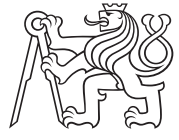
Základy programování v Matlabu

Petr Pokorný

Fakulta stavební ČVUT v Praze, katedra fyziky

2. února 2020

Osnova

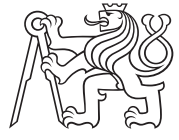


Kontakt

Ing. Petr Pokorný, Ph.D.

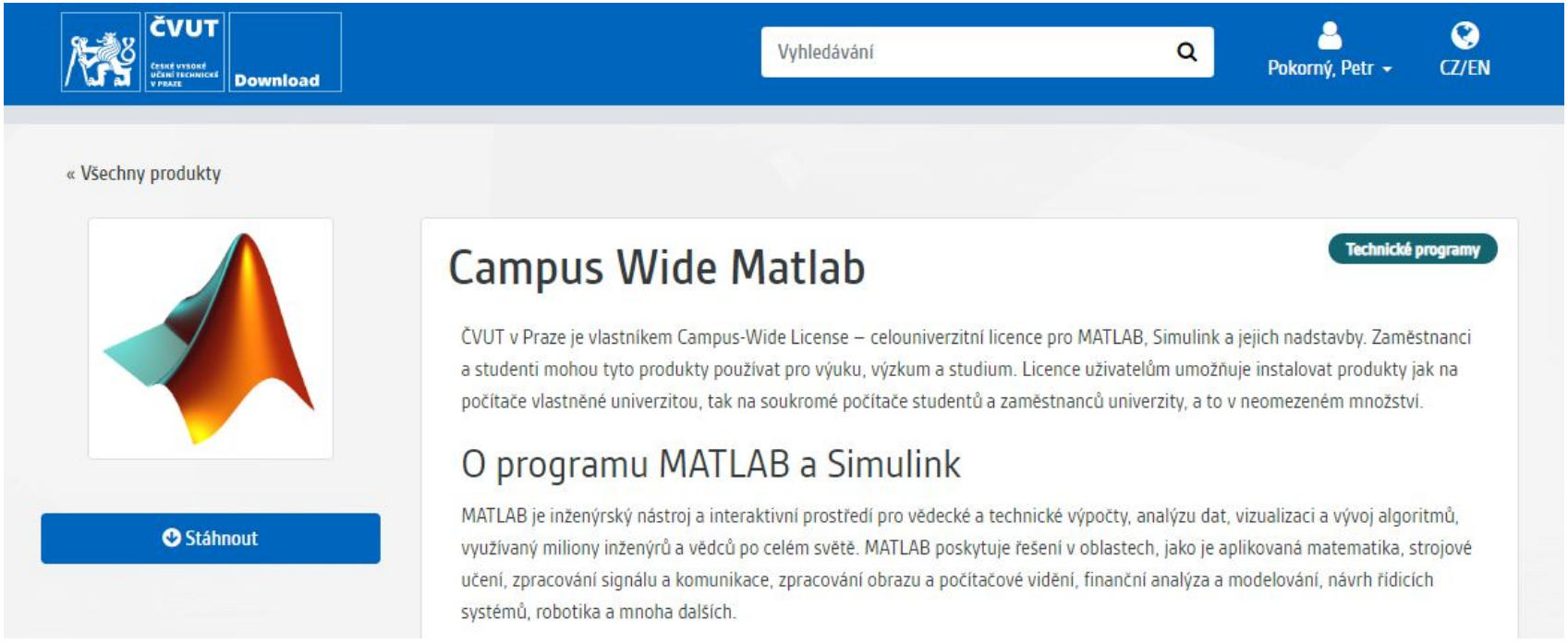
Fakulta stavební ČVUT v Praze, katedra fyziky
místnost A-618

petr.pokorny@fsv.cvut.cz



Kde sehnat Matlab?

