Wykład podstawowy z APD nr 3A
(obejmujący wprowadzenie do środowiska, poleceń
oraz listy operandów Matlab'a w wersji 6.x
do zajęć od 2 to 5 tygodnia zajęć)

prowadzący: mgr inż. Artur Bernat
Politechnika Koszalińska
Wydział Mechaniczny
Katedra Mechaniki Precyzyjnej
rok akademicki 2004/2005
semestr zimowy

Zgodnie z obietnicą, na łamach zakładki zasobow materiałów dydaktycznych mojej strony domowej (funkcjonującej w ramach domeny:Katedry Mechaniki Precyzyjnej: kmp.tu.koszalin.pl), prezentuję III wykład podstawowy-suplement z APD, dotyczący przeglądu listy poleceń podstawowych środowska Matlab. Jest to ostatni wykład podstawowy z APD i jednocześnie w swojej treści, podjęta pewna próba spójnego, zwięzłego przedstawienia możliwości środowiska Matlab pod kątem wykorzystania przedstawionego zbioru wiedzy w przekształceniach na danych stanowiących cyfrową reprezentację dwuwymiarowych obrazów.

Poniżej przedstawiony wykład III z APD powinien dzielić się na opis zagadnień podstawowych dotyczących środowiska obliczeniowego Matlab oraz na zagadnienia specyficzne dla zajęć z APD, a dotyczących głównie wywołań funkcji z *Image Processing Toolbox*. Jednakże presja obecnej chwili, to jest konieczność elekttronicznego wsparcia rzeszy Szanownych Studentów w gorącym okresie zaliczeń międzysemestralnych każe mi łączyć przydatność i wszechstronność prezentowanego zasobu informacji ze zwięzłością przekazu i jego spójnością.

1.Zaganienia podstawowe w nawigacji we środowisku Matlab

W orientacji w liście wywołań poleceń i funkcji pomocne są następujące polecenia kontekstowej pomocy (neologizmy anglo-saskie kazałyby ją nazwać *manual'em*, *help'em*, itd):

help	-	podaje informacje dotyczące wybranej funkcji lub toolbox'a (*help help)*
helpdesk	-	dokumentacja w formacie hypertekstu (wymagana przeglądarka Netscape lub podobne narzędzie programowe)
helpwin	-	pomoc bieżąca, zapewniana w oddzielnym oknie
lookfor	-	bardzo pożyteczne polecenie kontekstowej pomocy, w orientacyjnym stawianiu pierwszych niepewnych kroków na nieznanym gruncie setek i tysięcy poleceń oraz wywołań funkcyjnych i składniowych zawiłości środowiska Matlab, jakkolwiek bardzo wolne w wywołaniu
which	-	ułatwia lokalizację funkcji oraz plików
what	-	wyświetla listę plików roboczego katalogu, ustawionego w ścieżce bieżącej
exist	-	sprawdza istnienie zadanej funkcji lub zmiennej w przestrzeni roboczej
who	-	ogólnikowe, poszlakowe listowanie nazw zmiennych w przestrzenii roboczej
whos	-	listowanie szczegółowe z nazwą, typem i rozmiarem zmiennej
path	-	wyświetla listę bieżących ściezek przeszukiwań katalogów zainstalowanej wersji pakietu Matlab
addpath	-	uaktualnia listę ścieżek przeszukiwanych
pwd	-	wyświetla bieżacy katalog roboczy sesji roboczej w środowisku Matlab
dir	-	listowanie zawartości bieżacego katalogu Matlab'a
type	-	polecenie listowania plikow m-skryptów
clc	-	czyści okno główne poleceń środowiska Matlab
clear all	-	z argumentem *all* lub bez żadnych argumentów usuwa wszystkie zmienne z przestrzenii roboczej
clear X Y Z	-	przykładowo, usuwa z przestrzenii roboczej zmienne *X*, *Y*, *Z* (jeśli tylko

		zmienne o wyszczególnionych nazwach istnieją)
save sesja	-	realizuje zapis zbioru zmiennych przestrzeni roboczej na dysk do pliku o nazwie *sesja*, zazwyczaj z domyślnym rozsszerzeniem *.mat*
load sesja	-	realizuje iimport do przestrzenii roboczej środowiska Matlab wartości zmiennych utrwalonych wcześniej w pliku o nazwie *sesja*

