6 Elementy programovania v MATLABe

Týmto článkom začína druhá časť "Prvých krokov v MATLABe". Predpokladáme, že čitateľ sa do istej miery s programovaním v nejakom prostredí už zapodieval, a tomu je prispôsobený aj výklad. Základným údajovým typom systému MATLAB je matica, čo sa odráža na konštrukciách programovania: argumentami logických, či relačných operátorov sú matice. Navyše, systém ponúka špecifické funkcie (napr. "any", či "all"), ktoré umožňujú pružnú prácu a zvyšujú čitateľnosť M-funkcií. Na mnohých univerzitách sveta sa kurzy *Computational Science* realizujú v prostredí MATLABu.

6.1 Relačné a logické operátory

Medzi základné konštrukcie programovania patria podmienené príkazy a príkazy cyklu. V MATLABe sa im hovorí *riadiace* príkazy. Typickými predstaviteľmi sú príkaz "if" a cyklus "while". Ich tvary sú:

```
if  výraz                    while výraz
    príkazy                      príkazy
end                          end
```

kde *výraz* je taký výraz, v ktorom vystupujú relačné a logické operátory. Preto elementy programovania začneme relačnými a logickými operátormi. Dostaneme sa k nim príkazom `help ops` (ops = operators) a odozvou sú dve obrazovky výpisu všetkých operátorov. Hneď za aritmetickými nasledujú relačné a logické. Z výpisov vidíme, pod akým označením ich v systéme nájdeme. Tu sú relačné operátory:

eq	- Equal ==	ne	- Not equal ~=
lt	- Less than <	gt	- Greater than >
le	- Less than or equal <=	ge	- Greater than or equal >=

Ak chceme informáciu na operátor "==", resp. "<", dostaneme ju rýchle cez "help eq", resp. "help lt".

Druhá možnosť je obrátiť sa na HelpDesk, ponuku MatlabFunctions, "by Index" a potom pod písmenom "R" nájdeme "relational operators". Text je ilustrovaný aj príkladmi a cez "copy/paste" sme získali tieto dve ilustrácie:

```
prvá  situácia: » X = 5; X >= [1 2 3; 4 5 6; 7 8 10]
druhá situácia: » X = 5*ones(3,3); X >= [1 2 3; 4 5 6; 7 8 10]
```

Odozva oboch príkazov je tá istá matica 0/1-tiek (rozmyslite si, prečo!). Takýmto spôsobom získate informáciu o všetkých relačných operátoroch. Naviac, ako sme už viac razy povedali, je možná cesta experimentovania: rozmyslite si situáciu, v ktorej viete výsledok a opýtajte sa systému ...

Ako vidíme, operandami relačných operátorov sú:

- skalár s maticou (ako v prvej situácii) – odozvou je matica rovnakého rozmeru,
- dve matice rovnakých rozmerov (ako v druhej situácii) – odozvou je matica rovnakého rozmeru.

S relačnými operátormi sa stretneme hneď v kap.1, pri uvažovaní o aritmetike MATLABu, keď sa budeme zaujímať o rozloženie bezprostredných susedov rôznych bodov číselnej osi. Výpis obrazovky nám úplnú informáciu neposkytuje, ako ilustruje príkaz:.

```
» [ 1/3, eps,  1/3 - eps,  1/3 - eps == 1/3 ]
ans =
      0.333333333333333    2.22044604925031e-016    0.333333333333333    0
```

Tu je zoznam logických operátorov:

and	- Logical AND &	or	- Logical OR |
not	- Logical NOT ~	xor	- Logical EXCLUSIVE OR
any	- True if any element of vector is nonzero		
all	- True if all elements of vector are nonzero		

Pri priorite logických operácií, musíme byť zvlášť opatrní a smelo používať zátvorky, resp. overiť si na príklade, ako systém zamýšľanú kombináciu logických operátorov vyhodnocuje. Verzia MATLABu 5.3 (a nižšie) výraz "a | b & c" vyhodnocujú ako výraz "(a | b) & c", kým verzia 6 ako "a | (b & c). Ak má byť M-súbor prenositeľný medzi rôznymi verziami MATLABu, je nutné použiť zátvorky, a tak dosiahnuť to, čo zamýšľame.

Prvé štyri logické operátory sú známe (z logiky, resp. z programovacích jazykov), posledné dve logické funkcie "any" a "all" krátko ilustrujme (najprv si, pravda, prečítajte odozvu na "help any"):

```
» a = [1 0 0 0 1; 0 1 0 1 1; 1 1 0 1 1]
a =
     1     0     0     0     1
     0     1     0     1     1
     1     1     0     1     1

» any(a(:,1))
ans =
     1

» any(a(:,3))
ans =
     0

» any(a)
ans =
     1     1     0     1     1
```

ako vidíme, "any" operuje na matici "po stĺpcoch". Analogicky "all" (najprv ale: "help all"):

```
» all(a)
ans =
     0     0     0     0     1
```

Rozmyslite si a vyskúšajte, čo dávajú príkazy:

```
all(all(A == B))
any(any(A == B))
```

Zaujímavou logickou funkciou je "find" (pozri "help find"). Na ilustráciu vezmime:

```
» x = [ -1  0  5  2  0  -5  0  0  10]; ix = find(x), u = x(ix)
ix =
     1     3     4     6     9
u =
    -1     5     2    -5    10
```

