DSP - Cyfrowe przetwarzanie sygnałów
Laboratorium 0 – wprowadzenie do Matlab'a

Jakub Moroń, Paweł Hottowy

Utworzono: Styczeń 2022

Ostatnia zmiana: 6 marca 2023

Spis treści

Konwencja

Ten zapis oznacza ważne spostrzeżenie bądź wniosek do wykonanego zadania.

Ten zapis oznacza zadanie do samodzielnej realizacji.

1 Dla zaawansowanych

Uczestnicy kursu znający Matlab'a mogą podjeść do tego laboratorium "na odwrót" – podjąc się od razu realizacji zadań z wymienionych poniżej podrozdziałów, konsultując się jedynie z resztą instrukcji (i prowadzącym) w razie problemów.
Zadania do samodzielnej realizacji znajdują się w podrozdziałach:

- 4.3
- 5.6
- 7.5
- 8.2

2 Uruchamianie Matlab'a

Matlab jest dostępny na taurus-ie, oraz lokalnie w pracowni 101.

W czasie zajeć proszę używać tylko i wyłącznie lokalnej instalacji z pracowni 101.
Aby uruchomić Matlab'a, należy wykonać komendę

```
/opt/matlab/MATLAB_2022/bin/matlab
```

Alternatywnie można na stałe dodać ścieżkę do Matlab'a do zmiennej systemowej `$PATH`. W tym celu należy wyedytować plik

```
~\.bashrc
```

i dodać na jego końcu

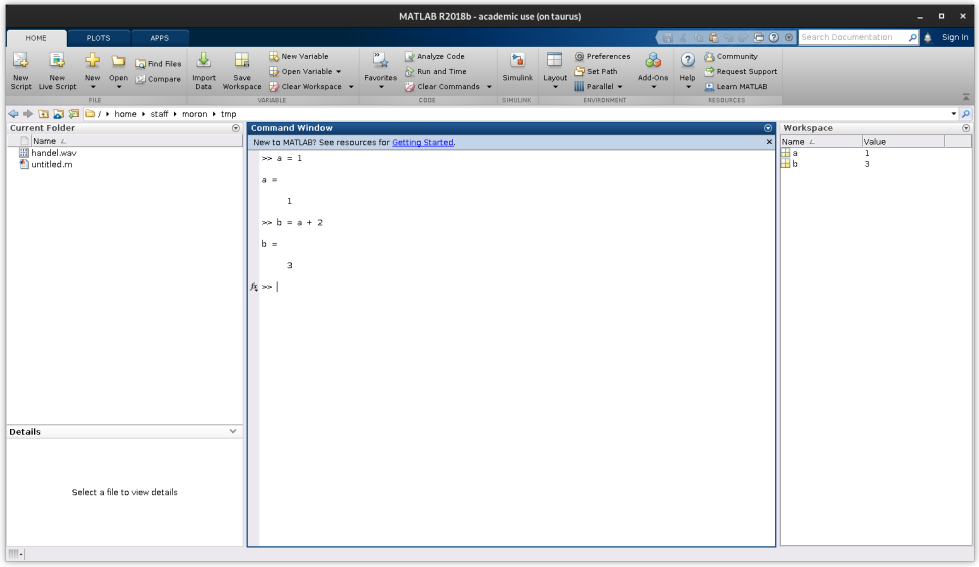
```
export PATH=$PATH:/opt/matlab/MATLAB_2022/bin/
```

Po ponownym zalogowaniu się lub otworzeniu nowej konsoli, Matlab'a można uruchomić komendą

```
matlab
```

3 Podstawy Matlab'a

3.1 Okno Matlab'a



Rysunek 1: Główne okno Matlab'a

Przy pierwszym uruchomieniu Matlab'a otwarte zostanie okno w standardowym układzie, zawierającym trzy panele:

- "Current folder" – dostęp do plików na dysku;
- "Command Window" – interaktywna konsola w której wpisujemy i wykonujemy komendy Matlab'a;
- "Workspace" – zbiór zmiennych stworzonych w danej sesji i/lub załadowanych z pliku.

W nagłówku panelu "Command Window" znajduje się link "Getting Started", otwierający podręcznik z podstawami Matlab'a. Stronę internetową zawierającą ten sam podręcznik można otworzyć poprzez dostępny na UPEL-u link "MATLAB – documentation". Niniejsza instrukcja, w wielu miejscach, jest wzbogaconym tłumaczeniem tej strony.

