Matlab

Stručné seznámení se základními výpočetními funkcemi

Aleš Čepek

155GIT1

2021-02-17

Copyright © 2013, 2020 Aleš Čepek
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Matlab

The Language of Technical Computing

Kde získat?

ČVUT http://download.cvut.cz

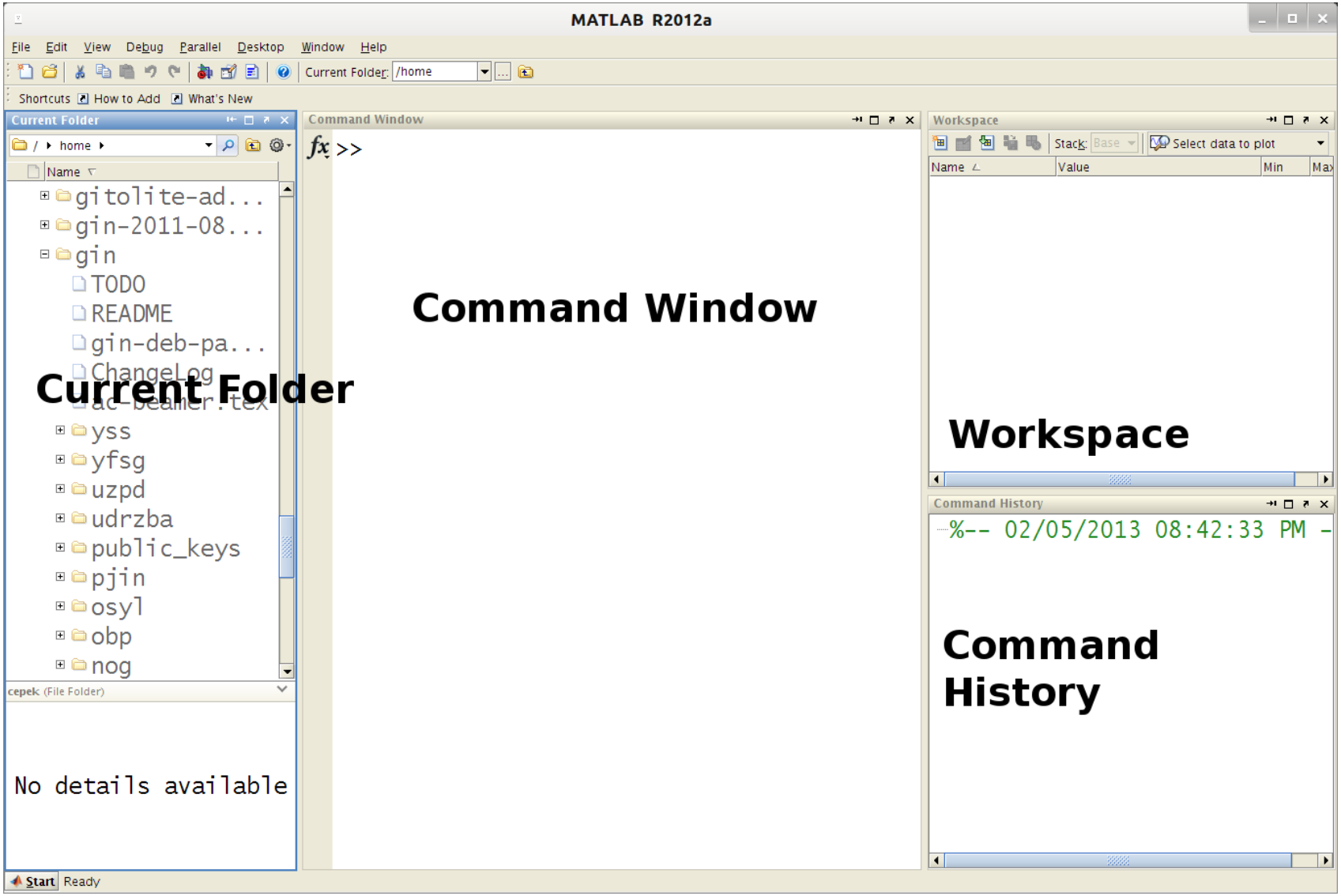
Jaké jsou alternativy?

Octave http://www.gnu.org/software/octave/

Pro první seznámení a experimenty dostatečně kompatibilní.
Legálně k dispozici pro MS Windows i GNU/Linux

Instalace pro Windows http://octave.sourceforge.net

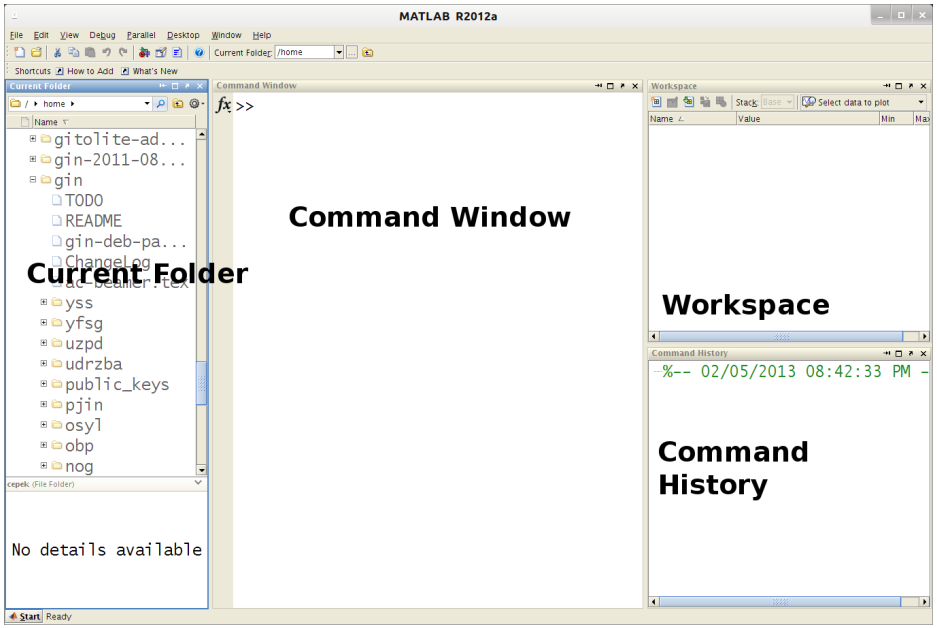
Matlab Desktop



Matlab Desktop

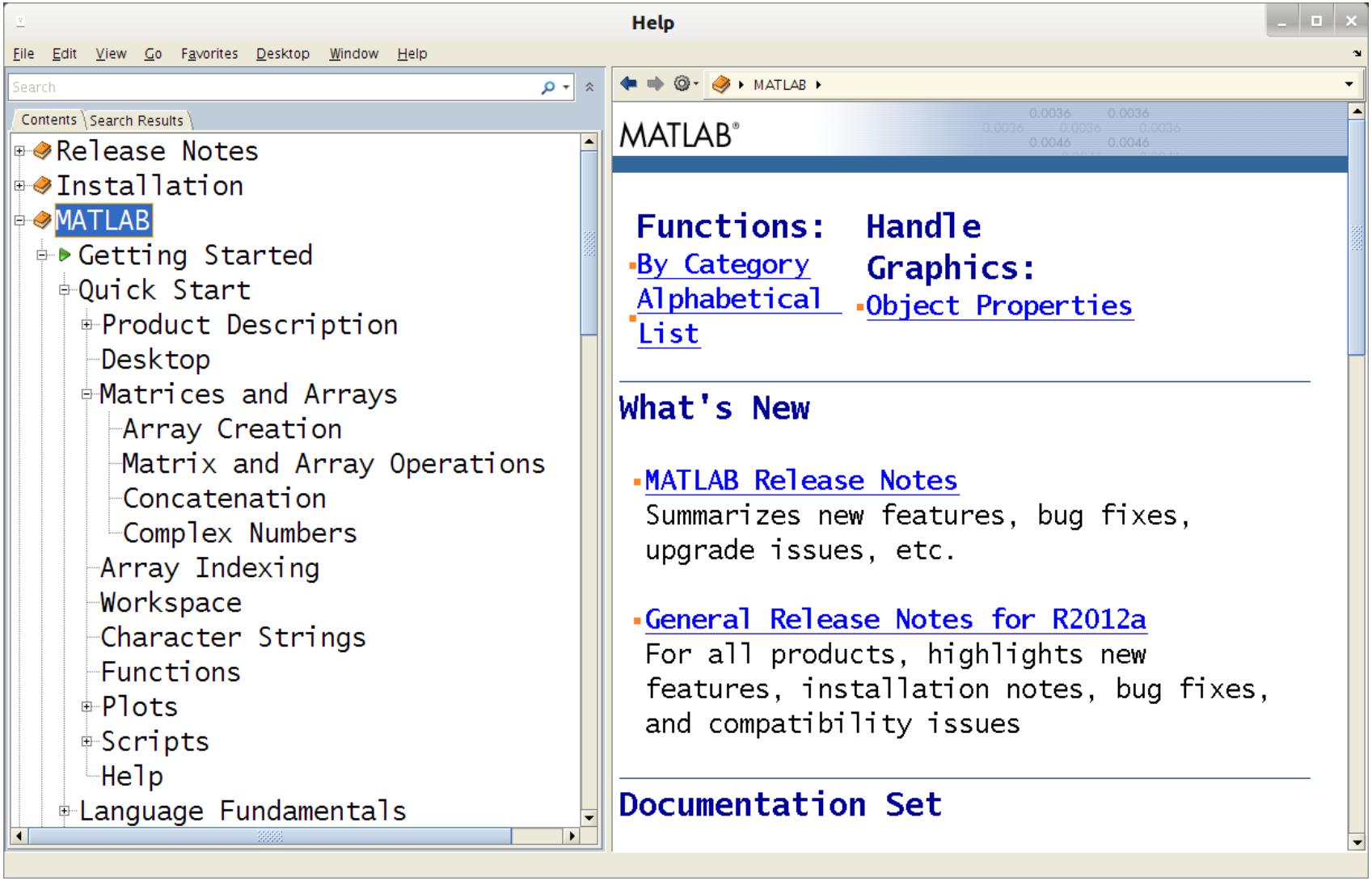
Zobrazní čtyř základních oken matlabu je ovládáno z menu Desktop, jejich přepínání myší nebo pomocí klávesových zkratek

0	Command Window	Ctrl+0
1	Command History	Ctrl+1
2	Current Folder	Ctrl+2
3	Workspace	Ctrl+3



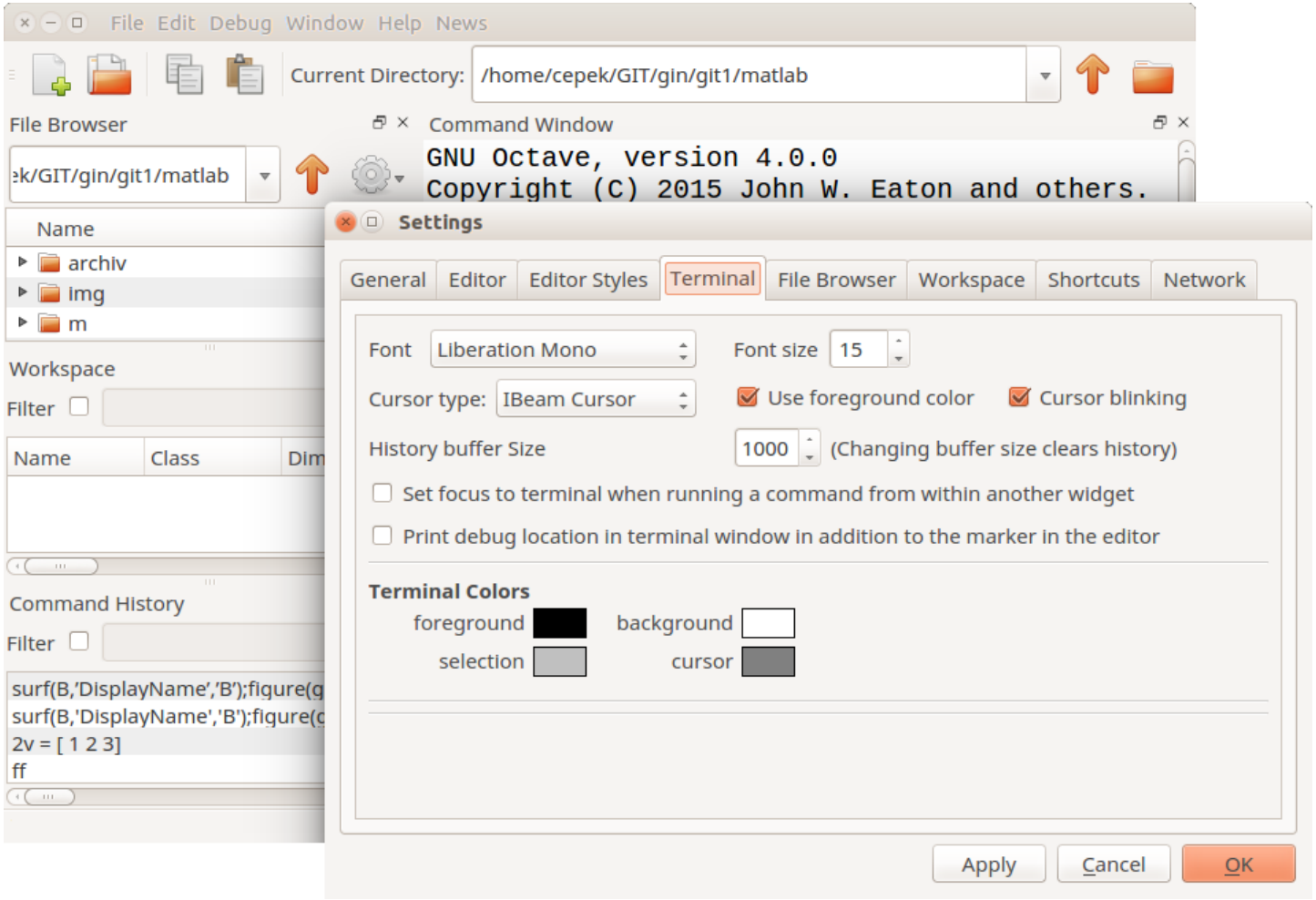
Základní okna mohou být umístěna na ploše Desktopu nebo samostatně (příkazy dock a undock window pomocí ikonky v horní liště okna)

Matlab Help


Help
File Edit View Go Favorites Desktop Window Help
Search
Contents Search Results
Release Notes
Installation
MATLAB
Getting Started
Quick Start
Product Description
Desktop
Matrices and Arrays
Array Creation
Matrix and Array Operations
Concatenation
Complex Numbers
Array Indexing
Workspace
Character Strings
Functions
Plots
Scripts
Help
Language Fundamentals
MATLAB
MATLAB®
Functions:
By Category
Alphabetical List
Handle Graphics:
Object Properties
What's New
MATLAB Release Notes
Summarizes new features, bug fixes, upgrade issues, etc.
General Release Notes for R2012a
For all products, highlights new features, installation notes, bug fixes, and compatibility issues
Documentation Set

Nastavení velikosti fontu v GNU Octave

Octave GUI $\longrightarrow$ Preferences $\longrightarrow$ Terminal



Octave v prostředí editoru GNU Emacs

```
emacs@dodo3
File  Edit  Options  Buffers  Tools  Octave  Help
Save  Undo

A = [1 2 3 5]

2*A - A'

-:**-  aaa.m          Top (4,9)      (Octave
;; This buffer is for text that is not savec
;; To create a file, visit it with C-x C-f a

Klávesové zkratky

C-c C-i C-l        zpracuje jeden řádek
C-c C-i C-a        zpracuje celý buffer

C-x h dell         vymaže buffer

C-o                přepínání bufferů

M-x octave-mode

U:**-  *scratch*      All (10,0)     (Lisp In
```

```
octave> A = [1 2 3 5]
A =

   1   2   3   5

octave>
octave> 2*A - A'
ans =

   1   3   5   9
   0   2   4   8
  -1   1   3   7
  -3  -1   1   5

octave>

U:**-  *Inferior Octave*   All (17,0)
```

Matice

Všechny proměnné v Matlabu jsou matice.[1] Řádkové a sloupcové vektory jsou matice s jedním řádkem, resp. sloupcem. Skalární hodnoty jsou matice s jedním řádkem a jedním sloupcem.

```
>>   A = [1, 2, 3, 4; 5, 6, 7, 8]
A =

     1     2     3     4
     5     6     7     8
```

Matice musí být obdélníkové.

```
>> M = [1, 2, 3, 4;
        1, 2, 3]
Error using vertcat
CAT arguments dimensions are not consistent.
```

[1] Přesněji řečeno, všechny proměnné Matlabu jsou vícerozměrná pole, matice je zvláštním případem dvourozměrného pole.

Vektory a skaláry

Vektory jsou speciálním případem matic s jedním řádkem nebo jedním sloupcem.

```
>> b = [1; 3; 5; 7; 9]     % sloupcovy vektor (matice 5x1)
b =
     1
     3
     5
     7
     9

>> c = [1, 2, 3, 4, 5]     % radkovy vektor (matice 1x5)
c =
     1     2     3     4     5
```

Skaláry jsou matice s jedním jediným prvkem.

```
>> a = 15
a =
    15
```

Proměnné

Proměnné jsou v Matlabu vytvářeny při prvním použití. Jméno musí začínat písmenem, mohou následovat písmena, číslice a znak podtržítko. Velká a malá písmena se rozlišují.

```
>> 2b = [10; 11; 12; 13]
 2b = [10; 11; 12; 13]
 |
Error: Unexpected MATLAB expression.
```

Přehled definovaných proměnných je uveden v *Desktopu* v okně *Workspace*, kde je lze přejmenovávat, editovat nebo graficky zobrazit jejich hodnoty (např. z nabídky přes pravé tlačítko myši).

```
B = rand(4,6);
surf(B,'DisplayName','B');figure(gcf)
```

Zrušit vytvořené objety umožňuje příka `clear`,

Indexování prvků

Prvky matice se indexují pomocí čísla řádku a čísla sloupce, např. `A(1,1)` je prvek na prvním řádku v prvním sloupci (indexování začíná od 1), odpovídá tedy prvku $\mathbf{a}_{1,1}$ z níže uvedeného příkladu. Prvek `A(2,3)` označuje hodnotu $\mathbf{g}_{2,3}$.

