I Prvé kroky v MATLABe

Časť 1

Časť 2

MATLAB (= **MAT**RIX **LAB**ORATORY, teda, maticové laboratórium, pre nás krátko: ML)

je interaktívny systém vyvinutý pre vedecko-technické výpočty. Prvá verzia vznikla v roku 1978 ako software pre podporu vyučovania numerických metód pod vedením C.B. Molera, známeho odborníka v tejto oblasti. Pre jeho priateľov je Cleve Moler – Mr. Matlab.

V 90-tych rokoch (a dodnes) sa inovácie systému MATLAB diali na pôde The MathWorks, Inc., keď pôvodnú fortranovskú verziu nahradila verzia v jazyku C. Vylepšenia MATLABu kráčali ruka v ruke s novými možnosťami infotechnológie a systém si získaval tisíce priaznivcov na univerzitách a výskumných pracoviskách celého sveta. Dnes existujú stovky vysokoškolských učebníc a príručiek, ktoré bezprostredne využívajú MATLAB. Veľa informácií ponúka Web-site http://www.mathworks.com

O všeobecných vlastnostiach a prednostiach systému MATLAB sa tu nejdeme rozširovať. Úlohou tohoto textu je pomôcť absolútnemu začiatočníkovi s prvými krokmi v tomto prostredí. Namiesto opisného výkladu budeme preberať konkrétne situácie, cez ktoré pochopíte, ako systém funguje. Poradíme kde a ako v ňom potrebnú informáciu hľadať. Čitateľ sa bude učiť sám (využívaním Help-u a experimentovaním).

Predpokladáme, že sedíte pred niektorou z verzií radu 5 (resp. 6) a že sú vám známe základné črty práce s dnešnými intuitívnymi prostrediami (v našich zemepisných šírkach pracujúcimi pod Windows). Prvú informáciu o ML dávajú príkazy odoslané z promptu (znak ">>") z okna príkazov (command window):

```
» intro
```

alebo

```
» demo
```

(za znak promptu stačí napísat "intro", alebo "demo" a odklepnúť). Predstavu o možnostiach systému získame príkazom: "tour". Pravda, obsah "tour" osloví len toho, kto z matematiky už dosť veľa vie. Pre poslucháčov druhého ročníka je mnohé z toho, čo prezentuje systém jednoducho nové a neznáme, ale, nevadí. Začneme základmi.

1 Práca s výrazmi (so skalárnymi premennými a jednoduchými funkciami)

Skutočnosť, že MATLAB je interaktívny systém znamená, že zadaním príkazu a odklepnutím vyvoláme realizáciu príkazu a výsledok sa objaví na obrazovke. Začnime výpočtami, ktoré obvykle spájame s prácou na bežnej vedeckej kalkulačke. Postupovať budeme komentovaním série príkladov, na ktorých získate predstavy o tom, ako riešiť bežné úlohy.

Aby sa odozva vášho prostredia MATLABu zhodovala s tou, ktorú tu uvádzame, je potrebné cez Menu **`File/Preferences/General`** vyznačiť voľbu **`Short`** (označenú ako default). Tým sme zvolili formát, v ktorom systém uvádza výsledky operácií na obrazovku.

Začnime uvažovaním o výrazoch typu: 3.12^4*5.87/98.4

Predpokladáme, že čitateľ vie, čo je priorita operácií a má aktívne skúsenosti s nejakým programovacím jazykom. Preto nie je treba strácať čas s tým, ako editovať číslo (napr., že nesmie obsahovať medzeru, alebo, že desatinnou je bodka a nie čiarka...:-)). Ale povedzme, že sme si nie istí, ako je to s prioritou

aritmetických operácií v MATLABe. Čo je pre systém výraz 4^3*2, vari 128 (=64*2), alebo 4096 (=4^6)?

Pre prvé kroky v MATLABe, ako v každom inom prostredí (či krajine ...) sú dôležité dve veci:

1) vedieť kde a ako hľadať informáciu
2) neokúňať sa, ... a veriť v "silu" experimentovania.

Začnime experimentovaním. Ak do promptu naťukáme 4^3*2 , tak odozvou je

```
ans =
     128
```

"ans" je meno premennej, ktorá nesie hodnotu posledne vykonanej operácie z promptu (ans – ako answer, dovtípili sme sa ...). Vidíme, že operácia umocnenia má väčšiu prioritu ako operácia násobenia. Jednoduchým experimentovaním s výrazmi, ktorých výsledok dopredu vieme, ľahko zistíme odpovede na podobné otázky.

Druhou cestou je hľadať informáciu v samotnom prostredí. Dnešné verzie systému sú vybavené pružným Help on-line, ktorý je prístupný viacerými spôsobmi. Teraz hovorme o najjednoduchšej ceste: kľúčové slovo "priorita", v angličtine označuje slovo "precedence", preto do promptu naťukajme:

```
» help precedence
```

Systém dáva tento výpis:

```
PRECEDENCE Operator Precedence in MATLAB

MATLAB has the following precedence for the built-in operators when
       evaluating expressions (from highest to lowest, with operators
       in the same class having equal precedence):
    1. transpose (.'), power (.^), complex conjugate transpose ('),
       matrix power (^)
    2. unary plus (+), unary minus (-), logical negation (~)
    3. multiplication (.*), right division (./), left division (.\),
       matrix multiplication (*), matrix right division (/),
       matrix left division (\)
    4. addition (+), subtraction (-), .....
```

(tu sme výpis odstrihli...) Vidíme, že všetkému nerozumieme, ale umocňovanie – "power" je hneď pod bodom 1, kým "multiplication" v bode 3.

Samozrejme, zátvorky musíme použiť, keď potrebujeme zmeniť poradie vykonania operácií. Posledná rada ohľadne priority operácií (platí aj pre ďalšie situácie, napr. maticové výrazy):

AK STE SI NIE ISTÍ, POUŽITE ZÁTVORKY – VÝRAZY V ZÁTVORKÁCH SA VYHODNOCUJÚ PREDNOSTNE.

Nie je chyba, keď bude vo vašom výraze jeden pár, alebo hoci aj dva páry, zátvoriek naviac! K výrazom, ktoré obsahujú čísla, poznamenajme už len to, že číslo môže byť zadané aj v semilogaritmickom tvare: (mantissa **.**10^exponent), ktorý je na kalkulačkách označovaný ako "scientific notation". Napr. 144 sa môže editovať takto: 1.44e2 (zase, samozrejme, bez medzier medzi prvým a posledným znakom).

A teraz hovorme o výrazoch, ktoré obsahujú aj premenné. Niektoré mená (mená premenných) sú už v systéme použité, medzi nimi napr. `pi` je známe 3.141592653 ... Pri našej voľbe **`Short format`** na obrazovke vidíme 3.1416. Berieme teda na vedomie, že jedna vec je vnútorná reprezentácia čísla v systéme, ktorá je pevná, nemenná a druhá vec je formát, v ktorom je číslo zobrazené na obrazovke (typ formátu si volíme sami).

Aj premenné "i", "j" sú v systéme definované a majú rovnakú hodnotu, a to hodnotu imaginárnej jednotky, avšak my ich často budeme používať ako indexy (predefinujeme ich tak ľahko, že si to ani nevšimneme). Pre zaujímavosť, overme:

```
» sqrt(-1) - i
ans =
     0
```

Na poslednom príkaze (sqrt(-1) – i) vidíme, že MATLAB pozná funkciu sqrt(.), dokonca pripúšťa pre ňu aj záporný argument (informácia o funkciách príde neskôr). Možno ste si uvedomili, že pred chvíľou sme symbol "e" použili na označenie exponentu (pre základ 10) vo vedeckej notácii reálneho čísla. Číslo, ktoré znamená základ prirodzeného logaritmu nemá v systéme MATLAB svoje meno, jednoducho je to hodnota exponenciálnej funkcie **`exp(.)`** v argumente 1, t.j. **`exp(1)`**.