3.2 Zmienne

Pracując z Matlab'em, przede wszystkim tworzymy zmienne i wywołujemy funkcje wpisując odpowiednie komendy w konsoli "Command Window".
Przykładowo, poniższa komenda tworzy zmienną `a` i przypisuje jej wartość 1:

```
a = 1
```

Zaobserwuj, że zmienna pojawiła się w panelu "Workspace", ponadto wartość zmiennej została wypisana w konsoli.

Kolejne zmienne możemy też utworzyć poprzez operacje matematyczne, bądź wywołanie funkcji, na uprzednio utworzonych zmiennych

```
b = a + 2
c = 10
d = b * c
e = cos( b )
```

Jeżeli nie podamy wprost nazwy zmiennej do której ma zostać wpisany wynik operacji, Matlab utworzy automatycznie zmienną `ans` aby przechować wynik.

Potrawisz przewidzieć jaką wartośc przyjmie zmienna **`y`** *w wyniku następujących komend?*

```
x = 10
x / 2
ans + 5
y = ans + 3
```

Jeżeli komendę zakończymy znakiem średnika, Matlab wykona obliczenie i umieści wynik w stosownej zmiennej, ale nie wyświetli go w konsoli.

```
zmienna = 1;
```

3.3 Czyszczenie okna i sesji

Aby wyczyścić jedynie konsolę "Workspace", bez utraty dotychczasowych wyników, należy użyć komendy

```
clc
```

Aby wyczyścić wszystkie zmienne (przywrócić Matlab'a do stanu początkowego) należy użyć komendy

```
clear
```

4 Skrypty i zapisywanie zmiennych

4.1 Skrypty

Zamiast wpisywać polecenia wprost w konsoli, można stworzyć skrypt który może być wielokrotnie uruchamiany i modyfikowany. W tym celu należy wybrać "New -> Script" z górnego menu. Otworzy to czwarty panel Matlab'a – "Editor" oraz dodatkowe menu górne.
Powyższe można też uzyskać poleceniem

```
edit nazwa_skryptu
```

Do skryptu można następnie wpisać wymagane komendy oraz zapisać skrypt na dysku ze standardowym rozszerzeniem `*.m`.

Aby uruchomić skrypt należy kliknąć ikonę "Run" w menu "Editor".

4.2 Zapisywanie zmiennych

Aktualną wartość wszystkich zmiennych można zapisać do pliku (standardowe rozszerzenie `''*.mat''`) poleceniem, lub wywołaniem funkcji (koniecznie gdy nazwa pliku jest zmienną):

```
save zmienne.mat
save( 'zmienne.mat' )
```

Można również zapisać jedynie wybrane zmienne:

```
a = 1;
b = 2;
c = 3;
save zmienne.mat a
save( 'zmienne.mat', 'a' )
```

Aby zapisać wartości w postaci tekstowej, należy użyć przełącznika `-ascii`

```
save zmienne.mat a -ascii
save( 'zmienne.mat', 'a', '-ascii' )
```

Do załadowania wartosci zmiennych z pliku służy polecenie

```
load( 'zmienne.mat' )
```

4.3 Zadania

Napisz skrypt który

1. utworzy zmienne `x = 10` , `y = x * 2.5` , `z = sqrt( x + y )`;
2. zapisze wartość zmiennej `z` do pliku (i **tylko** tej jednej zmiennej);
3. wyczyści wszystkie zmienne;
4. załaduje z powrotem zmienną `z` z pliku;
5. wypisze wartość `z` w konsoli. Zadbaj by było to **jedyne** co zostanie wypisane przy wykonaniu skryptu.

5 Operacje na macierzach i wektorach

5.1 Podstawowe zagadnienia

Matlab standardowo operuje na macierzach i wektorach, nie tablicach jakie znamy z języków programowania. Oczywiście wektor reprezentować będzie jednowymiarową tablicę, macierz - tablicę dwuwymiarową, jednak należy pamiętać że Matlab, wykonując operacje matematyczne na tych obiektach, będzie natywnie traktował je jako macierze i wektory.
Wektor (jednowymiarową tablicę) tworzymy następująco

```
a = [ 1 2 3 ]
```

zaś macierz (tablicę dwuwymiarową) poprzez podanie kolejnych wierszy rozdzielonych średnikami

```
b = [ 1 2 3; 4 5 6; 7 8 9 ]
```

