MATLAB

Matlab (**Mat**rix **Lab**oratory) to interakcyjne środowisko do obliczeń naukowo-inżynierskich oraz wizualizacji danych. Matlab dostępny jest na większość platform sprzętowych i systemowych (wartstwa GUI wykonana jest w Javie).
Historia Matlaba rozpoczęła się od zbiorów procedur do obróbki macierzy (biblioteki fortranowskie LINPACK i EISPACK) połączone wspólnym interaktywnym programem dzieła Cleve'a Molera pozwalającym na wykorzystanie procedur bez znajomości Fortranu. Komercyjną wersję tego programu wydała firma The MathWork, Inc. (C. Moler, S. Bangert, J. Little) pod nazwą Matlab.
Niezwykle istotną cechą Matlaba jest jego otwarta strukturę – producent oferuje dodatkowe pakiety, dodatkowe funkcje i procedury można również pisać samemu (zarówno w samym języku Matlab, jak i w C).

Uwaga! Plik ze spakowanymi procedurami i funkcjami omówionymi poniżej można pobrać ze strony:
http://www.phys.uni.torun.pl/~jacek/dydaktyka/matlab.zip

Część I.
Polecenia podstawowe

1. Pomoc
   a) help polecenie - wywołuje opis znajdujący się w nagłówku pliku polecenie.m, np. **help cos**
   b) **help general** – ogólne informacje o poleceniach Matlaba, **help punct** – informacje o znakach spec.
   c) W GUI graficznym można korzystać polecenia (dłużej się czeka): **helpwin cos**
   d) Ikona „?" na pasku narzędzi (pliki pomocy html) równoznczna z helpwin, indeks, szukanie
   e) lookfor cosine – przeszukiwanie pomocy tekstowej
   f) polecenie **demo** – sporo ciekawych prezentacji
   g) **ver** – lista wersji poszczególnych elementów Matlaba

2. Elementy środowiska Matlab
   a) LaunchPad
   b) Workspace (zob. również **who**, **whos**)
   c) Command History (zob. również strzałki ↑↓ , polecenie **diary** *nazwa_pliku* zapisuje historię do pliku)
   d) Current Directory
   e) Editor/Debugger (zob. polecenia **edit** *nazwa_pliku*, **type** *nazwa_pliku*)

3. Wybrane polecenia podstawowe języka Matlab
   a) Wpisanie nazwy zmiennej powoduje wyświetlenie jej wartości. np. **i=5, i** (nadpisana została stała i) Program rozróżnia wielkość liter.
   b) Disp np. **disp('Hello world!'), disp(['sample rate fs=',num2str(fs),' Hz'])**
   c) ! wykouje polecenie systemowe (DOS/Windows lub Linux/Unix) np. **!dir**
   d) Dodanie ; za poleceniem powoduje nie wyświetlenie wyników, np. **i=5;**
   e) Pętla – for np. **for i=1:10, disp(num2str(i)), end** (jeżeli każda część w osobnej linii – czeka na end)
   f) Operator przypisania =, i równości == są jak w C/C++, np. **i=9**, **i==9** (wartość logiczna)
   g) Instrukcja warunkowa – **if i==10 disp('10'), else disp('nie 10'), end**
   h) Instrukcja wyboru **switch**
   i) **clear i, clear** – usuwanie wybranej zmiennej lub wszystkich z pamięci
   j) pokaz tworzenia zbioru Mandelbrota

4. Predefiniowane stałe i zmienne Matlaba
   a) ans – zmienne, do której zapisywany jest nieskojarzony output (np. **1+1** jest równoznaczne z **ans=1+1**)
   b) pi – stała 3.1416
   c) i, j – stała urojona (możliwy jest zapis **z=1+i*2**)
   d) zmienne określane przez środowisko: date, clock, pwd, computer

5. Wybrane funkcje Matlaba:
   a) operatory działań algebraicznych: +, -, *, /, ^ (potęga)
   b) inne funkcje elementarne: abs, sign, rem (reszta z dzielenia), exp, log (log. nat.), log10, sqrt (= ^0.5)
   c) zaokrąglenia: round (do najbliższej), floor (do podłogi), ceil (do sufitu), fix (w kierunku zera)
   d) funkcje trygonometryczne: sin, asin, sinh, asinh, analogicznie dla cos, tan, cot.
      Uwaga! Różnica między atan i atan2 (funkcje rodem z C).