2 Podstawowe operacje arytmetyczne i bitowe, scalarne i macierzowe:

W środowisku Matlab, pozornie, na pierwszy rzut oka jest 'zamazywana' istotna różnica pomiędzy operacjami na skalarnych zmiennych, a operacjami na zmiennych macierzowych (oraz tensorowych, jak sądzę). Przynajmniej wrażenie to odnosi się w przypadku prostych operacji kończących się na dodawaniu i odejmowaniu. Ponadto deklaracja, inicjalizacja oraz reinicjalizacja zmiennych w porównaniu do typowych języków programowania 'epoki' *fortranowo-paskalowo-Cowej* jest dla istoty ludzkiej, nie zaznajomionej z klasyczną sztuką programowania, zupełnie naturalna i samorzutna (I przypadek poniżej):

```
>> a=2;
>> b=6;
>> c=b/a
c =
     3
```

,lub (II przypadek):

```
>> a=2;
>> b=6;
>> a\b;
>> ans

ans =
      3
```

W pierwszym przypadku dokonano deklaracji i inicjalizacji zmiennych *a* i *b* (a=2, b=6) z typem tych zmiennych określonym automatycznie przy inicjalizacji, a inicjalizację trzeciej zmiennej *c* dokonano wraz z przekazaniem wyniku dzielenia lewego argumentu przez prawy. Brak średnika 'sprowokował' wyświetlenie rezultatu bieżacej operacji, co może być uciążliwe w operacjach na danych większego formatu, np. danych wektorowych i macierzowych wymiaru drugiego lub wyższego.

W drugim przypadku powyżej, dzielenie prawego argumentu przez lewy zakończone zostało 'niemą' reakcją, tj. z koniecznością wywołania domyślnej zmiennej funkcjonującej w przestrzeni roboczej Matlab'a *ans*, która przechowuje rezultat wykonania ostatniej operacji.

Dla prezentacji elastyczności, spójności i ogólności punktu widzenia twórców Matlab'a w 'traktowaniu' zmiennych oraz przekształceń na ich wartościach, warto do rozważanego wątku wprowadzić co nieco zmiennych wektorowych i macierzowych. Polecenie *ones* to instrukcja tworząca domyślnie macierz kwadratową wypełnioną jedynkami, polecenie *zeros* to instrukcja środowiska Matlab tworząca (domyślnie) macierz kształtu kwadratowego wypełnioną zerami:

```
>> a=ones(3)
a =
     1     1     1
     1     1     1
     1     1     1

>> b=zeros(3)
b =
     0     0     0
     0     0     0
     0     0     0
```

Gdyby jednak na zmiennych *a* i *b* dokonać teraz próby obliczeń podobnych do tych dokonanych powyzej na zmiennych skalarnych:

```
>> c=a/b
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 0.000000e+000.

C =
   Inf   NaN   NaN
   Inf   NaN   NaN
   Inf   NaN   NaN
```

,to całość operacji zakończy się jedynie bezgłośnym 'miękkim' lądowaniem zamiast zdarzeniem-*crash*'*em* sesji uruchomieniowej programu. Dzielenie przez macierz osobliwą, nie jest dla środowiska Matlab niczym nowym. Najwyżej to tu, to tam we wartościach niby-wynikowej macierzy *c* pojawi się *NaN* (czytaj: *not-a-number*) lub *Inf (Infinity)*. Matlab, jako środowisko obliczeniowe jest dobrze usposobione do 'trudnych' przypadków dzielenia przez zero, odwracania macierzy osobliwych, itp. Mięciutkie lądowania to jego specjalność chyba, że można jeszcze uciec przypadkiem w domenę liczb zespolonych, jeśli początkujący adept sztuk obliczeniowych i algorytmów przetwarzania danych nie zadał zbyt trudnego zadania:

```
sqrt(-49) %komentarz:pierwiastkowanie 2-go stopnia liczby niedodatniej
ans =
        0 + 7.0000i
```