Matice jsou interně ukládány po sloupcích, v pořadí, které udává levý horní index z obrázku. Vektory, které jsou maticemi, můžeme proto indexovat jen jediným indexem, pořadovým číslem prvku.

Jednorozměrný index bychom mohli použít i pro matici, `A(8)` je totéž co `A(2,3)` a označuje prvek $\mathbf{g}_{2,3}$.

$$\mathbf{A} = \begin{pmatrix} {}^{1}\mathbf{a}_{1,1} & {}^{4}\mathbf{b}_{1,2} & {}^{7}\mathbf{c}_{1,3} & {}^{10}\mathbf{d}_{1,4} \\ {}^{2}\mathbf{e}_{2,1} & {}^{5}\mathbf{f}_{2,2} & {}^{8}\mathbf{g}_{2,3} & {}^{11}\mathbf{h}_{2,4} \\ {}^{3}\mathbf{i}_{3,1} & {}^{6}\mathbf{j}_{3,2} & {}^{9}\mathbf{k}_{3,3} & {}^{12}\mathbf{l}_{3,4} \end{pmatrix}$$

Indexování prvků - příklad

Pro matici **A** s hodnotami

```
A = [1.11 4.12 7.13 10.14
     2.21 5.22 8.23 11.24
     3.31 6.32 9.33 12.34]
```

jsou hodnoty jednorozměrných prvků

```
[A(1), A(2), A(3), A(4), A(12)]

1.1100    2.2100    3.3100    4.1200   12.3400
```

Stejné hodnoty odpovídají dvourozměrným indexům matice

```
[A(1,1), A(2,1), A(3,1), A(1,2), A(3,4)]

1.1100    2.2100    3.3100    4.1200   12.3400
```

Viz soubor `indexovani_prvku.m`

Operátor dvojtečka (:)

Operátor dvojtečka slouží vytváření submatic, vektorů, polí indexů a programování cyklů. Výraz `j:k` generuje posloupnost hodnot od `j` do `k` s krokem 1 (nebo prázdnou posloupnost, pokud `j > k`).

```
>> j=-2; k=5; j:k
ans =
    -2    -1     0     1     2     3     4     5
>> j=8; k=5; j:k
ans =
   Empty matrix: 1-by-0
```

Výraz `j:i:k` generuje obdobně posloupnost s krokem `i` pro hodnoty menší nebo rovno `k` (nebo prázdnou posloupnost, pokud `i == 0`, nebo `i > 0` a `j > k`, nebo `i < 0` a `j < k`).

```
>> j=1; i=3; k=11; j:i:k
ans =
     1     4     7    10
>> j=0; i=0.2; k=1; j:i:k
ans =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Submatice

Operátor dvojtečka umožňuje vytváření submatic pro zadaný interval, resp. množinu, řádků a sloupců.

Mějme matici A se třemi řádky a čtyřmi sloupci, jejíž hodnoty korespondují s příslušnými indexy

```
>> A = [11 12 13 14; 21 22 23 24; 31 32 33 34]
A =
    11    12    13    14
    21    22    23    24
    31    32    33    34
```

Submatici tvořenou řádky `r1:r2` a sloupci `s1:s2` získáme s použitím operátoru dvojtečka jako `A(r1:r2,s1:s2)`, například

```
>> A(2:3,3:4)
ans =
    23    24
    33    34
```

Submatice (pokračování)

Pokud v definici submatice požadujeme všechny hodnoty daného indexu, můžeme použít operátor dvojtečka bez parametrů, jak ukazuje následující příklad (druhý řádek matice A).

```
>> A(2,:)

ans =
    21    22    23    24
```

Pokud se v submatici vyskytuje horní hodnota daného indexu, můžeme ji nahradit symbolickým označením `end`, což je konstrukce, kterou budeme později používat při programování (poslední sloupec matice A)

```
>> A(:,end)

ans =
    14
    24
    34
```

Rozšiřování matic

Pokud se pokusíme odvolat na prvek mimo dané dimenze matice, dojde k chybě.

```
>> A(5,6)
Index exceeds matrix dimensions.
```

Pokud ale na tuto pozici vložíme hodnotu, rozšíří se dimenze matice tak, aby bylo prvek možno uložit a ostatní doplněné prvky se nastaví na nulu.

```
>> A(5,6) = 56

A =
    11    12    13    14     0     0
    21    22    23    24     0     0
    31    32    33    34     0     0
     0     0     0     0     0     0
     0     0     0     0     0    56
```

Tato zdánlivě užitečná vlastnost může být v praxi velmi nebezpečná.

Maticové sčítání

K matici můžeme přičíst skalární hodnotu, která se přičte ke všem jejím prvkům.

```
>> A + 1000
ans =
        1011        1012        1013        1014
        1021        1022        1023        1024
        1031        1032        1033        1034
```

Sčítat můžeme pouze matice stejného rozměru.

```
>> p = [1 2; 3 4]; q = [10 20; 30 40]; p+q
ans =
    11    22
    33    44
```

```
>> A + p
Error using  +
Matrix dimensions must agree.
```

Maticové násobení

Matici můžeme násobit skalárem (přenásobí se všechny její prvky).

Při násobení matic musí být počet sloupců první matice roven počtu řádků druhé matice.

```
>> p*q
ans =                    % p =                q =
    70   100             %      1   2          10  20
   150   220             %      3   4          30  40
```

```
>> A*p
Error using  *
Inner matrix dimensions must agree.
```

Matlab poskytuje operátor .* který vynásobí jednotlivé prvky se stejnými indexy.

```
>> p.*q
ans =                    % p =                q =
    10    40             %      1   2          10  20
    90   160             %      3   4          30  40
```

Další příklady operací po prvcích

```
p./q
ans =                                   % p =                    q =
    0.1000    0.1000                    %       1    2              10   20
    0.1000    0.1000                    %       3    4              30   40
```

```
>> p.^2
ans =
     1     4
     9    16
```

```
>> q.^p

ans =
        10         400
     27000     2560000
```

Transponování matice

Mějme komplexní matici Z

```
>> Z = [ 2+3i, 0, 5, 7i;  21, 6+2i, 0, 14; 8, i, 1-i, 9+4i ]

Z =  2+3i    0       5      0+7i
    21       6+2i    0     14
     8       0+i     1-i    9+4i
```

Operátor transponování Z' (apostrof) vytváři matici transponovanou a komplexně sdruženou k matici Z, tj. matici konjugovanou. Potřebujeme-li pouze transponovanou matici bez komplexně sdružených hodnot, použijeme operátor Z.' (tečka apostrof).

```
Z' = 2-3i   21      8                Z.'= 2+3i   21      8
     0       6-2i   0-i                    0       6+2i   0+i
     5       0      1+i                    5       0      1-i
     0-7i   14      9-4i                   0+7i   14      9+4i
```

Spojování matic (concatenation)

Matice je dvojrozměrné obdélníkové schéma. Pokud mají dvě matice stejný počet řádků, resp. sloupců, můžeme je spojit horizontálně, resp. vertikálně.

```
>> a = [1 2 3 4; 5 6 7 8]; b = [-9 -7 -5 -3; -1 -3 -5 -7];
>> [a, b]

ans =

     1     2     3     4    -9    -7    -5    -3
     5     6     7     8    -1    -3    -5    -7

>> [a; b]

ans =

     1     2     3     4
     5     6     7     8
    -9    -7    -5    -3
    -1    -3    -5    -7
```

Odstranění řádků/sloupců matice

Vybrané řádky/sloupce v matici můžeme odstranit tak, že do nich přiřadíme prázdný řádek/sloupec `[]`.

```
>> T = magic(4)
T = 16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
>> T(2,:) = []                % odstranit druhy radek
T = 16     2     3    13
     9     7     6    12
     4    14    15     1
```

```
>> T(:,[2 4]) = []            % odstranit druhy a ctvrty sloupec
T = 16     3
     9     6
     4    15
```

Maticové funkce a speciální proměnné

`zeros`	matice se samými nulami
`ones`	matice se samými jedničkami
`eye`	jednotková matice
`rand`	matice s prvky s rovnoměrným náhodným rozdělením <0,1)
`diag`	diagonální matice
`size`	rozměry matice
`length`	délka vektoru (dimenze)
`det`	determinant
`inv`	inverzní matice
`eig`	vlastní čísla a vlastní vektory matice
`rank`	hodnost matice
`find`	vyhledávání prvků (např. find(A > 0.4))
`pi`	hodnota π
`eps`	*počítačové epsilon*
`inf`	nekonečno
`NaN`	*Not a number*, 0/0
`i, j`	imaginární jednotka $\sqrt{-1}$
`realmin`	nejmenší použitelné reálné číslo
`realmax`	největší použitelné reálné číslo

Relační a logické operátory

`<`	menší než
`<=`	menší nebo rovno
`>`	větší než
`>=`	větší nebo rovno
`==`	je rovno
`~=`	není rovno

`~`	`NOT` (nejvyšší priorita)
`&`	`AND` (stejná priorita jako `OR`)
`|`	`OR` (stejná priorita jako `AND`)

Základní matematické funkce

`abs`	absolutní hodnoty prvků matice
`sign`	funkce *signum*
`sin, cos,...`	goniometrické funkce
`asin ,acos,...`	inverzní goniometrické funkce
`exp`	exponenciála o základu e, tj. e^x
`log, log10`	přirozený a dekadický logaritmus
`ceil, floor`	zaokrouhlení směrem k +nekonečnu, resp. -nekoneč
`round`	zaokrouhlení k nejbližšímu celému číslu
`fix`	zaokrouhlení k nejbližšímu celému číslu směrem k nu
`real, imag`	reálná a imaginární část komplexního čísla
`sort`	setřídění prvků vzestupně
`sum, prod`	suma, součin prvků pole
`max, min`	maximum, minimum pole
`mean, median`	průměr, mediám prvků pole
`std, var`	směrodatná odchylka, variance prvků pole

. . . a mnoho dalších.

Textové řetězce

Matlab nabízí bohatou sadu funkcí pro práci s řetězci. V úvodním seznámení se omezíme na základní operace s texty. Řetězec (*string*) je posloupnost znaků, se kterými budeme pracovat analogicky jako s číselnými vektory. Typ řetězce je v Matlabu `char`.

```
>> retezec = 'posloupnost znaku uzavrena do apostrofu (')'

retezec =
posloupnost znaku uzavrena do apostrofu (')

>> retezec(1:11)

ans =
posloupnost
```

```
>> class(retezec)

ans =
char
```

```
>> class(654.37)

ans =
double
```

Skripty (první zmínka)

Všechny příkazy, které zadáváme manuálně v příkazové řádce Matlabu (*Command Window*) můžeme zapsat do textového souboru a opakovaně je dávkově spouštět. Takovému textovému souboru se říká *skript*.

- Soubor skriptu musí mít příponu `.m` a pro jeho jméno musí být splněny stejné požadavky jako pro jméno proměnné (aby jej mohl Matlab spouštět).
- Skript můžeme vytvořit příkazem `edit` *`jméno souboru`*, otevře se okno editoru, ve kterém můžeme skript spouštět a ladit.
- Skripty nejsou soubory Matlabu s příponou `.mat`
- Komentáře začínají znakem % a jsou ukončeny koncem řádku. Blok textu můžeme označit jako komentář, pokud začíná dvojicí znaků `%{` na samostatném řádku a končí symetricky dvojicí znaků `}%` (také na samostatném řádku).
- Skript se z příkazové řádky spouští zadáním jména, případně příkazem `run` *`jméno`*, pokud se nenachází v pracovním adresáři (nebo na cestě). Pracovní adresář lze změnit příkazem `cd`. Cestu lze rozšířit příkazem `addpath`.

Příklad - řešení soustavy lineárních rovnic

$$\mathbf{Ax} = \mathbf{b}$$

```
A = [17    24     1     8    15
     23     5     7    14    16
      4     6    13    20    22
     10    12    19    21     3
     11    18    25     2     9]
b = [175; 190; 245; 190; 175]

x = inv(A)*b
```

```
x =

    1.0000
    2.0000
    3.0000
    4.0000
    5.0000
```

Příklad - řešení soustavy lineárních rovnic (pokračování)

Soustavu z předchozího příkladu $\mathbf{Ax} = \mathbf{b}$ je ale možné (a vhodné) řešit i jinak

```
x = A\b;
```

Pro čtvercovou matici **A** řeší operátor \ soustavu pomocí *LU* rozkladu, pro obdélníkové matice pomocí *QR* rozkladu.

Pro řešení soustav lineárních rovnic je k dispozici funkce linsolve

```
  x = linsolve(A, b)
```

nebo

```
  opts = struct("TRANSA", true);
  x = linsolve(A', b, opts)
```

kde volitelný parametr (struktura opts) umožňuje specifikovat vlastnosti matice A (LT dolní trojúhelníková, UT horní trojúhelníková, UHESS horní Hessenbergova, SYM symetrická, POSDEF pozitvně definitní, TRANSA transponovaná).

Příklad - lineární transformace souřadnic v rovině

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix} + \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

```
% vektor souradnic bodu (1. radek x_i, 2. radek y_i
x = [100.00 200.00 315.65 587.65 417.35;
     100.00 489.32 780.65 249.35 105.16]
x(3,:) = 1;

% koeficinety transformace
a =  0.987;    b = -0.008;    c =  0.002;
d =  1.032;    e = 10.321;    f = 20.084;

T = [a b e; c d f];  % transformacni matice

X = T*x   % linearni transformace souradnic v rovine
```

```
X =
  108.2210  203.8064  315.6223  588.3367  421.4042
  123.4840  525.4622  826.3461  278.5885  129.4438
```

Příklad - kolineární (projektivní) transformace ve 2D

$$X = \frac{ax + by + c}{gx + hy + 1}, \qquad Y = \frac{dx + ey + f}{gx + hy + 1}$$

```
x = [100.00 200.00 315.65 587.65 417.35;
     100.00 489.32 780.65 249.35 105.16]
x(3,:) = 1;

a =  0.987;    b = -0.008;    c =  0.002;
d =  1.032;    e = 10.321;    f = 20.084;
g =  0.000087; h = -0.000154;

T = [a b e; c d f];

X = T*x
q = [g h 1]*x;    X(1,:) = X(1,:)./q;    X(2,:) = X(2,:)./q;
```

```
X =
    108.9510  216.3448  347.8923  580.9439  413.0948
    124.3169  557.7891  910.8337  275.0878  126.8914
```

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Vstup a výstup externích dat

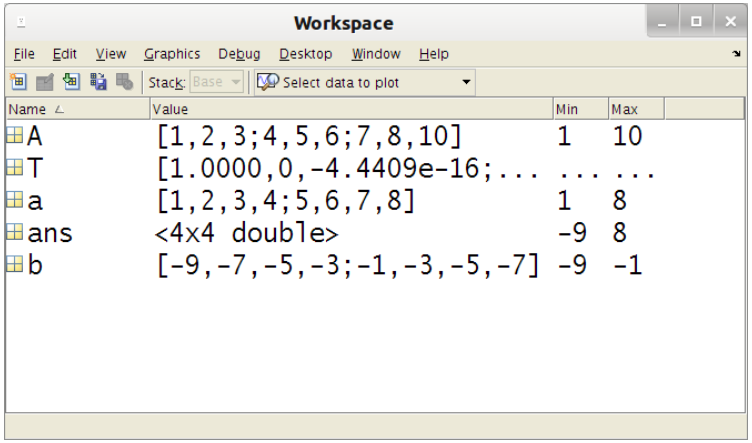
Matlab poskytuje širokou škálu nástrojů pro správu dat, základní kategorie jsou následující (členění podle oficiální dokumentace)

- Data and File Management
- Data import and export, workspace, files and folders
- Workspace Variables
- Data Import and Export
- File Operations
- Search Path
- Operating System Commands
- Internet File Access
- Serial Port Devices

V našem stručném úvodu se v této sekci zmíníme o správě proměnných *Workspace*, funkcích pro správu souborů a adresářů a zaměříme se na **čtení a zápis matic z/do textových souborů**.

Workspace

Workspace (pracovní prostor) obsahuje všechny proměnné vytvořené v Matlabu nebo importované ze souborů nebo jiných programů. Proměnné jsou zobrazeny ve stejnojmenném panelu, kde je můžeme rušit, editovat, přejmenovávat a také vizualizovat jejich hodnoty.



Obsah workspace zobrazíme příkazem `whos`.

```
>> whos
  Name        Size                Bytes  Class     Attributes

  A           3x3                    72  double
  T           3x3                    72  double
  a           2x4                    64  double
  ans         4x4                   128  double
  b           2x4                    64  double
```

Workspace (pokračování)

- proměnné nejsou persistentní, existují pouze po dobu běhu Matlabu
- proměnné workspace můžeme uložit to souboru příkazem `save` *`jmémo souboru`*
- proměnné uložené příkazem `save` jsou uloženy v komprimovaném souboru s příponou `.mat`
- proměnné uložené v souboru `.mat` můžeme načíst příkazem `load` *`jmémo souboru`*
- všechny proměnné uložené ve workspace odstraníme příkazem `clear all`
- příkazem `clear` *`seznam jmen`* můžeme vymazat pouze uvedené proměnné

Příkazy `load` a `save` lze použít i pro import a export dat z textových souborů; soubory `.mat` mohou být uloženy jako textové, implicitně se zapisují jako binární, je to ale řešení poměrně nesystémové a nebudeme je proto používat.

Příkazy pro práci se soubory a adresáře

cd	Změna pracovního adresáře
copyfile	Kopírování souboru nebo adresáře
delete	Odstranění souborů nebo grafických objektů
dir	Výpis obsahu adresáře
fileattrib	Nastavení atributů pro soubor nebo adresáře
filebrowser	Otevření panelu pro prohlížení běžného adresáře
isdir	Určuje zda se jedná o adresář
lookfor	Hledá klíčové slovo ve všech položkách nápovědy
ls	Výpis obsahu adresáře
matlabroot	Kořenový adresář
mkdir	Vytvoření nového adresáře
movefile	Přesun souboru nebo adresáře
pwd	Jméno běžného adresáře
recycle	Nastavuje volbu pro přesunuté zrušených souborů do koše
rmdir	Odstranění adresáře
tempdir	Jméno systémového dočasného adresáře
toolboxdir	Kořenový adresář pro zadaný toolbox
type	Zobrazí obsah souboru
visdiff	Porovná dva souboru, MAT, binarní, Zip soubory nebo adresáře
what	Vypíše soubory z adresáře MATLAB

Formátovaný zápis matice do textového souboru

```
M = [magic(3) [10;20;30]]

M =
     8     1     6    10
     3     5     7    20
     4     9     2    30
```

Matice se do výstupního proudu zapisují po sloupcích.