Co się stanie w wyniku następującego polecenia? Dlaczego?

```
zle_b = [ 1 2 3; 4 5; 6 7 8 ]
```

Typ i rozmiar danej zmiennej możemy sprawdzić komendmi

```
whos a
size( b )
```

Wykonaj następujące operacje. Potrawisz wytłumaczyć ich wyniki?

```
a + b
a * b
b * a
```

Ostatnie z poleceń można naprawić zmieniając orientację wektora `a` z jednego wiersza o trzech kolumnach na jedną kolumnę o trzech wierszach, czyli dokonując transponowania:

```
a
a'
b * a'
```

Zwróć uwagę na rozmiar (orientację) wyników operacji `a*b` *oraz* `b*a'`.
Macierz można też przemnożyć przez stałą, dodać stałą do wszystkich jej elementów, a nawet wykonać funkcję na poszczególnych elementach macierzy (lub wektora)

```
b * 2
b + 100
sin( b )
```

Potrafisz wytłumaczyć wynik poniższej operacji (^ to potęgowanie)?

```
b ^ 2
```

5.2 Podstawowe operacje na elementach macierzy

Ponieważ mnożenie, potęgowanie i tym podobne operacje wykonywane są w Matlab'ie w sposób macierzowy, aby wykonać operację mnożenia (albo przykładowo potęgowania) każdego z elementów macierzy niezależnie, należy użyć operatorów matematycznych poprzedzonych kropką.

```
b .^ 2
```

Podobnie możemy zrobić mnożąc macierz przez wektor. Zamiast mnożenia macierzowego Matlab przemnoży każdy element danego wiersza macierzy `b` przez odpowiedni element wiersza wektora `a`

```
b .* a
```

Zwróć uwagę że wyrażenia `b .* a` *oraz* `a .* b` *dają ten sam wynik, zaś mnożenie macierzowe (operator bez kropki) – nie.*
Zwróć uwagę że nie istnieją operatory `.+` *czy* `.-` *(dlaczego?).*

5.3 Funckje wspomagające inicjalizację macierzy

Poniższe funkcje tworzą macierze o rozmiarze `rozmiar` x `rozmiar` (wiersze x kolumny), wypełnione wstępnie zerami lub jedynkami.

```
rozmiar = 10;
zera = zeros( rozmiar )
jedynki = ones( rozmiar )
```

Aby utworzyć wektor, należy jeden z wymiarów macierzy ustawić na 1:

```
wektor_zer = zeros( 1, rozmiar )
wektor_transponowany_zer = zeros( rozmiar, 1 )
```

Użyj polecenia `help zeros` *i dowiedz się dlaczego dwa ostatnie polecenia utworzyły wektory.*
Utwórz matrycę o trzech wierszach i czterech kolumnach (3x4) zainicjalizowaną jedynkami.

Inicjalizacja (automatycznie ustawiająca rozmiar) wektora wartościami z przedziału 1–10 i domyślnym krokiem 1:

```
w = [ 1:10 ]
```

oraz to samo, tylko z krokiem 0.25:

```
w_f = [ 1:0.25:10 ]
```

Inicjalizacja wektora zaierającego `rozmiar` elementów, równomiernie rozłożonych między wartościami 1–10:

```
w_lp = linspace( 1, 10, rozmiar )
```

Zwróć uwagę że powyższe polecenie zrobiło to samo, co `w = [ 1:10 ]`. Co się jednak stanie w wyniku poniższych poleceń?

```
[ 1:11 ], size( ans )
linspace( 1, 11, rozmiar ), size( ans )
```

Poniższe polecenia generują wektory 1x5 o losowych wartościach z rozkładu jednorodnego od 0 do 1:

```
rand( 1, 5)
```

oraz z rozkładu normalnego o wartości średniej 0 oraz odchyleniu standardowym 1:

```
randn( 1, 5)
```

Wygeneruj wektor 10 wartości z rozkładu normalnego o wartości średniej 5 i odchyleniu standardowym 2.5. Podpowiedź – `help randn`.