No to jak jest z tym uniwersalizmem (tj. spójnym traktowaniem we składni działań i operacji podstawowych na skalarach, wektorach i macierzach)? Ano, spójrzmy:

```
>> a=3;
>> a^2
ans =
     9

>> a=3*ones(3)
a =
     3     3     3
     3     3     3
     3     3     3

>> a^2
ans =
    27    27    27
    27    27    27
    27    27    27

>> a.^2
ans =
     9     9     9
     9     9     9
     9     9     9
```

Jak widać, podobieństwa w operacjach na skalarnych i macierzowych typach zmiennych kończą się już przy mnożeniu, dzieleniu czy potęgowaniu. Inny jest bowiem wynik potegowania w ujęciu matematycznym macierzy (operator ^), a inny jest wynik potęgowania po-elementowego macierzy *a* (operator **.^**), bardzo podobny nadal w wyniku działania do operacji na zmiennej skalarnej. Inny przykład:

```
>> 1/b
ans =
   0.2000
```

```
>> b=5*ones(3)
b =
     5     5     5
     5     5     5
     5     5     5

>> 1/b
??? Error using ==> /
Matrix dimensions must agree.

>> 1./b
ans =
    0.2000    0.2000    0.2000
    0.2000    0.2000    0.2000
    0.2000    0.2000    0.2000
```

Operacja dzielenia *1/b* na zmiennej skalarnej jest możliwa do wykonania, lecz okazuje się niewypałem w przypadku zmiennej *b* w postaci macierzy, chyba że zastosujemy dzielenie po-elementowe tj. z wykorzystaniem operatora: **./**, lecz wówczas nie można mówić o uzyskaniu w wyniku macierzy odwrotnej do macierzy *b*. Nawiasem mówiąc, zaangażowana tutaj w rachunkach macierzowych algebra liniowa zezwala na prawo- i lewo-stronne dzielenie przez siebie macierzy:

```
>> a=diag([1 -5 7])+2*ones(3)
a =
     3     2     2
     2    -3     2
     2     2     9

>> b=ones(3)+diag([5 3 6])
b =
     6     1     1
     1     4     1
     1     1     7

>> a/b %macierz a przez macierz b
ans =
      0.41176      0.35294      0.17647
      0.43137     -0.94771      0.35948
     0.098039       0.1634       1.2484

>> a\b %macierz b przez macierz a
ans =
       1.8812      0.76238     -0.24752
      0.62376     -0.75248     0.049505
     -0.44554      0.10891      0.82178
```

W technice rozwijania wartości szeregów arytmetycznych jako zmiennych przestrzeni roboczej, które z punktu widzenia użytkownika środowiska Matlab, w praktyce przekładają się na tworzenie zmiennych wektorowych, istnieje spora dowolność składni i duże pole do własnej kreatywności:

```
>> a=[1:2:9]
a =     1     3     5     7     9
>> a=[1:2:9]
a =     1     3     5     7     9
>> a=linspace(1,9,5)
a =     1     3     5     7     9
```

Oprócz *zeros* oraz *ones* w podstawowych operacjach na macierzach może być pomocne wywołanie *ones* oraz *diag*:

```
>> a=eye(3)
```

```
a =
     1     0     0
     0     1     0
     0     0     1

>> b=diag([1:3])
b =
     1     0     0
     0     2     0
     0     0     3


>> c=rot90(b)
c =
     0     0     3
     0     2     0
     1     0     0

>> c'
ans =
     0     0     1
     0     2     0
     3     0     0
```