`https://download.cvut.cz`

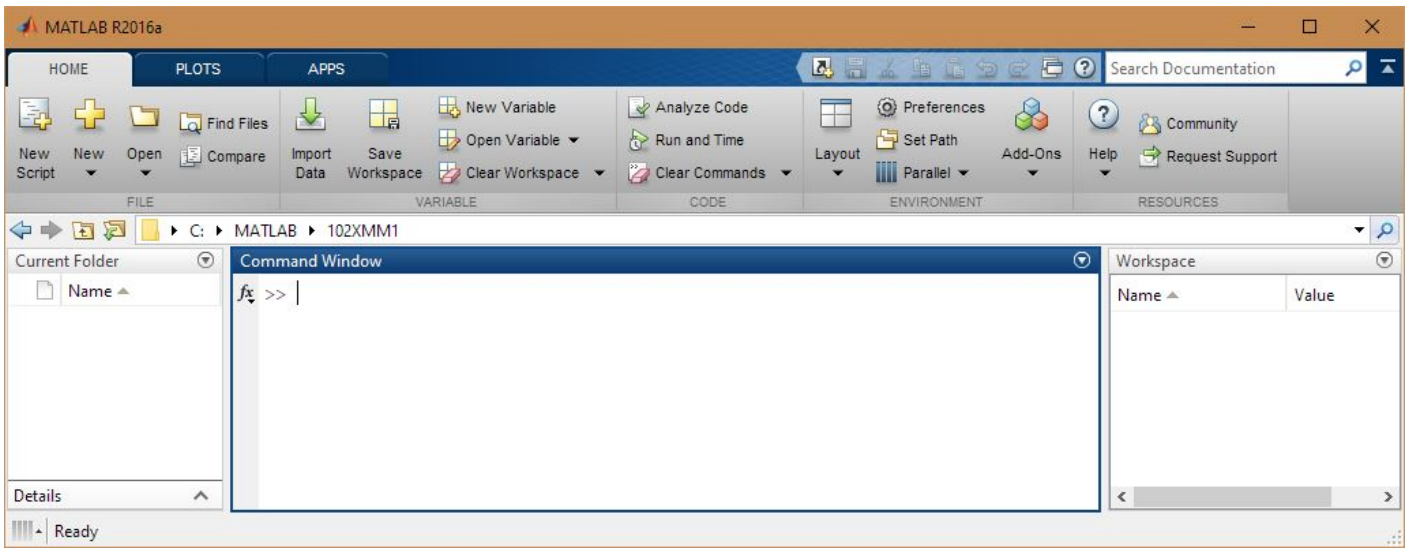


(oficiální komerční distributor: `http://www.humusoft.cz`)

První seznámení

Pracovní prostředí

- **Pracovní lišta**
- **Command Window** (příkazové okno)
- **Current Folder** (současný pracovní adresář)
- **Workspace** (aktivní proměnné)
- **History** (prohlížeč historie)

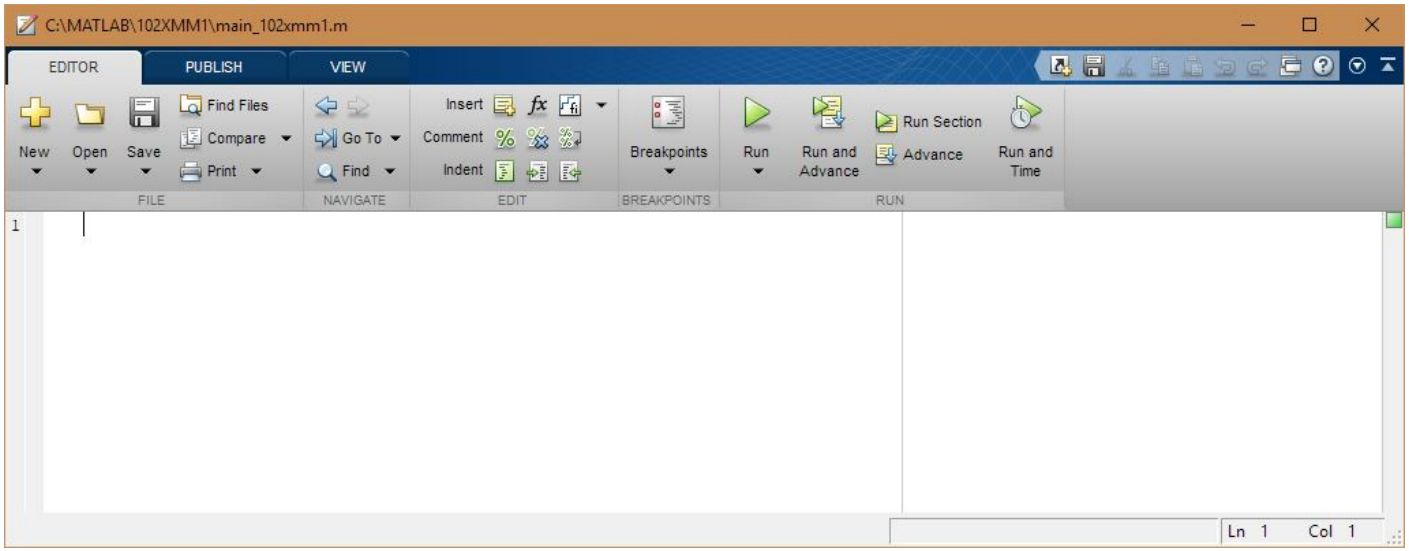


Příkazový řádek versus Editor

Do příkazového řádku můžeme zapisovat příkazy, jako bychom využívali "kalkulačku". Přitom lze využít veškeré nástroje a volat všechny funkce Matlabu.
Pro složitější (delší) výpočty je ale vhodnější práci postupně ukládat v **Editoru**. Soubory (**m-fily**) lze pak opakovaně snadno volat a případně korigovat zadání jednotlivých příkazů.

Editor

- **File** - správa souborů
- **Navigate** - navigace mezi soubory
- **Edit** - možnosti editace
- **Breakpoints** - zarážky
- **Run** - spuštění výpočtu



Nápověda

Pro podrobné seznámení se syntaxí a příklady použití jednotlivých funkcí má Matlab výborně zpracovanou nápovědu. Dialogové okno lze volat stisknutím tlačítka **Help** v Pracovní liště.
Další možností dotazu na podrobnosti k jednotlivým funkcím jsou příkazy `help` a `doc`.

```
1 % Vypis strucne informace k funkci sin na terminal
2 help sin
3
4 % Otevreni kompletni dokumentace k funkci sin
5 doc sin
```

Základy syntaxe

Příprava prostředí, komentáře

Při práci v Editoru i mimo něj je v řadě případech vhodné použít funkce `clc`, `clear`, `close`.
Komentář uvozujeme znakem %. Vše za ním Matlab ignoruje.

```
1 % toto je komentar
2 clc;                  % vymaze vse na terminalu
3 clear variables;      % vymaze vsechny promenne z ...
      pameti
4 clear globals;        % vymaze vsechny globalni ...
      promenne
5 close all;            % zavre vsechna otevrena okna
```

Matice a vektory (numerická vícerozměrná pole)

Matlab umožňuje zápis **matice** mnoha způsoby. Přiřazení hodnot proměnné uvedeno **hranatými závorkami**.

```
1  % Zapis "WYSIWYG"
2  A = [ 1 2 3
3        4 5 6
4        7 8 9 ];
5
6  % Strednik oznacuje prechod na novy radek
7  B = [ 1 2 3; 4 5 6; 7 8 9 ];
8
9  % Carka oddeluje hodnoty v radku
10 C = [ 10, -54, 123; 52, 8, -45; 22, 47, -8];
```

Poznámka: Středník na závěr každého příkazu = operátor potlačení výstupu.