Pokročme ďalej. Do promptu naeditujme napr.:

```
» v = g*m*(1 - exp(-c*t/m))/c
```

je to vzťah pre rýchlosť (v čase *t*) parašutistu, padajúceho voľným pádom, keď *m* je jeho hmotnosť a *c* je koeficient odporu (vzorec je získaný istým zjednodušením skutočného problému). Po odklepnutí tohoto príkazu systém reaguje takto:

```
» v = g*m*(1 - exp(-c*t/m))/c
??? Undefined function or variable 'g'.
```

Preto priraďme premennej *g* hodnotu gravitačného zrýchlenia (použijeme známu aproximáciu: 9.81)

```
» g = 9.81
g =
    9.8100
```

zopakujme:

```
» v = g*m*(1 - exp(-c*t/m))/c
??? Undefined function or variable 'm'.
```

Pochopili sme, že MATLAB pracuje s premennými v bežnom zmysle a na vyčíslenie výrazu je potrebné zadať hodnoty všetkých premenných. Ako vidíme, systém spracováva výraz zľava doprava, ako očakávame... Naraz (v jednom riadku promptu) pre *m*, *c* aj *t*, priradenie urobíme oddelením priraďovacích príkazov čiarkou:

```
» m = 70, c = 10, t = 15
m =
    70
c =
    10
t =
    15
```

Opätovné vyvolanie vzťahu dostaneme rolovaním (vo ver. 5 neviditeľnej) postupnosti už predtým zadaných príkazov, čo dosiahneme klávesou "šipka hore" ("arrow up") a, samozrejme, odklepnutím Enter:

```
» v = g*m*(1 - exp(-c*t/m))/c
v =
    60.6137
```

Namiesto pamäťových registrov (memory keys) vedeckej kalkulačky máme k dispozícii prakticky ľubovoľne veľa premenných, ktoré môžu predstavovať buď skalárne veličiny (ako v našom príklade), alebo vektory, či matice (uvidíme neskôr). Takto za 20 minút práce môžeme do pracovného priestoru ML napchať veľa premenných a stratiť prehľad o tom, aké sme už definovali. Zoznam aktuálnych premenných zobrazí príkaz "who":

```
» who
Your variables are:
ans       g         t
c         m         v
```

Identifikátor "ans", ako sme už povedali, slúži na uchovanie (ostatného) výsledku (pokiaľ sme ho nepriradili ako hodnotu nejakej premennej). Podstatné je upozorniť, že MATLAB je "case sensitive", čo znamená, že premenná g je rôzna od G. Vyskúšajte: pom nie je Pom, či POM. Ako ľahko zistíte experimentovaním, meno premennej môže mať až 31 znakov. Syntax identifikátora sa riadi známymi pravidlami, ktoré poznáme z Cčka, či Pascalu (musí začať písmenom a môže obsahovať aj podtrhovník).

K detailom základných akcií (vo verzii 5) sa dostaneme cez: ZákladnéMenuML – Help / HelpWindow. Help ver. 6 je štrukturovaný inak, je však vybavený HelpNavigatorom a nepotrebuje komentár. Venujeme sa preto HelpWindow ver. 5, ktorý ponúka desiatky tematických okruhov. Nám zatiaľ stačí prvých päť:

`matlab/general`	všeobecné príkazy
`matlab/ops`	operátory a špeciálne znaky
`matlab/lang`	konštrukcie programovacieho jazyka MATLABu
`matlab/elmat`	základné matice a maticové operácie
`matlab/elfun`	elementárne matematické funkcie (= funs)

Dvojklik na riadok vyvolá základnú informáciu o príslušnom tematickom okruhu. Ďalší dvojklik na konkrétny príkaz vedie k detailnej informácii. Avšak k helpu na konkrétny príkaz, či funkciu, vedie aj rýchlejšia cesta rovno z promptu, ak vieme meno príkazu, resp. operátora, či funkcie. Povedzme, že sme si

všimli existenciu príkazu "clc", ale teraz by sme si radi oživili informáciu o ňom, lebo napr. nevieme ... je to "clear" premenných, alebo "clear" okna? Stačí do promptu napísať:

```
» help clc
```

a odozva prostredia dáva:

```
CLC    Clear command window.

   CLC clears the command window and homes the cursor.

   See also HOME.
```

Vyskúšajte napr.:

```
help help
help whos
help clear
help lookfor
```

Všimnite si, že informácie končia pokynom "See also ...", čo zrejme pomáha nájsť to, čo chceme. Napr. "help help" informuje aj o príkaze "lookfor" (asi v strede výpisu) a v poslednom riadku vyzýva užívateľa, aby si pozrel "lookfor". Pre začiatočníka je "lookfor" zvlášť užitočný, čo manuály ML ilustrujú na probléme hľadania funkcie na invertovanie matice. Vskutku, vyskúšajte: "help inverse" nám nepomôže, ale "lookfor inverse" nájde (voľne povedané) všetko, čo nejako súvisí s kľúčovým slovom "inverse" (pokiaľ to leží v adresároch, kam vedie "path"). Mechanizmus týchto vecí sa objasní neskôr.

Všetky matematické funkcie bežnej vedeckej kalkulačky nájdeme v HelpWindow pod heslom:

`matlab/elfun` (elementárne matematické funkcie)

Prístup k nahliadnutiu čo všetko téma "matlab/elfun" obsahuje, dovoľuje aj príkaz "help elfun". Čitateľ má čo robiť, je tam veľa materiálu. Neprehliadnite, že logaritmická funkcia s prirodzeným základom má meno **`log(.)`**, kým so základom 10 má meno **`log10(.)`**! Napr.:

```
» log(exp(3))
ans =
     3
```

resp.:

```
» log10(10^5)
ans =
     5
```

Možno zo zvedavosti ste odklepli "log(0)" a zistili, že systém vypísal varovanie a **`ans = -Inf`**. Help na "Inf" vysvetľuje zmysel tohoto symbolu. Vyskúšajte príkaz "1/0", resp. "0/0" a cez Help na "NaN" sa dozviete, že NaN je označenie pre niečo, čomu nemôžeme priradiť hodnotu (napr. operácii: Inf – Inf sa nedá (vo všeobecnosti) priradiť hodnota (môže ňou byť všeličo ... čo, to závisí od kontextu).

Zaiste ste si všimli, že pod Menu: Help – je okrem ponuky Help Window aj Help Desk (HTML). Z mnohých MATLAB Topics, ktoré prezentuje Help Desk, upozorňujeme na tri (v ľavej časti panelu):

- Getting Started
- MATLAB Functions
- Online manuals (PDF).

Getting Started je manuál pre prvé kroky v MATLABe a jeho text sprevádza mnoho príkladov. Je to naozaj perfektný text, odladňovaný už od verzie 4 (r.1992). Snažíme sa v duchu "Getting Started" písať riadky tejto príručky. Komu tento text nestačí, má možnosť siahnuť po "Getting Started" (uvedomujeme si, že práca s MATLABom prirodzene zlepšuje našu angličtinu).

MATLAB Functions Help Desk ponúka v podstate to isté, čo dávajú tematické okruhy Help Window, ale pre začiatočníka je podstatné, že uvádzajú viac príkladov ("Examples") ako Help Window.

Online manuals vo formáte PDF poskytujú kompletný súbor manuálov celého systému. Pre začiatok stačí ten prvý: Getting Started, avšak, ako sme povedali, ten je pohodlne prístupný cez úvodné okno HelpDesk.

Zdá sa, že "kalkulačkový mód" MATLABu je už za nami. Presvedčme sa:

Úloha: Nájdite dĺžku prepony c trojuholníka ABC, ak $a = 4.6$ cm, $b = 3.5$ cm a uhol $\gamma = 65°$. (Vezmite na zreteľ, že argumenty trigonometrických funkcií musia byť v radiánoch. Pre kontrolu: $c = 4.45$ cm.)

Nakoniec malé potešenie. Editujte do príkazového riadku (a odklepnite):

```
» load earth, image(X); colormap(map), axis image, shg
```

2 Vektory, tabelovanie a grafy jednoduchých funkcií

Matica je základný dátový typ systému MATLAB. Aj skalárne veličiny, aj vektory, sú pre ML matice. Skalár (teda číslo, hoci komplexné...) je matica s rozmermi [1, 1]. Namiesto známeho "A je typu m n", (resp. A má rozmery: m n), sa MATLAB "vyjadruje" takto: size (A) = [m, n].
Presvedčme sa! Editujme v prompte, napr. "size(1+i)" , t.j. pýtajme sa na rozmer komplexného skaláru:

```
» size(1+i)
ans =
     1     1
```

Vektory sú matice, ktoré majú jeden riadok (= riadkové vektory, size = [1, n]), resp. sú to matice, ktoré majú jeden stĺpec (= stĺpcové vektory, size = [m, 1]). Prechod od riadkových k stĺpcovým, resp. naopak, zariadi operátor transponovania. Riadkový vektor zadáme celkom prirodzene; zložky vektora oddeľujeme medzerou, alebo čiarkou. Zadajme ho napr. pod menom rv.

```
» rv = [1 2 3]
rv =
     1     2     3
```

Transponovaním získame stĺpcový vektor, nech má meno sv:

```
» sv = rv'
sv =
     1
     2
     3
```

Ako vidíme, je to jednoduché. Rozmyslime si, čo budú matice `rv*sv` , resp. `sv*rv`? Operácia "*" je operáciou násobenia matíc (ale aj skalárov, veď so skalárom pracuje MATLAB ako s maticou).

Nech mat1 = rv*sv; rozmer rv*sv bude: [1, 3].[3, 1], teda mat1 má rozmer [1, 1], je to skalár.
Nech mat2 = sv*rv; rozmer sv*rv bude: [3, 1].[1, 3], teda mat2 má rozmer [3, 3]. Zadajme:

```
» mat1 = rv*sv, mat2 = sv*rv
mat1 =
    14
mat2 =
     1     2     3
     2     4     6
     3     6     9
```

Editovanie stĺpcového vektora je možné, samozrejme, aj bez transponovania takto:

```
» cv = [11; 12; 13]
cv =
    11
    12
    13
```

teda oddeľovačom zložiek je bodkočiarka. Bodkočiarka v matici oddeľuje jej jednotlivé riadky, ako uvidíme neskôr. Berme na vedomie, že matice budeme editovať v zhode s tým, ako editujeme vektory.
Zopakujme: vektory v MATLABe sú matice, ktorých jeden rozmer sa rovná 1.