Úloha. Nech y = [-1, 2, -3, -5, 1, -2, 4, 4, 2, -1, 3] a nech n = length(y). Použime príkaz `find` a nájdime počet znamienkových zmien v zložkách vektora y.

```
» sucin = y(1:n-1).*y(2:n), poc_znam_zmien = length(find(sucin<0))
sucin =
    -2    -6    15    -5    -2    -8    16     8    -2    -3
poc_znam_zmien =
     7
```

Ako uvádza Help, je možné aj takéto volanie funkcie "find":

```
» a = [-2  1 -1 -3; 1 -3  4  0], [i,j] = find(a > 1)
a =
    -2     1    -1    -3
     1    -3     4     0

i =           j =
     2           3
```

6.2 Riadiace príkazy

MATLAB má štyri riadiace príkazy: "if ", resp. "if ( ) elseif ( )", cykly "for", "while" a príkaz "switch". Help Window uvádza materiál v `matlab/lang`. Listovanie všetkého čo sem patrí, dáva v prompte príkaz: "`help lang`". Veľa príkladov uvádza tak HelpWindow, ako aj HelpDesk.
Začneme s najjednoduchšou formou príkazu "if". Keď používame tieto konštrukcie v scripte, píšeme obyčajne (pre prehľadnosť) príkazy tela konštrukcie pod sebou (premenné považujeme zatiaľ za skalárne):

```
if a > b
   t = a;
   a = b;
   b = t;
end
```

(Príkaz vymenil obsahy premenných, ak premenná **a** mala väčšiu hodnotu ako **b**.) Ak chceme písať príkaz "if" do riadku promptu, je nutné jednotlivé časti príkazu "if " oddeliť čiarkou, resp. bodkočiarkou:

```
» if a > b, t = a; a = b; b = t; end
```

Nasledujúci script vymení prvky matíc `mat1, mat2`, ak všetky prvky `mat1` sú väčšie ako im odpovedajúce prvky matice `mat2`:

```
% ilustrácia "if": ak všetky prvky mat1 sú väčšie ako prvky mat2
%                  tak script vymení obsah premenných mat1, mat2
if mat1 > mat2
   temp = mat1;
   mat1 = mat2;
   mat2 = temp;
end
```

Je tomu tak preto, lebo výraz "mat1 > mat2" je relačná matica a tá sa v príkazoch "if" a "while" vyhodnocuje ako pravdivá, len ak každý jej prvok sa rovná 1. Ak by sme tento fakt nevedeli, písali by sme podmienku takto: **`if all(all( mat1 > mat2))`**

Úloha. Napíšte script, ktorý vymení prvky matíc vtedy, ak existuje prvok v `mat1`, ktorý je väčší ako jemu odpovedajúci prvok matice `mat2`.

Na príkaze "if-elseif" si všimnite syntax ("elseif" sa píše spolu). Na ilustráciu uvádzame:

```
if sin(1) < cos(1)
         disp('sin(1) < cos(1)')
elseif sin(1) == cos(1)
         disp('sin(1) = cos(1)')
else
         disp('sin(1) > cos(1)')
end
```

Cyklus "for" sa nachádza v mnohých príkladoch HelpWindow, aj v HelpDesk, a preto uveďme len dva:

```
» sucet = 0;
» for x = 2.3:1.3:5
      sucet = sucet + x
   end
```

Situácia chcela ilustrovať, že výrazom za "for" býva často vektor tvaru "**`a:krok:b`**", avšak, môžeme použiť aj menej časté:

```
» sucet = 0;
» for x = [1 3 4 7]
      sucet = sucet + x*x
  end
```

Cyklus "while" začneme objasnením príkladu z HelpWindow, ktorý uvádzajú manuály MATLABu už dlhé roky. Ide o výpočet exponenciály matice z jej definície – ako súčtu maticového radu. Je prirodzené definíciu maticovej exponenciály motivovať známym Taylorovým radom skalárnej funkcie exp(.). Preto začneme skalárnym prípadom:

```
% script     : Taylorov (konecny) rad: 1 + x + x^2/2 + x^3/3! + x^4/4! + ...
% VARS       : tol, sucet, clen, n, x
%inicial.:
tol=0.001;   % tolerancia, pre zastavovaciu podmienku cyklu
sucet = 0;
clen = 1;
n = 1;
x = 1;       % koniec inicializacii, nasleduje telo:
while clen > tol
   sucet = sucet + clen;
   clen  = clen*x/n;
   n = n + 1;
end
sucet
```

Vyskúšajte fungovanie tohoto scriptu. Potom zmeňte hodnotu argumentu x na -1. Odozva bude možno neočakávaná, lebo to nebude hodnota blízka exp(-1). Pre záporné x je script chybný. Uvedomte si, v čom je háčik a modifikujte script tak, aby pracoval dobre tak pre nezáporné, ako aj pre záporne argumenty!

Teraz to isté, pre maticu "mx" (namiesto skaláru x, majme maticu "mx"). Člen radu bude matica, a preto v zastavovacej podmienke použijeme normu matice. Inak sa toho veľa nezmení:

```
% script       : Taylorov (konecny) rad pre exp(matica)
% VARS         : tol, sucet, clen, n, mx, dim
% inicializacie:
  tol = 0.001;        % tolerancia pre zastavovaciu podmienku
  dim = size(mx);     % len aby sme dva razy nevolali "size"
  sucet= zeros(dim);  % sucet (konecneho) radu
  clen = eye(dim);    % inicializacia clena radu
  n = 1;              % pomocny index
%
while norm(clen) > tol
   sucet = sucet + clen;
   clen  = clen*mx/n;
   n = n + 1;
end
sucet
```

Porovnajte odozvu scriptu s odozvou "built-in" funkcie "expm" (teda nie "exp", ale "expm" !).