Matice a vektory - *pokračování*

Matice s jedním rozměrem $= 1 \rightarrow$ **vektor**

```
1  % Zapis "WYSIWYG" (radkovy vektor)
2  a = [ 1 2 3 4 5 ];
3
4  % Carka oddeluje hodnoty v radku  (radkovy vektor)
5  b = [ 10, -54, 123, 52, 8 ];
6
7  % Sloupcovy vektor
8  s1 = [1
9        2
10       3 ];
11
12 s2 = [1; 2; 3; 4; 5];
```

Matice a vektory - *pokračování*

Skalár je matice rozměru 1x1.

```
1 % Skalarni hodnota
2 K = 54;
```

Určení rozměru numerického pole – funkce `size`

```
D = size(X), for M-by-N matrix X, returns the
two-element row vector D = [M,N] containing the
number of rows and columns in the matrix. For N-D
arrays, size(X) returns a 1-by-N vector of dimension
lengths. Trailing singleton dimensions are ignored.
```

Matice a vektory - *pokračování*

Zadání vektoru pomocí operátoru **dvojtečka** a funkce `linspace`

```
1  % Vektor od 1 po 2 do 100
2  a = 1:2:100;
3
4  % Vektor od 100 po -5 do 50
5  b = 100:-5:50;
6
7  % Diferenci +1 neni treba zapisovat (vektor od 1 ...
       po 1 do 100)
8  c = 1:100;
9
10 % Vektor o 1000 prvcich od 1 do 5 vcetne
11 d = linspace(1,5,1000);
```

Spojování matic a vektorů

Matice a vektory mohou sloužit jako submatice pro další použití. Nutné je však pamatovat na jejich rozměry a rozměr výsledné matice.

```
1 % Definice submatic
2 A = [ 1 2 3 ]; B = [ 3 4 ]; C = 15;
3
4 % Spojeni do jedne matice
5 D = [A; B C];
```

Pro spojování dále slouží funkce: `cat`, `horzcat`, `vertcat`, ...

Odstranění řádků/sloupců matice

Odstranění řádků nebo sloupců provedeme zavedením tzv. **prázdné proměnné** [].

```
1 % Definice matice
2 A = [ 1 2 3; 4 5 6; 7 8 9];
3
4 % Odstraneni prvniho radku
5 A(1,:) = [];
6
7 % Odstraneni druheho a tretiho sloupce
8 A(:,[2 3]) = [];
```

Textové řetězce

Textové řetězce uvozujeme **jednoduchými uvozovkami**.
Zobrazení textového řetězce - funkce `disp`

```
1 % Textovy retezec
2 jmeno = 'Petr';
3 prijmeni = 'Pokorny';
4 zamestnavatel = 'CTU in Prague';
5
6 % Vypis retezce na terminal
7 disp(jmeno)
8 disp(['Prijmeni: ',prijmeni])
```

Indexování

Indexování = výběr hodnot matice (řetězce) z dané pozice.
Indexujeme formou **vektoru**, který obsahuje čísla pozic, které chceme vybrat.
Pro označení maximálního rozměru dané matice použijeme funkci `size` nebo klíčové slovo `end`.

```
1 % Indexovani vyctem jednoho prvku
2 A = 1:50;    % zadani matice
3 A(1)         % vyber z pozice 1
4 A(33)        % vyber z pozice 33
5 A(48)        % vyber z pozice 48
```

Indexování - *pokračování*

```
1 % Indexovani vektorem
2 A(1:25)
3 A(1:5:25)
4 A([5 10 45])
```

```
1 % Pouziti end
2 A(10:end)
3 A(end-10:end)
```

Indexování - *pokračování*

U výběru prvků matic můžeme použít operátor **dvojtečka** jako označení **celého rozsahu**.

```
1 % Indexovani matice
2 A = rand(30,30);     % matice nahodnych cisel
3 A(1:5,end-3:end)
4 A(:,[1 3 20 25])
```

Indexování - *pokračování*

Výběr prvků matice můžeme provést jedním číslem nebo vektorem. V tu chvíli platí výběr tzv. **po sloupcích** – indexu 1 odpovídá pozice (1,1), indexu 2 odpovídá pozice (2,1), atp.

```
1 % Indexovani matice vektorem
2 A = rand(30,30);     % matice nahodnych cisel
3 A(1:5)
4 A(26:35)     % vybere poslednich pet prvku prvniho ...
      sloupce a prvnich pet druheho
```

Indexování - *pokračování*

Indexováním můžeme vybírat také prvky z textových řetězců.

```
1  % Zadani retezce
2  lorem = 'Lorem ipsum dolor sit amet, consectetur ...
       adipiscing elit, sed do eiusmod tempor ...
       incididunt ut labore et dolore magna aliqua. ...
       Ut enim ad minim veniam, quis nostrud ...
       exercitation ullamco laboris nisi ut aliquip ...
       ex ea commodo consequat. Duis aute irure dolor ...
       in reprehenderit in voluptate velit esse ...
       cillum dolore eu fugiat nulla pariatur.';
3
4  lorem(1:100)
```