Vymenovanie zložiek vektora je prvou možnosťou ako do prostredia dostať vektor. Ďalšou cestou je použiť operátor "**:**" (dvojbodka – colon, pozri: `help colon`). Povedzme, že chceme vytvoriť vektor [0 .2 .4 .6 .8 1], teda vektor, ktorého zložky sú rovnomerne rozložené čísla s krokom h (teraz $h = .2$). Označme ho x (x je identifikátor uvažovaného vektora). Dosiahneme to príkazom:

```
» x = 0:.2:1
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Všimnime si, že rovnaký účinok má príkaz:

```
» x2 = 0:.2:1.1
x2 =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Opýtajme sa na rozmer vektora x:

```
» size(x)
ans =
         1     6
```

Všimnime si, že argumenty funkcií (teraz išlo o funkciu "size") píšeme do okrúhlych zátvoriek, kým prvky vektorov do hranatých. Jednotlivé zložky vektora sú prístupné takto (index prvej zložky sa rovná 1):

```
» x(3)
ans =
       0.4000
```

a predefinovanie hodnoty zložky vektora, napr. tretej, dosiahneme priraďovacím príkazom:

```
» x(3) = .4111
x =
       0    0.2000    0.4111    0.6000    0.8000    1.0000
```

Pri predefinovaní jedinej súradnice, MATLAB zobrazuje celý vektor, pokiaľ mu v tom nezabránime bodkočiarkou. Teraz smerujeme k objasneniu práce *memory managera*. Ten má na starosti automatické dimenzovanie zadávaných vektorov a matíc. Najprv si všimnime, že príkaz

```
» x(12) = 0.9876
x =
  Columns 1 through 7
       0    0.2000    0.4000    0.6000    0.8000    1.0000    0

  Columns 8 through 12
       0         0         0         0    0.9876
```

spôsobí automatickú zmenu dimenzie! Nie je treba sa o tieto veci starať tak, ako sme zvyknutí z Basicu, Pascalu, či C-čka. Mimochodom, na zistenie dimenzie vektora môžeme použiť funkciu "length". Neskôr, keď budeme hovoriť o efektívnej práci so systémom, ukážeme, ako pomôcť memory managerovi, aby sa trápil so zmenami dimenzovania čo najmenej (ušetríme čas, výpočty budú efektívne).

Naozaj základná (a pôvabná) črta prostredia je "vektorizovanie" funkcií. Využijeme ju pri tabelovaní funkčných hodnôt, resp. pri vytváraní grafov funkcií. V MATLABe sú funkcie implementované tak, aby pracovali aj s vektorovým argumentom, a to takto:

Vráťme sa k pôvodnému vektoru x, čo dosiahneme priradením:

```
» x = x2
x =
       0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Všimnime si, že memory manager predefinoval dimenziu vektora x späť na pôvodnú, o čom sa presvedčíte, keď sa opýtate na hodnotu `x(12)`, alebo na `size(x)`, ale teraz sa s tým nezdržujme, chceme vektorizovať funkciu "sqrt( )" ! Máme vektor x, ktorý má 6 zložiek (resp. 6 súradníc, ako chcete). Priradenie:

```
» y = sqrt(x)
y  =
       0    0.4472    0.6325    0.7746    0.8944    1.0000
```

vytvorí vektor y, ktorého i-tá súradnica je odmocnina z i-tej súradnice vektora x. Zopakujme tento obrat s inou funkciou, napr. "round" (aplikovanou na vektor x):

```
» z = round(x)
z =
       0       0       0       1       1       1
```

Je zrejmé, že toto "vektorizovanie" funkcií je veľmi vhodná vlastnosť, pokiaľ ide o tabelovanie funkcií, alebo o získanie grafov funkcií (v MATLABe na získanie grafu je treba vektor funkčných hodnôt). Vezmime napr. funkciu "sin( )", trebárs na intervale [0, 2*pi]. Potrebné akcie:

1) vytvorenie vektora x rovnomerne rozložených argumentov vo zvolenom intervale [0, 2*pi]
2) vytvorenie vektora y, ktorého zložky sú hodnoty funkcie sínus v týchto argumentoch.

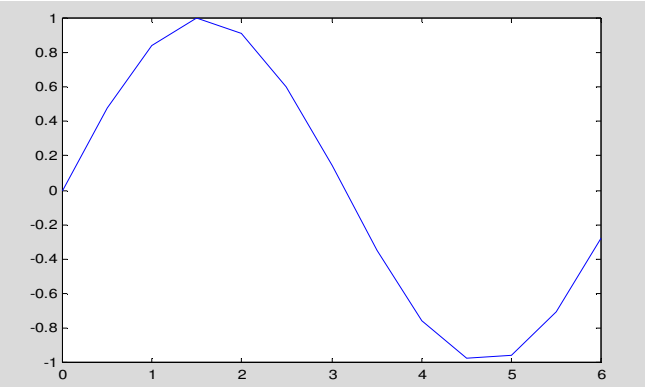
Vektory `x,y` vytvoríme príkazom (výpis tu neuvádzame):

```
» x = 0:.5:2*pi, y = sin(x);
```

Tabuľka hodnôt bude oveľa prehľadnejšia vo vertikálnej podobe. Vektory x, y chceme dostať vedľa seba, a preto pôjde o maticu. Označme ju "mtabsin". Jej prvý stĺpec bude vektor x' a druhý stĺpec vektor y' (všimnime si, že oddeľovačom stĺpcov je čiarka):

```
» mtabsin = [x', y']
```

```
mtabsin =
         0         0
    0.5000    0.4794
    1.0000    0.8415
    1.5000    0.9975
    2.0000    0.9093
    2.5000    0.5985
    3.0000    0.1411
    3.5000   -0.3508
    4.0000   -0.7568
    4.5000   -0.9775
    5.0000   -0.9589
    5.5000   -0.7055
    6.0000   -0.2794
```



Odozva príkazu plot (prvý pokus ...).

Je to jednoduché, ale priveľmi ... uvedomme si, že nemôžme pohodlne meniť krok nezávislej premennej a nevidíme záhlavie tabuľky, aj keď meno "mtabsin" nám signalizuje o čo ide. Zakrátko sa naučíme robiť oveľa lepšie tabelovanie.

Ako získame graf? Aplikujme príkaz "plot" na dvojicu vektorov x, y; "plot" spojí bod `[x(i),y(i)]` s bodom `[x(i+1),y(i+1)]` úsečkou (pre všetky i, samozrejme), a tak vykreslí graf:

```
» plot(x, y)
```

Obrázok sa grafu funkcie sínus podobá len zhruba ..., chyba je v tom, že sme nepoužili dostatočne veľa bodov intervalu [0, 2.pi]. Použime desať razy viac bodov (obrázok teraz vidíte na svojej obrazovke...).

```
» x = 0:.05:2*pi; y = sin(x); plot(x, y)
```

Potlačenie výpisov vektorov x, y (za príkazmi sú bodkočiarky...) je na mieste; zakrátko sa naučíte dávať si pozor... :-). Použitie príkazu "help plot" dáva kompletnú informáciu o ďalších parametroch (a teda ďalších možnostiach) tohoto príkazu.

Úloha: Znázornite priebeh paraboly $y = x^2$ na intervale [-2, 2].
Prvý pokus by dopadol asi takto:

```
» x = -2:0.05:2; y = x^2;
??? Error using ==> ^
Matrix must be square.
```

V čom je problém? Je v nepatričnom vektorizovaní funkcie "druhá mocnina". Namiesto x^2 máme editovať x.^2 (neprehliadnite bodku!), pretože v MATLABe príkaz: x^2 znamená násobenie matice x maticou x, a to je možné len pre štvorcové matice (v našom prípade je však x riadkový vektor). Tieto veci sa objasnia v článku 4 pri výklade rozdielu medzi maticovými operáciami a operáciami typu "array". To, čo zamýšľame, dosiahneme príkazmi:

```
» x = -2:0.05:2;
» y = x.^2;
» plot(x, y)
```

Úloha: Znázornite graf polynómu $y = x^3 - 6.x^2 + 11.x - 6$ na intervale [0, 4].

Nakoniec poznamenávame, že namiesto príkazu "a:1:b" možno písať "a:b", lebo v takom prípade sa pre krok h berie (default) hodnota 1. Ak napr. $a = 2.35$, tak oba nasledujúce priraďovacie príkazy dávajú to isté (na výpis vektora u nie je priestor, preto sme ho potlačili):

```
» a = 2.35; u = a:1:9; v = a:9
v =
    2.35    3.35    4.35    5.35    6.35    7.35    8.35
```

Ešte informáciu o "prázdnom" vektore, ktorý získame príkazom "w = [ ]". Rozmer w sa rovná [0 0] (overte!). Použitím [ ] môžme "vypustiť" jednu, resp. viac súradníc vektora, čo ilustrujú príkazy:

```
» u(2) = [ ], v(3:5) = [ ]
u =
    2.35    4.35    5.35    6.35    7.35    8.35
v =
    2.35    3.35    7.35    8.35
```

3 M-súbory typu: script

Uvažujme nad zlepšením tabelovania funkcie sínus (na intervale [*a*, *b*] s krokom *h*), ktoré sme v článku 2 dosiahli definovaním matice "mtabsin". Prajeme si, aby tabuľka mala záhlavie, aby výpis argumentu x neobsahoval zbytočné nuly (v ľavom stĺpci matice) a aby prípadná zmena hodnôt *a*, *b*, *h* bola jednoduchá. Toto všetko je možné dosiahnuť sériou príkazov, ktoré "patria k sebe" (ich zmyslom je vytvoriť tabuľku). Aj keď je pravda, že by sme tie príkazy, na viackrát – časť po časti, do riadku promptu naeditovali, modifikovanie takej kolekcie príkazov by bolo veľmi ťažkopádne. MATLAB preto ponúka možnosť napísať celú postupnosť príkazov do súboru, ktorému hovoríme – *script*.

Script je (po formálnej stránke) ASCII súbor. Po obsahovej stránke je to postupnosť príkazov zapísaná do súboru pod nejakým menom.V našom prípade to bude "TabSin.m" (script musí mať príponu .m). Preto sa scriptom hovorí M-súbory a podľa prípony sú zvonku ľahko identifikovateľné. Napísaním mena súboru (bez prípony) do promptu a odklepnutím, sa vyvolá vykonanie postupnosti príkazov scriptu. Pravda, za predpokladu, že ML vie nájsť náš M-súbor, teda že ten je uložený buď v aktuálnom adresári, alebo v takom, ku ktorému "vedie cesta" (info: help dir, help path – resp. pozrite záverečnú úlohu tohoto článku).

Príkazy scriptu môžu používať premenné definované v pracovnom priestore ML (t.j. tie, ktoré sme predtým definovali v prompte) a, pochopiteľne, tiež tie, ktoré sú definované v samotnom scripte. Script napíšeme v editore MATLABu, ktorý vyvoláme cez Menu: `File/New/m-file`

Aby bol script pre nás (aj po nejakom čase) zrozumiteľný, je vhodné písať komentár. Riadky, ktoré začínajú symbolom "%", interpreter MATLABu ignoruje. Teraz objasnime príkazy nášho scriptu:

Ako vidíme, prvé dva riadky tvoria komentár.
Tretí, štvrtý a piaty riadok – nič nové, sú to tri priraďovacie príkazy (s potlačením výpisu).
Šiesty riadok definuje riadkový vektor x, siedmy riadok je riadkový vektor sínusov zložiek vektora x.
Ďalší riadok definuje riadkový vektor k, ktorého zložky budú indexovať riadky tabuľky.
Deviaty riadok definuje maticu s tromi riadkami (v matici je bodkočiarka oddeľovač riadkov).

Riadky 10:12 – pozrite: "help disp".
Príkaz v poslednom riadku je príkaz, ktorý poznáme z jazyka C. Umožňuje riadenie formátovania výpisu (je použitý vo vektorizovanej verzii – pozrite: "help fprintf").

```
% nas prvy script: tabelovanie hodnot sinusu na
%                  intervale [a, b] s krokom h
a = 0;
b = 2*pi;
h = 0.5;
x = a:h:b;
y = sin(x);
k = 1:length(x);
mat = [k; x; y];    % toto je akoby horizontalna tabulka
disp(' ')           % fprintf z nej vyrobi vertikalnu
disp('  k    x(k)    sin(x(k))')
disp(' ------------------------')
fprintf(' %2.0f    %4.2f     %7.4f \n', mat)
```

Riadky komentára môžeme vyvolať (priamo do okna command window) príkazom: `help TabSin`. To je užitočné, pretože aj keď meno m-súboru sa snažíme vhodne voliť, po čase nám obsah scriptu v hlave "vybledne" (zvlášť, keď medzitým napíšeme niekoľko iných scriptov...).

Zrejme si uvedomujeme, že premenné scriptu a pracovného priestoru sú spoločné a to má svoje výhody aj nevýhody. Preto je dobré v komentári scriptu uvádzať mená tých premenných, ktoré sú v scripte definované (stávajú sa totiž spoločnými a predefinujú hodnoty premenných pracovného priestoru rovnakého mena). A ako sme už povedali, komentár je ľahko dostupný (`help menoscriptu`).

Úloha: Napíšme script, ktorý do aktuálnych ciest ML (do systémovej premennej "path") pridá adresár "mydir" a nastaví "mydir" ako aktuálny adresár. Meno pre súbor vhodne zvoľte (nutne s príponou `.m`).

```
% path to: c:\MATLABR11\mydir
%      cd  c:\MATLABR11\mydir
path(path,'c:\MATLABR11\mydir');
cd c:\MATLABR11\mydir;
```

4 Matice, maticové operácie a operácie s prvkami poľa

4.1 Definovanie matíc

Matice malých rozmerov definujeme vypísaním ich prvkov po riadkoch, pričom oddeľovačmi riadkov je bodkočiarka (prvky riadku oddeľuje medzera, alebo čiarka, veď riadok chápeme ako vektor):

```
» a = [1 2; 3 4]
a =
    1    2
    3    4
```

Maticu je možné editovať v prompte aj tak, že riadky oddeľujeme klávesou Enter a pomocou klávesy "medzera" dosiahneme už pri zadávaní známy (prehľadný) tvar matice. Mini-ukážka:

```
» b = [11  22
       33  44]
b =
    11  22
    33  44
```

Prvkami matíc (samozrejme aj vektorov) môžu byť aj výrazy (ukážme na vektore):

```
» c = [sqrt(5)  (1+2+4)/3]
c =
    2.2361  2.3333
```

MATLAB umožňuje "prilepiť" vektor c k matici b ako jej riadok, alebo stĺpec. Keď ako riadok, treba ho oddeliť oddeľovačom riadkov:

```
» d = [ b; c ]
d =
   11.0000   22.0000
   33.0000   44.0000
    2.2361    2.3333
```

a teraz ako stĺpec. Stĺpec dostaneme transponovaním (riadkového) vektora c:

```
» e = [ b  c']
e =
   11.0000   22.0000   2.2361
   33.0000   44.0000   2.3333
```

Možno nás napadlo vyskúšať:

```
» A = [ a   b   c' ]
A =
    1.0000   2.0000   11.0000   22.0000   2.2361
    3.0000   4.0000   33.0000   44.0000   2.3333
```

alebo, prečo nie, napríklad:

```
» B = [ a  b;  b' a ]
B =
    1   2   11   22
    3   4   33   44
   11  33    1    2
   22  44    3    4
```

Ako vidíme, maticu možno vytvoriť po blokoch z matíc, už predtým definovaných. A naopak, z "veľkej" matice možno ľahko vytvárať nové matice, ako jej submatice (t.j. submatice tej veľkej):

```
» C = B( 1:3, 2:4 )
C =
    2   11   22
    4   33   44
   33    1    2
```

Operátor "**:**" je veľmi silný "nástroj" ML (pozri: `help colon`). Z článku 2 vieme, že príkaz "**a:h:b**" vytvorí vektor hodnôt (počnúc bodom **a** s krokom **h**). Teraz však nejde o interval $[a, b]$ reálnych čísel, ale o indexovú množinu pre prvky matíc. Vyskúšajme, čo je odozva príkazu: **"1:3"**

```
» 1:3
ans =
     1   2   3
```

teda vektor (veď opýtajte sa na `size(ans)`). A čo je napr.: **`1:2:8`**

```
» 1:2:8
ans =
     1    3    5    7
```

tentoraz je krokom hodnota 2 (v notácii: "a : h : b" je teraz a = 1, h = 2, b = 8). Preto:

```
» D = B( 1:2:4, 2:2:4 )
D =
     2    22
    33     2
```

Zrejme sme nadšení... flexibilita MATLABu pri práci s maticami je omračujúca (a veľa zaujímavého je zrejme len pred nami!) Konvenciu, ktorá sa spája s operátorom "**:**" ukazuje príkaz:

```
» E = B(:, 2)
E =
     2
     4
    33
    44
```

teda B(**:**, 2) predstavuje druhý stĺpec matice B, kým napr. B(3, **:**) jej tretí riadok

```
» F = B(3, :)
F =
    11    33    1    2
```

Vyskúšajte: B(: , 2:2:4), potom B(: , 4:-1:1), resp. experimentujte! Nakoniec ešte takáto zaujímavosť: pre našu maticu a = [1 2 ; 3 4] použime príkaz "a(**:**)":

```
» a =[1 2; 3 4]; a(:)
ans =
     1
     3
     2
     4
```

Teda príkaz: "**`mat(:)`**" dáva stĺpcový vektor, vytvorený stĺpcami matice napísaných "pod seba". Naopak, zo stĺpca možno vytvoriť pôvodnú maticu (detaily získate cez `help reshape`):

```
» reshape(u, 2, 2)
ans =
     1    2
     3    4
```

Operátor "**:**" môže vystupovať aj na ľavej strane priraďovacieho príkazu! Sledujte (a rozmyslite si ...)