```
fid = fopen('demo.txt','w');
fprintf(fid,'%6.2f  %6.2f  %6.2f\n',M);
fclose(fid);
```

```
>> type demo.txt

  8.00    3.00    4.00
  1.00    5.00    9.00
  6.00    7.00    2.00
 10.00   20.00   30.00
```

Otevření a zavření souboru fopen a fclose

Na soubor se v našem kurzu budeme dívat jako na posloupnost bytů, které budeme zpracovávat jako *proud dat*, tj. budeme je sekvenčně číst nebo zapisovat.[2]

- Každý soubor musíme před jeho použitím otevřít příkazem `fopen`, kde povinným parametrem je jméno souboru, druhým volitelným parametrem jsou atributy připojení.
- Návratovou hodnotou funkce `fopen` je celé číslo, identifikátor souboru nebo -1 (pokud se otevření souboru nepovedlo). Hodnoty 0, 1 a 2 jsou rezervovány pro standardní vstup, standardní výstup a standardní chybový výstup.
- Volání funkce `fopen` pouze s jedním parametrem otevře soubor pouze pro čtení a je ekvivalentní volání `fopen(`*jméno souboru*`,'r').`
- Funkce `fclose(fid)` zavře daný soubor, návratová hodnota je 0 nebo -1 (došlo k chybě). Volání funkce `fclose('all')` zavře všechny otevřené souboru (kromě fid 0, 1 a 2).

[2] Většina moderních operačních systémů pod vlivem Unixu považuje soubor za prostou posloupnost bytů a další interpretaci přenechává na aplikačních programech nebo knihovnách. http://cs.wikipedia.org/wiki/Soubor

Atributy otevření souboru

Druhý volitelný parametr funkce `fopen` určuje atributy otevření souboru

```
FID = fopen('jmeno_souboru', 'atributy')
```

- 'r' otevření souboru pro čtení (implicitní hodnota)
- 'w' otevření souboru pro zápis; zruší existující obsah
- 'a' otevře nebo vytvoří soubor pro zápis; připojuje data na konec souboru
- 'r+' otevře existující soubor pro čtení a zápis (nevytváří nový soubor)
- 'w+' otevře nebo vytvoří soubor pro čtení a zápis; zruší existující obsah
- 'a+' otevře nebo vytvoří soubor pro čtení a zápis; připojuje data na konec souboru
- 'W' otevře soubor pro zápis bez automatického vyprazdňování
- 'A' otevře soubor pro *append* bez automatického vyprazdňování

Další parametry, vesměs specifické pro různé operační systémy, nebudeme používat.

Funkce fprintf(FID, FORMAT, seznam objektu)

Formátovací funkce `fprintf` má dva povinné parametry (FID identifikátor souboru a FORMAT formátovací řetězec), následuje seznam objektů, které mají být zapsány do soboru FID (seznam může být i prázdný), je-li v seznamu matice (resp. pole), je zapisována po sloupcích. Formátovací řetězec je používán cyklicky opakovaně, dokud nejsou zapsány všechny prvky objektů uvedených v seznamu `fprintf`. Návratovou hodnotou je počet zapsaných bytů.

```
x = 0:.1:1;
y = [x; exp(x)];
fid = fopen('demo.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);
```

```
>> type demo.txt
  0.00    1.00000000
  0.10    1.10517092
     ......
  0.90    2.45960311
  1.00    2.71828183
```

Funkce fprintf a sprintf

- Funkce `fprintf` bez parametru FID (identifikátor souboru) zapisuje výstup na obrazovku (na standardní výstupní zařízení číslo 1).

```
>> fprintf('%d %f\n', 2, 34.4)
2 34.400000
>>
```

- Funkce `sprintf` má stejné parametry jako `fprintf`, návratovou hodnotou je ale řetězec (jinak řečeno, zapisuje formátovaný výstup do textového řetězce).

```
>> s = sprintf('%d %f\n', 2, 34.4);
>> s
s =
2 34.400000
```

```
>> class(s)
ans =
char
```

Formátovací řetězec funkce fprintf

Formátovací řetězec popisuje formátování výstupních polí a je tvořen kombinací následujících tří kategorií:

- Formátovací položky tvoří znak % s konverzním znakem (d, i, o, u, x, f, e, g, c, nebo s). Mezi znakem % a konverzním znakem mohou být zapsána jedno nebo dvě čísla, která udávají šířku výstupního pole a počet míst za desetinnou tečkou.
- Text, který má být vytištěn
- Escape sekvence se používá pro zápis následujících speciálních znaků

  `\b` *Backspace* (návrat o jeden znak zpět)
  `\f` *Form feed* (nová stránka)
  `\n` *New line* (nový řádek)
  `\r` *Carriage return* (návrat vozíku)
  `\t` Horizontální tabulátor
  `''` Znak apostrof
  `%%` Znak procento
  `\\` Znak zpětné lomítko

Konverzní znaky

Celá čísla se znaménkem

- `%d`
- `%i` Dekadická čísla

Celá čísla bez znaménka

- `%u` Dekadická čísla
- `%o` Oktalová čísla
- `%x` Hexadecimální čísla s malými písmeny a-f

Čísla v pohyblivé řádové čárce

- `%f` Zobrazení s pevnou desetinnou tečkou
- `%e` Exponenciální notace jako `3.141593e+00`
- `%g` Kompaktnější zápis než `f` a `e` bez úvodních nul

Znaky

- `%c` Jeden znak
- `%s` Textový řetězec (posloupnost znaků)

Konverzní znak X pro výstup velkými písmeny A-F, obdobně E a G

Příklady formátovaného výstupu

```
A1 = [9.9, 9900];
A2 = [8.8,  7.7 ;
      8800, 7700];
formatSpec = 'X je %4.2f metru tj. %8.3f mm\n';
fprintf(formatSpec,A1,A2)
```

```
X je 9.90 metru tj. 9900.000 mm
X je 8.80 metru tj. 8800.000 mm
X je 7.70 metru tj. 7700.000 mm
```

- Specifikace formátu je uložena do proměnné `formatSpec` typu `char`.
- V příkazu `fprintf` tvoří seznam dvě matice `A1` a `A2`, jejich prvky jsou zapisovány postupně po sloupcích.
- Protože je na vstupu více prvků než konverzních znaků, je formát používán opakovaně.

Příklady formátovaného výstupu

```
a = [1.02, 3.04, 5.06];
fprintf('%d\t',round(a));
```

```
>> a = [1.02, 3.04, 5.06];
fprintf('%d\t',round(a));
1        3        5        >>
```

- Tisk hodnot typu *double precision* konvertovaných funkcí `round` na celá čísla.
- Za každým číslem následuje jako oddělovač znak tabulátor.
- Výstup není ukončen znakem *konec řádku (\n)* a není proto *zalomen.*
- Bez konverze na celá čísla použije Matlab pro typ *double precision* namísto celočíselné konverze `%e)`.

```
>> fprintf('%d\t',a);
1.020000e+00    3.040000e+00    5.060000e+00    >>
```

Příklady formátovaného výstupu

```
x = 0:.1:1;      y = [x; exp(x)]
fprintf('%6.2f  %12.8f\n',y);
```

```
y =
 Columns 1 through 7
         0  0.1000  0.2000  0.3000  0.4000  0.5000  0.6000
    1.0000  1.1052  1.2214  1.3499  1.4918  1.6487  1.8221

 Columns 8 through 11
    0.7000  0.8000  0.9000  1.0000
    2.0138  2.2255  2.4596  2.7183

  0.00    1.00000000
  0.10    1.10517092
     ......
  0.90    2.45960311
  1.00    2.71828183
```

- Matice (a pole) se zapisují po jednotlivých prvcích po sloupcích (u vícerozměrných polí se první sloupec zleva mění nejrychleji).

Příklady formátovaného výstupu

```
fprintf('Specialni znaky: ');
fprintf('%% procento   '' apostrof   \\ zpetne lomitko\n');
```

```
Specialni znaky: % procento   ' apostrof   \ zpetne lomitko
```

- Protože výstup prvního příkazu `fprintf` není ukončem znakem *konec řádku*, vypíše se oba příkazu do jednoho řádku (druhý `printf` pokračuje tam, kde skončil první příkaz).

```
fprintf('%d dekadicky / %o oktalove / %x sestnactkove\n',
        [123 123 123])
```

```
123 dekadicky / 173 oktalove / 7b sestnactkove
```

- Zápis čísla 123 v různých soustavách.

Čtení matic z textových souborů

Předpokládejme, že máme textový soubor `magic5.txt`, který jsme vyvořili následovně.

```
fid = fopen('magic5.txt', 'w');
fprintf(fid,'%3g %3g %3g %3g %3g\n', magic(5)');
fclose(fid);
```

```
 17  24   1   8  15
 23   5   7  14  16
  4   6  13  20  22
 10  12  19  21   3
 11  18  25   2   9
```

Tento soubor pro nás představuje příklad vstupního datového proudu, ve kterém je posloupnost čísel 17, 24, 1, . . . 2, 9 oddělených *bílými znaky (mezery, tabulátory, konce řádků).* Řádkování pro nás přitom nebude podstatné (zde pouze pro lepší čitelnost).

Pro čtení takovýchto nestrukturovaných dat budeme používat funkci `fscanf`, která je doplňkovou funkcí k `fprintf`.

Načtení matice o zadaných rozměrech SIZEA

```
[A, COUNT] = fscanf(fid, FORMAT, SIZEA);
```

kde `SIZEA` může mít následující tvar:

- **Inf** Čti do konce souboru (implicitní hodnota)
- **[N]** Čti maximálně N prvků do sloupcového vektoru.
- **[M, N]** Čti po sloupcích maximálně `M*N` prvků (případně chybějící prvky v posledním sloupci jsou doplněny nulami). N může být Inf, M nemůže.

`FORMAT` byl popsán v souvislosti s funkcí `fscanf`.

Příklad

Načtení všech čísel ze vstupního proudu do vektoru `A`, `COUNT` je počet načtených elementů.

```
[A, COUNT] = fscanf(fid,'%g ');
```

Dva příklady načtení matice

Načtení matice o dvou řádcích a šesti sloupcích. Načteno 12 prvků.

```
[A, COUNT] = fscanf(fid,'%g ', [2, 6]);
```

```
A =
    17     1    15     5    14     4
    24     8    23     7    16     6
```

Načtení matice o třech řádcích a neznámém počtu sloupců. Načteno všech 25 prvků, chybějící dva v posledním sloupci doplněny nulami.

```
[A, COUNT] = fscanf(fid,'%g ', [3, inf]);
```

```
A =
    17     8     5    16    13    10    21    18     9
    24    15     7     4    20    12     3    25     0
     1    23    14     6    22    19    11     2     0
```

Zápis a načtení matice (opakování)

```
A = magic(3); A(3,:) = []; A   % kontrolni vypis matice A
fid = fopen('demo.txt', 'w');
fprintf(fid, '%g ', A);
fclose(fid);
type demo.txt                  % vypis souboru demo.txt

fid = fopen('demo.txt');
[B, n] = fscanf(fid, '%f ', size(A));
fclose(fid);

A == B                         % porovnani matic A a B (A rovno B)
```

```
A =   8     1     6     % vstupni matice A 2x3
      3     5     7

8 3 1 5 6 7              % soubor demo.txt
                         % posloupnost cisel neclenena po radci
ans =
      1     1     1     % porovnani matice A a nactene matice
      1     1     1     %     vsechny prvky jsou si rovny
```

Formátovací řetězec a přeskakování dat

Pokud mezi znakem % a konverzním znakem uvedeme znak hvězdičku (*), je odpovídající hodnota při čtení přeskočena.

```
A = magic(3); A(3,:) = [];
fid = fopen('demo.txt', 'w');
fprintf(fid, '%g ', A);
fclose(fid);
type demo.txt                    % vypis vstupnich dat

fid = fopen('demo.txt');
B = fscanf(fid, '%f %*f ');      % kazde druhe cislo ignorovan
fclose(fid);
B'
```

Ve vstupních datech je přeskakováno každé druhé číslo 9, v daném příkladu hodnoty 3, 5 a 7.

```
8 3 1 5 6 7                      % soubor demo.txt

ans =
     8     1     6               % transponovany vektor B
```

Čtení z textového řetězce (funkce sscanf)

```
[A,COUNT,ERRMSG,NEXTINDEX] = sscanf(S,FORMAT,SIZE)
```

- Funkce `sscanf` umožňuje načítat data z textového řetězce `S`. Parametry `A, COUNT, FORMAT a SIZE` mají stejný význam jako u funkce `fscanf`.
- `ERRMSG` obsahuje chybovou zprávu, pokud při načítání doško k chybě
- `NEXTINDEX` je počet zpracovaných znaků vstupního řetězce plus 1.

Příklad

```
S = '2.7183  3.1416';
A = sscanf(S,'%f')
```

```
A =
    2.7183
    3.1416
```

Změna polohy kurzoru (funkce fseek)

Funkce `fseek` umožňuje měnit pozici *kurzoru* v souboru.

```
status = fseek(fid, offset, origin);
```

fid	Identifikátor otevřeného souboru z volání `fopen`	
offset	Posun, která je interpretován následovně:	
	offset > 0 offset $= 0$ offset < 0	Posun směrem ke konci souboru od pozice *origin*. Pozice se nemění (nastavena na *origin*). Posun směrem k začátku souboru.
origin	Řetězec, který určuje pozici, od které se počítá *offset*	
	'bof' 'cof' 'eof'	-1 : Začátek souboru 0 : Stávající pozice v souboru 1 : Konec souboru
status	Návratová hodnota 0, pokud operace proběhla úspěšně a -1 pokud došlo k chybě. Při výskyty chyby použijte funkci `ferror` pro získání podrobnějších informací o vzniklé chybě.	

Změna polohy kurzoru (funkce fseek) - příklad

```
A = magic(6);
fid = fopen('demo.txt','w+');
fprintf('kurzor po otevreni souboru %3d\n', ftell(fid));
fprintf(fid, '%g ', A);
fprintf('kurzor po zapisu A         %3d\n', ftell(fid));
fseek(fid, 0, 'bof');  % 'bof' / 'cof' / 'eof'
fprintf('kurzor po volani fseek     %3d\n',ftell(fid));
B = fscanf(fid, '%g', size(A));
if A == B
    fprintf('cteni/zapis do souboru      OK\n');
else
    fprintf('chyba pri zapisu/cteni matice\n');
end
fclose(fid);
```

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Třídění

Matlab poskytuje řadu speciálních operací, jednou z nich je třídění vektorů a matic. Třídění vektoru je přímočaré

```
octave> A = [ 2 5 3 7 9 6 1 4]
A =

   2   5   3   7   9   6   1   4

octave> B = sort(A)
B =

   1   2   3   4   5   6   7   9
```

Druhý volitelný parametr udává směr třídění `'ascend'` nebo `'descend'`

```
octave> B = sort(A,'descend')
B =

   9   7   6   5   4   3   2   1
```

Třídění matic

Pokud použijeme funkci `sort` pro matici, setřídí její sloupcové vektory

```
octave> A = [8 3 9 1; 2 4 3 6; 7 5 0 5]
A =
   8   3   9   1
   2   4   3   6
   7   5   0   5

octave> B = sort(A)
B =
   2   3   0   1
   7   4   3   5
   8   5   9   6
```

Opět můžeme určit směr třídění

```
octave> B = sort(A,'descend')
B =
   8   5   9   6
   7   4   3   5
   2   3   0   1
```

Třídění

Matice je dvourozměrný objekt dimenze $m \times n$. Funkce `sort` má druhý volitelný parametr, který určuje podle které dimenze se má třídit

```
octave> A = [8 3 9 1; 2 4 3 6; 7 5 0 5]
A =
   8   3   9   1
   2   4   3   6
   7   5   0   5

octave> S = sort(A,1)  # třídíme sloupce (podle první dimenz
S =
   2   3   0   1
   7   4   3   5
   8   5   9   6

octave> R = sort(A,2)  # třídíme řádky (podle druhé dimenze)
R =
   1   3   8   9
   2   3   4   6
   0   5   5   7
```

Třídění

Jinou funkcí je `sortrows`, která třídí řádky vzestupně.