Základní maticové operace

- **sčítání** - přičtení skalární hodnoty, součet matic stejného rozměru

```
1 A = rand(5,5); B = rand(5,5);
2 C = A + 5;       % pricte 5 vsem prvkum matice
3 D = A + B;       % soucet matic
```

- **násobení** - pravidla maticového násobení (rozměry matic)

```
1 E = rand(5,3); F = rand(3,2);
2 G = E*F;         % [5x3]*[3x2] = [5x2]
```

Základní maticové operace - *pokračování*

- **tečkovaný operátor** - operace prvek po prvku pro matice stejného rozměru (násobení, mocnění, dělení prvek po prvku)

```
1  A = rand(5,3); B = rand(5,3);
2  C = A.*B;    % prvek A(i,j) vynasobi s B(i,j)
3  D = A.^B;    % dtto pro mocneni
4  E = A./B;    % a deleni
```

- matice musí mít shodné rozměry

Základní maticové operace - *pokračování*

- **transpozice** komplexní matice (*apostrof* komplexně sdružená k transponované, *s tečkovaným operátorem* bez komplexně sdružených hodnot)

```
1  A = [1 + 2i, 3, 2 - 5i; 2, 4, 6i; 1 + 1i,  5, 3];
2  B = A';      % matice transponovana komplexne ...
       sdruzena
3  C = A.';     % matice pouze transponovana
4
5  D = rand(3,3);
6  E = D';      % pro matici realnych hodnot nema ...
       vliv (komplexni cast = 0)
```

Vybrané užitečné funkce

- speciální proměnné

```
1 pi       % hodnota pi
2 eps      % strojove epsilon
3 inf      % nekonecno
4 NaN      % Not a number
5 i, j     % imaginarni jednotka
6 realmin % nejmensi realne cislo
7 realmax % nejvetsi realne cislo
```

Vybrané užitečné funkce - *pokračování*

- matice a maticové funkce

```
1  ones          % matice jednicek
2  zeros         % matice nul
3  eye           % jednotkova matice
4  rand          % matice nahodnych cisel
5  diag          % diagonalni matice
6
7  size          % velikost matice
8  length        % delka vektoru
9  det           % determinant
10 inv           % inverze
11 eig           % vlastni cisla
12 rank          % hodnost
13 find          % vyhledani prvku (A == 5, B ≥ 3)
14 all           % pravda pro vsechny nenulove
15 any           % pravda pro nejaky nenulovy
```

Vybrané užitečné funkce - *pokračování*

- matematické a statistické funkce

```
1  abs                % absolutni hodnota
2  sign               % signum
3  sin, sind          % sinus (radiany), sinus (stupne)
4  cos, cosd, ...     % cosinus
5  asin, acos, ...    % inverzni goniometricke funkce
6  exp                % exponenciala zakladu e
7  log, log10         % prirozeny/dekadicky logaritmus
8  round              % zaokrouhleni na nejblizsi cele
9  fix                % zaokrouhleni na nizsi cele
10 ceil, floor        % zaokrouhleni k +inf, -inf
11 real, imag         % realna a imaginarni cast
12 sort               % razeni matice
13 sum                % soucet prvku matice
14 min, max           % minimum, maximum
15 mean, median       % prumer, median
16 std, var           % smerodatna odchylka, variance
```

Vybrané užitečné funkce - *pokračování*

- aproximace a interpolace

```
1 polyfit      % polynomicky fit
2 polyval      % vypocet polynomu z koeficientu
3 spline       % spline interpolace
4 interp1      % 1-D interpolace
5 interp2      % 2-D interpolace
6 interpn      % n-D interpolace
7 griddata     % interpolace dat nerovnomerneho ...
      gridu
```

Vybrané užitečné funkce - *pokračování*

- relační operátory
  - $==$ je rovno
  - $\sim=$ není rovno
  - $<$ je menší
  - $<=$ je menší nebo rovno
  - $>$ je větší
  - $>=$ je větší nebo rovno
- logické operátory
  - && `AND` – stejná priorita jako `OR`
  - || `OR` – stejná priorita jako `AND`
  - $\sim$ `NOT` – nejvyšší priorita