e) operacje na liczbach zespolonych: **real(z)**, **imag(z)**, **conj(z)**, **norm(z)** (=abs)
oraz wszystkie powyższe np. **cos(z)**

6. Wektory, macierze, tensory
   a) **:**, np. **w=1:10** (ogólna postać pocz:koniec lub pocz:krok:koniec)
   b) [ ], np. **m=[1 2 3; 4 5, 6]**
   c) Ponadto można tworzyć macierze poleceniami eye, ones, zeros, rand.
   d) **rank(m)** - ilosc wymiarow, **size(m)** - rozmiar
   e) **m'** - transpozycja
   f) Mnożenie macierzy/wektorów (liczba): **m'*m**
   g) Mnożenie macierzy/wektorów (macierz): **b=m*m'**
   h) Mnożenie elementów całej macierzy: **c=m.*m** (wynik: macierz o elementach będących iloczynami)
   i) **det(b)** – wyznacznik macierzy
   j) **b(1,1)=10** – dostęp do elementów macierzy
   k) **b, det(b)**
   l) **det(m'*m)** - operacje złożone
   m) **E=eig(b)** - wartości własne
   n) **[V,D]=eig(b)** - wartości i wektory własne
   o) Wektoryzacja operacji znacznie przyspiesza dzialanie:
      i) 1:10
      ii) od:step:do
      iii) b(:,2)=3
      iv) b(1:2:3,1)=4

7. Operacje wejścia-wyjścia (zmienne mozna zobaczyc w workspace) i operacje na plikach
   a) load/save *nazwa_pliku zmienna* –ascii, np. **save b.txt b -ascii**
   b) load/save *nazwa_pliku zmienna* –mat, np. **save b.bin b -mat**
   c) imread/imwrite - czytanie i zapisywanie obrazu z pliku w (niemal wszystkich formatach w czytaniu)
   d) bmpread/bmpwrite (przykłady będą przy okazji operacji na macierzach)
   e) wavread/wavwrite (przykłady będą przy okazji operacji na wektorach)

   f) Katalogi: pwd, cd, mkdir, dir
   g) Pliki: copyfile, delete, edit

Część II.
Wektory, proste wykresy

1. **Trivia**.
   Analiza poleceń w pliku „**dzwiek1.m**” – kopiować do podstawowego okna Matlaba
   a) komentarz na początku pełni rolę pomocy (wyświetlany jest gdy wpiszemy **help dzwiek1**)
   b) cały plik można wykonać (jako zbiór poleceń) wpisując **dzwiek1**
   c) ustalamy stałe **Amp=3**, **freq1=1**, **freq2=0.01**, **g=10**
   d) deklaracja wypełnionych zerami wektorów A, B, C, D o długości N=10000 poleceniem **A=zeros(N,1);**. Polecenie zeros(N) z jednym parametrem wyprodukowałoby domyślną macierz N *x* N. Można to prześledzić w **Workspace**.
   e) Do wektorów zapisujemy różne funkcje periodyczne z modulacjami fazy i częstości

```
for t=1:N
      A(t,1)=Amp*sin(freq1*t);
      B(t,1)=Amp*sin((freq1+freq2*freq2*t)*t);  %wzrost czestosc
      C(t,1)=Amp*sin(freq2*t)*sin(freq1*t);     %modulacja amplitudy
      D(t,1)=Amp*sin(freq1*t+g*sin(freq2*t));   %modulacja czestosci
end
```

   f) Pokazać wektor (wykres) można poleceniem plot: figure(1), **plot(A)**, fragment **plot(A(1:1000))**
   g) Interpretacja dźwiękowa: **sound(A)**
   h) Transformata Fouriera (prawdziwa – tj. zespolona): **F=real(fft(A));, plot(F)**
   i) Manipulacje na transformacie Fouriera pozwalają na stosowanie różnego rodzaju filtrów (dolno-, górno przepustowych, „korektorów graficznych” itp.).
      W naszym prostym przypadku można np. dodać dodatkową częstość:

```
F(6000:7000)=F(8000:9000); %kopiowanie z prawej
```

```
F(3000:4000)=F(1000:2000); %kopiowanie z lewej
plot(F)
sound(real(ifft(F))) %transformata odwrotna i sound( )
```