```
octave> A = [8 3 9 1; 2 4 3 6; 7 5 0 5]
A =

   8   3   9   1
   2   4   3   6
   7   5   0   5

octave> B = sortrows(A)
B =

   2   4   3   6
   7   5   0   5
   8   3   9   1
```

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data

Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Příklad — řešení kvadratické rovnice

$$ax^2 + bx + c = 0, \qquad s = \sqrt{b^2 - 4ac}, \qquad x_{1,2} = \frac{-b \pm s}{2a}$$

Řešení můžeme naprogramovat jako (skript) `kvadrov1.m`

```
1  % Reseni kvadraticke rovnice (verze 1)
2  disp('Reseni kvadraticke rovnice ax^2 + bx + c = 0');
3  a = input('Koeficient a: ');
4  b = input('Koeficient b: ');
5  c = input('Koeficient c: ');
6  d = b*b - 4*a*c;
7  s = sqrt(d);
8  x1 = (-b + s)/2/a
9  x2 = (-b - s)/2/a
10 return;
```

- Funkce `disp` na řádku 2 vypisuje zadaný řetězec v příkazové řádce.
- Funkce `input` na řádcích 3–5 zobrazuje nápovědu, zpracuje interaktivně zadaný vstup a výsledek předává jako návratovou hodnotu. Funkce umožňuje zadávat i komplexní koeficienty rovnice.

Příklad — řešení kvadratické rovnice

```
>> kvadrov1
Reseni kvadraticke rovnice ax^2 + bx + c = 0
Koeficient a: 1
Koeficient b: 2
Koeficient c: -3

x1 =
     1
x2 =
    -3
```

```
Reseni kvadraticke rovnice ax^2 + bx + c = 0
Koeficient a: 1
Koeficient b: 2
Koeficient c: 3

x1 =
  -1.0000 + 1.4142i
x2 =
  -1.0000 - 1.4142i
```

Příklad — řešení kvadratické rovnice

```
>> kvadrov1
Reseni kvadraticke rovnice ax^2 + bx + c = 0
Koeficient a: 0
Koeficient b: 1
Koeficient c: 1

x1 =
   NaN
x2 =
  -Inf
```

- Pokud zadáme nulovou hodnotu parametru `a`, skončí výpočet chybou
- Program neumožňuje postupné vkládání koeficientů více rovnic
- Program neumožňuje ukončit předčasně výpočet (například zadáním prázdné odpovědi)

Řešení $\longrightarrow$ doplníme výpočetní cyklus a příkazy pro ošetření podmínek.

Příklad — řešení kvadratické rovnice

```
1  % Reseni kvadraticke rovnice (verze 2)
2  disp('Reseni kvadraticke rovnice ax^2 + bx + c = 0');
3  while true
4      a = input('Koeficient a: ');
5      if isempty(a)
6          disp('Vypocet ukoncen.');
7          return;
8      elseif a == 0
9          disp('Koficient ''a'' nesmi byt nulovy!');
10         continue;
11     end
12     b = input('Koeficient b: ');
13     c = input('Koeficient c: ');
14     s = sqrt(b*b - 4*a*c);
15     x12 = [(-b + s)/2/a; (-b - s)/2/a]
16 end
```

- Cyklus 3 – 16 (ukončen příkazem `return` na řádku 7)
- Příkaz `continue` na řádku 10 znamená nové pokračování cyklu od příkazu `while` na ř. 3
- Podmínka 5 – 11 (pokud je koeficient `a` nulový, pak 9 – 10)

Algoritmus

Algoritmus je přesný návod či postup, kterým lze vyřešit daný typ úlohy.[3]

Konečnost Každý algoritmus musí skončit v konečném počtu kroků. Z praktického hlediska mají smysl algoritmy, které skončí v *přijatelném čase*, hovoříme proto o algoritmické složitosti (viz dále)

Obecnost Algoritmus řeší danou obecnou úlohu pro zadaný obor vstupních hodnot a ne úlohu typu *součin matice 3x4 s vektorem.*

Determinovanost V každém kroku zpracování algoritmu pro daná data musí být přesně definováno, který krok následuje.

Výstup je nezbytným výsledkem činnosti algoritmu. Podobně jako v případě konečnosti může být výstupem i pseudonáhodné řešení, jako například v metodě *Least Median Squares*.

Elementárnost Algoritmus se skládá z konečného počtu elementárních kroků.

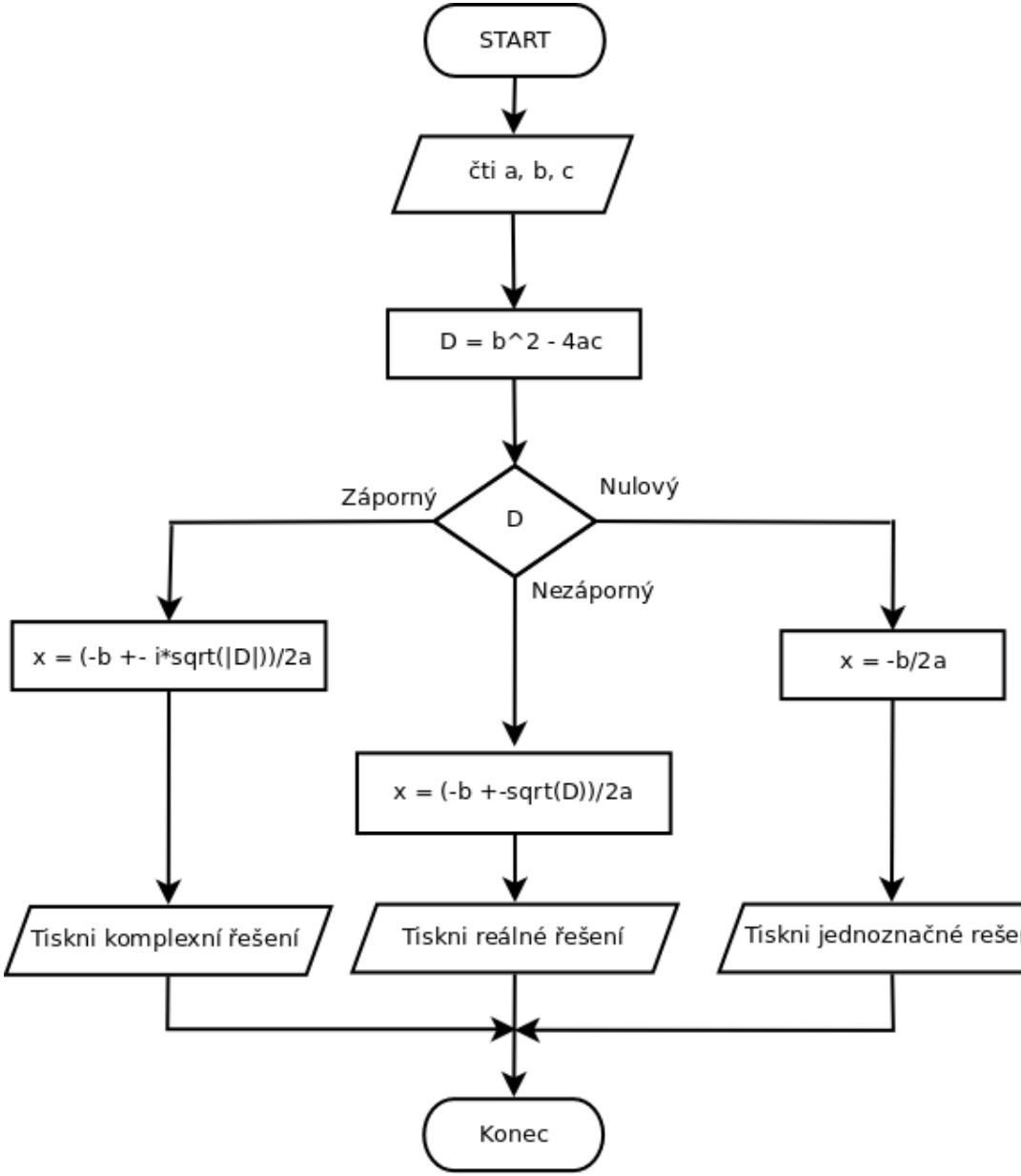
[3] http://cs.wikipedia.org/wiki/Algoritmus, http://en.wikipedia.org/wiki/Algorithm

Příklad jednoduchého vývojového diagramu

Výpočet kvadratické rovnice

- příkazy se implicitně provádějí postupně
- základní řídící konstrukce jsou podmínky, které umožňují větvení programu a cykly pro opakování daného kódu
- přenesení řízení chodu programu s návratem zpět zajišťuje volání funkcí, které v tomto smyslu můžeme také počítat mezi řídící konstrukce

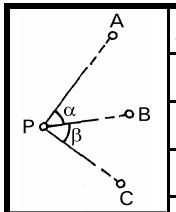
Vývojové diagramy jsou jedním z grafických nástrojů pro vyjádření chodu



Svébytným jazykem popisu algoritmů bývaly formuláře

Vloženo na výslovné přání studentů ze SPŠZ v Praze

PROTÍNÁNÍ ZPĚT

	$\Delta Y_{AB} = Y_A - Y_B$	$\Delta X_{AB} = X_A - X_B$			$\Delta Y_{MB} = \Delta Y_{AB} - a\Delta X_{AB}$	$\Delta X_{MB} = \Delta X_{AB} + a\Delta Y_{AB}$
	Y_A	X_A	α	$a = \mathrm{cotg}\ \alpha$	$p = \Delta Y_{MB} + \Delta Y_{BC} + b\Delta X_{BC}$	$q = \Delta X_{MB} + \Delta X_{BC} - b\Delta Y_{BC}$
	Y_B	X_B			$K = -p/q$	$L = -q/p$
	Y_C	X_C	β	$b = \mathrm{cotg}\ \beta$	$J = K + L$	$Q = \frac{\Delta Y_{MB} + K\,\Delta X_{MB}}{J}$
P | A, B, C	$\Delta Y_{BC} = Y_B - Y_C$	$\Delta X_{BC} = X_B - X_C$			$Y_P = Y_B + L\,Q$	$X_P = Q + X_B$
(1)	(2)	(3)	(4)	(5)	(6)	(7)
A						
B						
C						
P						
A						
B						
C						
P						
A						

Podmíněný příkaz if

```
if výraz
    příkazy
end
```

```
if výraz
    příkazy
else
    příkazy
end
```

```
if výraz_1
    příkazy
elseif výraz_2
    příkazy
else
    příkazy
end
```

- *Příkazy* jsou prováděny, pokud reálná část *výrazu* má všechny elementy nenulové.
  - Pokud stačí pouze některé nenulové prvky, použijeme funkci *any(výraz)*.
  - Implicitní chování tedy je *all(výraz)*.
- Části `else` a `elseif` jsou volitelné a mohou chybět.
- Klíčové slovo `elseif` je nedělitelné, tedy nesmí obsahovat mezeru mezi slovy else a if.
- *Výraz* má typicky tvar *expr rop expr*, kde relační operátor *rop* je `==`, `<`, `>`, `<=`, `>=` nebo `~=`.

Podmíněný příkaz if

Příklady vyhodnocení výrazů typy pole

```
a = [1 2 3 4];    b = [2 1 0 4];

if a < b
    disp('''a'' je mensi nez ''b''');
else
    disp('''a'' je vetsi nebo rovno ''b''');
end;

if a       % totéž jako  if all(a)
    disp ('pole ''a'' obsahuje pouze nenulove cleny');
end;

if any(b)
    disp('pole ''b'' obsahuje alespoň jeden nenulovy clen')
end;
```

```
'a' je vetsi nebo rovno 'b'
pole 'a' obsahuje pouze nenulove cleny
pole 'b' obsahuje alespon jeden nenulovy clen
```

Příkaz switch

Příkaz `switch` je určen podobně jako příkaz `if` k větvení programu. Používá se tam, kde vybíráme podle hodnoty jednoho výrazu (zápis je pak stručnější a hlavně kompaktnější než pomocí `if`).

```
switch switch_výraz
  case case_výraz1
    příkaz, ..., příkaz
  case {case_výraz2, case_výraz3, case_výraz4,...}
    příkaz, ..., příkaz
 ...
  otherwise
    příkaz, ..., příkaz
end
```

- Po vyhodnocení se *switch_výraz* porovnává sekvenčně s výrazy uvedenými z klíčovým slovem `case` a provede se první blok příkazů, pro který je nalezena shoda.
- Větev označená klíčovým slovem `otherwise` je volitelná a provádí se tehdy, pokud nebyla nalezena explicitní shoda.

Příkaz switch - demo

```
for i = 1:5
    switch i
        case 1
            fprintf('proměnná i je rovna jedné\n');
            fprintf('-------------------------\n')
         case 2
            fprintf('proměnná i je rovna dvěma\n');
            fprintf('=========================\n');
        case 3
            fprintf('proměnná i je rovna třem\n');
            fprintf('#########################\n');
        otherwise
            fprintf('proměnná i není z intervalu <1,3>\n');
   end;
end
```

Příkaz cyklu for

```
for proměnná = výraz
    příkazy
end
```

- V příkazu `for` je *výraz* typicky ve tvaru M:N

```
for r = 1:N
    for c = 1:N
        A(r,c) = 1/(r+c-1);
    end
end
```

mohou to ale být obecné seznamy.

```
for k = [1 3 5 7 11 13]
```

- Příkaz `break` ukončí cyklus.
- Příkaz `continue` ukončí probíhající iteraci a přejde na následující.

Příkaz cyklu for — příklady

Skalární součin

$$s = \sum_{i=1}^{N} a_i b_i$$

```
N = 1000; a = rand(N); b = rand(N);

s = 0;
for i = 1:N
    s = s + a(i)*b(i);
end
```

Sumace prvků matice

$$s = \sum_{r=1}^{M} \sum_{s=1}^{N} M_{r,s}$$

```
R = 1000;  S = 1500;
M = rand(R,S);

s = 0;
for r = 1:R
    for s = 1:S
        s = s + M(r,s);
    end
end
```

Příkaz cyklu while

```
while výraz
    příkazy
end
```

- *Výraz* je podmínka, který se vyhodnocuje stejně jako v příkazu `if`, dokud platí, jsou postupně prováděny *příkazy*.
- Ve většině případů se proměnné použité ve *výrazu* v těle cyklu mění, protože jinak by cyklus nikdy neskončil. Příkazy `while` s konstantní podmínkou musí být explicitně ukončeny jinak (o tom později).[4]

Příklad

```
x = 0.1; s = 1+x; t=x; n=2;
while t > eps
    t = t*x/n; s = s + t; n = n + 1;
end
[s, n, (s - exp(x))*1e16]
```

```
    1.1052   11.0000   -4.4409
```

[4] viz příkazy `break` a `continue`

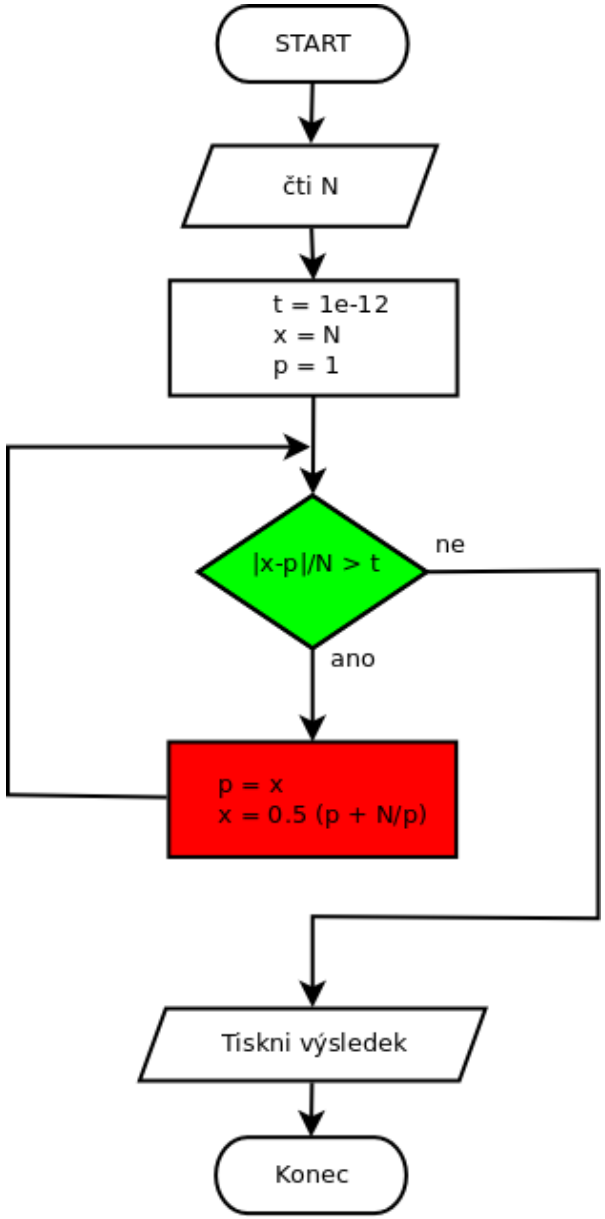
Algoritmus pro výpočet druhé odmocniny

Výpočet druhé odmocniny x z čísla N, tj. $x = \sqrt{N}$, podle Heronova vzorce

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{N}{x_i}\right), \qquad \text{kde} \quad N > 0$$

```
N = 2;
t = 1e-12;
x = N;
p = 1;

while abs(x-p)/N > t
    p = x;
    x = (p + N/p) / 2;
end;

[x, (x-sqrt(N))*1e16]
```

```
    1.4142   -2.2204
```



Eratosthenovo síto

Algoritmus pro nalezení všech prvočísel menších nebo rovných N.[5]

Inicializace: Vytvoříme prázdný seznam prvočísel S a pracovní seznam K čísel $2 \ldots N$.

Krok 1: Dokud seznam K není prázdný, je první číslo ze seznamu K prvočíslo, uložíme je jako p.

Krok 2: Pokud platí $p > \sqrt{\max K_i}$, obsahuje K již pouze prvočísla. Připojíme K k seznamu S. Zpracování končí.

Krok 3: Do S přidáme prvočíslo p. Ze seznamu K odebereme p a všechny jeho násobky. Návrat na krok 1.