```
» z = 5:5:50, y = [13 23 33 43]
z =
     5    10    15    20    25    30    35    40    45    50
y =
    13    23    33    43

» z(1:2:8) = y
z =
    13    10    23    20    33    30    43    40    45    50
```

Vyskúšajme, ako bude vektor **z** vyzerať po priradení, v ktorom operátor "**:**" je na oboch stranách priraďovacieho príkazu:

```
» z = 5:5:50, y = [13 23 33 43], z(1:2:6) = y(1:3)
z =
     5    10    15    20    25    30    35    40    45    50
y =
    13    23    33    43
z =
    13    10    23    20    33    30    35    40    45    50
```

Tipnite si, odozvu príkazu (a potom svoj tip overte!):

```
» z([1 3 5 6])=[ ]
```

Nie je možné ukázať všetko, avšak na to, aby ste k ML pocítili "kladný vzťah", je toho už asi dosť ...;-)

4.2 Maticové operácie

Začneme s unárnymi operáciami: transponovaním a invertovaním. Operáciu transponovania sme už mali, síce pre vektory, ale to je jedno, rovnaká syntax platí pre matice. Invertovanie matice je definované, ako vieme, len pre regulárne matice. Inverziu matice A získame funkciou "inv", teda: `inv(A)`.

Binárne operácie začneme operáciou sčítania a odčítania. Tie sú definované len ak ide o matice rovnakých rozmerov. V opačnom prípade systém hlási chybu: `Matrix dimensions must agree`. Presne vzaté, okrem prípadu, keď jednou z matíc je skalár. Skalár je možné pričítať k ľubovoľnej matici (spomeňte si na demo MATLABu, tam je prezentovaná operácia: a + 2, kde a je vektor).

Operácia násobenia sa už objavila v článku 2, keď sme uvažovali `sv*rv`. Z lineárnej algebry vieme, že súčin matíc je definovaný len v prípade, keď počet stĺpcov prvej matice sa rovná počtu riadkov druhej. V opačnom prípade systém hlási chybu: `Inner matrix dimensions must agree`. Znovu je výnimkou prípad, keď jedným z činiteľov je skalár – vyskúšajte!

Operácia "delenia matíc" je v systéme označovaná ako "matrix division". Aj keď termín "delenie matice maticou" sa v slovenskej terminológii nepoužíva, sem-tam ho možno použijeme (pre jednoduchosť vyjadrovania). Dôležitejšie než terminológia je upozornenie, že ak A, B sú matice, tak

A\B (ľavé delenie matice B maticou A) je matica X, ktorá je riešením maticovej rovnice A*X = B.

B/A (pravé delenie matice B maticou A) je matica X, ktorá je riešením rovnice X*A = B.

Teoreticky vzaté, môžeme povedať, že ak A je štvorcová, tak A\B = inv(A)*B, kým B/A = B*inv(A). Avšak MATLAB – ako výpočtový systém – nerieši uvedené maticové rovnice použitím inverzií (ide o skutočnosti z numerickej lineárnej algebry, ktorým sa budeme venovať v kap. 3 numerických výpočtov).

Ak matica A nie je štvorcová, tak sústava A*X = B je buď *preurčená*, alebo *podurčená*. Čo považujeme za riešenie sústavy v takom prípade nebýva témou základného kurzu lineárnej algebry. Vezmime na vedomie, že MATLAB nám dáva odpoveď aj v situáciách, v ktorých to neočakávame. Má totiž v sebe zakomponované všetky dodnes známe maticové algoritmy. Na ukážku vyskúšajte (nedajte sa zneistiť, o čo vlastne ide, sa dozviete v kap. 3 numerických výpočtov):

```
» A = rand(3,2), B = rand(3,4), X = A\B, A*X - B
```

Zmieňme sa o maticových výrazoch obsahujúcich mocniny matice. Ak A je matica, tak jej tretiu mocninu A*A*A môžeme dostať príkazom A^3. Avšak upozorňujeme, že ML *k*-tú mocninu matice nepočíta ako A*A* ... *A (*k*-činiteľov), o čom sa presvedčíme premyslene navrhnutým príkladom – opäť ide o otázku z numerickej lineárnej algebry. Pre zaujímavosť, exponent mocniny matice nemusí byť prirodzené číslo. Aká je odozva na príkaz a^0.5, alebo a^2.4 ? Vyskúšajme:

```
» s = [13  20; 5  8],  t = s^0.5   % matica t je "odmocninou" matice s
s =
    13   20
     5    8
t =
    3.0000    4.0000
    1.0000    2.0000
```

Očakávame, že **t*t** sa rovná matici **s** (druhá mocnina matice [3 4; 1 2] sa rovná matici [13 20; 5 8], avšak výpis matice **t** sa nezhoduje s vnútornou reprezentáciou matice **t** !). Čo je **t*t - s** ?

```
» t*t  - s
ans =
  1.0e-014 *
   -0.3553   -0.3553
    0         0
```

výpis – `ans` – treba čítať tak, že skalár 1.0e-014 násobí všetky prvky matice, ktorá nasleduje, teda výsledok operácie **t*t - s** je "prakticky" nulová matica. Pre zaujímavosť, sledujte odozvu na príkazy:

```
» w = sqrtm(s), [norm(t*t-s), norm(w*w-s)] %pozri help sqrtm (resp. cast 4.5)
w =
    3.0000    4.0000
    1.0000    2.0000
ans =
    1.0e-014 *[0.5024,  0.4580]
```

4.3 Generovanie matíc funkciami MATLABu

V systéme ML máme k dispozícii funkcie, ktorými ľahko vytvoríme špeciálne matice. Vyskúšajte napr.:

zeros, **zeros(3),** alebo **zeros(5,3),** resp. **zeros(3,4),**
ones, **ones(4),** resp. **ones(3,6),**
eye, **eye(3)** a tiež napr. **eye(3,5),**
rand, **rand(3,5)** – prvky matice sú "náhodné čísla" z intervalu [0, 1],
randn, **randn(4,3)** – teraz sú prvkami matice náhodné čísla pochádzajúce z normálneho rozdelenia.

Ak A je matica, tak tieto príkazy fungujú aj takto (ilustrujeme len na prvom): `zeros(size(A))`, vyskúšajte a dovtípite sa ... Ak u je vektor (riadkový, či stĺpcový, je to jedno...), tak príkaz: `diag(u)` generuje diagonálnu maticu, pričom vektor u tvorí jej diagonálu. Vyskúšajte!

Repertoár špeciálnych matíc zabudovaných do MATLABu ako "built-in" je naozaj široký. Viac nájdete v `HelpWindow/Elementary matrices and matrix manipulation`, podnadpis: `specialized matrices`. Na ukážku sa zmieňme len o týchto troch (všetky tri sú štvorcové):

magic(n) generuje populárne *magické štvorce*
pascal(n) generuje *Pascalove matice*
vander(x) generuje *Vandermondovu maticu*, známu z interpolácie; vektor x je vektor uzlov interpolácie.

Ako uvidíme neskôr, M-funkciami si dokážeme vytvoriť akúkoľvek maticu, ktorej štruktúru dokážeme algoritmizovať. Avšak aj z promptu sa dá dosiahnuť neočakávane veľa. Pre ilustráciu ukážeme definovanie trojdiagonálnej matice, ktorá má na hlavnej diagonále súradnice vektora q a na bezprostredne susedných diagonálach prvky vektora r. Vektory q, resp. r vytvorme pre jednoduchosť takto:

```
» q = 10:3:22,  r = 1:2:7
q =
    10    13    16    19    22
r =
     1     3     5     7

» H = diag(q,0) + diag(r,-1) + diag(r,1) % diag(matica,cislo)pozri:help diag
H =
    10     1     0     0     0
     1    13     3     0     0
     0     3    16     5     0
     0     0     5    19     7
     0     0     0     7    22
```

Úloha. Nech W = zeros(10, 10). Definujte maticu Z tak, aby sa od matice W líšila v prvom a poslednom riadku a tiež na vedľajšej diagonále, pričom na týchto miestach má mať matica Z jednotky. Inými slovami: pole samých núl (= matica W) treba modifikovať tak, aby jednotky vykreslili písmeno Z.

4.4 Operácie s prvkami poľa ("array" operations)

Najprv objasnime nadpis tohto článku. Okrem maticových operácií MATLAB disponuje aj operáciami, či funkciami, ktoré "operujú" na jednotlivých prvkoch matice. Význam týchto operácií zakrátko oceníme pri vytváraní grafov funkcií. V originálnom manuáli sa tieto operácie popisujú slovami – "array" operations – (aby sa odlíšili od štandardných maticových operácií).

Najjednoduchšie bude začať príkladom. V nasledujúcom riadku si pripomeňme maticu **a** a definujme novú maticu A príkazom "A = a.^2" (neprehliadnite bodku):

