LAK - ćwiczenia nr 1

Uwagi wstępne

Po uruchomieniu programu `SCILAB` należy ustawić katalog roboczy - polecenie `chdir 'nazwakatalogu'` zmienia katalog na podany, jeśli jest dostępny.

Pierwszą wykonywaną komendą jest:

```
--> diary('nazwisko_imie_album.cwicz1')
```

Ostatnią wykonywaną podczas ćwiczeń komendą będzie:

```
--> diary(0)
```

UWAGA!
Polecenie to przerywa zapis sesji, zatem należy je użyć dopiero po zakończeniu wszystkich działań związanych z realizacją zadań.

Po zakończeniu zajęć uczestnik kopiuje zawartość katalogu roboczego do wskazanego katalogu na komputerze prowadzącego.

Elementy środowiska obliczeniowego Scilab

1. Podstawowym obiektem matematycznym jest macierz. Nie trzeba jej deklarować, zostanie utworzona automatycznie w momencie podstawienia. Na przykład komenda

```
--> a(3,5)=11
```

tworzy macierz o 3 wierszach i 5 kolumnach, przy czym $a_{3,5} = 11$, a pozostałe elementy są zerami (wypróbuj rownież tą komendę ze średnikiem na końcu). Można też zadać macierz w całości, podając ciąg wierszy oddzielonych średnikami, według składni:

```
--> a= [1,2 ; 3,4]
```

Jeśli teraz podstawimy coś na nieistniejący element, to macierz zostanie automatycznie rozszerzona:

```
--> a(3,3)=1
```

Używając podobnej składni możemy utworzyć macierz z podmacierzy jako części składowych. Wypróbuj:

```
--> b=[1,2;3,4]
--> c1=[b,b,b]
--> c2=[b,b;b,b]
```

Istnieją również specjalne funkcje tworzące macierze. Wypróbuj:

```
--> d1=eye(3,3)
--> d2=zeros(3,3)
--> d3=ones(3,3)
```

Ważną rolę pełni funkcja pozwalająca poznać wymiary macierzy. Wypróbuj:

```
--> [nw,nk]=size(b)
--> [nw,nk]=size(c1)
--> x=1; [nw,nk]=size(x)
```

Jak widać, skalar traktowany jest jak macierz o 1 wierszu i 1 kolumnie. Przy okazji zobaczyłeś, że w jednym wierszu można napisać kilka rozkazów odzielonych średnikami.

2. Podstawową operacją wykonywaną na macierzach jest mnożenie według reguł algebry oraz jego pochodne. Oczyść teraz przestrzeń roboczą

```
--> clear
```

a następnie wypróbuj

```
--> A=[1,2;3,4]
--> x=[10;11]
--> b=A*x
--> A2=A*A
```

Operacje pochodne to (dla macierzy kwadratowych) potęgowanie i obliczanie odwrotności

```
--> A3=A^2
--> A1=A^(-1)
```

Dzielenie macierzy może być lewostronne lub prawostronne. Zapisujemy to jako

$$A^{-1} \cdot B \quad \equiv \quad A \setminus B$$

$$B \cdot A^{-1} \quad \equiv \quad B/A$$

Wypróbuj:

```
--> z=A\b
```

Czy otrzymałeś x tak jak powinieneś? Na marginesie: przed chwilą rozwiązałeś układ równań liniowych. Scilab jest w stanie poradzić sobie z wymiarami rzędu wielu tysięcy.

Macierze można też mnożyć przez skalar oraz dodawać (wypróbuj) według znanych reguł. Transpozycję oznaczamy znakiem prim:

```
--> C=A'
```

3. Oprócz działań wykonywanych według reguł algebry, są też działania wykonywane kolejno na wszystkich elementach macierzy. Odróżniają się one kropką. Wypróbuj

```
--> B=A.*A
--> B=A.^3
--> B=A.^(-1)
--> B=A./A
```

Tą samą cechę mają procedury funkcyjne: jeżeli argumentem jest macierz, to wynikiem jest również macierz z elementów poddanych działaniu funkcji. Np.

```
--> B=log(A)
--> B=sin(A)
--> z=sin(x)
```

4. Powróćmy jeszcze do tworzenia macierzy. Wektor wierszowy można utworzyć według składni:

```
pierwszyelement : przyrost : ostatnielement
```

Spróbuj:

```
--> w=1:2:20
--> w=0:0.1:1
--> w=1:10
```

Jak widać, jeśli opuścimy środkowy parametr, to domyślnie zostanie przyjęty przyrost 1.

Użyteczna jest też funkcja zmieniająca kształt macierzy. Najpierw prostuje ona macierz wejściową, ustawiając jej **kolumny** jedna za drugą, a następnie z powstałego wektora wycina kolejne kolumny macierzy wynikowej. Wypróbuj

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

```
--> b=matrix(A,6,1)
```

Z kolei wektory indeksów pozwalają również wycinać prostokątne części macierzy

```
--> B=A([1,2],[1,3])
```

Ten sam wynik można też uzyskać jako

```
--> B=A(:,[1,3])
--> B=A(1:2,1:2:3)
```

czyli dwukropek zastępuje cały zakres indeksu. Poeksperymentuj:

```
--> B=A(1,:)
--> B=A(2,2:3)
--> B=A(2,1:2:3)
```

5. Omówimy teraz najprostsze instrukcje sterujące. Po pierwsze, ciąg instrukcji można wprowadzić nie tylko interakcyjnie, ale również ze zbioru tekstowego komendami

```
exec('nazwazbioru',0)
exec('nazwazbioru',1)
```

lub przez menu. Znaczenie parametru: 0 – wyświetlane są tylko wyniki działania (domyślne), a 1 – wyświetlane są też wykonywane instrukcje.