```
S = [];
K = [2:N];
while ~isempty(K)
    p = K(1);
    m = max(K);
    if p > sqrt(m)
        S = [S K];
        K = [];
    else
        S = [S p];
        q = p;
        while q <= N
            K(K==q) =[];
            q = q + p;
        end;
    end;
end;
```

[5] http://cs.wikipedia.org/wiki/Eratosthenovo_síto

Eratosthenovo síto - jiná implementace

```
S = [];
K = [2:N];
while ~isempty(K)
    p = K(1);
    m = max(K);
    if p > sqrt(m)
        S = [S K];
        K = [];
    else
        S = [S p];
        q = p;
        while q <= N
            K(K==q) =[];
            q = q + p;
        end;
    end;
end;
```

viz `esito_demo.m`

```
K = [1:N];   K(1) = 0;
for i = [2:N/2+1]
    if K(i) ~= 0
        j = 2*i;
        while j <= N
            K(j) = 0;
            j = j + i;
        end;
    end;
end;
S = find(K ~= 0);
```

N	čas 1	čas 2
1000	0.02	0.00
25000	3.88	0.02
50000	15.09	0.03
100000	64.57	0.07
200000	302.97	0.14
400000	1300.56	0.27

Příklad - Taylorův rozvoj funkce sinus

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \dots$$

```
suma = 0;
t = x;
n = 1;
z = 1;
while abs(t) > diff
    t = z*x^n/factorial(n);
    suma = suma + t;
    n = n + 2;
    z = -z;
end
```

```
Elapsed time is 66.079 secs.
```

```
suma = x;
t = x;
n = 2;
x2 = -x*x;
while abs(t) > diff
    t = t*x2/n/(n+1);
    suma = suma + t;
    n = n + 2;

end
```

```
Elapsed time is 3.49 secs.
```

Odhad času pro 100 000 výpočtů rozvoje (viz `rozvoj_sin.m`).

Příklad - Taylorův rozvoj funkce arkus sinus

$$\arcsin(x) = x + \frac{1}{2}\cdot\frac{x^3}{3} + \frac{1\cdot 3}{2^2\cdot 2!}\cdot\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2^3\cdot 3!}\cdot\frac{x^7}{7} + \ldots \qquad x \in < -1, 1 >$$

```
s = x;                  % soucet n clenu
n = 1;                  % index clenu 1, 2, 3, ...
m = 1;                  % mocnina x^m, 3, 5, 7, ....
z = x;                  % hodnota n-teho clenu
q = x*x;
while abs(z) > tol
    z = z * m;          % v citateli 1, 3, 5, ...
    m = m + 2;
    z = z * q;          % mocniny x3, x5, x7, ....
    z = z / 2;          % mocniny 2, 2^2, 2^3, ....
    z = z / n;          % ve jmenovateli n!
    n = n + 1;
    s = s + z/m;        % ve jmenovateli 3, 5, 7, ...
end;
```

Poznámka: pro x blížící se ± 1 je vhodné použít
$\arcsin(x) = \pm\left(\pi/2) - 2\arcsin\sqrt{(1-|x|)/2}\right)$

$\mathcal{O}$ notace

Poznámka ke klasifikaci náročnosti algoritmů

Asymptotická složitost (konstantní, lineární, kvadratická, logaritmická, ...)

$$f(x) = \mathcal{O}(g(x)) \quad \text{pro} \quad x \to \infty$$

pokud existuje x_0 takové, že

$$|f(x)| \leq M|g(x)| \quad \text{pro} \quad x > x_0$$

Příklady

- $\mathcal{O}(1)$ indexování prvku v poli
- $\mathcal{O}(\log_2 N)$ vyhledání prvku v seřazeném poli metodou půlení intervalu
- $\mathcal{O}(N)$ vyhledání prvku v neseřazeném poli lineárním (sekvenčním) vyhledáváním
- $\mathcal{O}(N \log N)$ seřazení pole reálných čísel podle velikosti (např. algoritmem Mergesort)
- $\mathcal{O}(N^2)$ diskrétní Fourierova transformace (DFT)
- $\mathcal{O}(2^N)$ přesné řešení problému obchodního cestujícího (či jiného NP-úplného problému hrubou silou)

Gaussova eliminace

Rozšířená matice soustavy

Gaussova eliminační metoda řeší soustavu lineárních rovnic

$$\mathbf{Ax} = \mathbf{b}$$

převodem na ekvivalentní soustavu s horní trojúhelníkovou maticí.

Obvykle se přitom pracuje s tzv. rozšířenou maticí soustavy, kde k matici $\mathbf{A}$ se připojí vektor $\mathbf{b}$ jako $n+1$-ní sloupec

$$(\mathbf{A}|\mathbf{b})$$

Lze takto připojit i více sloupců, obecně matici.

Řádkové úpravy mohou být:

- výměna dvou řádku
- vynásobení řádku nenulovou hodnotou
- přičtení násobku řádku k jinému řádku

Gaussova eliminace

Příklad

Pro soustavu $\mathbf{Ax} = \mathbf{b}$

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 \\ -3 & 1 & 1 & -2 & -3 \\ -6 & 7 & 11 & -3 & -4 \\ 0 & -5 & -1 & 6 & 5 \\ -3 & -14 & -22 & 3 & 7 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 62 \\ -21 \\ 9 \\ 36 \\ -50 \end{pmatrix}$$

vytvoříme rozšířenou matici soustavy

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ -3 & 1 & 1 & -2 & -3 & -21 \\ -6 & 7 & 11 & -3 & -4 & 9 \\ 0 & -5 & -1 & 6 & 5 & 36 \\ -3 & -14 & -22 & 3 & 7 & -50 \end{pmatrix}$$

Gaussova eliminace

Příklad — první eliminační krok

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ -3 & 1 & 1 & -2 & -3 & -21 \\ -6 & 7 & 11 & -3 & -4 & 9 \\ 0 & -5 & -1 & 6 & 5 & 36 \\ -3 & -14 & -22 & 3 & 7 & -50 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 15 & 23 & 1 & 6 & 133 \\ 0 & -5 & -1 & 6 & 5 & 36 \\ 0 & -10 & -16 & 5 & 12 & 12 \end{pmatrix}$$

Gaussova eliminace

Příklad — druhý eliminační krok

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 15 & 23 & 1 & 6 & 133 \\ 0 & -5 & -1 & 6 & 5 & 36 \\ 0 & -10 & -16 & 5 & 12 & 12 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 6 & 6 & 7 & 77 \\ 0 & 0 & -2 & 5 & 16 & 94 \end{pmatrix}$$

Gaussova eliminace

Příklad — třetí eliminační krok

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 6 & 6 & 7 & 77 \\ 0 & 0 & -2 & 5 & 16 & 94 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 0 & 3 & 7 & 47 \\ 0 & 0 & 0 & 6 & 16 & 104 \end{pmatrix}$$

Gaussova eliminace

Příklad — čtvrtý eliminační krok

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 0 & 3 & 7 & 47 \\ 0 & 0 & 0 & 6 & 16 & 104 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 0 & 3 & 7 & 47 \\ 0 & 0 & 0 & \color{red}{0} & 2 & 10 \end{pmatrix}$$

Gaussova eliminace

Příklad — zpětná substituce

$$\begin{pmatrix} 3 & 4 & 6 & 2 & 5 & 62 \\ 0 & 5 & 7 & 0 & 2 & 41 \\ 0 & 0 & 2 & 1 & 0 & 10 \\ 0 & 0 & 0 & 3 & 7 & 47 \\ 0 & 0 & 0 & 0 & 2 & 10 \end{pmatrix}$$

Řešení

$$\begin{aligned} x_5 &= 10/2 = 5 \\ x_4 &= (47 - 7 \times 5)/3 = 4 \\ x_3 &= (10 - 0 \times 5 - 1 \times 4)/2 = 3 \\ x_2 &= (41 - 2 \times 5 - 0 \times 4 - 7 \times 3)/5 = 2 \\ x_1 &= (62 - 5 \times 5 - 2 \times 4 - 6 \times 3 - 4 \times 2)/3 = 1 \end{aligned}$$

Obecně

$$x_k = \left(\mathbf{b}_k - \sum_{j=k+1}^{n} \mathbf{A}_{kj} x_j \right) \Big/ \mathbf{A}_{kk}$$

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Příkaz figure - vytvoření grafického objektu (okna)

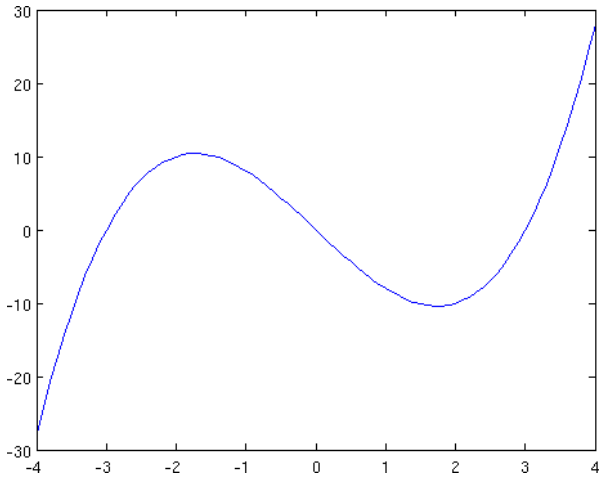
```
1 figure
2 figure('JménoParametru',hodnotaParametru,...)
3 figure(h)
4 h = figure(...)
```

Příkaz `figure` vytváří grafické objekty (okna), ve kterých Matlab zobrazuje grafické výstupy, a vrací ukazatel *(handle)* na vytvořený objekt.

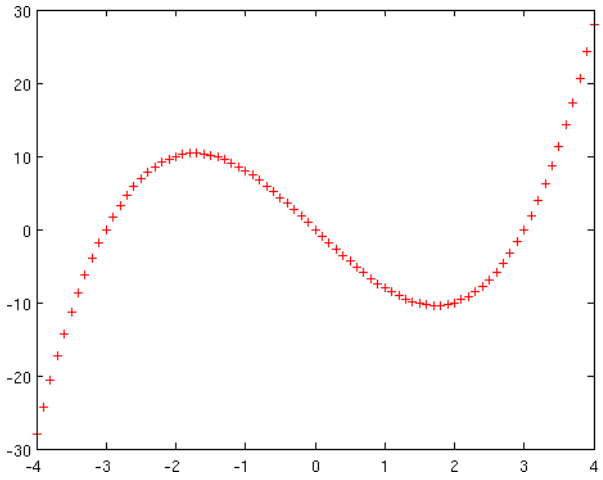
- Příkaz (1) vytváří nové okno s implicitními hodnotami parametrů, (2) vytváří nový grafický objekt s explicitně nastavenými hodnotami vybraných parametrů.
- Příkaz (3) pracuje v závislosti na tom, zda *ukazatel "h" na grafický objekt (handle)* existuje nebo ne. Pokud je *h* ukazatel na existující grafický objekt, zajistí příkaz `figure(h)`, že daný objekt se stává běžným objektem, je *viditelný (visible)* a je přesunut nad všechna okna na obrazovce. Pokud `h` není ukazatelem na grafický objekt a je to celé číslo, vytvoří příkaz (3) nové okno a přiřadí mu hodnotu `h`.
- Přikaz (4) vrací ukazatel na grafický objekt.

Příkaz plot - graf funkce

```
x = -4:0.1:4;
y = x.^3 - 9*x;
plot(x,y);
```




```
x = -4:0.1:4;
y = x.^3 - 9*x;
plot(x,y, 'r+');
```

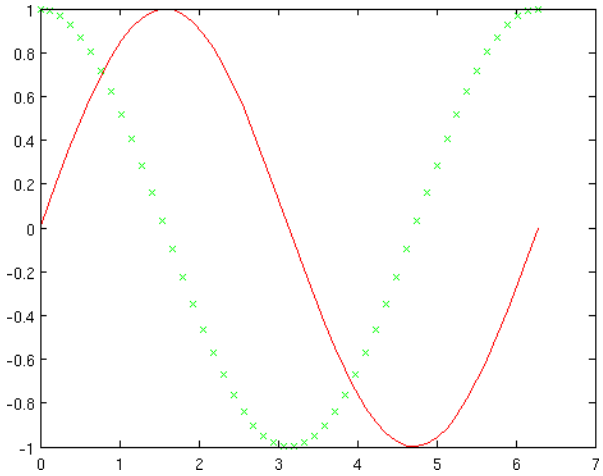


Příkaz plot - parametry

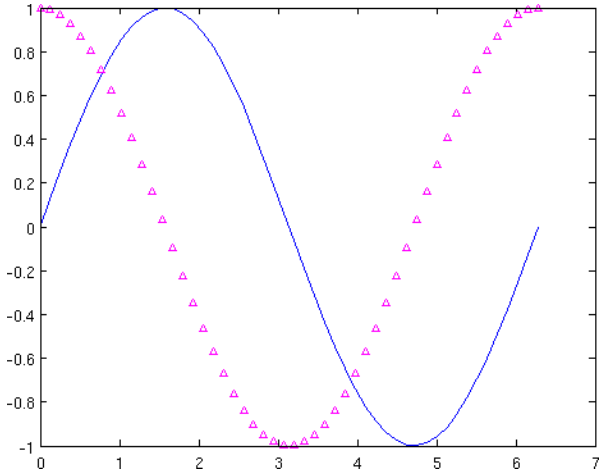
y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		ˆ	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

Příkaz plot - dva grafy funkce v jednom obrázku

```
x = linspace(0,2*pi,50);
y = sin(x);
z = cos(x);
plot(x,y,'r', x,z,'gx');
```

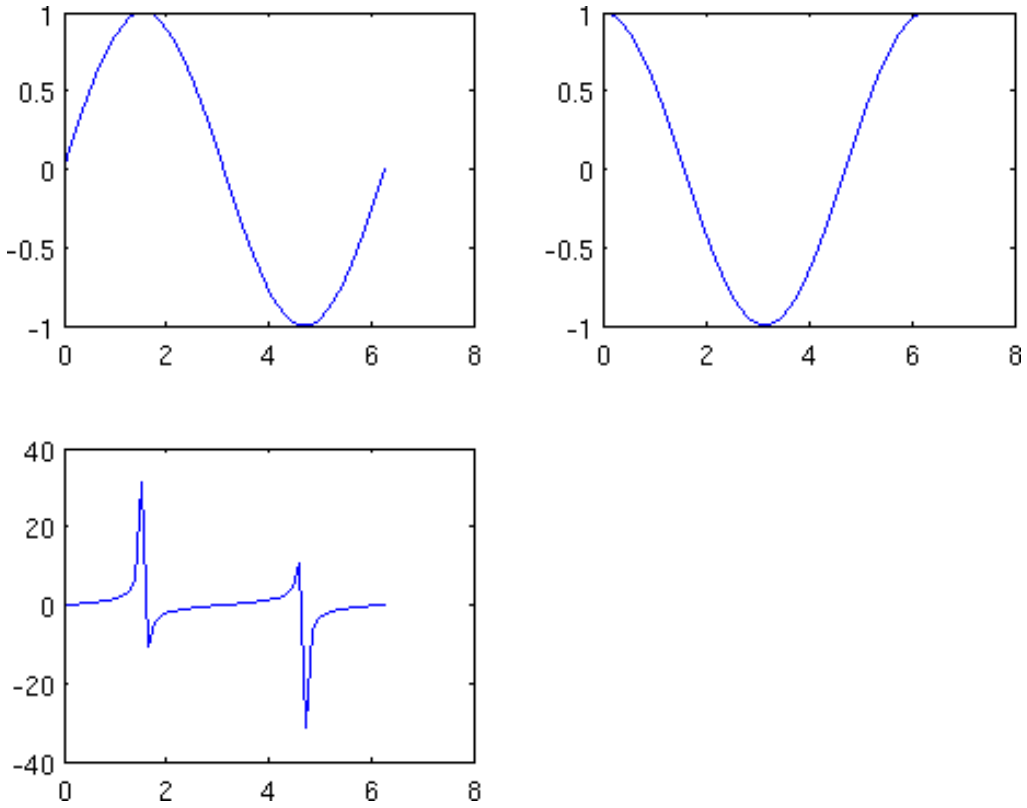



```
x = linspace(0,2*pi,50);
y = sin(x);
plot(x,y,'b')
z = cos(x);
hold on
plot(x,z,'m^')
hold off
```



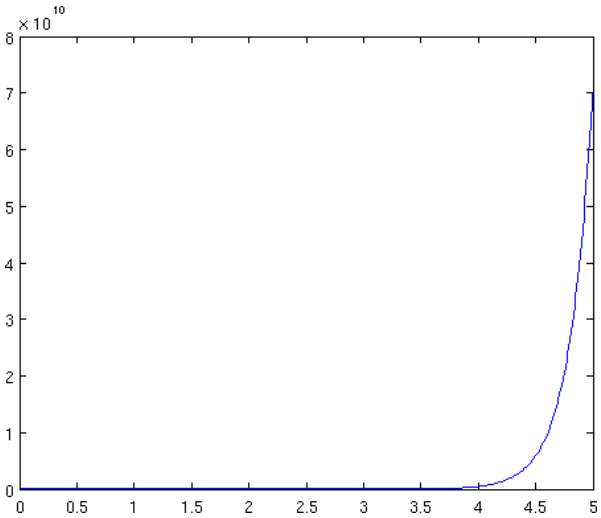
Příkaz subplot

```
x = linspace(0,2*pi,50);
y = sin(x);
z = cos(x);
w = tan(x);

subplot(2,2,1)
plot(x,y)
subplot(2,2,2)
plot(x,z)
subplot(2,2,3)
plot(x,w)
```



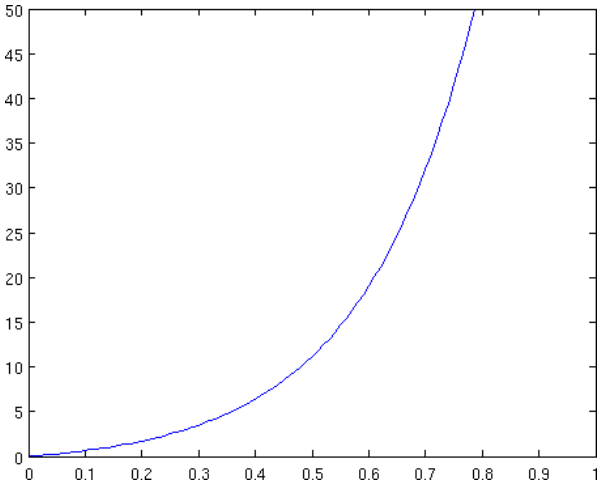
Příkaz plot - axis

```
x=0:0.01:5;
y=exp(5*x)-1;
plot(x,y)
```



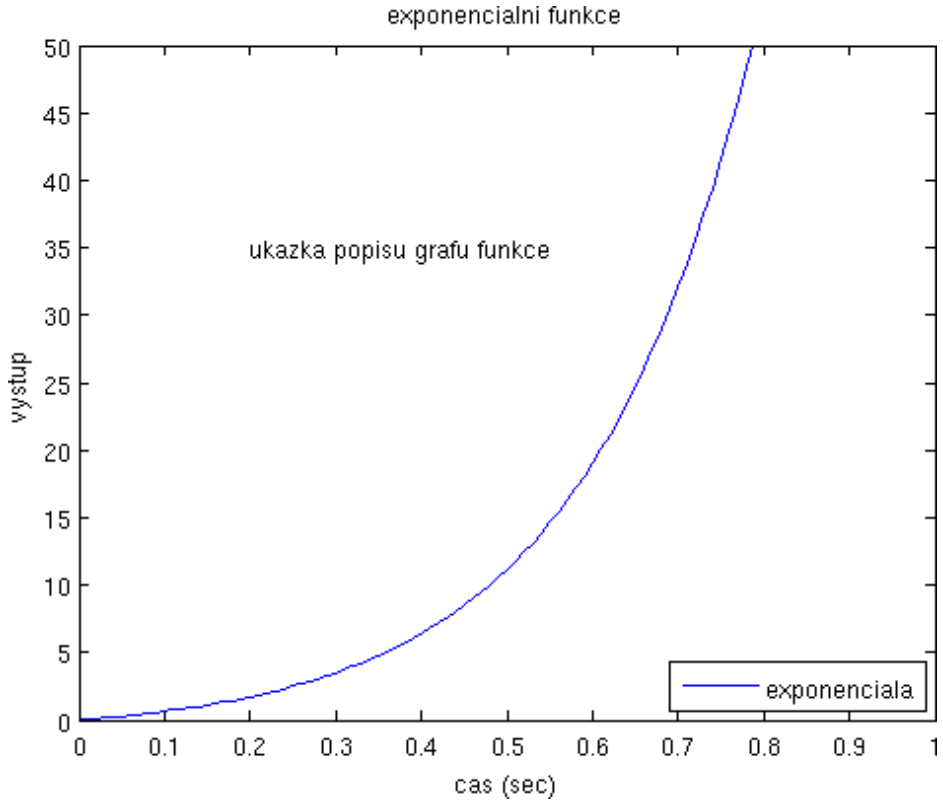
Omezení oblasti zobrazeného grafu funkce.