```
» a = [1 2; 3 4]; A = a.^2
A =
     1     4
     9    16
```

Vidíme, že prvky A sú mocninami odpovedajúcich prvkov matice **a**. Matica A vznikla operáciou umocnenia, ale nie matice **a** ako matice, ale umocnenia jednotlivých prvkov poľa **a**. To isté sme mohli dosiahnuť príkazom: **a.*a** Ak matice **a, b** majú rovnaký rozmer, tak ich možno násobiť "po prvkoch" (čo

nemá nič spoločné s násobením matíc). Oproti **a.*a** je zmena v tom, že úlohu druhého činiteľa hrá matica **b**. Pripomeňme si maticu **b** a realizujme násobenie matíc **a, b** "po prvkoch":

```
» b,  a.*b
b =
    11     22
    33     44
ans =
    11     44
    99    176
```

Operácie **.+** (bodka plus), resp. **.–** (bodka mínus) MATLAB nepozná, nie sú potrebné, ničím sa totiž nelíšia od maticových operácií sčítania, resp. odčítania, lebo veď tie sa definujú "po prvkoch".

Pretože operáciu " **.*** " máme už za sebou, sú na rade operácie " **./** " a " **.** ". Sú jednoduché a fungujú tak, ako očakávame, vyskúšajte si to!

Zaujímavejšou je " **.^** ", pretože (na rozdiel od situácie a.^2, ktorou sme začínali) funguje aj s maticou v exponente! Všimnime si:

```
» c = 2.^a
c =
     2      4
     8     16

» c.^a
ans =
     2     16
   512  65536
```

4.5 Matice ako argumenty funkcií (maticové funkcie)

Už sme si uvedomili, že v MATLABe je mnoho funkcií definovaných tak, že pripúšťajú za svoj argument aj maticu. V rôznych manuáloch sa im hovorí maticové funkcie. Niektoré operujú

na prvkoch matice, resp.
na jej stĺpcoch, alebo
na celej matici ako celku.

Prvým príkladom maticovej funkcie, ktorá operuje na prvkoch, bola funkcia *sínus* (článok 2). Ako ste si mohli všimnúť v `HelpWindow-matlab/elfun`, prakticky všetky tam uvedené elementárne funkcie fungujú takým spôsobom, ako funkcia *sínus* (to, čo uvádzame je len preklad výpisu help-u pod heslom – SIN):

"SIN(X) je matica, ktorej prvky sú hodnoty funkcie sínus v prvkoch matice X."

Hodnotou týchto funkcií je opäť matica (rovnakého rozmeru ako tá, ktorá predstavuje argument uvažovanej funkcie). Samozrejme, nielen funkcie uvedené v "elfun" (teda elementárne) sú príkladmi maticových funkcií operujúcich na prvkoch matice, takými sú aj mnohé iné.

Ďalšie maticové funkcie operujú na stĺpcoch matice. Sú nimi napríklad funkcie, ktoré sa používajú pri analýze dát ako sú: **max, min, sum, prod, mean, median, ...** a mnohé ďalšie. Hodnotou funkcie **max(a)** je vektor, ktorého zložky sú maximá zo stĺpcov matice **a**. Analogicky pracujú aj tie ostatné funkcie. Napríklad:

```
» a = [1 2; 3 4]; vektmax = max(a), vektsum = sum(a), vektmean = mean(a)
vektmax =
          3     4
vektsum =
          4     6
vektmean=
          2     3
```

Nakoniec hovorme o maticových funkciách, ktoré operujú na matici "ako celku". Najprv niekoľko takých, ktorých hodnoty sú skalárne veličiny (t.j. čísla):

det(A)	determinant matice	**rank(A)**	hodnosť matice
trace(A)	stopa matice	**norm(A)**	norma matice

(Stopa matice sa definuje ako súčet diagonálnych prvkov matice.)

Príklady takých maticových funkcií, ktorých hodnoty sú vektory:

size(A) rozmery matice
poly(A) vektor koeficientov charakteristického polynómu matice
eig(A) vektor vlastných čísel matice

Hodnoty nasledujúcich maticových funkcií sú matice, t.j. aj argument funkcie a aj jej hodnota sú matice. Ako prvú uveďme funkciu `rref()`, dávajúcu známy redukovaný stupňovitý tvar (row reduced echelon form) matice. Vyskúšajte:

```
» c = rand(3,5), rref(c)
```

Ako druhú spomeňme `sqrtm( )`. Hodnotou `sqrtm(a)` je taká matica `w`, že `w*w = a`. Uvedomme si rozdiel medzi `sqrt( )`, ktorá operuje na prvkoch matice a `sqrtm( )`, ktorá operuje nad celou maticou:

```
ma = [9 4; 16 25]; sqrt(ma), sqrtm(ma)
ans =
     3     2
     4     5
ans =
    2.8146    0.5191
    2.0764    4.8910
```

Ako tretiu uveďme funkciu "eig". Ak ju voláme jednoduchým spôsobom:

```
» vc = eig(a)
vc =
   -0.3723
    5.3723
```

tak odozvou je vektor vlastných čísel matice (podrobnejšie v kap. 4 numerických výpočtov). Avšak, ak syntax príkazu je:

```
» [S, D] = eig(a)
S =
   -0.8246   -0.4160
    0.5658   -0.9094
D =
   -0.3723         0
         0    5.3723
```

tak hodnotou funkcie "eig" sú dve matice: prvou maticou je S, ktorej stĺpce sú vlastné vektory matice **a**, druhou je matica D, diagonálna matica obsahujúca (prislúchajúce) vlastné čísla matice **a**. Maticových funkcií je v systéme MATLAB naozaj veľa (pozri: `matlab/matfun`) a môžete si byť istí, že systém obsahuje všetky, ktoré majú v maticových výpočtoch nejaký význam. Preberať ich budeme v jednotlivých kapitolách numerických výpočtov.

Poznamenávame, že kombinovaním týchto "built-in" funkcií ľahko vytvárame ďalšie. Napr. funkcia, ktorej hodnotou je súčet všetkých prvkov matice "ma", je funkcia:

```
» sum( sum(ma) )
ans =
    54
```

Zrejme jednoduché kombinácie, ako vidíme, ľahko zvládneme v jednom riadku promptu. Zložitejšie budeme písať do M-funkcií (článok 7, časť 2), a tak zväčšovať možnosti MATLABu pre naše konkrétne potreby. Všimnite si ešte raz syntax volanie funkcie "eig":

```
» [S, D] = eig(a)
```

Takáto syntax sa vyžaduje vtedy, keď hodnotou funkcie je viac matíc, či vektorov, resp. skalárov. Napr. pri pozornom prečítaní "help max" sa dozvieme, že ak funkciu "max" voláme týmto spôsobom, tak okrem maximálnej zložky vektora dostaneme aj *index* maximálnej zložky:

```
» vx = [2 6 4 8 7 4 0 8]; [maximum, index] = max(vx)
maximum =
     8
index =
     4
```

Ako vidíme, tým indexom je index *prvého* výskytu maxima.

5 Vektorizovanie, práca s funkciami a ich grafmi

Vektorizovanie sa v MATLABe uplatňuje neustále (v článku o M-funkciách vysvetlíme, prečo vždy, ak je to možné, vyhýbame sa cyklom "for"). Vektorizovanie ilustrujme na príklade konverzie veľkosti uhla v stupňoch, na veľkosť v radiánoch. Ak x predstavuje veľkosť uhla v stupňoch a y veľkosť (toho istého uhla) v radiánoch, tak zrejme je vzťah medzi x a y lineárny, t.j. $y = a + b\,x$, naviac, zrejme $a = 0$. Na zistenie parametra b stačí jedna dvojica odpovedajúcich si hodnôt. Pre priamy uhol máme: $x = 180$, $y =$ pi, teda $b =$ pi /180. Konverziu možno predstaviť vertikálnou tabeláciou, ktorú predstavuje matica "konv":

```
» x = (0:30:180)'; y = (pi/180)*x; konv = [x, y]
konv =
         0         0
   30.0000    0.5236
   60.0000    1.0472
   90.0000    1.5708
  120.0000    2.0944
  150.0000    2.6180
  180.0000    3.1416
```

Nuly v ľavom stĺpci, má na svedomí formát "Short", ktorý je vhodný pre hodnoty uhla v radiánoch, ale nie je vhodný pre (nami volené, celočíselné) veľkosti uhla v stupňoch. Pomôže funkcia "fprintf" (objavila sa v článku 3). Využijeme fakt, že je vektorizovaná. Ak chceme získať vertikálnu tabeláciu je treba funkcii "fprintf" odovzdať tabuľku v horizontálnej podobe, a preto treba maticu "konv" transponovať!