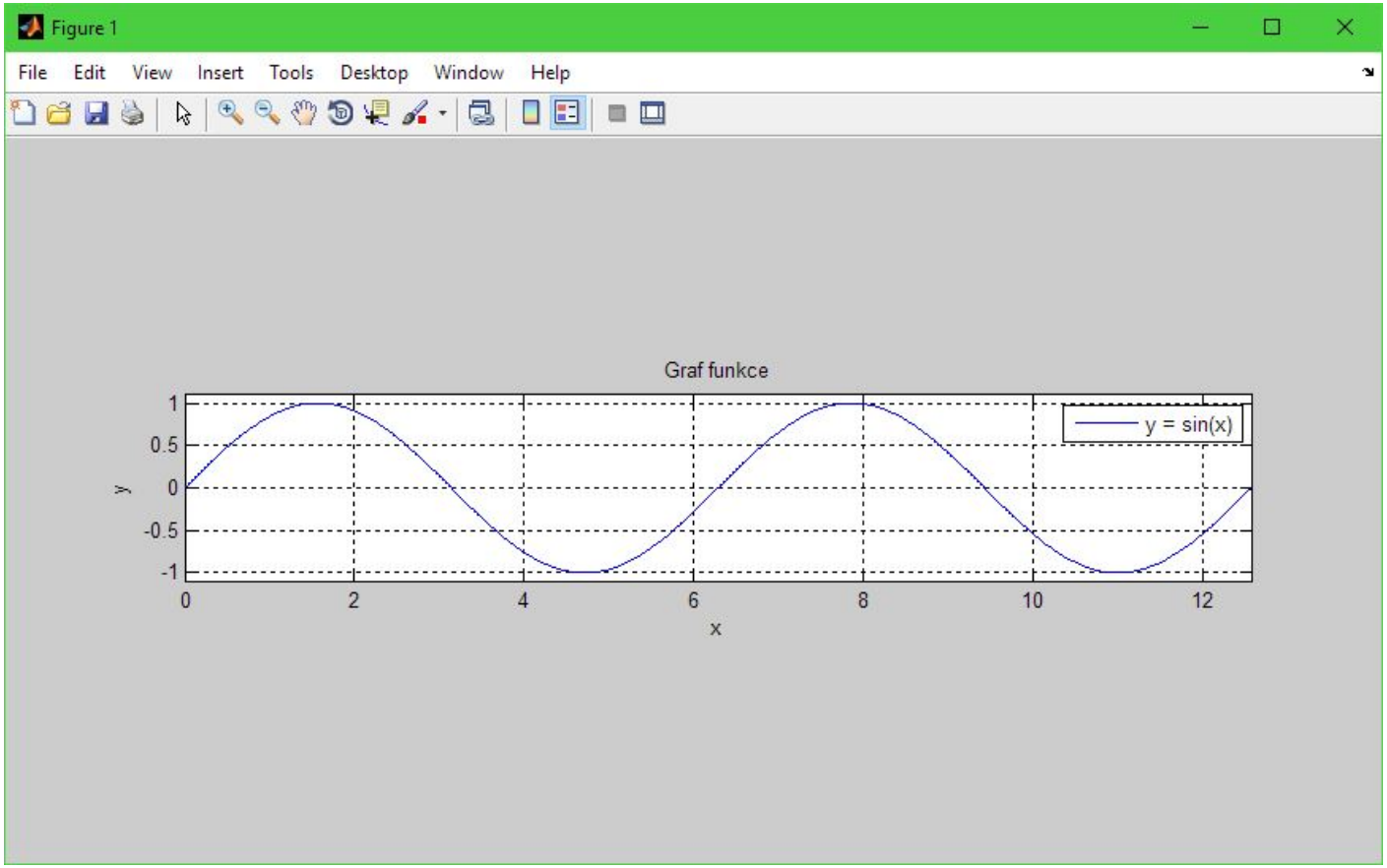
Základní grafy

Liniové 2D grafy

Základní 2D graf vytváříme příkazem `plot` (podrobně viz `doc plot`). Širokou škálou příkazů můžeme specifikovat zobrazované prvky.

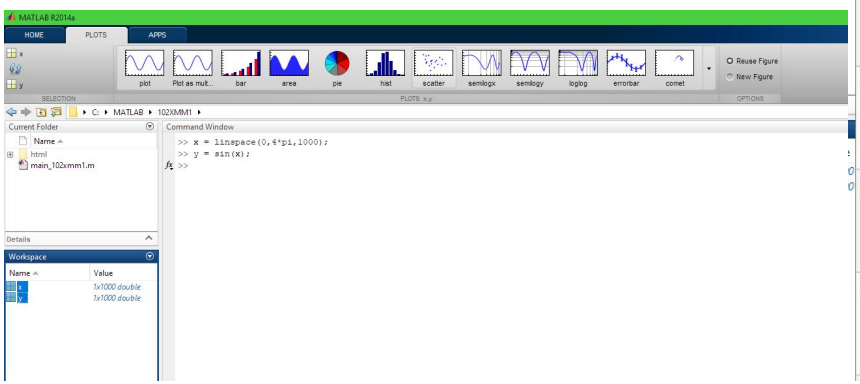
```
x = linspace(0,4*pi,1000);
y = sin(x);

figure(1)              % okno grafu
plot(x,y,'b-')         % modra linie
xlabel('x')            % popis osy x
ylabel('y')            % popis osy y
legend('y = sin(x)') % legenda
title('Graf funkce') % nadpis
grid on                % mrizka
axis equal             % rovnomerne osy
xlim([0 4*pi])         % limity osy x
ylim([-1.1 1.1])       % limity osy y
```

Liniové 2D grafy - *pokračování*



Graf z Workspace

Graf lze snadno generovat přímo pro proměnné z Workspace. Označíme-li vybrané proměnné, poté se odemknou možnosti v záložce Plots. V okně Figure následně můžeme měnit parametry grafu. Vše, co se provádí ve Figure, lze zapsat do kódu v Editoru. Kód grafu lze generovat z File/Generate Code.

