach, wykonuje większość funkcji z algebry liniowej takich jak wyznaczanie macierzy Jacobiego, macierzy odwrotnej, rozkładu LU czy wektorów własnych), **The Optimization Toolbox** (szeroki pakiet optymalizacyjny funkcji liniowych i nieliniowych), **SIMULINK** (rozbudowany pakiet symulacyjny np. modele myśliwca F14, miejskiej populacji, termodynamiki mieszkania,...)

7.2 Dostępność

MATLAB dostępny jest dla użytkowników zarówno systemu Microsoft Windows jak i sympatyków środowiska UNIX. Skorzystać można z niego także za pomocą Internetu dzięki odpowiednim serwerom, znajdującym się na terenach, przeważnie, placówek naukowych. Jednym z takich miejsc jest ACK Cyfronet w Krakowie.