5.4 Wyłuskiwanie elementów i łączenie macierzy

Wyłuskania elementu z i-tego wiersza i j-tej kolumny dokonujemy przez

```
b( i, j)
```

Można też użyć liniowego indeksu, np `b( 4 )`, jednak nie jest on intuicyjny – Matlab numeruje elementy w wieszach pierwszej kolumny, potem drugiej, itp. Tym samym `b( 4 )` jest tożsame `b( 1, 2)` .
Można także wyłuskać fragment macierzy, np dwa pierwsze wiersze i dwie pierwsze kolumny macierzy `b`:

```
b( 1:2, 1:2 )
```

albo pierwszą i trzecią kolumnę z dwóch pierwszych wierszy, itd:

```
b( [1 2], [1 3] )
```

Z użyciem operatorów wyłuskania można też dokonać przypisania wartości do danego elementu macierzy, np:

```
kopia_b = b;
kopia_b( 1, 3 ) = 100
kopia_b( 2:3, 1:2 ) = -1
```

Macierze można łączyć (konkatenować) operatorem nawiasów kwadratowych

```
c = ones( 3 );
[ b, c ]
```

Wykonaj poniższe operacje. Rozumiesz ich wyniki?

```
[ c, b ]
[ c, b' ]
[ c, b ]'
[ c; b ]
```

Poniższe funkcje wspomagają replikację macierzy i wektorów:

```
w_1x5 = repmat( w, 1, 5 );
w_5x1 = repmat( w, 5, 1 );
```

5.5 Operacje logiczne

Na całych macierzach można wykonywać operacje logiczne (np. porównanie). Wynikem jest macierz wartości logicznych reprezentujących wynik danej operacji dla każdego z elementów macierzy. Przykładowo: poniższe wyrażenie znajduje wszystkie elementy macierzy `b` większe od 4:

```
maska = b > 4
```

Macierzy wartości logicznych można następnie użyć, przykładowo, do wyzerowania elementów nie spełniających warunku:

```
b .* maska
% albo to samo w jednej linii. Na marginesie -znak "%" rozpoczyna komentarz
b .* ( b > 4 )
```

Mnożenie macierzy przez macierz wartości logicznych jest możliwe tylko dla niektórych typów danych. W innych przypadkach, szczególnie typu `uint8`, stosowanego natywnie do reprezentacji obrazu, wymagana jest uprzednia jawna konwersja typu maski z wartości logicznej do typu maskowanej macierzy. Przykładowo, jeśli macierz `b` zawiera dane typu `uint8`:

```
b .* uint8( maska )
% albo
b .* uint8( b > 4 )
```

Do wypisania indeksów elementów macierzy spełniających warunek służy funkcja:

```
find( b > 4 )
```

Funkcja ta wypisuje indeksy zdefiniowane jak podano na początku tego podrozdziału.

Macierzy wartości logicznych (maski) możemy też użyć do uzyskania wektora zawierającego te z elementów z macierzy B, dla których odpowiedni element maski był prawdą:

```
b( maska )
```

5.6 Zadania

Zadania do samodzielnej realizacji.

- Stwórz wektor `A` liczb całkowitych malejących od 5 do -5.
- Podnieś każdą z wartości wektora `A` do potęgi 3.
- Stwórz macierz B o wymiarach 4x11, w której każdy wiersz będzie zawierał to, co wektor A.
- Stwórz macierz C która będzie transpozycją macierzy B.
- Stwórz 1000-elementowy wektor D wartości pseudolosowych o rozkładzie normalnym, wartości średniej μ 10 i odchyleniu standardowym σ 1.5, a następnie:
  - Stwórz kopię wektora mającą wyzerowane wszystkie wartości mniejsze od 10;
  - Wypisz ile wartości jest mniejsze od 10 (μ), 8.5 ($\mu - \sigma$), 7.0 ($\mu - 2\sigma$) oraz 5.5 ($\mu - 3\sigma$). Czy otrzymane wyniki zgadzają się z oczekiwaniami co do wartości rozkładu normalnego?

6 Pętle i instrukcje warunkowe

W skryptach możemy umieszczać standardowe elementy znane z języków programowania – pętle `for` oraz `while`, a także instrukcje warunkowe `if` lub `switch`.