```
» fprintf('   %3.0f    %5.4f \n', konv')
          0    0.0000
         30    0.5236
         60    1.0472
         90    1.5708
        120    2.0944
        150    2.6180
        180    3.1416
```

Úloha: Zapíše tieto príkazy do scriptu "stup_rad.m" tak, aby hodnoty v stupňoch boli vytvorené vektorom "`a:h:b`", ktorý bude možné jednoducho pre-editovávať. Vytvorte analogické matice predstavujúce konverzie zo stupňov Celzia na stupne Fahrenheita, resp. naopak. Známy vzťah (zapísaný symbolicky) má tvar: C = (5/9)*(F – 32).

Úloha: Nech x sa rovná vektoru [0, 0.07, 0.14, 0.21, ... , 6.09], n = length(x). Využívajúc príkazy článku 2 (t.j. bez použitia cyklu "for") definujte vektor

1) u tak, aby `u(i)= 0.5(x(i)+ x(i+1)), i=1,2,..,n-1, u(n)= x(n).`
2) v tak, aby `v(i)= (x(i-1)+ 2.x(i)+ x(i+1)), i=2,3,..,n-1, v(1)=x(1), v(n)=x(n)`
3) y tak, aby `y(i)= sum(x(j): 1≤j≤i), i=1,2,..,n` (návod: `help cumsum`)
4) z tak, aby `z(i)= sin(x(i))+ cos(x(i+1)),i=1,..,n-1, z(n)= sin(x(n))+ cos(x(1))`

Ukážeme jedno z možných riešení pre definovanie vektora u. Pre väčšiu čitateľnosť, konštrukciu ilustrujeme pre x = [0, 0.2, 0.4, ... 1.0]:

```
» x = 0:0.2:1, n = length(x); u = [ 0.5*(x(1:n-1) + x(2:n))   x(n) ]
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
u =
    0.1000    0.3000    0.5000    0.7000    0.9000    1.0000
```

Je možné, že toto krátke riešenie nie je dobre čitateľné. Preto ešte raz – krok po kroku:

```
» x = 0:0.2:1, n = length(x);
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
» y = x(2:n)
y =
    0.2000    0.4000    0.6000    0.8000    1.0000
» u = 0.5*(x(1:n-1) + y)
u =
    0.1000    0.3000    0.5000    0.7000    0.9000
```

```
» u = [u  x(n)]
u =
    0.1000    0.3000    0.5000    0.7000    0.9000    1.0000
```

Analogicky nájdete riešenia ostatných prípadov. Teraz uvažujme nad nasledujúcimi príkazmi:

```
x = 0:5:40, a = [2 4 8 9], u = x(a)
x =
     0     5    10    15    20    25    30    35    40
a =
     2     4     8     9
u =
     5    15    35    40
```

ako vidíme, príkaz "**x(a)**" je ekvivalentný príkazu:

```
for i = 1:length(a)
  u(i) = x(a(i))
end
```

Všimnime si, že funguje aj toto:

```
» y = zeros(1,10)
y =
     0     0     0     0     0     0     0     0     0     0
» y(1:4) = x(a)
y =
     5    15    35    40     0     0     0     0     0     0
```

Úloha. Nech `w = 10:10:90`. Rozmyslite si, ako bude vyzerať vektor `w` po príkaze:

```
w(1:2:6) = ones(1,3)
```

resp. aké prvky bude mať vektor `w = 10:10:90` po príkaze:

```
w(2:3:9) = zeros(1,3)
```

Venujme sa teraz maticiam. V článku 4 sme videli, ako veľa sa dá dosiahnuť operátorom "**:**". Nech W je matica definovaná vymenovaním svojich prvkov:

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

Vytvorme maticu A ako submaticu matice W tvorenú druhým až štvrtým riadkom a stĺpcami 2, 4, 6, 7. To dosiahneme príkazom:

```
» A = W(2:4,[2 4 6 7])
A =
     8     6     8     2
     4     8     4     6
     6     4     6     8
```

Výmenu riadkov, napr. druhého a tretieho, zariadi príkaz:

```
» A = A([1 3 2], :)
A =
     8     6     8     2
     6     4     6     8
     4     8     4     6
```

Na matici A budeme ilustrovať skutočnosť, že operáciu násobenia matice vektorom možno chápať ako kombinovanie stĺpcov matice s váhami, ktoré predstavujú zložky vektora x. Definujme vektory **a1**, **a2**, **a3, a4** ako stĺpce matice A, teda

$$\mathbf{a1} = A(:, 1),\quad \mathbf{a2} = A(:, 2),\quad \mathbf{a3} = A(:, 3),\quad \text{resp. } \mathbf{a4} = A(:, 4).$$

Ak vektor x má štyri zložky, je možné uvažovať o vektore A.x, ktorý chápeme ako obraz vektora x pri zobrazení, ktoré predstavuje matica A. Operácia A.x je pre MATLAB triviálna operácia (pokiaľ dimenzia vektora x je zhodná s počtom stĺpcov matice A). To, na čo chceme upozorniť, je fakt, že vektor A.x možno chápať ako kombináciu stĺpcov A s váhami x(i), teda, že platí:

$$A.x = x(1).\mathbf{a1} + x(2).\mathbf{a2} + x(3).\mathbf{a3} + x(4).\mathbf{a4}$$

Symbolikou MATLABu:

$$A.x = x(1).A(:,1) + x(2).A(:,2) + x(3).A(:,3) + x(4).A(:,4)$$

Ilustrujme túto skutočnosť na stĺpcovom vektore x = [1; -2; 1; -1].

```
» a1 = A(:,1); a2 = A(:,2); a3 = A(:,3); a4 = A(:,4); x = [1; -2; 1; -1];
» w = x(1)*a1 + x(2)*a2 + x(3)*a3 + x(4)*a4 - A*x; w'
 ans =
      0    0    0    0
```

Pre úsporu priestoru sme nechali namiesto stĺpcového vektora w vypísať jeho transpozíciu. Chápanie operácie A.x ako kombinácie stĺpcov matice A budeme v ďalšom často využívať.

Teraz sa venujme funkciám (matematickým funkciám) a ich grafom.V článku 2 sme vytváranie grafov začali definovaním vektora argumentov, využívajúc operátor "**:**" v príkaze "a:h:b". Niekedy je výhodné použiť príkaz "linspace(a, b, n)", ktorý vytvorí *rovnomernú* sieť *n* uzlov, ktorá začína v bode a, končí v bode b, pričom krok siete je $h = (b - a)/(n - 1)$. To znamená, že:

funkcia "linspace(a,b,n)" je ekvivalentná funkcii "a:(b-a)/(n-1):b"

Máme teda dve možnosti a volíme podľa toho, či chceme predpísať veľkosť kroku *h*, alebo počet bodov siete (do počtu bodov siete sa počítajú aj krajné body a, b). Vyskúšajme:

```
» x = 3:0.2:4, u = linspace(3, 4, 6)
x =
    3.0000    3.2000    3.4000    3.6000    3.8000    4.0000
u =
    3.0000    3.2000    3.4000    3.6000    3.8000    4.0000
```

Ak ale vezmeme napr. a = 0, b = 10, n = 1001, tak ak x = linspace(a, b, n), y = 0:0.01:10, odozva funkcie sum(x == y) nás možno prekvapí (odpoveď nájdete v kapitole 1, článok 4: aritmetické operácie v ML).

Ako sme videli v článku 2, vektorizovanie niektorých (napr. trigonometrických) fukcií je automatické, a preto ich grafy nám nerobia problémy. Na ilustráciu vezmime funkciu *f*, $f(x) = \exp(-x/1.4).\cos(6x)$ na intervale [0, 10]. Graf získame príkazmi:

```
» x = linspace(0, 10, 1001);
» y = exp(-x/1.4).*cos(6*x);  % neprehliadnite typ súčinu ".*"
» plot(x, y)
```

Teraz ukážeme vektorizovanie polynómov a racionálnych lomených funkcií. Povedzme, že chceme znázorniť grafy funkcií:

f 1, $f1(x) = 3.2x^2 + 0.78x - 2.64$ na intervale [0, 1]

f 2, $f2(x) = 1/(25x^2 + 1)$ na intervale [-1, 1]

f 3, $f3(x) = (3.2x^2 + 0.78x - 2.64)/(x^3 + x^2 - 2x + 3.8)$ na [-2, 2]

a nakoniec funkcia definovaná "vidličkou":

f 4, $f4(x) = -x^3 + x^2 + 2x$, ak $x \in [-3, 0]$, resp. $f5(x) = x.\cos(5.x)$, ak $x \in [0, 6]$.

Nasledujú riešenia:

f 1:

```
» x = linspace(0,1,51);
» y = 3.2*x.^2 + 0.78*x - 2.64;
» plot(x,y), shg            % shg: show graph window
```

f 2:

```
» x = linspace(-1,1,401);
» y = 1./(25*x.^2 + 1);     % je možné aj y = ones(1,length(x))./(25*x.^2+1)
» close all                 % tento príkaz zavrie všetky grafické okná
» plot(x,y,'g.-')           % help plot je okamžite poruke...
```

f 3:

```
» x = linspace(-2,2,201);
» cit = 3.2*x.^2 + 0.78*x - 2.64;
» men = x.^3 + x.^2 - 2*x + 3.8;
» y = cit./men;
» plot(x,y,'k-'), shg
```

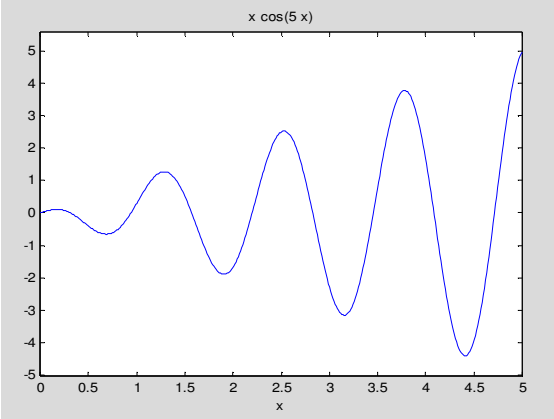
f4:
```
» x1 = linspace(-3, 0, 301);  y1 = -x1.^3 + x1.^2 + 2*x1;
» x2 = linspace( 0, 6, 601);  y2 = x2.*cos(5*x2);
» x = [x1 x2];
» y = [y1 y2];
» close all, plot(x,y), grid on
```

Teraz ukážme, že takéto matematické funkcie možno definovať príkazom "inline" ako tzv. *inline objekty* a pohodlne s nimi pracovať (napr. odovzdávať ich ako argumenty iným funkciám, ako uvidíme neskôr). Napr. príkazom

```
f = inline('x*cos(5*x)'), y = f(1)
f =
     Inline function:          y =
     f(x) = x*cos(5*x)              0.2837
```

definujeme funkciu a celkom prirodzene získame jej hodnoty. Použitím príkazu "ezplot" (easy plot) dostaneme jej graf na intervale [a, b], napr. na [0, 5] takto:

```
ezplot(f, [0, 5])
```



Príkazom "inline" je možné definovať aj funkciu dvoch a viac premenných. Napr.

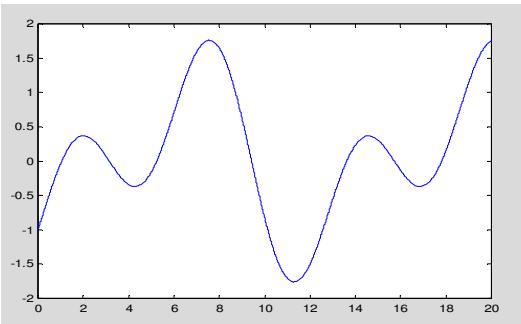
```
g = inline('sin(a*t + b)')
g =
     Inline function:
     g(a,b,t) = sin(a*t + b)
```

Všimnime si, že systém sám identifikoval premenné a zvolil ich poradie. Ak nám také poradie nevyhovuje, napr. chceme, aby premenné *a*, *b* vystupovali skôr ako parametre, tak zvolíme syntax s vyznačením poradia. Prvou premennou nech je *t* a až za ňou parametre *a*, *b*:

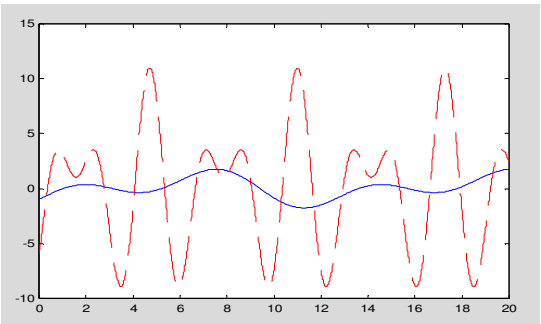
```
h = inline('sin(a*t + b)','t','a','b'), u = h(3,2,1), v = h(1,2,3)
h =
     Inline function:
     h(t,a,b) = sin(a*t + b)

u =               v =
     0.6570            -0.9589
```

Do jedného grafického okna je možné znázorniť grafy viacerých funkcií. To je možné dosiahnuť napr. tak, že príkazu "plot" odovzdáme viac funkcií (začneme jednou):

```
» x = linspace(0, 20, 2001);  y = sin(x) - cos(x/2);
» plot(x, y), shg
```



Graf funkcie y = sin(x) – cos(x/2)



Čiarkovane graf z = 5.sin(3x) – 6.cos(2x)

A teraz definujme ďalšiu funkciu:

```
» z = 5*sin(3*x) - 6*cos(2*x); plot(x,y,'b-', x,z,'r--'), shg
```

Odozva je na obrázku vpravo. Inú možnosť ako zobraziť viac funkcií do jedného grafického okna ponúka príkaz "hold on". V našej situácii by sme ho použili takto (začíname od začiatku):

```
» y = sin(x) - cos(x/2);  plot(x, y), shg
» hold on, plot(x, z,'r-'), shg
```

Pokiaľ nezadáme príkaz "hold off", dovtedy MATLAB kreslí grafy do toho istého okna. Všimnime si, že pridaním grafu funkcie $z = z(x)$ sa automaticky zmenila mierka na osi y. Mechanizmus automierky prispôsobil mierku osi y tak, aby bolo možné spolu s grafom funkcie $y(x)$ znázorniť aj graf funkcie $z(x)$. Je možno prekvapujúce, ako zmena mierky osi y ovplyvnila tvar grafu funkcie $y = y(x)$.

Sú však situácie, ktoré automierka nemôže zvládnuť. Takou je prípad funkcie, ktorá je na uvažovanom intervale neohraničená. Pokúste sa vytvoriť graf funkcie tangens na intervale [-pi/2, 5.pi/2].

Použitím "plot" ste pravdepodobne nedosiahli to, čo ste očakávali. Príkazy a komentár k vytváraniu grafu funkcie tangens sme napísali do textového súboru "graf_tan.txt" prostredníctvom príkazu "diary" (pozrite "help diary"). Príkaz "diary" umožňuje automatické zapisovanie obsahu príkazového riadku do textového súboru. Súbor "graf_tan.txt" otvorte NotePad-om a veľkosť okna upravte podľa prvého riadku (len preto, aby sa jeho obsah dal pohodlne čítať). Potom umiestnite príkazové okno ML tak, aby sa obe okná "nevyrušovali". Editujte príkaz za príkazom zo súboru "graf_tan.txt" do ML (môžete použiť aj copy/paste) a sledujte odozvy systému. Porovnavajte reakcie MATLABu s komentárom, ktorý sprevádza príkazy v súbore "graf_tan.txt".

Úloha: Zopakujte postup, ktorým ste získali graf funkcie tangens na znázornenie grafu funkcie cotangens na intervale [0, 3*pi]. Prácu začnite príkazom "diary on" (alebo "diary menosuboru", teda napr. "diary pokus"). Po skončení práce zadaním príkazu "diary off" uzavriete súbor "diary" (resp. súbor "pokus"). Potom si súbor pozrite, napr. v NotePad-e, je to iste poučné. Pre tých, ktorí potrebujú pomoc:

```
» x = linspace(0, pi, 301); y = cot(x); plot(x,y, x+pi,y, x+2*pi,y),
  axis([0,3*pi,-20,20]), grid on, shg
```

Poznamenávame, že hore spomenutý príkaz "ezplot" znázorní graf tangensu pohodlnejšie. Vyskúšajte:

```
ezplot('tan',[-10, 10])
```

Pravdepodobne ste si uvedomili, že v príkaze "inline" sme nemuseli definitorický výraz napísať vo vektorizovanej podobe. Všimnite si však, že potom argumentom nemôže byť vektor. Napr.

```
f = inline('1/(1 + x^2)'), f([1, 2, 3])
f = Inline function:
    f(x) = 1/(1 + x^2)
??? Error using ==> inline/subsref
Error in inline expression ==> 1/(1 + x^2)
??? Error using ==> ^
Matrix must be square.
```

Avšak funkciu je možné definovať aj vektorovo a v takom prípade môžeme chcieť napr. g([1, 2, 3]):

```
g = inline('1./(1 + x.^2)'), g([1, 2, 3])
g =
    Inline function:
    g(x) = 1./(1 + x.^2)
ans =
    0.5000    0.2000    0.1000
```

Dokonca môžeme funkciou "vectorize" pôvodne nevektorizovanú funkciu f vektorizovať:

```
f = vectorize(f)
f =
    Inline function:
    f(x) = 1./(1 + x.^2)
```

Na záver priznajme, že v našom výklade (od začiatku až doteraz) sme často používali termín "príkaz", hoci išlo o funkciu, resp. o použitie operátora MATLABu. Slovo funkcia môže znamenať funkciu tak, ako ju rozumieme v matematike, alebo tak, ako ju rozumieme v programovacích jazykoch, resp. v samotnom MATLABe. Pritom či už ide o príkazy, alebo o funkcie, editujeme ich do príkazového riadku. Veríme, že čitateľa táto nedôslednosť nepoplietla. Poznamenávame, že typickými príkazmi sú napr. "clear", "save", "load", či "format". Častejšie pracujeme s funkciami, napr. "linspace" je funkcia, ktorá má dva, resp. tri vstupné argumenty a jej hodnotou je, ako vieme, vektor.

Tieto riadky uzatvárajú prvú časť "Prvých krokov v MATLABe". Teraz sa čitateľ môže pustiť do čítania prvých kapitol numerických výpočtov. Predpokladáme, že s druhou časťou "Prvých krokov" sa budete oboznamovať priebežne popri štúdiu ďalších kapitol textu.