2. **Analiza plików wave** (*.wav) gotowymi procedurami Matlaba
   Plik „**dzwiek2.m**”.
   a) Czytanie pliku wave: **[y,fs,bits]=wavread('heartbeat.wav');**
      y – wektor zawierający dźwięk (plik jest mono, więc tylko jedna ścieżka)
      fs – odczytana z nagłówka pliku wave częstość próbkowania (sample rate) – wyznacznik jakości
      bits – ilość bitów w próbce
   b) wykres **plot(y)**
   c) analiza spektralna: **specgram(y)**, procedura rozbija sygnał na nakładające się na siebie okna (np. 1:100, 2:101, 3:102 itd.), następnie liczy transformatę FFT dla każdego okna, uśrednia po długości okna
   d) transformata Fouriera z przesunięciem, tak, żeby niskie częstości były po środku. I tak powinna być wyświetlana połowa wykresu: **f1=real(fftshift(fft(y)));, plot(f1)**
   e) Obliczanie spectrum: **spectrum(y)** – wykres spektrum („całka” po specgram).
   f) Odtworzenie pliku wave poleceniem sound z dodatkowymi parametrami: **sound(y,fs,bits)**
   g) Przykłady zastosowania w medycynie (w katalogu dzwiek2 jest plik **firstbeatcomp.m** porównujący zapis pracy serca w kilku chorobach).

3. Pokaz możliwości obrazowania i filtrowania dźwięku **stereo** (słowa, muzyka)
   Odtworzyć i zanalizować plik „**dzwiek3.m**”

Część III
Macierze, prosta obróbka obrazu

1. **Trivia** (plik **obraz1.m**)
   a) Czytanie formatu Windows bmp za pomocą funkcji bmpread: **[x1,map1]=bmpread('obraz1a.bmp');**
      x1 – macierz kolorów N *x* M dla każdego piksela (obraz nie przechowuje współrzędnych punktu)
      map1 – macierz 256 *x* 3 – mapa kolorów (skala kolorów R, G i B dla przedziału 0 – 255).
      Każdy element macierzy x1 ma wartość od 0 do 255. Kolor odpowiadający tej wartości należy sprawdzić w map1. Np. Dla punktu *x* = 1, y = 1 uzyskujemy poleceniem **x1(1,1)** wartość 154. Odpowiednia pozycja w **map1(154,:)** zwraca kolor RGB (0.5647, 0.6431, 0.4039). Ten sposób zapisu, nazywany obrazem indeksowanym (ang. *indexed image*), zmniejsza rozmiar zajmowany w pamięci, ale ogranicza ilość dostępnych jednocześnie kolorów do 256.
   b) Wyświetlenie obrazu indeksowanego: **image(x1)**
   c) Zastosowanie odpowiedniej mapy kolorów: **colormap(map1)**
   d) Żeby zobaczyć mapę kolorów można użyć **colorbar** – w przypadku zdjęć zazwyczaj nie jest w żaden sposób monotoniczna.
   e) Predefiniowane mapy: gray, hot, cool, bone, copper, pink itp. Przy zdjęciu zastosowanie tych map nie ma większego sensu. Natomiast jeżeli obraz jest w skalach szarości zastosowanie map jest bardziej sensowne.
   f) Przykładowa funkcja zamknięta w pliku M-file
      i) nagłówek ze słowem kluczowym function – tylko jedna funkcja może znaleźć się w pliku
      ii) funkcja ind2rgb zmienia obraz indeksowany ze skojarzoną mapą kolorów w macierz RGB, tj. macierz N *x* M *x* 3, która przechowuje kody kolorów RGB.
      iii) następnie oblicza się dla każdego punktu stopień szarości np. BW=(R+G+B)/3

```
%Funkcja z tablicy indeksowanej NxM robi tablice NxM w skali szarosci
%(c) Jacek Matulewski 2003

function value = rgb2gray(we,map)
disp('Z tablicy indeksowanej NxM robi tablice NxM w skali szarosci')
disp('Prosze czekac...')

nx=size(we,1);
ny=size(we,2);
disp(['Obraz wejsciowy, rozmiar=',num2str(nx),'x',num2str(ny)])
```

```
rgb=ind2rgb(we,map); %zmienia format z macierzy + mapa kolorow do RGB
value=zeros(nx,ny);
for i=1:nx
    for j=1:ny
        r=rgb(i,j,1); %wartosc koloru czerwonego (R)
        g=rgb(i,j,2); %wartosc koloru zielonego (G)
        b=rgb(i,j,3); %wartosc koloru niebieskiego (B)
        value(i,j)=255*(r+g+b)/3.0; %obliczanie stopnia szarosci
    end
end
```