Przykładowo poniższy skrypt oblicza pierwsze `N` elementów ciągu Fibonacci-ego i wypisuje "poza zakresem" gdy wartość wyliczonego elementu jest większa od 10^{20}, wraz z indeksem tego elementu:

```
N = 100;
f( 1 ) = 1;
f( 2 ) = 1;

for n = 3:N
       f( n ) = f( n-1 ) + f( n-2 );

       if f( n ) > 1e+20
              n + " -poza zakresem"
       end
end

f( 1:10 )
```

7 Wykresy

7.1 Podstawy

Do rysowania wykresu służy funkcja `plot` . Najpierw, oczywiście, należy stworzyć zmienne zawierające dane do wykresu:

```
x = linspace( 1, 2*pi, 100 );
y = sin( x );
plot( y );
```

Wykonaj poniższe polecenie – w czym tkwi różnica między oboma wykresami?

```
plot( x, y );
```

Zauważ że nowy wykres nadpisał stary. Aby otworzyć nowe okno, należy użyć polecenia

```
figure, plot( x, y );
```

7.2 Formatowanie wykresów

Wykonaj zadania z poprzedniego podpunktu tak, aby skrypt tworzył dwa niezależne okna z wykresami.

Zauważ, że jeżeli nie zamkniesz ręcznie okien otwartych przez poprzednie wykonanie skryptu, kolejne wykonanie doda nowe okna pozostawiając stare.
Aby zamknąć wszystkie istniejące okna z wykresami należy dodać na początku skryptu

```
close all;
```

Dodawanie podpisów osi:

```
xlabel( "Faza [rad]" );
ylabel( "sin(x) [au]" );
```

Dodawanie tytułu wykresu:

```
title( "Wykres funkcji sin(x)" );
```

Zauważ że podpisy dotyczą ostatnio utworzonego wykresu.
Aby zmienić podpisy na pierwszym z wykresów, należy go uprzednio wybrać funkcją `figure()`, której argumentem jest numer wykresu (np. dla "Figure 1" będzie to 1):

```
figure( 1 );
xlabel( "numer probki [-]" );
ylabel( "sin(x) [au]" );
title( "Wykres funkcji sin(x)" );
```

Alternatywnie można skorzystać z funkcji `gca`, zwracającej uchwyt do ostatnio utworzonego wykresu, np:

```
plot( x, y );
handler = gca;
```

Wywołanie funkcji `get` na uchwycie wypisuje w konsoli listę wszystkich własności obiektu. Można tego użyć do dalszej konfiguracji wykresu, np:

```
get( handler )
set( handler, 'FontSize', 14 )
set( handler, 'XTick', [-0.1:0.1:1.1] )
```

Wiele z opcji wykresów można ustawić na kilka sposobów. Przykładowo, zakresy na osiach OX i OY można ustawić poleceniem

```
axis([-0.2 1.2 -6 6]);
```

bądź poprzez własności

```
set( handler, 'XLim', [ -0.2 1.2 ] )
set( handler, 'YLim', [ -6 6 ] )
```

Jeżeli uchwyt do wykresu nie został pobrany w chwili jego tworzenia, można poleceniem `gca` uzyskać uchwyt do aktualnego wykresu. Analogicznie `gcf` pobiera uchwyt do aktualnego okna (figure) w którym wyświetlany jest wykres.

Zauważ że polecenie `plot` natywnie nadpisuje poprzedni wykres.
Aby umieścić wiele krzywych na jednym wykresie należy użyć polecenia `hold on`. Spowoduje to że kolejne komendy `plot` będą się dodawać do wykresu, zamiast go nadpisywać:

```
y2 = cos( x );

figure, plot( x, y, "r--" );
hold on;
plot( x, y2, ":" );

legend( "sin", "cos" );
xlabel( "Faza [rad]" );
ylabel( "Wartosc [au]" );
title( "Wykres funkcji sin(x) i cos(x)" );
grid on;
```

W powyższym skrypcie pojawiły się dodatkowe elementy, jak przypisanie legendy do krzywych funkcją `legend` oraz zmiana stylu wyświetlania krzywej poprzez trzeci argument funkcji `plot`. Więcej o stylach znajdziesz w dokumentacji (link na UPEL-u "MATLAB - formatowane wykresów"). Ponadto włączona została siatka poleceniem `grid on`.