Príkaz "switch", ktorý realizuje podmienené vetvenie, zatiaľ nekomentujeme a odkazujeme čitateľa na Help Desk, ktorý upozorňuje, čím sa "switch" v MATLABe odlišuje od "switch" v jazyku C. Jednu situáciu použitia tohto príkazu uvedieme v článku o M-funkciách.
Zábavnú ilustráciu príkazu "switch" získate príkazom "why" (v prípade, že ste "why" doteraz nepoužili, pár razy za sebou odklepnite "why"). Príkaz "type why" ukáže kód "why" a použitie "switch".

7 M – funkcie

M-súbory systému MATLAB (t.j. súbory *.m) sú dvoch typov – buď sú to scripty, alebo M-funkcie, krátko – funkcie. Vlastnosti a význam scriptov sme popísali v článku 3. Teraz hovorme o funkciách. Vytvárame ich ako užívatelia systému a rozširujeme nimi možnosti systému vzhľadom na naše konkrétne potreby. Práca s nimi je rovnako pružná ako s tými funkciami, ktoré sú do systému zabudované (sú "built-in", ako napr. `size`, `sin`, `rank` ... atď.). Rozdiel je v tom, že kým zdrojový kód (užívateľom) napísanej funkcie je dostupný, zdrojový kód "built-in" funkcií dostupný nie je.

Čo odlišuje funkciu od scriptu? Samozrejme, líšia sa po formálnej stránke, ale podstata je v tom, že:

1) funkcia môže pracovať s (ľubovoľne veľa) vstupnými a výstupnými argumentami a "komunikuje" s okolím len prostredníctvom nich (jedinou výnimkou sú "globálne" premenné pracovného priestoru, ale ich existenciu nateraz ignorujme)
2) všetky premenné vystupujúce vo funkcii (včítane vstupných a výstupných argumentov) sú lokálnej povahy. Hoci by ich mená boli zhodné s menami premenných pracovného priestoru, nemajú s nimi nič spoločné.

Čo sa týka objasňovania formy funkcie, začnime príkladom. Vezmime náš prvý script "TabSin" a urobme z neho funkciu. Tá nám umožní tabelovať hodnoty funkcie sínus na intervale $[a, b]$ s krokom h bez toho, aby bolo nutné rezervovať tri premenné pracovného priestoru (a pamätať si to!).

```
function TabSinf(a,b,h)
%TABSINF TabSinf(a,b,h) tabelovanie hodnot sinusu na [a, b] s krokom h
%
x= a:h:b;
y= sin(x);
disp(' ')
disp('  k     x(k)     sin(x(k))')
disp('-------------------------')
for k = 1:length(x)
   disp(sprintf(' %2.0f     %4.2f     %7.4f ',k, x(k), y(k) ))
end
disp(' ')
```

Prvý riadok M-funkcie (po formálnej stránke) odlišuje funkciu od scriptu tým, že má predpísanú formu: musí začínať kľúčovým slovom **function** a za ním: vektor výstupných argumentov, meno funkcie a v okrúhlych zátvorkách mená vstupných argumentov (všímajte si syntax na príkladoch!). Naša funkcia nemá výstupné argumenty, pretože jej zmyslom je akcia (nie hodnota) – akcia: tabelovať sínus.

Všimli ste si, že meno funkcie sme (oproti scriptu) zmenili – totiž funkciu zapíšeme do súboru, ktorý musí(!) mať rovnaké meno, ako funkcia (a v našom adresári už existuje script "TabSin.m" – z článku 3). Preto teraz pribudne súbor: "TabSinf.m".

Druhý riadok (a prípadne ďalšie riadky) funkcie začínajú znakom % a obsahujú komentár. Vrelo sa odporúča písať komentár aj pre krátke funkcie. Z dvoch dôvodov – jednak preto, že po čase nám náš vlastný kód už veľa nehovorí a tiež preto, lebo na správne volanie funkcie si potrebujeme pripomenúť nielen význam vstupných a výstupných argumentov, ale aj ich poradie. Na prečítanie komentára nie je treba prácu editora, lebo opäť príkaz `help meno-funkcie` zobrazí riadky komentára do Command Window.

Druhý riadok funkcie, je prvým riadkom komentára a hovorí sa mu H1 line preto, lebo je to prvý riadok nami vytváraného Helpu. Ak dodržíte nasledujúcu formu pre H1 riadok, bude fungovať príkaz "lookfor" (spomenuli sme ho v článku 1, význam "lookfor" si oživte cez "help lookfor").
Forma H1 riadku je takáto:
znak %, potom názov funkcie veľkými písmenami, potom jedna (alebo viac) medzier a krátky opis funkcie. Zvykom je opis začať veľkým písmenom a riadok ukončit bodkou (ale to veľké písmeno a bodka je už len "štábna" kultúra ... :-) a nie je to nutné.