```
x=0:0.01:5;
y=exp(5*x)-1;
plot(x,y)
axis([0, 1, 0, 50])
```



Příkaz plot - doplnění popisu

```
title('exponencialni funkce')
xlabel('cas (sec)')
ylabel('vystup')
legend('exponenciala','Location','SouthEast')
text(0.2,35,'ukazka popisu grafu funkce')
```



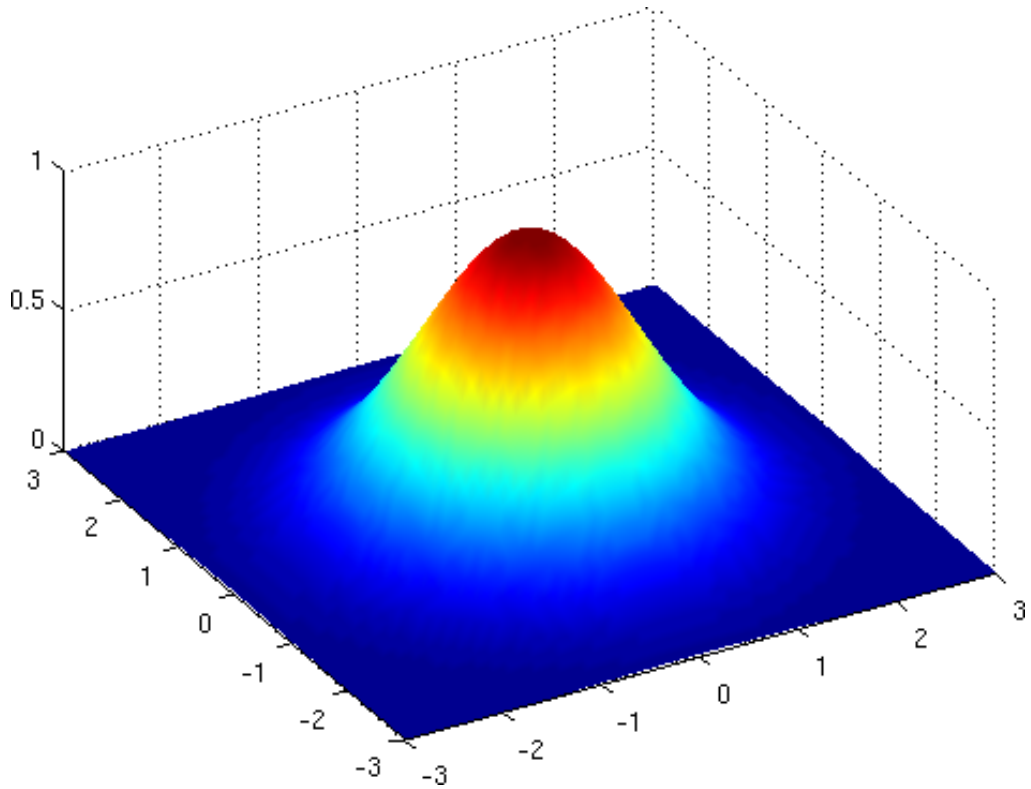
Další příkazy

loglog	stejný jako plot, s tím, že obě osy mají logaritmické měřítko
semilogx	stejný jako plot, s tím, že osa x má logaritmické měřítko
semilogy	stejný jako plot, s tím, že osa y má logaritmické měřítko
plotyy	vykresluje dvě křivky se dvěma osami, jednu vpravo, jednu vlevo.
grid	přidává do výstupu mřížku.
surf	zobrazení plochy.

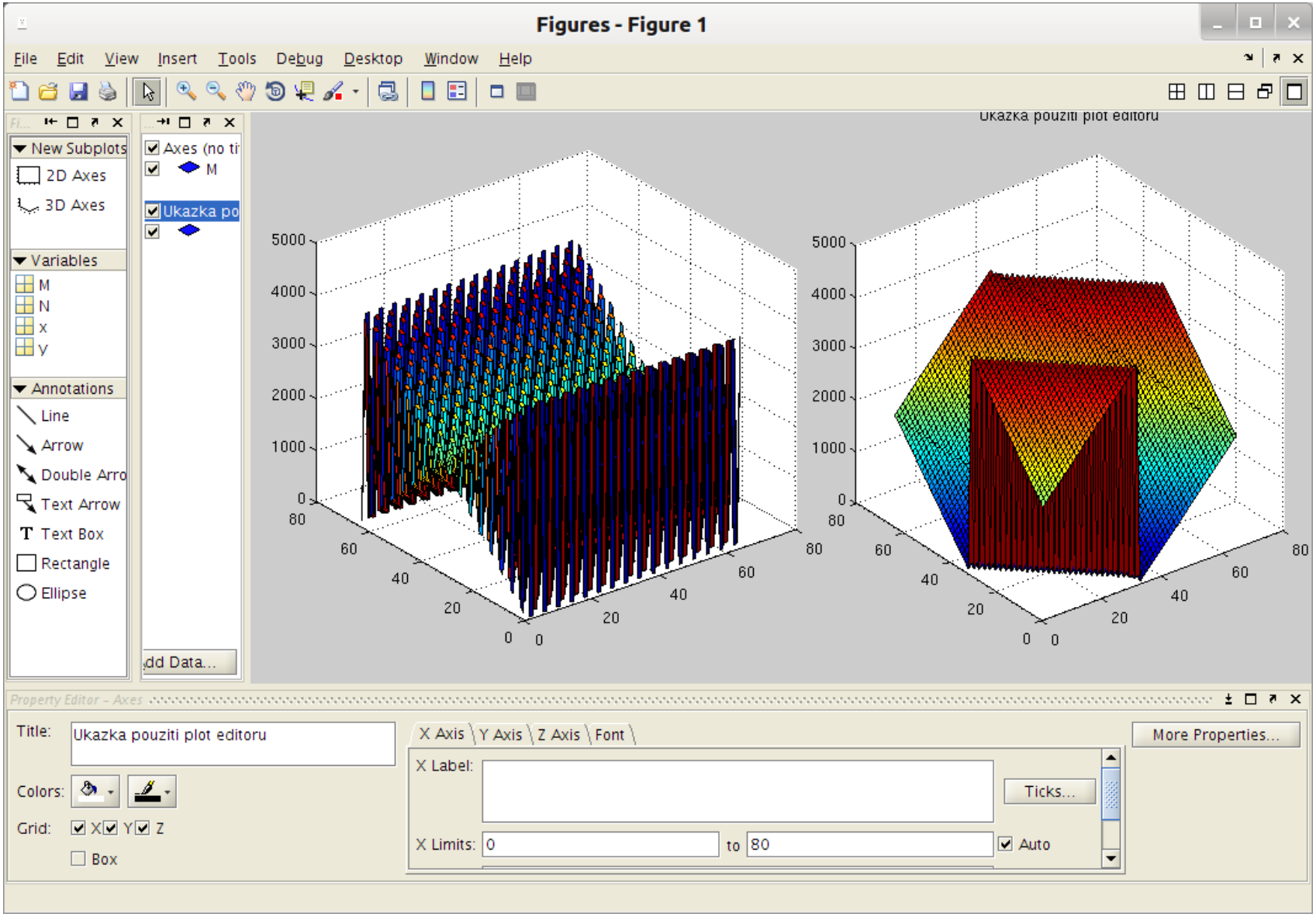
Zobrazení plochy Gaussova rozdělení ve 2D

```
A=1; x0=0; y0=0; sx=1; sy=1;      body = linspace(-3, 3, 60)
[X, Y] = meshgrid(body, body);
Z = A*exp(-(X-x0).^2/(2*sx^2) - (Y-y0).^2/(2*sy^2) );
figure(1), surf(X,Y,Z), shading interp, view(-30,50)
```

$$f(x,y) = Ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2}\right)}$$



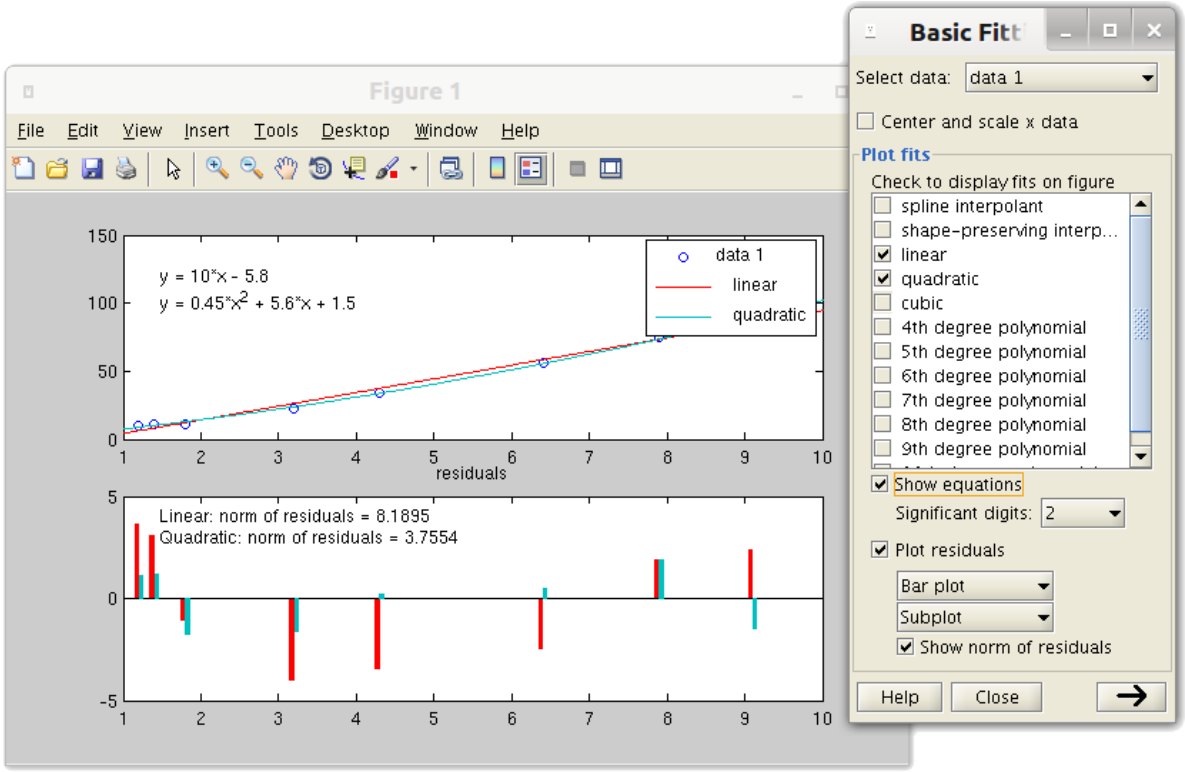
Ukázka použití plot editoru



Aproximace křivek v matlabu

Jednoduché aproximace: `Figure -> Tools -> Basic Fitting`

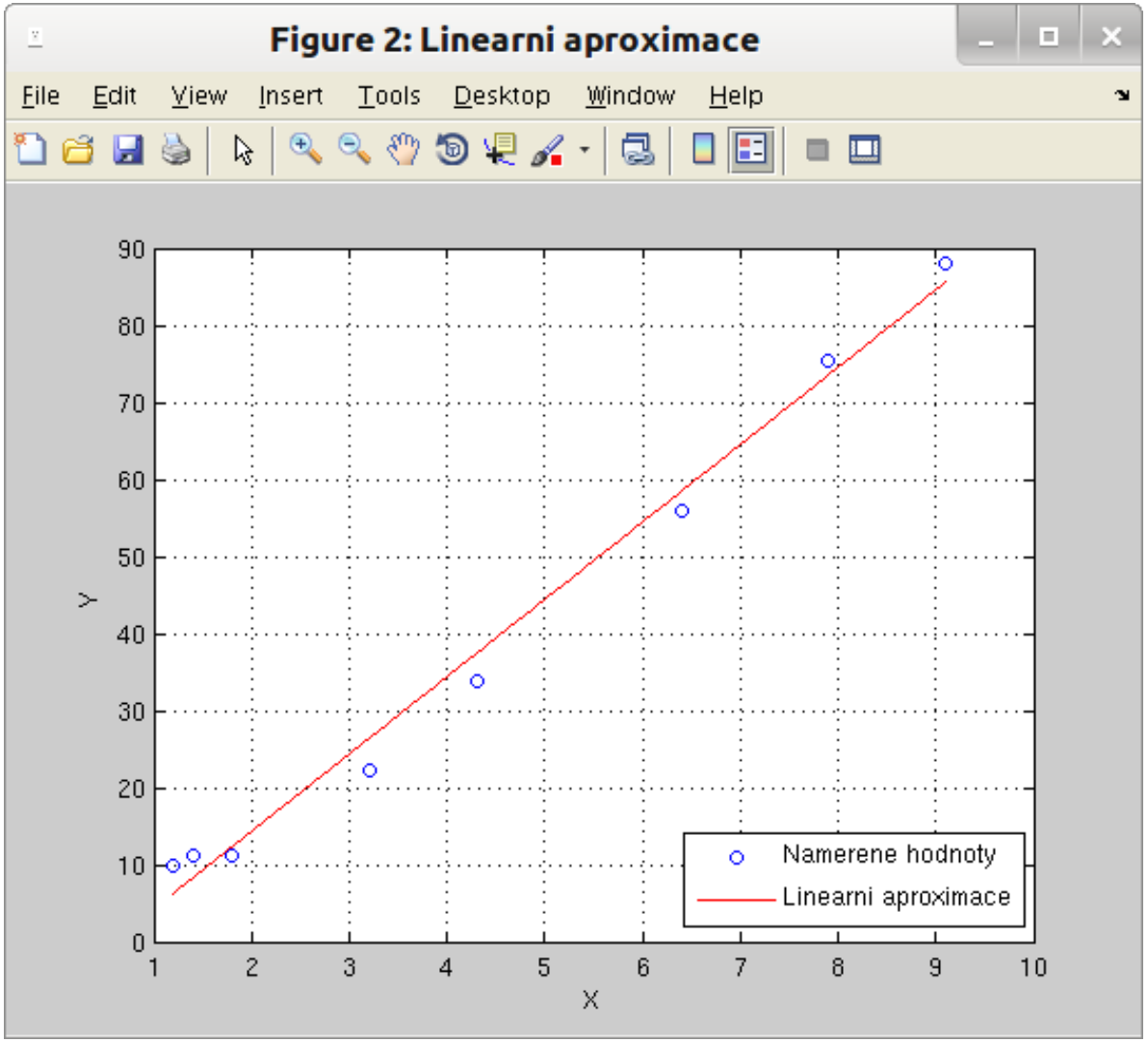
```
x = [ 1.2   1.4   4.3   7.9   9.1   6.4   3.2   1.8];
y = [ 9.9  11.3  33.9  75.4  87.9  56.0  22.3  11.2];
plot(x,y, 'o');    % zobraz data jako jednotlive body
```



Příklad — lineární aproximace

```
% priprav data pro aproximaci krivkou
[X, Y] = prepareCurveData(x, y);

% nastav typ aproximace a volitelne parametry
ft = fittype('poly1');
op = fitoptions(ft);
op.Lower = [-Inf -Inf];
op.Upper = [Inf Inf];

% aproximace zvolenym typem krivky
[aprox, gof] = fit(X, Y, ft, op);

figure(2), set(2, 'Name', 'Linearni aproximace' );
h = plot(aprox, X, Y, 'o');
legend( h, 'Namerene hodnoty', 'Linearni aproximace', ...
           'Location', 'SouthEast');
xlabel( 'X' );
ylabel( 'Y' );
grid on
```

Příklad — lineární aproximace



Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Definice funkce

```
function [výstupní_parametry] = jméno(vstupní_parametry)
```

- Definice funkce povinně začíná klíčovým slovem function
- následuje seznam výstupních parametrů uzavřených do hranatých závorek `[ ]` (pokud má funkce jeden výstupní parametr, závorky se nemusí uvádět). Seznam výstupních parametrů může být prázdný.
- Seznam parametrů a jméno funkce odděluje *rovnítko* **=** (neuvádí se, pokud je seznam výstupních parametrů prázdný).
- Každá funkce musí být zapsána v samostatném souboru, který se jmenuje stejně jako daná funkce a má příponu `.m`
- Komentáře následující za řádkem definujícím jméno a parametry funkce tvoří její dokumentaci, která je zobrazována příkazy `help` a `doc` (dokumentace je ukončena prvním prázdným nebo nekomentářovým řádkem).
- Vstupní parametry funkce by neměly být ve funkci měněny.

Příklad - výpočet obsahu a obvodu trojúhelniku

```
function [obsah, obvod] = trojuhelnik(a, b, c)
% funkce [obsah, obvod] = trojuhelnik(a, b, c)
% pocita obsah a obvod trojuhelniku zadaneho stranami a, b,
%
%   obsah = sqrt( s*(s-a)*(s-b)*(s-c) ), kde s = obvod/2

obvod = a + b + c;
s = obvod/2;
obsah = sqrt( s*(s-a)*(s-b)*(s-c) );
```

```
>> [plocha, obvod] = trojuhelnik(10, 20, 25)

plocha = 94.9918
obvod  = 55

>> help trojuhelnik
  funkce [obsah, obvod] = trojuhelnik(a, b, c)
  pocita obsah a obvod trojuhelniku zadaneho stranami a, b,

    obsah = sqrt( s*(s-a)*(s-b)*(s-c) ), kde s = obvod/2
```

Příklad - výpočet Taylorova rozvoje

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

```
function y = sinus(x)
% ukazka vypoctu sumace
% a implicitniho vypoctu
% mocnin a faktorialu
diff = 1e-16;
y = x;
% prvni clen
t = x;     % n-ty clen
n = 2;
x2 = -x*x;
while abs(t) > diff
    t = t*x2/n/(n+1);
    y = y + t;
    n = n + 2;
end
```