7.3 Wiele wykresów w jednym oknie

7.3.1 subplot

Niezależnie od wersji Matlab'a do umieszczenia wielu wykresów w jednym oknie możemy użyć funkcji `subplot`. Jednakze jeżeli dysponujemy Matlab'em w wersji >= R2019b, narzędziem wygodnieszym w użyciu może być `tiledlayout`.
Funckja `subplot` przyjmuje następujące argumenty: ilość wierszy wykresów umieszczanych we wspólnym oknie, ilość kolumn, oraz numer wykresu. Żeby nie było za prosto, numeracja wykresów w siatce jest ortogonalna do numeracji elementów w macierzy – wykresy numerowane są od 1 do k w pierwszym wierszu, $k+1$ do $2k$ w drugim, itp.
Przykładowo, poniższy kod umieszcza obie krzywe *sin* i *cos* na dwóch wykresach w jednym rzędzie:

```
subplot( 1, 2, 1); plot( x, sin( x ) ), title( "Sinus" );
subplot( 1, 2, 2); plot( x, cos( x ) ), title( "Cosinus" );
```

Zauważ że wykresy pojawiły się w pierwszym z okien nadpisując jego zawartość. Czy wiesz dlaczego? Czy wiesz jak spowodować, by stworzyło się nowe, trzecie okno z tym wykresem?

Umieść wykresy jeden pod drugim (w jednej kolumnie).

7.3.2 tiledlayout - Matlab >= R2019b

W Matlab'ie od wersji R2019b efekt z poprzedniego punktu można uzyskać z użyciem `tiledlayout`. Jest to nowsza, nieco bardziej rozbudowana funckja:

```
t = tiledlayout( 2, 2 );
title( t, "Trigonometric Functions" );

nexttile
plot(x,sin(x))
title("Sinus")

nexttile
plot(x,cos(x))
title("Cosinus")
```

7.4 Zapis wykresu do pliku

Aby zapisać okno z wykresami do pliku, należy użyć polecenia

```
exportgraphics(gcf, 'nazwa_pliku.png' );
```

Zauważ że użyliśmy tutaj komendy `gcf` *pobierającą uchwyt do ostatnio wybranego okna. W tym miejscu może też być wprost uchwyt do okna, ale też i pojedynczego wykresu (albo* `gca`*).*

Rozmiar obrazka można zdefiniować zmieniając rozmiar okna, np w calach:

```
set(gcf,'Units','inches','Position',[0 0 7 3]);
```

a następnie eksportując obrazek z podaniem rozdzielczości na cal (tutaj 300). Rozmiar w pikselach to iloczyn rozdzielczości i rozmiaru okna.

```
exportgraphics(gcf, 'nazwa_pliku.png', 'Resolution', 300 );
```

Istnieje również możliwość użycia funkcji `print`, eksportującej zawsze ostatnio wybrane okno:

```
print( 'nazwa_pliku.png', '-dpng', ['-r' '300'] );
```

Funkcja ta udostępnia znacznie więcej formatów, w szczególności wektorowych, niż `exportgraphics`. Więcej informacji na jej temat – `help print` oraz dokumentacja online.

7.5 Zadania

Zadania do samodzielnego wykonania
Napisz skrypt który:

- Na początku zamknie wszystkie okna pozostałe po uprzednio wykonanych skryptach oraz usunie wszystkie zdefiniowane zmienne;
- Utworzy odpowiedni wektor `x` oraz obliczy wartość funkcji $sin(x)$ oraz $cos(x)$ w przedziale obejmującym 5 okresów funkcji sinus.
- Utworzy okno z przebiegami obu funkcji oraz ich iloczynem $(sin(x) * cos(x))$, umieszczonymi na jednym wykresie, z różnym formatowaniem (tak, aby się od siebie odróżniały);
- Utworzy drugie okno w wykresami w siatce 2x2: sinus i cosinus w górnym rzędzie, $1 - sin(x)$ oraz $1 - cos(x)$ w dolnym;
- Utworzy trzecie okno wyświetlające sinus w przedziale jednego okresu $(0 - 2\pi)$. Uwaga - należy zmodyfikować wykres, nie dane do niego;
- Zapewni prawidłowe podpisy osi, jednostek, oraz tytuły wszystkich wykresów, a także legendę do krzywych na pierwszym wykresie;
- Zapisze wszystkie wykresy do plików o rozdzielczości co najmniej 640x480 pikseli, najlepiej w formacie wektorowym.

8 Obrazy i grafika

8.1 Podstawowe komendy

Do wczytania obrazu z pliku służy funckja

```
dane = imread( "nazwa_pliku.rozszerzenie" );
```

której argumentem jest nazwa pliku (opcjonalnie ze ścieżką). Funkcja ta potrafi wczytać wiele różnych formatów plików, standardowo rozpoznając format na podstawie rozszerzenia (można też wymusić format – patrz `help imread`). Dla formatów nie używających indeksowania kolorów (a takich będziemy używać), funkcja zwraca tablicę kolorów RGB.