V našom prípade to znamená, že H1 riadok vyzerá takto:

```
% TABSINF  Tabelovanie hodnot funkcie sinus.
```

Ďalšie riadky komentára sú ľubovoľné a všetky budú zobrazené príkazom "help menofuncie". Formu komentára nakoniec môžte odpozorovať zo stoviek M-funkcií Vašej verzie MATLABu. Nasleduje príklad veľmi jednoduchej funkcie s jedným vstupným a jedným výstupným argumentom:

```
function y = apriemer(x)
% APRIEMER   Aritmeticky priemer zloziek vektora x.
% y = apriemer(x)   x je riadkovy, alebo stlpcovy vektor
if min(size(x))~= 1
   error('chybny vstup! vstupny argument ma byt vektor!')
end
n = length(x);
y = sum(x)/n;
```

Nasledujúca funkcia má dva vstupné argumenty a ilustruje použitie príkazu "switch":

```
function y = normamat(mx,p)
% NORMAMAT   Norma matice mx, podla parametra p.
% ak p= 1,   tak norma stlpcova
% ak p= 6,   tak norma Frobeniusova (F je 6. pismeno :-))
% ak p= inf  tak norma riadkova
switch p
case 1
   y = max(sum(abs(mx)));
case 6
   y = sqrt(sum(sum((abs(mx)).^2)));
case Inf
   y = max(sum(abs(mx')));
otherwise
   error('chybny vstup! druhy argument musi byt 1,6, alebo inf')
end
```

Teraz príklad funkcie s dvomi výstupnými argumentami:

```
function [y1,y2] = minmax2(a)
%MINMAX Najmensi a najvacsi prvok matice abs(a)
y1 = min(min(a));
y2 = max(max(a));
```

volanie funkcie vyzerá takto:

```
» a = [1 2; 3 4]; [u, v] = minmax(a)
```

```
u =           v =
     1             4
```

Nasleduje obmena funkcie minmax, ilustrujúca použitie funkcie "nargout" (= number of arguments-out). Funkcie "nargin" a "nargout" sú veľmi šikovné nástroje pre vytváranie užívateľsky príjemných M-funkcií, pozrite Help! Ilustrácia:

```
function [y1,y2] = minmax2(a)
%MINMAX2  Najmensi a pripadne! aj najvacsi prvok matice abs(a)
%         y1 je najmensi prvok matice abs(a)
%         ak nargout= 2, tak y2 je najvacsi prvok matice abs(a)
y1 = min(min(a));
if nargout > 1, y2 = max(max(a)); end
```

Teraz môžme funkciu volať dvomi spôsobmi: s dvomi argumentami (ako predtým), ale tiež s jedným:

```
» a = [1 2; 3 4]; y = minmax2(a), [u,v] = minmax2(a)
y =
     1
u =           v =
     1             4
```

Nasledujúca funkcia je prepísaním scriptu z článku 6. Vstupnými argumentami sú `x` a `tol`.

```
function [y, n] = expTskf(x,tol)
% EXPTSKF         Taylorov (konecny) rad exp funkcie - skalarny pripad.
% expTskf(x,tol)  x - argument, tol (real > 0) pre zastavov. podmienku
%                 y - sucet radu, n - pocet clenov radu
sucet = 0;        % sucet (konecneho) radu
clen = 1;         % clen radu
n = 1;            % pomocny index
while abs(clen) > tol
   sucet = sucet + clen;
   clen = clen*x/n;
   n = n + 1;
end
y = sucet;
n = n - 1;
```

Nasleduje obmena, v ktorej uplatníme funkciu nargin (= number of arguments-in, t.j. počet vstupných argumentov). Funkciu "expTskf2" je možné volať aj s jedným vstupným argumentom `x` a v takom prípade sa za argument "tol" berie hodnota `eps*x`.

```
function [y, n] =  expTskf2(x,tol)
% EXPTSKF2         Taylorov (konecny) rad exp funkcie skalarny pripad.
% expTskf2(x,tol)  x - argument, tol (real > 0)- pre zastav.podmienku
%                  y - sucet radu, n- pocet clenov radu
%
if nargin == 1     % ak sa nezada "tol" default hodnota = eps*x
   tol = eps*x     % eps  je globalna premenna
end
sucet = 0;         % sucet (konecneho) radu
clen = 1;          % clen radu
n = 1;             % pomocny index
%
while abs(clen) > tol
   sucet = sucet + clen;
   clen = clen*x/n;
   n = n + 1;
end
y = sucet;
n = n - 1;
```

Na záver poznamenajme, ako sa vyvarovať toho, že použijeme (nechtiac!) ako meno našej funkcie také, ktoré sa zhoduje s menom nejakého existujúceho M-súboru (nejakého scriptu, či M-funkcie), prípadne s menom nejakej zabudovanej funkcie MATLABu. Napr., ako overiť, že meno "minmax" je možné použiť ? Je viac ciest: použime príkaz `type` (teda: `type minmax`), alebo `help minmax`, alebo `exist`, teda: `exist('minmax')`. Poznamenávame, že môže pomôcť aj príkaz "which" (viď `help which`).

Optimalizovanie M-súborov

Vektorizovanie sme realizovali od samého začiatku tohoto textu pri každej príležitosti. Teraz ukážeme, že je to v systéme MATLAB skutočne "to pravé – orechové". Predstavuje totiž využívanie "built-in" funkcií, kým používanie cyklov "for" a realizovanie operácií so zložkami vektorov (či prvkami matíc) je mnohonásobne (niekedy rádovo) pomalšie. Tento fakt chceme teraz ilustrovať na jednoduchom príklade. Čas potrebný na realizáciu výpočtov zabezpečia funkcie `tic` a `toc`. Všetko bude jasné z nasledujúceho:

```
» n = 100000; x = rand(n,1); s = 0;
» tic, for i = 1:n, s = s + x(i)^2; end; toc
elapsed_time =
    2.5300
» tic, s = sum(x.^2); toc
elapsed_time =
    0.2200
```