Ten powyższy przykład zakończono na inicjalizacji zmiennej *c* zawartością macierzy b obróconej przeciwnie do ruchu wskazówek zegara o 90 stopni. Następnie podano wynik transpozycji macierzy *c* (operator: '). Wywołanie *ones* powoduje inicjalizację macierzy o zerowych elementach z wyjątkiem tych położonych na głównej przekątnej, które otrzymują wartość równą 1. Wywołanie *diag* powoduje inicjalizację w przestrzenii roboczej Matlab'a macierzy zerowej, z niezerowymi jednak elementami, zadanymi w pierwszym argumencie wywołania, a położonymi (domyśnie) na głównej przekątnej.

A propos analogicznych operacji na elementach macierzy, można tu przytoczyć wywołania *flipud* oraz *fliplr* mające na celu realizację odpowiednio: odwrócenia lustrzanego elementów macierzy na kierunku góra-dół oraz odbicia lustrzanego na kierunku lewa-prawa strona macierzy (wschód-zachód)

W praktyce, często w użyciu pozostają wprowadzone na użytek składni poleceń Matlab'a mechanizmy wyliczania oraz selektywnego wyboru i reinicjalizacji elementów macierzy:

```
>> a=zeros(3,5)
a =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

>> a(2,:)=1
a =
     0     0     0     0     0
     1     1     1     1     1
     0     0     0     0     0

>> a(:,3)=2
a =
     0     0     2     0     0
     1     1     2     1     1
     0     0     2     0     0

>> a(2,:)=[1:1:5]
a =
     0     0     2     0     0
     1     2     3     4     5
     0     0     2     0     0
```

```
>> a(:,3)=[10;10;30]
a =
     0     0    10     0     0
     1     2    10     4     5
     0     0    30     0     0
```

Dziwne na pozór wywołania typu: *a(2,:)* oraz *a(:,3)* mają na celu 'zaadresowanie' czy też wyszczególnienie zarówno w reinicjalizacji jak i kopiowaniu wartości, odpowiednio: w pierwszym przypadku wszystkich elementów z 2 wiersza macierzy a (rozmiar macierzy 3 wiersze na 5 kolumn), w drugim przypadku wszystkich elementów z 3 kolumny macierzy *a.* W kontynuacji powyższego przykładu, jeszcze tylko przykład z sumowaniem 'wyszczególnionych elementów macierzy a za pomocą wywołania operacji sumowania *sum:*

```
>> a
a =
     0     0    10     0     0
     1     2    10     4     5
     0     0    30     0     0

>> sum(a(:,3))
ans =
    50

>> sum(a(2,:))
ans =
    22

>> sum(a(:))
ans =
    62
```

Ostatnie wywołanie z treścią wewnętrznego wyrażenia typu *a(:)* umożliwia wyliczenie w szeregu, tj. w postaci wektora wierszowego wszystkich elementów macierzy potencjalnie dwu-, trzy- lub wielo-wymiarowej, co w rezultacie po zsumowaniu dało wynik 62. Inna demonstracja możliwości wyliczeniowych składni poleceń Matlab'a operujących na typach macierrzowych:

```
>> a=ones(3,3,3)

a(:,:,1) =
     1     1     1
     1     1     1
     1     1     1

a(:,:,2) =
     1     1     1
     1     1     1
     1     1     1

a(:,:,3) =
     1     1     1
     1     1     1
     1     1     1

>> sum(a(:))
ans =
    27
```

W przykładzie stworzono 3-wymiarową macierz jedynek *a*, której suma elementów wynosi 27.