Funkcje definiuje się według składni

```
function [w1,...,wn] = nazwa(a1,...,am)
........
........
w1=......
w2=......
endfunction
```

gdzie `w1,...,wn` to lista wyników (dowolnego typu), natomiast `a1,...,am` to argumenty, również dowolnego typu. Funkcja taka powinna stanowić zbiór tekstowy `nazwa.sci` i zostać przed użyciem załadowana do przestrzeni roboczej instrukcją `exec` lub przez menu.

Dla ilustracji utwórz zbiór `proba.sci` zawierający:

```
function [x1,x2]=proba(x)
x1=1/x;
x2=x^2;
endfunction
```

a następnie wykonaj polecenia:

```
--> exec('proba.sci')
--> [a,b]=proba(4)
```

Argumenty do funkcji przekazywane są poprzez wartość. Nie ma zmiennych globalnych.

Szczególnie ważną pozycję w każdym programowaniu zajmuje instrukcja pętli. Ma ona składnię:

```
for j=wek
  do .....
end
```

gdzie wektor `wek` może mieć postać:

```
wek = 1:10
wek = [1,7,3]
```

Spróbuj:

```
--> for i=1:10 do
```

```
--> x=i^2;
--> disp(x)
--> end
```

Powtórz to samo, podstawiając zamiast 1:10 wektor [1,7,3]. Przy okazji poznałeś instrukcję `disp(x)` która wyświetla postać publikowalną dowolnego obiektu. Poeksperymentuj z nią.

Podamy jeszcze składnie dwu instrukcji warunkowych, które działają w oczywisty sposób.

```
if wl1 then
  .....
elseif wl2 then
  ....
else
  ....
end
```

z opcjonalnymi częściami `elseif, else` oraz

```
while wl do
  .....
else
  .....
end
```

Wyrażenie logiczne wl ma zwykłą postać, przy czym:
`==` testowanie równości,
`&` operacja 'and'
`|` operacja 'or'
`~` operacja negacji.

6. Opiszemy teraz kilka najbardziej przydatnych instrukcji graficznych. Spróbuj:

```
--> x=0:0.1:5;
--> y=sin(x);
--> plot2d(y')
--> clf
--> plot2d(x',y')
--> clf
```

a następnie

```
--> z=cos(x);
--> plot2d(x',[y',z'])
--> clf
```

Jak widać, komenda `plot2d` rysuje wykresy kolumn drugiego argumentu jako funkcji kolumn pierwszego argumentu. Jeśli `x` brak, jako `x` traktowane są kolejne numery wierszy `y`.

Następna instrukcja to `plotframe([xmin,ymin,xmax,ymax],[nx,mx,ny,my])`.
Spróbuj:

```
--> plotframe([0,0,10,10],[2,5,2,5])
```

Polecenie tworzy układ współrzędnych o odpowiednich wymiarach, przy czym osie podzielone są na odpowiednio `m` przedziałów i `n` podprzedziałów. Z kolei `xfrect(xleft,ytop,width,height)`rysuje wypełniony prostokąt. Spróbuj:

```
--> xfrect(7,5,3,1)
```

Podobnie działa instrukcja `xrects( R , fill )`, tylko że rysuje ona `n` wypełnionych prostokątów jednocześnie. W tym przypadku:

```
R = [ xleft , ......  ]
    [ ytop  , ......  ]
    [ width , ......  ]
    [ height, ......  ]
fill = [ k1, ....,kn ]
```

gdzie każda kolumna `R` reprezentuje jeden prostokąt, natomiast `k1,...,kn` to numery kolorów. Zazwyczaj k=1 oznacza czerń.

Wyczyść okno graficzne, utwórz ponownie ten sam układ współrzędnych i spróbuj:

```
--> R=[3,3,2,1;8,7,1,1]'
--> f=[1,2]
--> xrects(R,f)
--> clf
```

Ostatnia komenda to rysowanie wielu odcinków `xsegs(P1,P2)`, gdzie `P1,P2` zawierają `n` kolumn postaci:

```
P1 = [ xp1, .....  ]  P2 = [ yp1, .....  ]
     [ xk1, .....  ]       [ yk1, .....  ]
```

Każda para kolumn odpowiada odcinkowi pomiędzy punktami:

`[xp1,yp1] <----> [xk1,yk1]` .

Zadania

Zadanie 1

Wykorzystując operatory arytmetyczne `Scilaba` rozwiąż układ równań

$$\begin{cases} x + 2\,y + 3\,z & = 1 \\ 2\,x + 3\,y + z & = 3 \\ 3\,x + y + 2\,z & = 2 \end{cases}$$

Zadanie 2

Dane są sumy następujących szeregów:

$$\begin{aligned} \sum_{k=1}^{\infty} \frac{1}{k^2} &= \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ... \\ &= \frac{\pi^2}{6}, \end{aligned} \tag{1}$$

$$\begin{aligned} \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^2} &= \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + ... \\ &= \frac{\pi^2}{12}. \end{aligned} \tag{2}$$

— W oparciu o sumy skończone n elementów szeregów (1) i (2) skonstruuj funkcje `MojePi1` oraz `MojePi2` o argumencie n, zwracające przybliżenia wartości liczby π.
— Porównaj szybkość zbieżności obu funkcji do wartości π.
— Utwórz wykres zależności zwracanych przez obie funkcje wartości od argumentu n.

Wskazówka: w Scilabie wartość “dokładna” stałej π to `%pi`.