Ako vidíme, vektorizovaním sme významne získali. Pri optimalizovaní kódu hrá úlohu aj pre-alokovanie poľa (vektora, resp. matice). V druhom článku prvej časti sme ocenili prácu *memory managera*, vďaka ktorému sme sa nemuseli starať o dimenzovanie vektorov a matíc. Avšak v prípadoch, keď dimenzia matíc je veľká, realizáciu našich výpočtov urýchlime vhodným "alokovaním vopred" (t.j. pre-alokovaním) vektorov a matíc. Začnime veľmi jednoduchou funkciou (v jej komentári objasňujeme):

```
function y = alok_nie(n)
% alok_nie Ilustracia prace s vektorom bez alokovania
% memory manager musi pracovat v kazdej! slucke cyklu
aux = [1];
for i= 2:n
   aux(i)= 1/i;
end
y = sum(aux);
```

volajme:

```
» tic, nie = alok_nie(5000), toc
nie =
    9.0945
elapsed_time =
    5.3900
```

a teraz s alokáciou:

```
function y = alok_ano(n)
% ilustracia prace s vektorom ked alokujeme, ked memory manager
% nemusi! pracovat v kazdej slucke cyklu vdaka prikazu:
aux = ones(1,n);   % tento prikaz pre-alokuje miesto pre vektor
for i = 2:n
   aux(i)= 1/i;
end
y = sum(aux);
```

volajme:

```
» tic, ano = alok_ano(5000), toc
ano =
    9.0945
elapsed_time =
    0.2200
```

V tejto situácii sme ignorovali vektorizovanie, pracovali sme so zložkami vektora, lebo sme chceli ilustrovat pre-alokáciu. Pre zaujímavosť, vektorizujme kód a nemusíme písať žiadne cykly:

```
» tic, x = 1:5000; x = 1./x; sum(x), toc
ans =
    9.0945
elapsed_time =
    0.1100
```

A ako vidíme, dostali sme ďaleko najlepší čas.

Na nasledujúcej situácii ukážeme, že alokovanie nemožno robiť "mechanicky", bez rozmyslu. Uvažujme o postupnosti $(a_n)_n$ definovanej rekurentne tak, že jej *n*-tý člen je súčtom dvoch predchádzajúcich členov, pričom $a_1 = 1$ aj $a_2 = 1$.
$(a_n)_n$ je známa *Fibonnaciho* postupnosť: 1, 1, 2, 3, 5, 8, 13, ... Na vytvorenie takej postupnosti sa nedá uplatniť vektorizovanie, avšak, prealokovať priestor na vektor, ktorý ideme vytvárať, je možné. Napíšme dve funkcie: `fibon` (bez alokácie) a `fibona` (to isté, ale s alokáciou). Tu sú ich kódy:

```
function vf = fibon(n)
% FIBON  generuje prvych n clenov Fibonnaciho postupnosti.
%        postupnost zacina takto: 1, 1, 2, 3, 5, 8, 13 ...
if n == 1
   vf = [1];
elseif n == 2
   vf = [1, 1];
else
   vf = [1, 1];
   for i= 3:n
      vf(i) = vf(i-1) + vf(i-2);
   end
end
```

Nasleduje kód s alokáciou, ale so zlou alokáciou... vlastne, s nealokáciou, a tak funkcia má aj taký názov:

```
function vf = fibonaomyl(n)
% FIBONAOMYL  generuje prvych n clenov Fibonnaciho postupnosti.
%             alokovanie tu nie je alokovanim!!
vf = ones(1,n);
if n == 1
   vf = [1];
elseif n == 2
   vf = [1, 1];
else
   vf = [1 1];
   for i = 3:n
      vf(i) = vf(i-1) + vf(i-2);
   end
end
```

Z údivom zistíte, že (zlé) "alokovanie" nedáva lepšie výsledky! Naopak! Dáva horšie!

```
» tic, fibon(5000); toc
elapsed_time =
    5.4300
» tic, fibonaomyl(5000); toc
elapsed_time =
    5.5000
```

Samozrejme, niekde musí byť chyba ... a tá chyba je v nesprávnom alokovaní.... Čo sa deje? Pozrime na piaty riadok zdola – príkaz: `vf = [1 1];`
zmenil dimenziu vektora `vf`, takže to, čo sa v cykle `for i = 3:n` rozbehlo, bolo zase to, čo predtým! Totiž predimenzovávanie (t.j. zmena dimenzovania) vektora `vf` v každej slučke cyklu! Žiadna alokácia sa neuskutočnila, naopak, kód je horší ako ten prvý, pretože príkaz `vf = ones(1,n)` je tam (tam, kde je) celkom zbytočný! Má to byť takto (komentár funkcie sme už vystrihli):

```
function vf = fibona(n)    % uz spravne alokovanie
if n == 1
   vf = [1];
elseif n == 2
   vf = [1, 1];
else
   vf = ones(1,n);
   for i = 3:n
      vf(i)= vf(i-1) + vf(i-2);
   end
end
```

A teraz volajme funkciu a merajme čas...

```
» tic, fibona(5000); toc
elapsed_time =
    0.2700
```

Ako vidíme, stojí za to správne alokovať vektory a matice veľkých rozmerov. Pravda, záleží na okolnostiach. V bežných situáciách (a na dnešných počítačoch) si memory manager MATLABu ľahko poradí.

8 Grafické možnosti MATLABu

8.1 2D grafy

Začneme krátkou rekapituláciou známeho: ak `v` je vektor, tak príkaz `plot(v)` je ekvivalentný príkazu `plot(1:length(v),v)`, čo znamená, že `plot(v)` dáva znázornenie hodnôt zložiek vektora vzhľadom na množinu indexov, t.j. vzhľadom na vektor `1:length(v)`.