g) Operacje na macierzach: mnożenie przez liczbę, dodawanie i odejmowanie obrazów
h) Przygotowywanie filmów:
   i) **n=200;, M = moviein(n+1);** – inicjuje pamięć na przechowywanie klatek filmu
   ii) w pętli od 0 do n-1 wyświetlanie zmodyfikowanych obrazów (np. przenikanie) za pomocą **image**. Ważne jest wpisanie **drawnow**, żeby wymusić natychmiastowe wyświetlanie
   iii) pobieranie klatek filmu z figure przygotowanego przez moviein za pomocą getframe: **M(:,i+1) = getframe;**
   iv) Film można obejrzeć poleceniem **movie(M)** lub zachować do pliku avi (co najmniej Matlab 6) **movie2avi(M,'obraz1-zkompresowany.avi')** **movie2avi(M,'obraz1-nie_zkompresowany.avi','COMPRESSION','None')**

2. Dwuwymiarowa szybka transformata Fouriera
   a) Przygotowanie dwuwymiarowej funkcji periodycznej:
   ```
   x=1:nx;      %wektor x
   y=1:ny;      %wektor y
   x1=(0.5+0.5*sin(0.1*x)') * (0.5+0.5*cos(0.03*y));     %macierz
   image(255*x1)
   ```
   b) wykonanie transformaty: **f1=fft2(x1,nx,ny);, image(real(fftshift(f1)))**
   c) transformacja odwrotna: **x2=real(ifft2(f1,nx,ny));, image(255*x2)**
   d) **obraz2b** (transfromacja dct2 – discrete cosinus transform 2D)

3. Wykrywanie krawędzi – pokaz: **obraz3** (ogólny schemat: **edge(i1,'prewitt');**)

4. Procedura do tworzenia „płaskorzeźby”: należy pokazać różnicę między skalami szarości sąsiednich punktów, kierunek, w którym odejmujemy sąsiednie punkty, wyznacza kierunek światła, jakim oświetlamy płaskorzeźbę (**obraz4**).

5. Filtrowanie obrazu funkcją na sąsiedztwie punktu (funkcja **filter** – należy zdefiniować macierz z wagami 9 punktów z otoczenia włączając sam punkt, które należy dodać w przefiltrowanym obrazie). Pozwala to na rozmycie obrazu, relief, wyostrzenie. Pokaz predefiniowanych filtrów – **obraz5**.

6. Sterowanie jasnością (ang. *intensity*)
   ```
   J = imadjust(I,[low_in high_in],[low_out high_out],gamma)
   ```
   a) wycina to co poza low_in i high_in
   b) to co przejdzie przez wstępny filtr przeskalowuje do granic low_out i high_out
   c) modyfikuje parametr gamma (potęgowanie: wejście^gamma, czasem z liniową funkcją dla malych wartosci; gamma=1 nic nie zmienia)
   d) identycznosc = **imadjust(***wejscie***,[0 1],[0 1],1)**

Część IV

Wizualizacja

1. Wykresy funkcji jednoargumentowych (**help graph2d**)
   a) Prosty wykres:

```
x=-2*pi:0.01:2*pi;              %wektor x
min(x), max(x), size(x)
y1=sin(x);                      %obliczanie wektora y
h=plot(x,y1);                   %wykres, h - uchwyt do wykresu
```