```
% test funkce sinus(x)
maxdif = 0;
for x = -1:0.01:1
    y1 = sin(x);
% system. fce
    y2 = sinus(x); % uziv.funkce
    if abs(y1-y2) > abs(maxdif)
        maxdif = y1-y2;
    end
end
fprintf('maxdif = %g\n', maxdif);
```

```
>> vypocet_sinus
maxdif = 2.22045e-16
```

Příklad - protínání zpět

```
function [x, y] = protinani_zpet(souradnice, uhly)
% Vypocet souradnic bodu protinanim zpet
% Dano: souradnice tri bodu (xy), dva uhly

dx_ab = souradnice(1,1) - souradnice(2,1);
dy_ab = souradnice(1,2) - souradnice(2,2);
dx_bc = souradnice(2,1) - souradnice(3,1);
dy_bc = souradnice(2,2) - souradnice(3,2);

a = 1/tan(uhly(1)/200*pi);   b = 1/tan(uhly(2)/200*pi);
dx_mb = dx_ab + a*dy_ab;     dy_mb = dy_ab - a*dx_ab;
p = dy_mb + dy_bc + b*dx_bc;
q = dx_mb + dx_bc - b*dy_bc;
K = -p/q;
L = -q/p;
J = K + L;
Q = (dy_mb + K*dx_mb)/J;

x = Q + souradnice(2,1);
y = souradnice(2,2) + L*Q;
```

Příklad - protínání zpět (výpočet)

```
xy = [853704.65 1011445.66;
      852501.72 1011270.35;
      851874.53 1010783.03];
uhly = [46.4243 61.7646];

[x, y] = protinani_zpet(xy, uhly);
fprintf('x = %10.3f   y = %11.3f\n', x, y);

x = 852360.681   y = 1010330.533
```

Příklad - plocha polygonu

Plocha polygonu

$$P = \frac{1}{2}\sum_{i}^{N-1}(x_{i+1} - x_i)(y_{i+1} + y_i)$$

```
function plocha = plocha_polygonu(body)
% vypocet plochy polygonu

[R,C] = size(body);
plocha = 0;
if R < 3
    return;
end
for i = 1:R-1
    plocha = plocha + (body(i+1,1)-body(i,1)) ...
                    * (body(i+1,2)+body(i,2));
end
plocha = plocha + (body(1,1)-body(R,1))*(body(1,2)+body(R,2)
plocha = abs(plocha/2);
```

Příklad - plocha polygonu (výpočet)

```
body = [1 1;  0 2;  -1 1;  -1 -1;  1 -1]

p = plocha_polygonu(body);
fprintf('plocha polygonu pro neuzavrena data je %.f\n', p);

[r,c] = size(body);
body(r+1,:) = body(1,:);

p = plocha_polygonu(body);
fprintf('plocha polygonu pro uzavrena   data je %.f\n', p);
```

```
plocha polygonu pro neuzavrena data je 5
plocha polygonu pro uzavrena   data je 5
```

- Funkce počítá plochu pro uzavřená data (poslední bod seznamu je totožný s prvním) i neuzavřená data (poslední bod je implicitně spojen s prvním), viz řádek 13 zdrojového kódu.

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Číselné soustavy

Zápis čísla 123 je pozičním vyjádřením hodnoty

$$(123)_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0.$$

Obdobně můžeme vyjádřit stejnou hodnotu i v jiných číselných soustavách

$$\begin{aligned}
(1111011)_2 &= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= (64 + 32 + 16 + 8 + 0 + 2 + 1)_{10} = (123)_{10} \\
(173)_8 &= (1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0)_{10} = (1 \times 64 + 7 \times 8 + 3) = \\
(7B)_{16} &= (7 \times 16 + 11)_{10} = (112 + 11)_{10},
\end{aligned}$$

kde B je *hexadecimální číslice* vyjadřující hodnotu 11 (písmena A, B, C, D, E, F se používají pro označení hodnot 10 až 15).

Obecně pro libovolný základ r v pozičním vyjádření platí

$$(d_k\ d_{k-1} \ldots d_2\ d_1\ d_0)_r = \sum_{i=0}^{k} d_i \times r^i.$$

Zobrazení celých čísel ve dvojkové soustavě

Kladná čísla, uložená v jednom bytu (8 bitů), můžeme znázornit následovně

$$(0)_{10} = (0000\ 0000)_2 \qquad (7)_{10} = (0000\ 0111)_2$$
$$(1)_{10} = (0000\ 0001)_2 \qquad (16)_{10} = (0001\ 0000)_2$$
$$(2)_{10} = (0000\ 0010)_2 \qquad (127)_{10} = (0111\ 1111)_2$$
$$(3)_{10} = (0000\ 0011)_2 \qquad (128)_{10} = (1000\ 0000)_2$$
$$(5)_{10} = (0000\ 0101)_2 \qquad (255)_{10} = (1111\ 1111)_2$$

- Záporná čísla bychom mohli označit nastavením *znaménkového bitu (první bit zleva)*, takovému zobrazení se říká přímý kód se znaménkem. Poslední uvedená hodnota by pak ovšem označovala číslo $(-127)_{10}$.
- Problémem přímého kódu je, že v něm existují dvě reprezentace nuly (kladná a záporná nula).
- V jazycích jako je C++ nebo Java se rozlišují celá čisla se znaménkem (typ `signed`) a celá čísla bez znaménka (typ `unsigned`). Matlab umožňovat podobně definovat typ prvků objektů pomocí *tříd*.

Dvojkový doplněk

Celá záporná čísla se obvykle vyjadřují ve dvojkovém doplňku:

- absolutní hodnota je převedena do dvojkové soustavy
- vypočte se její negace (nuly jsou nahrazeny jedničkami a naopak)
- je přičtena jednička

Příklad: Číslo -123 ve dvojkovém doplňku:

$$\begin{aligned} 123 = 0111\ 1011 &\quad \text{abs. hodnota} \\ 1000\ 0100 &\quad \text{negace} \\ 1000\ 0101 &\quad \text{plus jedna} \rightarrow \quad -123 \end{aligned}$$

Neexistuje *záporná nula*

$$\begin{aligned} (0)_{10} &= (0000\ 0000)_2 & (-0)_{10} &= (0000\ 0000)_2 \\ (1)_{10} &= (0000\ 0001)_2 & (-1)_{10} &= (1111\ 1111)_2 \\ (2)_{10} &= (0000\ 0010)_2 & (-2)_{10} &= (1111\ 1110)_2 \\ (3)_{10} &= (0000\ 0011)_2 & (-3)_{10} &= (1111\ 1101)_2 \\ (5)_{10} &= (0000\ 0101)_2 & (-5)_{10} &= (1111\ 1011)_2 \end{aligned}$$

Čísla v plovoucí řádové čárce

Nejpoužívanějším standardem pro výčty v *plovoucí řádové čárce* je IEEE 754 Standard for Floating-Point Arithmetic.

znaménkový bit	exponent	normalizovaná mantisa

Pro případ tzv. *jednoduché přesnosti* (4 byty) je formát uložení:

- jeden znaménkový bit
- 8 bitů pro uložení exponentu (rozsah od -126 do +127)
- 23 bitů normalizované mantisy

což odpovídá zhruba přesnosti 7 dekadických cifer, $\log_{10}(2^{23+1}) \approx 7.22$. Příkladem může bý C++ typ `float`.

Pro *dvojnásobnou přesnost* (8 bytů) a 52-bitovou mantisou je přesnost zobrazení řádově 16 dekadických cifer. Příkladem může být C++ typ `double`.

Čísla v plovoucí řádové čárce

Příklad: Jak je zobrazeno číslo -115.75 v jednoduché přesnosti.

$$(115)_{10} = 2^6 + 2^5 + 2^4 + 2^1 + 2^0 = (1110011)_2$$
$$(0.75)_{10} = (0.11)_2$$
$$(115.75)_{10} = (1110011.11)_2$$

V normalizovaném tvaru 1.11001111×2^6

IEEE osmibitový exponent $(2^7 - 1) + 6 = 133 = (10000101)_2$

Ve výsledném zobrazení se neukládá první binární jednička normalizovaného tvaru

1	10000101	11001111000000000000000

V daném příkladu byla zvolena desetinná část tak, aby měla ukončený binární rozvoj. Pokud binární rozvoje není ukončený nebo přesahuje rozsah dané mantisy, dochází při ukládání čísel ke ztrátě přesnosti.

Číslo podmíněnosti

- Funkce matlabu `cond(A)` počítá číslo podmíněnosti matice `A`.
- Pro soustavu $Ax = b$ určuje číslo podmíněnosti maximální poměr relativní chyby řešení x vzhledem k relativní chybě b.
- Pro čtercovou regulární matici A je číslo podmíněnosti

$$\kappa(A) = ||A^{-1}|| \cdot ||A||$$

  pro některou z maticových norem ($||\cdot||_1,\ ||\cdot||_\infty,\ ||\cdot||_2$).
- $\kappa(A) \geq 1$.
- Z praktického hlediska $\kappa(A) \approx 10^k$ znamená, že můžeme očekávat ztrátu až k dekadických cifer při řešení soustavy $Ax = b$.
- Soustavy s velkámi hodnotami $\kappa(A)$ jsou *špatně podmíněné* (a naopak soustavy s malým číslem podmíněnosti jsout *dobře podmíněné*).

Příklad špatně podmíněné soustavy

Soustava $Ax = b$, kde $A = \begin{pmatrix} 1 & 1 \\ 1 & 1+\epsilon \end{pmatrix}$ je pro malé hodnoty $\epsilon = 10^{-N}$ špatně podmíněná (pro $\epsilon = 0$ je singulární).

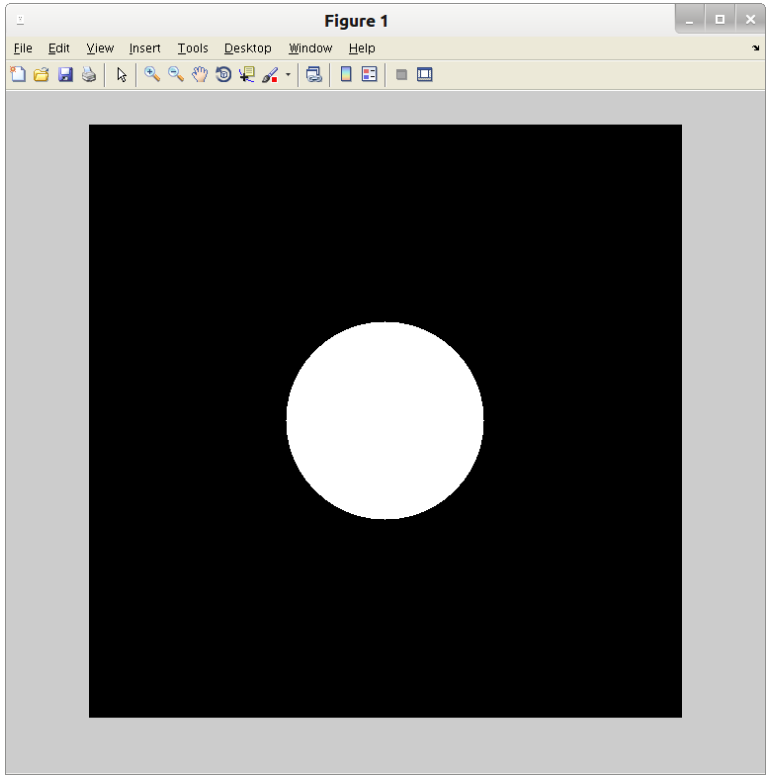
Pro zvolenou konstantu K a vektor $b = \begin{pmatrix} 2K \\ (2+\epsilon)K \end{pmatrix}$ je řešením soustavy vektor $x = \begin{pmatrix} K \\ K \end{pmatrix}$.

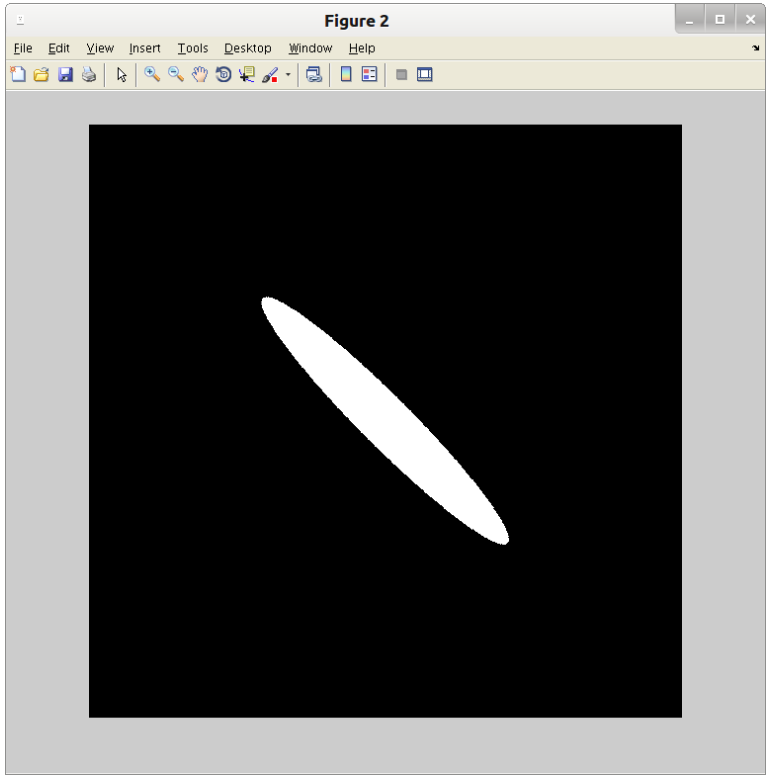
Inverzní matice $A^{-1} = \begin{pmatrix} 1+\epsilon & -1 \\ -1 & 1 \end{pmatrix}$ a odhad čísla podmíněnosti $\kappa(A)$ je $2 * (2+\epsilon)/\epsilon$.

$K = \pi,\ N =$	1	2	5	13	15
$\kappa(A)$	42.08	402.0	4.00e+05	4.01e+13	3.18e+15
chyba x_1	3.6e-15	1.0e-13	1.1e-10	0.0068	0.86
chyba x_2	-1.1e-14	-1.2e-13	-1.2e-10	-0.0088	-1.14

```
function [cpodm, chyba] = nestabilni1 (N, K)
  % funkce [cpodm, chyba] = nestabilni1 (N, K) generuje demo
  % soustavu s potencialne spatne podminenou matici soustavy
  % Vstupni parametr N udává mocninu cisla epsilon = 10^-N,
  % konstanta, ktera definuje velikost clenu vektoru absolut
  % clenu.
  %
  % e = 10^-N;                          epsilon
  % A = [ 1, 1;  1, 1+e];               Ax = b
  % b = [2*K; (2 + e)*K];               x je [K; K]
  %
  % Vystupni parametr cpodm je cislo podminenosti vygenerova
  % soustavy, chyba je skutecna chyba reseni soustavy Ax=b.

  e = 10^-N;                          % epsilon
  A = [ 1, 1;  1, 1+e];               % inverze  [1+e, -1; -1,
1 ]/e;
  b = [2*K; (2 + e)*K];               % Ax = b