Operacja wyliczenia elementów macierzy *a* może się okazać identyczna w skutkach wykonania z odpowiednio przeprowadzoną operacją przegrupowania elementów pierwotnej macierzy w macierz wynikową z pomocą wywołania *reshape:*

```
a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

>> b=a(:)'
b =
     1     5     9    13     2     6    10    14     3     7    11    15     4     8    12
16

>> b=reshape(a,1,16)
b =
     1     5     9    13     2     6    10    14     3     7    11    15     4     8    12
16
```

A propos postaci, czy też formatu wyświetlania wartości liczb:

```
>> a=3;
>> b=7;

>> a/b
ans =    0.4286

>> format long
>> ans
ans =   0.42857142857143

>> format short
>> ans
ans =    0.4286

>> format long e
>> ans
ans =    4.285714285714286e-001

>> format bank
>> ans
ans =           0.43

>> format hex
>> ans
ans =   3fdb6db6db6db6db

>> format rational
>> ans
ans =     3/7

>> format short g
>> ans
ans =      0.42857
```

, z zaprezentowanych oznaczeń różnych formatów danych wynikowych najwygodniejszym wydaje się być *format short g*, jakkolwiek czasami *format long e*, jest wymagany w zadaniach o zwiększonej precyzji wyświetlania. Z formalnego punktu widzenia:

FORMAT	-	domyślnie, wszystkie operacje wykonywane są w podwójnej precyzji
FORMAT SHORT	-	(domyślnie), ustawiony jest format stałoprzecinkowy z 5 cyframi znaczącymi wyświetlania (mantysa skalowalna)
FORMAT LONG	-	ustawiony jest format stałoprzecinkowy z 15 cyframi znaczącymi wyświetlania liczby (mantysa skalowalna)
FORMAT SHORT G-		najlepszy (najbardziej czytelny) ze zmienno- lub stało- przecinkowego formatu wyświetlania z 5 cyframi znaczącymi prezentacji
FORMAT LONG G -		najlepszy zmienno- lub stało- przecinkowy format wyświetlania z 15 cyframi znaczącymi prezentacji

Ogólnie, podsumowując w temacie omawianego punktu 2, najlepiej 'sprowokować' pomoc kontekstową do samoistnej prezentacji możliwości środowiska Matlab poleceniem o składni:

help +.

3.Operacje i przekształcenia na macierzach i wektorach (ciąg dalszy)

Wywołanie polecenia *size* ma na celu zbadanie rozmiaru (rozmiarów) zmiennej skalarnej , wektorowej, macierzowej (ogólnie n-wymiarowej):

```
>> a=ones(3,4)
a =
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> size(a)
ans =     3     4

>> size(a,1)
ans =     3

>> size(a,2)
ans =     4
```

Oczywiście, po braku wyszczególnienia wymiaru wywołanie *size* spowoduje wydruk na ekranie wiersza liczb odpowiadających odpowiednio ilości wierszy w macierzy (1 wymiar), ilości kolumn w macierzy (2 wymiar) oraz rozmiaru w kolejnych wymiarach dla macierzy 3- i więcej-wymiarowych.

W zakresie wyliczania lub operacji na wycinkach (tj. podmacierzach) zadanej macierzy 2-wymiarowej zwracam jeszcze raz uwagę na szczególną rolę operatora wyliczeniowego dwukropka (tak chyba mogę nazwać ten operator w pełnionej przez niego funkcji). Niech będzie zadana macierz *a* w elementach wynikająca z rozwinięcia szeregu liczb w kolejnych wierszach:

```
>> a=[1:4;5:8;9:12;13:16]
a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

Elementy podmacierzy zawierającej się w całości w wewnątrz macierzy *a* można 'zaadresować' w miarę jednoznacznie, tzn.:

```
>> a(2:3,2:3)
ans =
     6     7
    10    11
```

W przypadku jednak 'zaadresowania', np.:drugiej pionowej połowy macierzy *a*, istnieje pewna dowolność w formułowaniu wyrażenia:

```
>> a(:,3:4)% 1 przypadek
ans =
     3     4
     7     8
    11    12
    15    16

>> a(1:4,3:4) % 2 przypadek
```

```
ans =
     3     4
     7     8
    11    12
    15    16

>> [a(:,3) a(:,4)] % 3 przypadek
ans =
     3     4
     7     8
    11    12
    15    16