Do wyświetlenia danych reprezentujących obraz służy funkcja

```
imshow( dane );
```

Dla uproszczenia operacji, w czasie zajęć będziemy z reguły pracować na obrazach w skali szarości. Wynika to z tego że:

- spora część przetwarzania obrazów kolorowych pracuje w innej przestrzeni (z innym modelem) kolorów niż RGB (np. kompresja JPEG używa przestrzeni YCbCr, komputerowa analiza obrazu – najczęściej HSL lub HSV). Definicja tych przestrzeni i metody konwersji pomiędzy nimi wykracza poza ramy tego przedmiotu;
- niezależnie od wybranej przestrzeni, niemal wszystkie operacje i przekształcenia wykonywane są niezależnie dla każdej składowej. Dlatego obliczenie które należy wykonać raz dla skali szarości, w przypadku obrazu kolorowego wystarczy po prostu powtórzyć trzy razy. Komplikuje to jednak niepotrzebnie skrypt i wydłuża czas jego wykonania.

Aby przekowertować obrazek do skali szarości, należy użyć funkcji

```
szare_dane = rgb2gray( dane );
```

Obraz w Matlab'ie jest reprezentowany zwykłą macierzą (o odpowiedniej ilości wymiarów). Z tego też względu można:

- stosować na nim dowolne operacje poznane wcześniej (łącznie z wycinaniem fragmentów, zastępowaniem i maskowaniem);
- stworzyć i zainicjalizować własną macierz, a następnie wyświetlić ją jako obraz.

8.2 Zadania

Zadania do samodzielnego wykonania

1. Pobierz z UPEL-a, z sekcji poświęconej aktualnemu laboratorium, archiwum z plikami i rozpakuj je (najlepiej w katalogu w którym zapisujesz sktypty Matlab'a);
2. Wczytaj, z rozpakowanego archiwum, plik "ball.jpg". Plik ma rozdzielczość 680x680 pikseli. Następnie:
   a) Zobacz jaki rozmiar (i ile wymiarów) ma tablica przechowująca wczytany obraz, oraz jaki typ danych (liczb) przechowuje;
   b) Wyświetl wczytany obrazek;
   c) Rozłóż obrazek na trzy, każdy reprezetujący (w skali szarości) jedną ze składowych R, G oraz B;
   d) Wyświetl w osobnych oknach obrazy zawierające poszczególne składowe kolorów oryginalnego obrazka. Czy Twoim zdaniem są to rzeczywiście składowe RGB?
3. Stwórz w Matlab'ie obraz 256x256 pikseli, zawierający cztery kwadraty - dwa czarne w lewym górnym i prawym dolnym rogu, dwa białe w pozostałych.
4. *Dodatkowe, dla zainteresowanych:* Stwórz kolorowy obraz przedstawiający szachownicę albo znak rozpoznawczy polskiego lotnictwa.
5. Wczytaj plik "tree.jpg" a następnie:
   a) Wyświetl wczytany obrazek;
   b) Przekonwertuj go na skalę szarości i wyświetl wynik w drugim oknie;
   c) Dowiedz się Matlab'em jaki jest rozmiar obrazka w pixelach;
   d) Wytnij z lewego, górnego rogu obrazka fragment o wymiarach 200x200 pikseli i wyświetl go w nowym oknie;
   e) Zastąp kwadrat 300x300 pikseli w prawym dolnym rogu zerami i wyświetl rezultat w nowym oknie;
   f) Ustaw na zero wszystkie piksele, których wartość jest mniejsza od 128 i wyświetl wynik. Powtórz maskowanie odwracając warunek (zerujemy piksele których wartość jest większa od 128).
   g) *Dodatkowe, dla zainteresowanych:* Spróbój dokonać prostych modyfikacji kolorów (z użyciem dowolnego zdjęcia, choćby "tree.jpg") – np. usunąć (wyzerować) jedną ze składowych R, G lub B, zastosować maskowanie progowe z poprzedniego podpunktu do jednej (lub dwóch) składowych, odwrócić jeden z kolorów (256 - wartość każdego piksela) itp.