Automatické škálovanie sa postará o to, aby všetky zložky vektora boli zobrazené. Niekedy to ale nemusí byť najvhodnejšie, ako ukazuje príklad:

```
» v = [1  2  4  4  0  2]
» plot(v), shg
```

V takých prípadoch použijeme príkaz `axis([xmin xmax ymin ymax])`, ktorý vyraďuje automatiku škálovania. Argument príkazu "axis", t.j. vektor `[xmin xmax ymin ymax]` môžeme označiť napr. ako "okno":

```
» okno = [0  8  -1  5]; axis(okno), grid on, shg
```

Aktuálnu hodnotu vektorového parametra príkazu "axis" zobrazí príkaz "axis" (bez argumentu). Jeho odozvou je vektor aktuálnych hodnôt: `xmin, xmax, ymin, ymax`. Návrat automatického škálovania dosiahneme príkazom `axis auto`.
(Rozšírenie možností dávajú príkazy "xlim", "ylim" – pozrite help príkazov: `xlim, ylim`).

Ilustrovanie týchto príkazov prináša jednoduchý script. Bude čitateľnejší, ak si najprv vyskúšate príkazy:

```
» clf, t = linspace(0,2*pi,5)'; a = [cos(t) sin(t)]; plot(a(:,1), a(:,2))
» axis('square')
» hold on
» t = linspace(0,2*pi,9)'; a = [cos(t) sin(t)]; plot(a(:,1), a(:,2),'r')
```

V nasledujúcom scripte uplatníme funkciu "pause"; `pause(k)` spôsobí prestávku dĺžky asi *k* sekúnd:

```
% ilustracia plotovania stlpcov jednej matice
close all
for i = 2:5
   t = linspace(0, 2*pi,2^i+1)';
   a = [cos(t) sin(t)];
   plot(a(:,1), a(:,2));
   axis('square')
   hold on
   pause(1)
end
hold off
```

Ak si uvedomíte úlohu vrcholov znázornených mnohouholníkov, tak prechod od týchto pravidelných mnohouholníkov ku kružnici je celkom prirodzený. Ak diskretizovanú premennú "t" priblížime viac ku kontinuálnej "tc" prebiehajúcej spojite interval [0, 2*pi], dostávame parametrické vyjadrenie kružnice:

```
» clf, t = linspace(0, 2*pi, 501);
» plot(cos(t), sin(t)), axis square
```

Zaiste ste si všimli, že príkaz "plot" akceptuje aj maticové argumenty. Nakreslime elipsu najprv bez využitia parametrického vyjadrenia, ilustrujúc "plot", ktorého argumenty sú dve matice **`a,b`**:

```
» x = linspace(0, 2, 401)'; y = sqrt((4 - x.^2)/4);
» a = [-x -x  x  x]; b = [-y  y -y  y];
» close all, plot(a, b,'k'), axis([-3 3 -2 2]), grid on
```

A teraz jednoduchšie, využijúc parametrické vyjadrenie, naviac, umiestnime jej stred do bodu (11, 13):

```
» t = linspace(0, 2*pi, 101)';
» x = 11 + 2*cos(t); y = 13 + sin(t)
» figure(2), plot(x, y), axis auto, grid on
```

Zmyslom nasledujúceho scriptu je predstaviť príkazy: "input", "title", "xlabel", "ylabel" a "gtext".

```
% plotovanie rychlosti volneho padu (ako funkcie casu)
tmax = input('zadajte tmax (zvolte z intervalu (5,50)): ');
t = linspace(0, tmax, 101);
c = input('zadajte hodnotu koef.odporu [kg/s]: ');
m = input('zadajte hmotnost parasutistu  [kg]: ');
v = 9.81*m*(1 - exp(-c*t/m))/c;
close all
plot(t,v)
title('priebeh rychlosti volneho padu od casu')
xlabel('cas v sekundach')
ylabel('rychlost volneho padu [m/s]')
gtext('vidime, ze rychlost padu nerastie neobmedzene ...')
gtext('(tieto dva riadky len ilustruju prikaz "gtext"...)')
```

Ako sme videli, príkaz `c = input('text')` zobrazí text a čaká na zadanie numerickej hodnoty, ktorú priradí premennej `c`. Ak chceme zadať reťazec ako hodnotu premennej, použijeme syntax: `x = input('text','s')`. Kliknutie na vlasový kríž umiestnilo text – to je účinok príkazu `gtext`.

Príkaz **`subplot`** umožňuje v jednom grafickom okne prezentáciu niekoľkých grafov, okienka ktorých budú umiestnené vo forme matice m krát n (spolu $m.n$ okien) – syntax: `subplot(m,n,i)`. Odkazujeme na ne indexom "i", keď indexovanie sa berie po riadkoch (zhora dole, resp. v riadku, prirodzene, zľava doprava). V našom príklade vezmeme $m = 2$, $n = 3$ (t.j. dva rady a v každom z nich 3 okienka), tak napr. prostredné okienko v prvom rade je celkovo druhé ($i = 2$), kým prvé okienko v druhom rade je celkovo štvrté ($i = 4$). Všetko objasní príklad:

```
% ilustracia "subplot" - grafy mocninovych funkcii umiestnene v dvoch radoch
% a v kazdom rade tri okienka; ich indexovanie zlava doprava a zhora dole:
x = linspace(0,2,201);
subplot(2,3,1), plot(x, x, 'r'),  ylabel('prva mocnina')
subplot(2,3,2), plot(x, x.^2, 'g'), ylabel('druha mocnina')
subplot(2,3,3), plot(x, x.^3, 'b'), ylabel('tretia mocnina')
subplot(2,3,4), plot(x, x.^4, 'c'), ylabel('stvrta mocnina')
subplot(2,3,5), plot(x, x.^5, 'm'), ylabel('piata mocnina')
subplot(2,3,6), plot(x, x.^6, 'k'), ylabel('siesta mocnina')
```

Všimnime si, že automierka škáluje mierky v každom subokienku zvlášť.