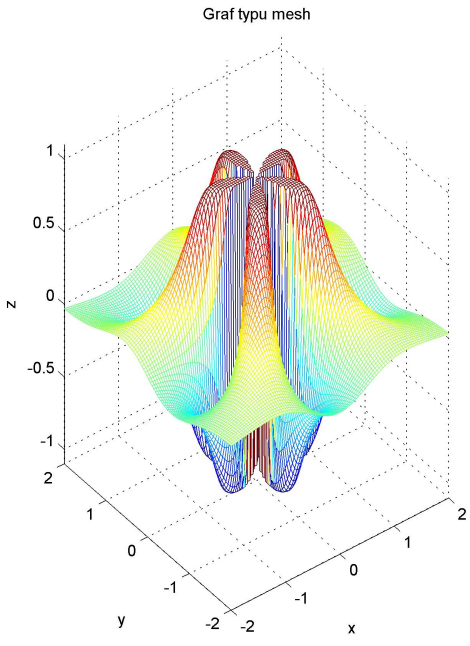


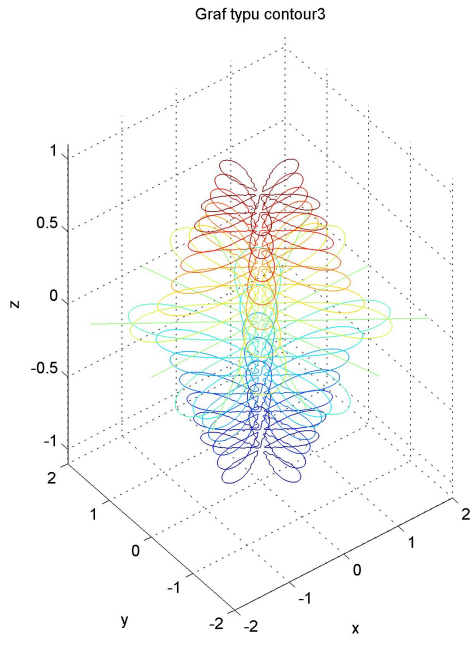
Další vybrané typy grafů

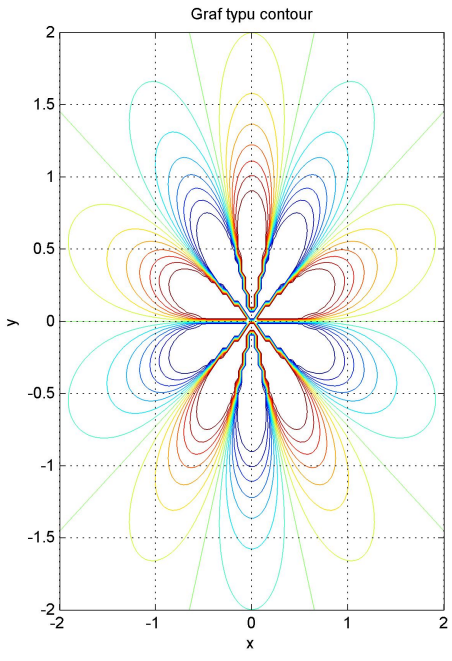
Existuje celá řada typů grafů pro různé vizualizace:

- liniové grafy – `plot`, `plotyy`, `semilogx`, `semilogy`, `loglog`, `area`, `errorbar`, `plot3`, `comet`,
- scatter grafy – `scatter`, `scatter3`,
- koláčové grafy – `pie`, `pie3`,
- histogramy – `hist`, `rose`,
- prostorové grafy – `surf`, `surfc`, `surfl`, `mesh`, `meshc`, `meshz`, `waterfall`, `ribbon`, `contour3`,
- a mnoho dalších ...

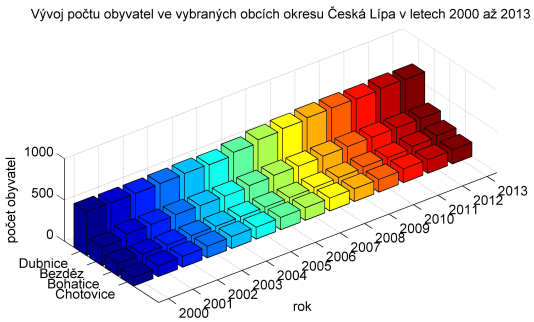
Další vybrané typy grafů - *pokračování*

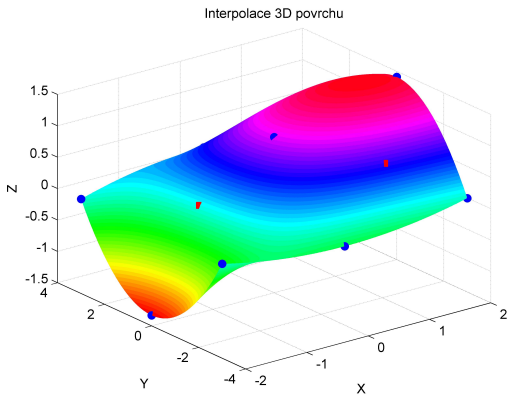


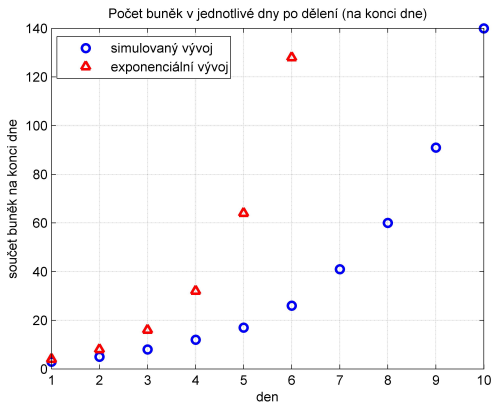




Další vybrané typy grafů - *pokračování*







Řízení toku programu

Řízení toku programu

- Podmínky – příkazy `if`, `switch`
- Cykly – příkazy `for`, `while`
- Vynucené zastavení cyklu – příkaz `break`
- Vynucené pokračování v další iteraci – příkaz `continue`

Podmínky – příkaz `if`

Testuje jednu nebo více podmínek na pravdu (`true`/`false`).

```
1  % Priklad jednoduche podminky
2  test = true;
3  if test
4      disp('Podminka byla splnena.');
5  end
6
7  % Priklad if/else
8  if test
9      disp('Podminka byla splnena.');
10 else
11     disp('Podminka nebyla splnena.');
12 end
```