   Dopuszczalna jest też forma h=plot(y1), ale wówczas osią x jest indeks odpowiedniego elementu y

   b) Formatowanie linii:

```
set(h,'Color','r')              %czerwona linia
set(h,'Color',[0,0,0])          %czarna linia
set(h,'Color',[1,0.5,0.75])     %linia o kolorze R=1,G=0.5,B=0.75
set(h,'LineWidth',2)            %grubosc linii 2
set(h,'LineStyle','--')         %linia przerywana
set(h,'LineStyle',':')          %linia kropkowana
set(h,'LineStyle','-.')         %linia dash-dot
set(h,'LineStyle','-')          %linia ciagla
```

   c) Formatowanie osi:

```
axis([-2*pi 2*pi -1 1]) %obciecie wykresu AXIS([XMIN XMAX YMIN YMAX])
grid on                         %siatka
xlabel('kat')                   %opis osi x
ylabel('sin')                   %opis osi y
set(gca,'XTick',-2*pi:pi:2*pi)                              %ticks
set(gca,'XTickLabel',{'-2pi','-pi','0','pi','2pi'})
```

   d) W niektórych opisach można stosować notację TeX:

```
xlabel('-2\pi < \theta < 2\pi')
```

   e) Kilka funkcji na tym samym wykresie:

```
hold on                         %dodawanie do istniejącego wykresu
y2=cos(x);
y3=-sin(x);
y4=-cos(x);
plot(x,y2,x,y3,x,y4);
axis([min(x) max(x) min(y1) max(y1)])
%Uwaga! Wykres wielu zmiennych powinien być podany parami z argumentem x.
%Inaczej plot(y1,y2) interpretowany jest jako wykres y2(y1)
legend(h,'Sin','Cos','Minus Sin','Minus Cos')  %legenda
```

   f) Wykres może składać się z kilku podwykresów:

```
figure(2)
subplot(2,2,1), plot(y1)
subplot(2,2,2), plot(y2)
subplot(2,2,3), plot(y3)
subplot(2,2,4), plot(y4)
```

2. Okno wykresu – interakcyjne konfigurowanie wykresu. Eksport wykresu (**Tools \ Edit Plot**)

3. Wykresy funkcji dwuargumentowych (**help graph3d**)
   a) Funkcje image, imagesc, imshow (używane w części III), pcolor

```
[x,y,z]=peaks(100);          %przykladowa funcja 2D
imagesc(x)                   %x i y to nie wektory tylko macierze
imagesc(y)                   %macierzy wymagaja funkcje wykresow 3D
imagesc(x+y)
imagesc(y*x)
imagesc(x*y)                 %stala
imagesc(x.*y)
```

   b)
```
figure(1), imagesc(z)        %pokazanie przeskalowanej macierzy z
colorbar
figure(2), pcolor(x,y,z), shading flat
```

   c) Wykresy konturowe:

```
figure(3), contour(x,y,z,20)       %wykres konturowy, il. poziomic=20
figure(4), contourf(x,y,z,20)      %wypelniony
figure(5), contour3(x,y,z,20)      %3D, warto przelaczyc grid off
figure(6), [c,h]=contourf(z,10);, clabel(c,h)
```

   d) Funkcje wykreślające powierzchnie:

```
figure(7), surf(x,y,z)
figure(8), mesh(x,y,z)
figure(9), meshc(x,y,z)

figure(10)
surf(x,y,z)
axis([min(min(x)) max(max(x)) min(min(y)) max(max(y)) min(min(z)) max(max(z))])
shading flat                       %gladkie cieniowanie
alpha(0.7)                         %przezroczystosc
```

4. Wykresy słupkowe i kołowe:

```
y=rand(3,10)                 %macierz 3 x 10
figure(1), bar(y)            %wykresy slupkowe
figure(2), bar3(y)
figure(3), bar3(y,'stack')
figure(4), area(y)           %wykres warstwowe
figure(5), pie(y)            %wykres kolowy
```

Wersja z 26 lutego 2003.

Najnowsza wersja dokument: http://www.phys.uni.torun.pl/~jacek/dydaktyka/matlab.pdf
Pliki z przykładami: http://www.phys.uni.torun.pl/~jacek/dydaktyka/matlab.zip