Významnú úlohu v dvojrozmernej grafike má príkaz `fplot`. Je užitočný najmä vtedy, keď graf uvažovanej funkcie sa na nejakej časti oboru rýchle mení, a preto príkaz plot nedáva dobrý obrázok. Ilustrácia:

```
» figure(2), x = 2:.05:5; y = x.*sin(x.^3); plot(x,y), grid on
```

V okolí bodu (4.5; 0) je graf dokonca nedôveryhodný. Príkaz `fplot` pracuje pomalšie, pretože adaptívne mení hustotu bodov vzhľadom na rýchlosť zmeny zobrazovanej funkcie, avšak dáva v takýchto prípadoch ďaleko lepšie výsledky, čo ilustruje takáto voľba:

```
» figure(3), okno = [2, 5]; fplot('x*sin(x^3)', okno)
```

Všimnime si, že v prípade použitia "fplot" nie je nutné písať vektorizovanú definíciu funkcie (porovnajte syntax "x.*sin(x.^3)" so syntaxou "x*sin(x^3)"). Samozrejme, že je možné zobraziť funkcie definované užívateľom. Pre ilustráciu definujme funkcie f 1, f 2 (buď ako "inline" objekty, ktoré sme uviedli v článku 5, alebo ako M-funkcie, do M-súborov "f 1.m", resp. "f 2.m"):

```
function y = f1(x)
% ilustracia fplot
y = exp(-x/3)*cos(x^2);
```

```
function y = f2(x)
% ilustracia fplot
y = sin(x^2)
```

```
» figure(4), fplot('f1',[0 6 -2 2]), hold on,  fplot('f2',[0 6 -2 2],'k')
```

Je možné pracovať s vektorom funkcií, (v tomto prípade teda bez "hold on") ako ukazuje príklad volania:

```
» figure(5), fplot('[f1(x), f2(x)]', [0 6 -2 2]), shg
```

8.2 3D grafy

Zavrime všetky grafické okná a začnime znázornením jednoduchej krivky danej parametricky:

```
» t = linspace(0, 10, 1001);
» x = 2*cos(5*t);
» y = 2*sin(5*t);
» z = t;
» plot3(x,y,z), grid on
» title('ukazka prikazu plot3')
```

Zrejme nás zobrazenie neprekvapilo, veď v rovine x-y ide o kružnicu s polomerom 2 (z-súradnica je snáď najjednoduchšou možnou funkciou parametra t). Ako vidíme, príkaz "plot3" je analógiou príkazu "plot", pričom (nepovinný) štvrtý argument je reťazec. Jeho prvý znak určuje farbu, druhý znak určuje typ označenia bodov a tretí znak typ čiary, ktorá body spája (o znakoch platí presne to, čo v príkaze "plot"). Jeden z možných výberov riadiaceho reťazca použijeme v nasledujúcej ilustrácii, keď znázornenie krivky bez pomoci MATLABu by už nebolo jednoduché:

```
» t = 0:0.001:4;
» x = t.*cos(25*t);
» y = t.*sin(25*t);
» z = sqrt(t);
» plot3(x,y,z,'r.:'), grid on, shg
```

Zobrazenie grafu funkcie dvoch premenných začneme opäť situáciou, v ktorej si dokážeme graf funkcie predstaviť aj bez podpory MATLABu. Nech f je na oblasti $[0, 2]\times[0, 1]$ daná vzťahom $f(x, y) = x^2 + 2y^2$. Zrejme graf funkcie je časť paraboloidu, ktorý prechádza bodmi (0,0,0), (2,0,4), (0,1,2) a (2,1,6).

V MATLABe zostrojenie grafu funkcie dvoch premenných znamená:

1. vytvorenie mriežky uzlových bodov v uvažovanej oblasti (teraz pôjde o obdĺžnik $[0, 2]\times[0, 1]$)
2. vytvorenie matice, ktorej prvky sú hodnoty funkcie f v uzloch mriežky
3. použitie príkazov "mesh", "surf", "contour", resp. ďalších, na znázornenie grafu (alebo jeho časti).

Mriežku uzlových bodov vytvoríme príkazom "meshgrid", keď najprv zvolíme vektor `x` pre x-ové súradnice a vektor `y` pre y-ové súradnice bodov mriežky. Pre pohodlnú čitateľnosť voľme:

```
» x = 0:0.4:2; y = 0:0.2:1;
```