>> [a(1:4,3) a(1:4,4)] % 4 przypadek
ans =
     3     4
     7     8
    11    12
    15    16
```

Ta dowolność w formułowaniu składni poleceń, wynika przede wszystkim z uogólnionej możliwości odwoływania się do kolumn macierzy *a* (1 przypadek) albo możliwości precyzyjnego określania zakresu danych występujących w kolumnach macierzy *a* (2 przypadek). W obu tych przypadkach wykorzystano jedynie parę nawiasów okrągłych (tj.:( )), pełniących w wyrażeniach ze zmiennymi typu macierzowego rolę indeksacji elementów (o zgrozo!, w składni języków: Pascal, Fortran oraz C, tę rolę pełni para symboli [], pozostająca w zupełnej opozycji i sprzeczności ze składnią znaczeniową środowiska Matlab).

W przykładach 3 i 4 dodatkowo zastosowano parę symboli [] pełniących tutaj (jak i w większości pozostałych zawiłości składniowych poleceń) rolę wyliczeniowo-kontatenującą (tzn. z grubsza po polsku mówiąc: rolę ramowego czynnika sklejającego lub łaczącego np.:elementy podwektorów lub wielkości skalarnych we wektor/macierz-wiekszy koglomerat danych). Innymi słowy, wewnętrzne 2 podwyrażenia czy to dla przypadku 3, albo dla przypadku 4, wyszczególniają jedynie kolumnę 3 i 4 macierzy *a.* Operator wyliczeniowo-łączeniowy [] z przecinkiem lub spacją pomiędzy tymi 2 podwyrażeniami pozwala na stworzenie z kolumny 3 i 4, macierzy składającej się z 2 kolumn każda po 4 elementy. Zamiana przecinka lub spacji pomiędzy tymi podwyrażeniami na średnik, zmienia rezutat wywołania wyrażenia 3 i 4 z kontatenacji poziomej na pionową:

```
>> [a(:,3); a(:,4)]
ans =
     3
     7
    11
    15
     4
     8
    12
    16

>> [a(1:4,3); a(1:4,4)]
ans =
     3
     7
    11
    15
     4
     8
    12
    16
```

W rozwijaniu wartości wektora danych liczb rzeczywistych, pomocne może być nie tylko wywołanie *linspace*, lecz przede wszystkim (z technicznego punktu widzenia) wywołanie

logspace, które to wymaga zasadniczo trzech argumentów: punktu startowego (wykładnik potęgi przy podst. 10), punktu końcowego (wykładnik potęgi przy podst. 10) zakresu wartości rozwijanego wektora liczb oraz ilości punktów równomiernie rozłożonych na wyobrażonej osi liczb rzeczywistych skalowanej logarytmicznie przy podstawie dziesiętnej:

a)

```
>> logspace(0,1,5)
ans =
            1       1.7783       3.1623       5.6234           10
```

b)

```
>>  logspace(log10(4),log10(64),5)
ans =
            4            8           16           32           64
```

Dwa powyższe przykłady ilustrują:
a) możliwość rozwinięcia szeregu liczb rzeczywistych na przestrzeni jednej dekady wartości w jej skrócie logarytmicznym bez konieczności wykonywania czasochłonnych obliczeń odręcznych oraz
b) możliwość rozwijania szeregów liczb logarytmicznie w oparciu o inną podstawę logartytmów niż dziesiętną.Poniżej analogiczny 'eksperyment' z podstawą logarytmu naturalnego:

```
>> vect=[1:5]
vect =
     1     2     3     4     5

>> e_vect=power(e,vect)
e_vect =
       2.7183       7.3891       20.086       54.598       148.41

>> logspace(log10(e),log10(e*e*e*e*e),5)
ans =
       2.7183       7.3891       20.086       54.598       148.41
```

-

Literatura:

[1] Matlab v.6.5, « Manual »
[2] Matlab v. 6.5, « Image Processing Toolbox »