Podmínky – příkaz `if` - *pokračování*

```
1  % Priklad if/elseif/else
2  test1 = false;
3  test2 = true;
4  if test1
5      disp('Podminka 1 byla splnena.');
6  elseif test2
7      disp('Podminka 1 nebyla splnena.');
8      disp('Podminka 2 byla splnena.');
9  else
10     disp('Nebyla splnena zadna z podminek.');
11 end
```

Podmínky – příkaz `if` - *pokračování*

Podmínka bude splněna, kdykoli je příkaz za klíčovým `if` splněn. K formulaci testovacích příkazů používáme *relační a logické operátory*:

$==$, $\sim=$	je rovno, není rovno
$<$, $<=$	je menší, menší nebo rovno
$>$, $>=$	je větší, větší nebo rovno
&&, $\|\|$, $\sim$	`AND`, `OR`, `NOT`

Dále lze využít další funkce (např. `all`, `any`), které vrací hodnoty `true`/`false` nebo jimi tvořené vektory.

Podmínky – příkaz `if` - *pokračování*

```
1  % Priklad if
2  testValue1 = 1e-5;
3  testValue2 = 1;
4  A = 0.05;
5  if ( A > testValue1 ) && ( A < testValue2 )
6      disp(['A je vetsi nez ', num2str(testValue1), ...
7          ' a mensi nez ',num2str(testValue2)]);
8  else
9      disp('A je mimo kriticky rozsah.');
10 end
```

Poznámka: Využili jsme funkci `num2str`*, která převádí numerickou hodnotu na textový řetězec.*

Podmínky – příkaz `switch`

Větvení programu na takové případy, kdy se dotazovaná proměnná **může rovnat** různým hodnotám.

```
1 switch dotazovana_promenna
2     case pripad_jedna
3     % telo programu pro pripad jedna
4     case pripad_dva
5     % ....
6     otherwise
7 end
```

Podmínky – příkaz `switch` - *pokračování*

Počet případů (`case`) není omezen.
Příkaz `otherwise` není povinný.

```
1 vek = 15;
2 switch vek
3     case 14
4         disp('Promenna vek je rovna 14.')
5     case 15
6         disp('Promenna vek je rovna 15.')
7     otherwise
8         disp('Promenna vek neni 14 ani 15.')
9 end
```

Cykly – příkaz `for`

Cykly využíváme, má-li se **opakovat** určitá část kódu (iterace, procházení prvků matice, načítání textu řádek po řádku, ...).
Za klíčovým slovem `for` následuje proměnná, jejíž hodnota se v průběhu cyklu (opakování) mění (konstantně narůstá, ...). Hodnoty této proměnné definujeme **vektorem**. **Délka** tohoto vektoru určuje **počet opakování**.

```
1 % Vypsani cisel 1 az 100
2 for n = 1:10
3     disp(['Cislo ' num2str(n)]);
4 end
```

Cykly – příkaz `for` - *pokračování*

```
1 % Vypis lichych radku matice
2
3 A = rand(100,5);     % vytvoreni matice
4
5 for idx = 1:2:size(A,1) % n je vektor 1, 3, ... ...
      az pocet radku matice
6
7       disp(A(idx,:));
8
9 end
```

Poznámka: Tento příklad lze řešit elegantně tzv. vektorizací (viz dále).

Cykly – příkaz `for` - *pokračování*

```
1  % Priklad jednoduche animace
2  x = linspace(0,2*pi,100);
3  y = sin(x);
4
5  figure(1)
6  xlabel('x'); ylabel('y');
7  grid on; hold on;
8  xlim([0 2*pi]); ylim([-1 1]);
9  for in = 1:length(y)
10     plot(x(in),y(in),'x');
11     pause(0.2);
12 end
13 hold off
```

Cykly – příkaz `while`

`While` cyklus probíhá do té doby, dokud je splněna vstupní podmínka. Nemusíme znát počet opakování cyklu.

```
1 while podminka_is_true
2     % kod vykonavany, je-li podminka splnena
3 end
```

```
1 value = 5;
2 while value > 0.1
3     value = value/2;
4     disp(['hodnota je: ',num2str(value)])
5 end
```

Cykly – příkaz `while` - *pokračování*

Newtonova metoda pro hledání druhé odmocniny

$$x = \sqrt{a} \quad \rightarrow \quad x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right), x_0 = a$$

```
1  a = 4;
2  x1 = a; diference = 10;      % nulta iterace
3  while abs(diference) > 1e-10
4      x2 = 1/2*(x1 + a/x1);    % update x2
5      diference = x2 - x1;     % update dif
6      x1 = x2;                 % update x1
7      disp(x2);                % zobrazeni
8  end
```

Vynucené zastavení cyklu – příkaz `break`

`break` – vynucené zastavení `for` nebo `while` cyklu

```
1 data = [1 2 3 NaN 5 6];
2 for in = 1:length(data)
3     if isnan(data(in))
4         disp(['NaN hodnota na pozici ', ...
              num2str(in)]);
5         break;
6     end
7 end
```

Vynucené pokračování v další iteraci – příkaz `continue`

`continue` – okamžité spuštění dalšího kroku cyklu s vynecháním zbytku aktuálního

```
1 vzdalenosti = [10 21 32 12 25 16];
2 for in = 1:length(vzdalenosti)
3     if vzdalenosti(in) > 15
4         disp(['Bod ' num2str(in) ' je moc daleko'])
5         continue;
6     end
7     disp(['Bod ' num2str(in) ' je ve vzdalenosti ...
          ' num2str(vzdalenosti(in)) ' m'])
8 end
```