Nasledujúcim príkazom "meshgrid" realizujeme prvý bod horeuvedeného postupu. Náročky príkaz neukončíme bodkočiarkou, aby sme videli jeho odozvu:

```
» [xg, yg] = meshgrid(x, y)
xg =
         0    0.4000    0.8000    1.2000    1.6000    2.0000
         0    0.4000    0.8000    1.2000    1.6000    2.0000
         0    0.4000    0.8000    1.2000    1.6000    2.0000
         0    0.4000    0.8000    1.2000    1.6000    2.0000
         0    0.4000    0.8000    1.2000    1.6000    2.0000
         0    0.4000    0.8000    1.2000    1.6000    2.0000
yg =
         0         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Vidíme, že "objekt" `[xg yg]` je dvojica matíc: matica `xg` a matica `yg`. Matica `xg` predstavuje vertikály v rovine x-y (tie sa od seba líšia x-hodnotou, predstavme si ich ako úsečky v $[0, 2]\times[0, 1]$ rovnobežné s osou y, pre $x = 0,\ 0.4,\ 0.8,\ 1.2,\ 1.6,\ 2.0$.

Matica `yg` predstavuje horizontálne úsečky rovnobežné s osou x, pre $y = 0,\ 0.2,\ 0.4, 0.6,\ 0.8,\ 1.0$.

MATLAB bude interpretovať tieto dve matice tak, že f-hodnoty vyčísluje v uzloch siete – v priesečníkoch pomyslených úsečiek, čím sa realizuje druhý bod hore uvedeného postupu. Definujeme maticu **`z`** príkazom, v ktorom je nutné použiť matice `xg, yg`:

```
» z = xg.^2 + 2*yg.^2
z =
         0    0.1600    0.6400    1.4400    2.5600    4.0000
    0.0800    0.2400    0.7200    1.5200    2.6400    4.0800
    0.3200    0.4800    0.9600    1.7600    2.8800    4.3200
    0.7200    0.8800    1.3600    2.1600    3.2800    4.7200
    1.2800    1.4400    1.9200    2.7200    3.8400    5.2800
    2.0000    2.1600    2.6400    3.4400    4.5600    6.0000
```

Overme, že toto pole čísel naozaj predstavuje *f*-hodnoty v uzloch zvolenej siete obdĺžnika [0, 2]×[0, 1].

Uvažujme bod [*x*(3), *y*(4)] našej siete. Je to bod roviny [0.8; 0.6] (pretože ML indexuje od 1). *f*-hodnota v bode [0.8; 0.6] sa rovná: (.8)^2 + 2*(.6)^2 = .64 + .72 = 1.36
Naozaj, v matici **z** je hodnota 1.36 v treťom stĺpci a štvrtom riadku. Pre prvky matice **z** teda platí:

```
z(i,j) = f(x(j),y(i))
```

Tretí bod postupu realizujeme príkazmi "mesh", alebo "surf":

```
figure(1), mesh(x,y,z)
figure(2), surf(x,y,z)
```

Všimnime si, že v "mesh" aj "surf" sme mohli použiť x, y, avšak matica **z** musí byť matica vytvorená príkazom "meshgrid"!

V nasledujúcom príklade (ktorý dáva zaujímavejší obrázok) použijeme skrátený postup, keď preskočíme medzikroky a výsledok dosiahneme rýchlejšie (z pochopiteľných dôvodov potlačíme výpisy):

```
» [x, y] = meshgrid(-2:0.02:2, -1:0.01:1);
» z = 1./(1 + x.^2 + 4*y.^2);
» figure(3), mesh(x,y,z)
```

Namiesto "mesh" vyskúšajte "surf" a uvidíte rozdiel. Ďalším príkazom 3D grafiky je "contour". Umožňuje znázorniť tie body oblasti, ktoré majú tú istú *f*-hodnotu:

```
» figure(4), contour(x,y,z)
```

Prirodzene, že by sme radi "riadili" hodnoty tých *f*-hodnôt. MATLAB to umožňuje tak, že zvolíme vektor želaných f-hodnôt a voláme príkaz "contour" v rozšírenom tvare. Povedzme, že nás zaujímajú tie body roviny, v ktorých má funkcia *f* hodnoty 0.3, 0.6, 0.9. Preto definujme "contour-hodnoty" vektorom `conh` a príkazom "contour(x, y, z, conh)" získame želané zobrazenie:

```
» conh = [.3 .6 .9];
» figure(5), contour(x,y,z,conh)
```

Naviac, je možné čiary popísať takto:

```
» conh = 0.2:0.1:0.9;              % pre lepsie znazornenie volime viac urovni
» figure(6), contour(x,y,z,conh)
» [c, h] = contour(x,y,z,conh);
» clabel(c, h, conh([2, 4, 6, 8]))
```

Ako vidíme, zvolené čiary (druhá, štvrtá, šiesta a ôsma) sú popísané odpovedajúcimi *f*-hodnotami.

Nakoniec sa zmieňme o modifikácii príkazov "mesh" a "surf", keď sa pod grafom funkcie zobrazia aj čiary *f*-hodnôt (ide o kombináciu týchto príkazov s príkazom "contour"). Vyskúšajme:

```
» close all, meshc(x,y,z)
```

Zvedavejší si určite všimli, že Menu Tools grafického okna ponúka "Rotate 3D" (alebo jeho ikonku máte "vytiahnutú" ako tlačítko lišty). Ak kliknete (do obrázku) a držíte ľavé tlačítko myši, tak pohybom ruky môžete meniť uhol, pod ktorým sa pozeráte na graf.

Úloha: Na oblasti [-3, 3]×[-3, 3] zobrazte graf funkcie $z = \exp(-(x^2 - 2.r.x\,y + y^2)/(2(1 - r^2)))$, $r = 0.8$.
Na oblasti [-8, 8]×[-8, 8] zobrazte graf funkcie $z = sin(\,sqrt(x^2 + y^2\,))\,/\,sqrt(x^2 + y^2\,)$.

Na záver zopakujeme, že veľa užitočného materiálu poskytuje stránka: http://www.mathworks.com , na ktorej (okrem iného) nájdete stovky M-funkcií, týkajúcich sa rôznych oblastí inžinierskych výpočtov.