Funkce

Uživatelsky definované funkce

- definice začíná klíčovým slovem `function`
- seznam volitelného počtu parametrů ohraničený `[ ]`, jeden parametr nemusí být v závorkách
- výstižné jméno funkce
- vstupní parametry za jménem v kulatých závorkách
- funkce ukládáme do samostatného souboru s příponou `.m`

```
1 function [vystupy] = jmeno(vstupy)
2 % popis funkce pro help a doc
3 vystupy = vstupy;
4 end
```

Uživatelsky definované funkce - *pokračování*

```
1  function I = intlich(x,y)
2  % I = intlich(x,y)
3  %    vypocte numericky integral lichobeznikovym ...
       pravidlem
4
5  a = x(1); b = x(end);
6  fa = y(1); fb = y(end);
7  N = length(x)-1;
8
9  I = (b - a)/N*((fa + fb)/2 + sum(y(2:end-1)));
10
11 end
```

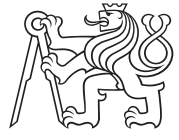
Funkce s volitelným počtem vstupních parametrů

- počet vstupních parametrů - `nargin`
- pole buněk vstupních parametrů - `varargin`

```
1  function N = myfcn(varargin)
2
3  N = nargin;
4
5  for in = 1:N
6      disp( varargin{in} );
7  end
8
9  end
```

In-line a lokálně definované funkce

- In-line funkce - definované v řádku
- Lokálně definované funkce (vnořené) - proměnné, které nejsou na vstupu, hledá v nadřazené funkci



In-line a lokálně definované funkce - *pokračování*

```
1  function main
2  % main   nadrazena funkce
3
4  % In-line funkce
5  fce1 = @(x) x.^2 + 3;
6
7  % Volani in-line funkce
8  a = 50;
9  b = fce1(a);
10
11 % Volani vnorene funkce
12 c = localFcn(b);
13
14     function out = localFcn(in)
15     % lokalne definovana funkce
16         out = in + a; % a je dedena promenna
17     end
18
19 end
```

Vektorizace

Vektorizace

- Vektorizace je jeden z nejsilnějších nástrojů Matlabu pro optimalizaci výpočtů na velkých souborech dat.
- Řada operací prováděných iterativně (pomocí cyklů apod.) lze řešit vektorizací a s využitím integrovaných funkcí Matlabu.
- Využíváme zpravidla **indexování** pomocí vektorů, které nahradí výběr pomocí cyklu.
- Pro hromadnou aplikaci operátoru na vybrané prvky používáme často **tečkovaný operátor**.

Vektorizace - *pokračování*

```
1  % Priklad vektorizace souctu kazdeho licheho ...
       prvku matice
2  A = rand(100000,1);
3
4  % Pomoci for cyklu
5  t1 = tic;                    % mereni casu
6  S1 = 0;
7  for in = 1:2:length(A)
8      S1 = S1 + A(in);         % scitani
9  end
10 S1
11 t1 = toc(t1)
12
13 % Vektorizaci
14 t2 = tic;
15 S2 = sum(A(1:2:end))         % soucet na vybrane prvky
16 t2 = toc(t2)
```

Vektorizace - *pokračování*

```
1  % Priklad souctu ctvercu vybranych prvku matice
2  A = rand(100,100);
3
4  % Pomoci for cyklu
5  t1 = tic;                    % mereni casu
6  S1 = 0;
7  for in = 1:2:size(A,1)   % cyklus pres radky
8      for jn = 1:2:size(A,2)   % cyklus pres sloupce
9          S1 = S1 + A(in,jn)^2;
10     end
11 end
12 S1
13 t1 = toc(t1)
14
15 % Vektorizaci
16 t2 = tic;
17 S2 = sum(sum( A(1:2:end,1:2:end).^2 ))   % mocneni ...
       teckovanym operatorem a soucet funkcemi
18 t2 = toc(t2)
```

Vektorizace - *pokračování*

```
1  % Nejkratsi vzdalenost mesta od bodu
2  X = [1034250 1034157 1034874];
3  Y = [ 733245  733105  733990];
4
5  X0 = 1034200; Y0 = 733100;
6
7  % For cyklem
8  R = zeros(size(X));
9  for n = 1:length(X)
10     R(n) = sqrt((X0 - X(n))^2 + (Y0 - Y(n))^2);
11 end
12 Rmin = min(R)
13
14 % Vektorizaci
15 Rmin2 = min( sqrt((X0 - X).^2 + (Y0 - Y).^2) )
```

Symbolické výpočty

Symbolické proměnné

V Matlabu lze pracovat i s analytickými vztahy, derivovat, integrovat, apod., pomocí `symbolických` proměnných.

```
1 % Definice symbolickych promennych
2 syms a b c
3
4 % Pouziti
5 M = a*b;
6 N = a/c;
```

Základní symbolické operace

```
1  % Definice symbolickych promennych
2  syms a b c
3
4  % Zakladni algebraicke operace
5  M = a*b; N = a/c;
6
7  % Derivace
8  diff( cos(b) )
9  diff( sin(a*b)*exp(c), b ) % parcialni derivace ...
       podle b
10
11 % Integrace
12 int( cos(b) )
13 int( sin(a)^2, a , 0 , 1 ) % urcity integral ...
       podle a od 0 do 1
```

Díky za pozornost :)

Ing. Petr Pokorný, Ph.D.
Fakulta stavební ČVUT v Praze, katedra fyziky
Skupina aplikované optiky
petr.pokorny@fsv.cvut.cz