  cpodm = cond(A);                    % odhad ||inv(A)||*||A|| je
  chyba = inv(A)*b - [K; K];          % skutecna chyba reseni
```

Geometrický význam čísla podmíněnosti ve 2D





Zdrojový kód `condNumDemo.m`

Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

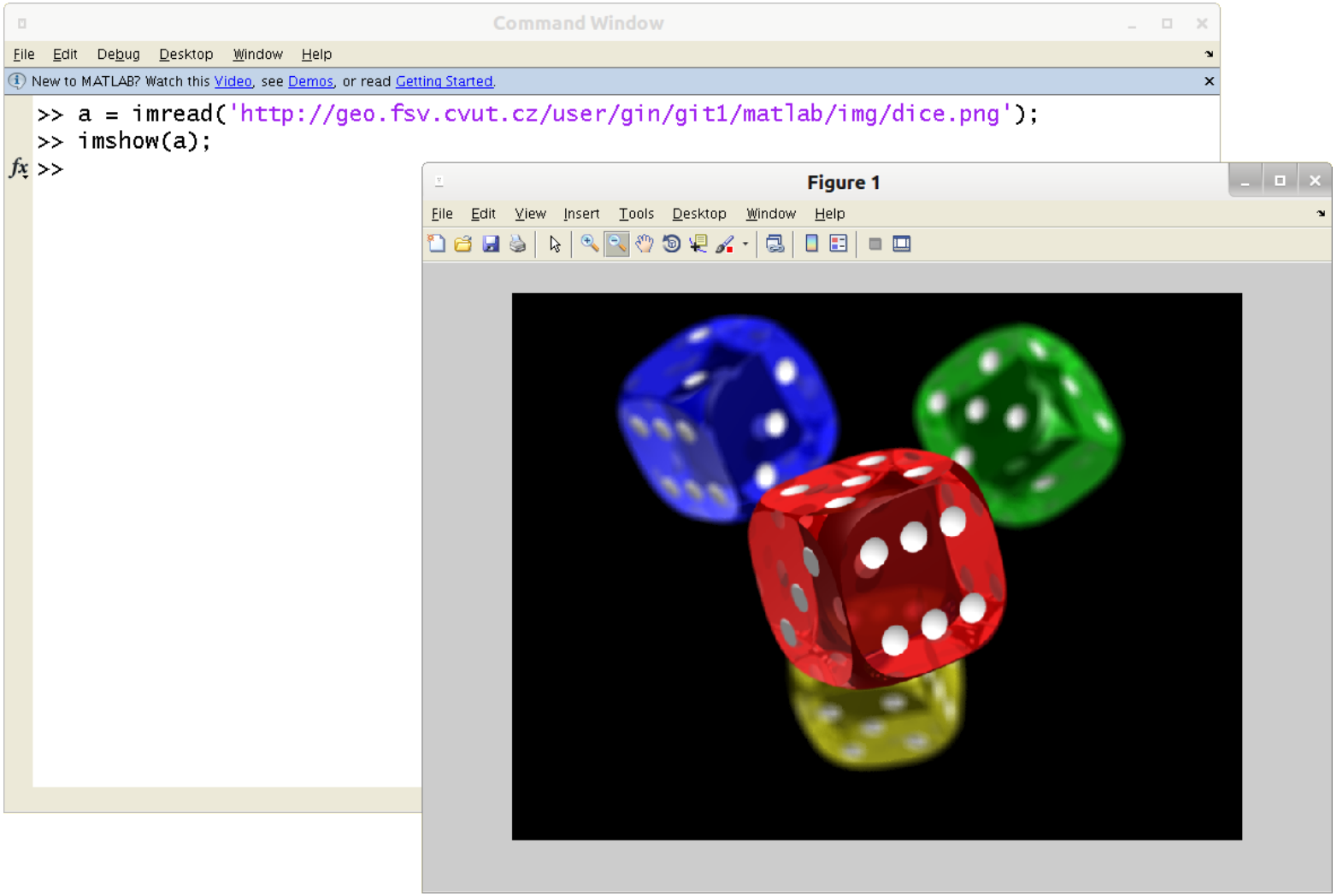
Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Obrazová data

dice



Obrazová data — rastry

Digitální obrazová data jsou sestavena z pixlů (*pixel*), pro digitální obraz je stanoveno, jak mají být jednotlivé pixly obarveny. Obecně ale rastrová data mohou vyjadřovat i jiné, než vizuální informace, například intenzitu, teplotu a pod. Pro zjednodušení budeme obrazová data a rastry považovat za volná synonyma.

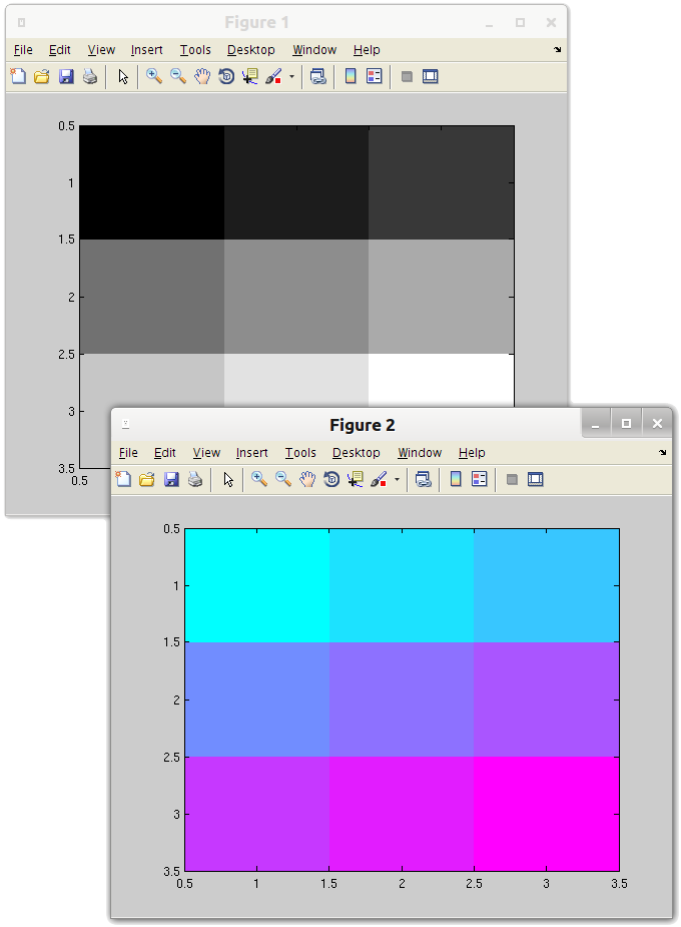
Z pohledu uživatel je rastr pole s $M \times N$ pixely, které lze indexovat stejně jako prvky matice. Matlab poskytuje obecné rozhraní, pro přístup k rastrovým obrázkům pro všechny běžné formáty, například

BMP	Microsoft Windows Bitmap
GIF	Graphics Interchange Format
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
TIFF	Tagged Image File Format

Příkazy image a imagesc

Příkaz `image` zobrazuje matici jako *obrázek*. Příkaz `imagesc` se liší pouze v tom, že škáluje data tak, aby bylo využito celé barevné schéma.

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

figure(1), imagesc(A);
colormap gray;

figure(2), imagesc(A);
colormap cool;
```

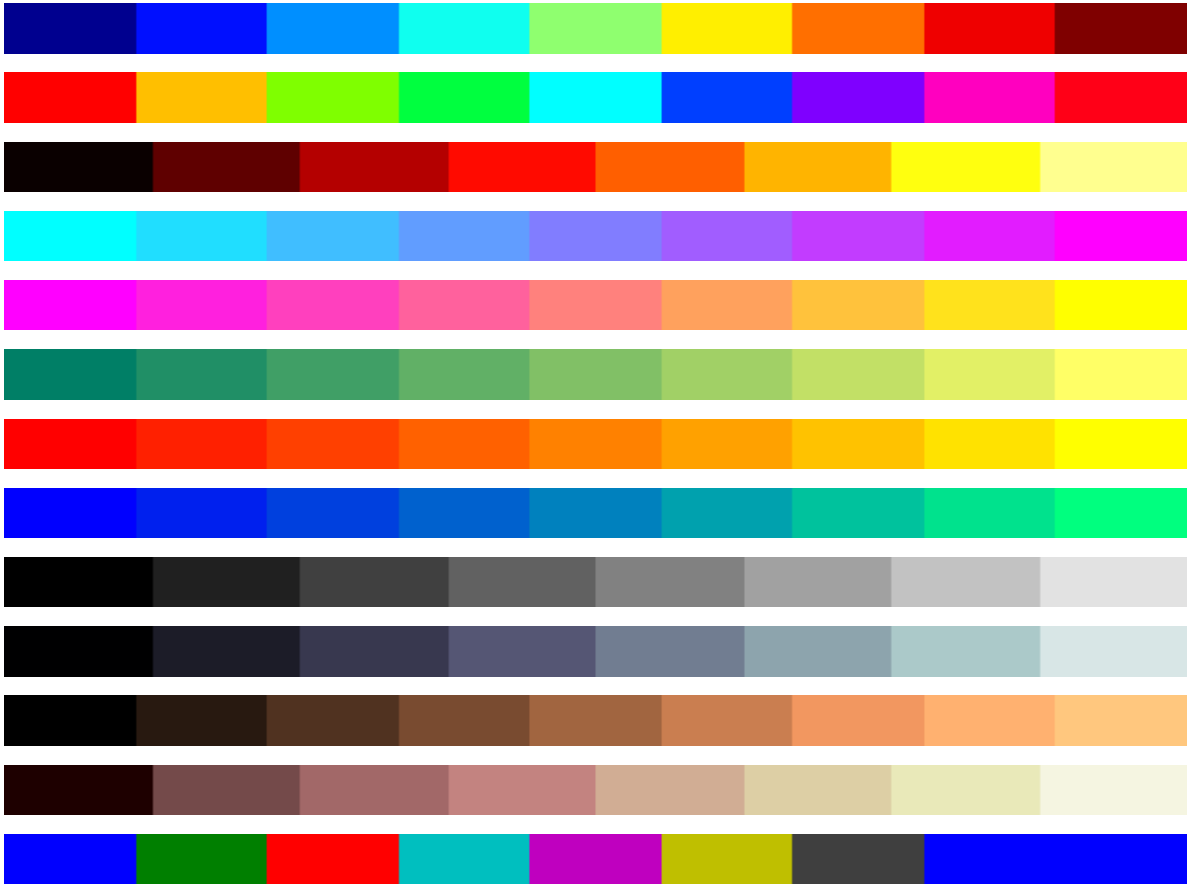


- Příkaz `image` poskytuje další možnosti, o kterých se zmíníme později.
- Osa *x* na obrázku směřuje *vpravo* (tj. ve směru čísel sloupců matice), osa *y* *dolů* (ve směru číslování řádků).

Colormap (barevné schéma)

Jména přednastavených barevných schémat.

jet (default)
hsv
hot
cool
spring
summer
autumn
winter
gray
bone
copper
pink
lines
custom



...

```
>> colormap cool;
```

Základní typy obrazových dat

The Image Processing Toolbox definuje čtyři základní typy obrazových dat.

Binary	*Binární obraz* je logické pole obsahující pouze nuly a jedničky, které jsou interpretovány jako černá a bílá.
Indexed	Obraz s *indexovanými barvami* je pole s prvky třídy `logical`, **`uint8`, `uint16`**, `single` nebo `double`, jehož hodnoty pixlů jsou indexy do barevného schématu (colormap). Colormap je pole $m \times 3$ třídy double. **Zeleně** jsou označeny typy používané v obrazové analýze.
Grayscale	Obraz se *stupni šedi* je pole prvků `uint8`, `uint16`, `int16`, `single` nebo `double`, jehož pixely vyjadřují intenzitu.
Truecolor	*Truecolor* nebo RGB obraz je pole dimenze $r \times s \times 3$ s prvky třídy `uint8`, `uint16`, `single` nebo `double`, jehož pixely udávají hodnoty intenzity.

Konverze matice na obraz ve stupních šedi

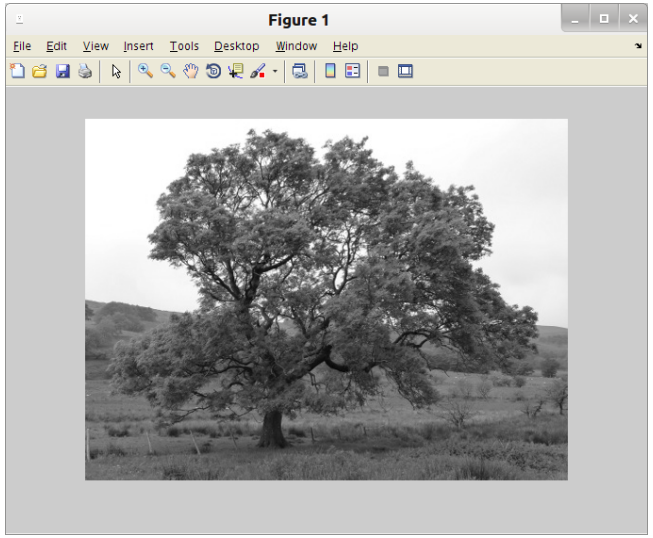
```
I = mat2gray(A, [amin amax])
I = mat2gray(A)
```

- Převádí matici A na intenzity v intervalu 0.0 (černá) až 1.0 (bílá). Hodnoty menší nebo rovno `amin` jsou převedeny na 0.0, hodnoty větší nebo rovno `amax` jsou převedeny na 1.0.
- Pokud je funkce volána pouze s jedním parametrem, jsou hodnoty `amin` a `amax` vypočteny ze vstupních dat.
- Vstupní pole A musím být logické nebo numerické, výstupem je matice s prvky typu double.

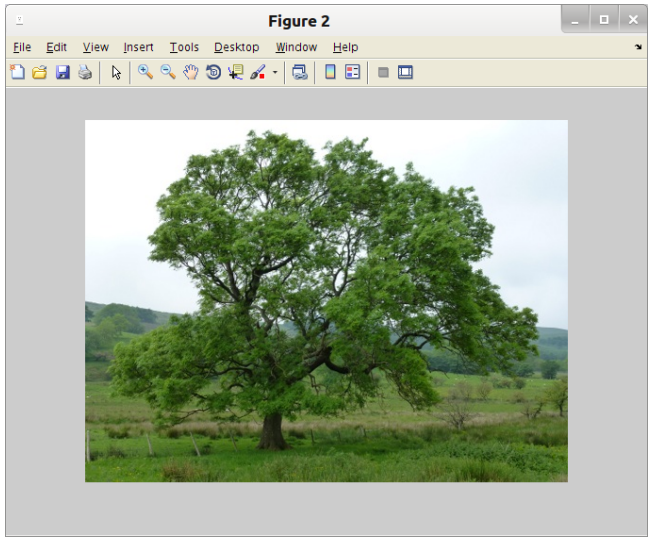
Poznámka: v softwaru GNU Octave je funkce `mat2gray` součástí balíčku *image*, který musí být nainstalován (`sudo apt-get install octave-image`) a zaveden příkazem `pkg load image`.

Obraz ve stupních šedi a truecolor (jedna matice)

```
>> gtree = imread('tree.png');
>> size(gtree)

ans =

   384   512

>> figure(1), imshow(gtree);
```

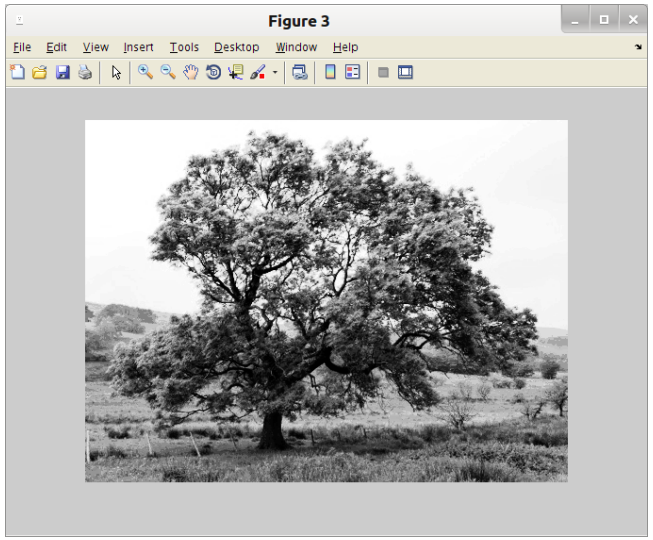



```
>> jtree = imread('tree.jpg');
>> size(jtree)

ans =

   384   512     3

>> figure(2), imshow(jtree);
```

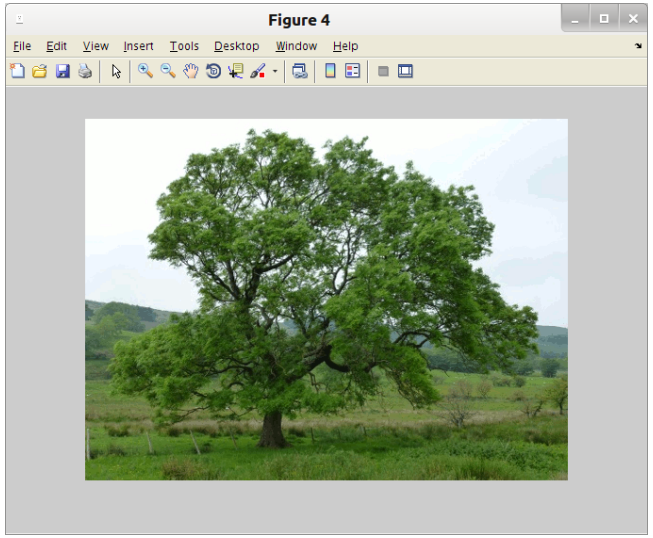


Obraz s indexovanými barvami truecolor (dvě matice)

```
>> ttree = imread('tree.tif');
>> size(ttree)

ans =
   384   512

>> figure(3), imshow(ttree);
>> % obraz je zobrazen chybně,
>> % protože chybí barevná mapa
```




```
>> [ttree tcmap]=imread('tree.tif');
>> size(tcmap)

ans =
   256     3

>> figure(4), imshow(ttree, tcmap);
>> % 256 barev ==> zkreslení mraků
```



imfinfo - informace o obrazu typu grayscale

```
>> imfinfo('tree.png')
ans =
                  Filename: 'tree.png'
               FileModDate: '03-Mar-2013 00:36:22'
                  FileSize: 118042
                    Format: 'png'
             FormatVersion: []
                     Width: 512
                    Height: 384
                  BitDepth: 8
                 ColorType: 'grayscale'
           FormatSignature: [137 80 78 71 13 10 26 10]
                  Colormap: []
                 Histogram: []
             InterlaceType: 'none'
              Transparency: 'none'
    SimpleTransparencyData: []
           BackgroundColor: []
           RenderingIntent: 'perceptual'
            Chromaticities: [1x8 double]
                     Gamma: 0.4545
               XResolution: 2835
```

imfinfo - informace o obrazu typu truecolor

```
>> imfinfo('tree.jpg')
ans =
            Filename: 'tree.jpg'
         FileModDate: '02-Mar-2013 20:03:36'
            FileSize: 110732
              Format: 'jpg'
       FormatVersion: ''
               Width: 512
              Height: 384
            BitDepth: 24
           ColorType: 'truecolor'
     FormatSignature: ''
     NumberOfSamples: 3
        CodingMethod: 'Huffman'
       CodingProcess: 'Sequential'
             Comment: {}
                Make: 'Panasonic'
               Model: 'DMC-TZ20'
         Orientation: 1
         XResolution: 72
         YResolution: 72
      ResolutionUnit: 'Inch'
```

imfinfo - informace o obrazu typu indexed

```
>> imfinfo('tree.tif')
ans =
                    Filename: 'tree.tif'
                 FileModDate: '02-Mar-2013 22:23:03'
                    FileSize: 137596
                      Format: 'tif'
                FormatVersion: []
                       Width: 512
                      Height: 384
                    BitDepth: 8
                   ColorType: 'indexed'
             FormatSignature: [73 73 42 0]
                   ByteOrder: 'little-endian'
              NewSubFileType: 0
               BitsPerSample: 8
                 Compression: 'LZW'
   PhotometricInterpretation: 'RGB Palette'
                 StripOffsets: [6x1 double]
             SamplesPerPixel: 1
                RowsPerStrip: 64
             StripByteCounts: [6x1 double]
                 XResolution: 72
```

Obraz truecolor, RGB komponenty a intenzity

Vytvoření obrázků z předchozího příkladu

rozklad_rgb.m

```
I = imread('http://geo.fsv.cvut.cz/user/gin/git1/matlab/img/
R = I;    R(:,:,[2 3]) = 0;
G = I;    G(:,:,[1 3]) = 0;
B = I;    B(:,:,[1 2]) = 0;

figure(1),
subplot(3,3,2), imshow(I),
subplot(3,3,4), imshow(R),
subplot(3,3,5), imshow(G),
subplot(3,3,6), imshow(B),
subplot(3,3,7), imshow(mat2gray(R(:,:,1),[0 255])),
subplot(3,3,8), imshow(mat2gray(G(:,:,2),[0 255])),
subplot(3,3,9), imshow(mat2gray(B(:,:,3),[0 255]));
```

Tři algoritmy pro konverzi RGB na stupně šedi

lightness (světlost)

$$(\max(R, G, B) + \min(R, G, B))/2$$

avarage (průměr)

$$(R + G + B)/3$$

luminosity (jas) Protože je lidské oko citlivější na zelenou, než na jiné barvy, má zelená složka vyšší váhu.

$$0.2989 * R + 0.5870 * G + 0.1140 * B$$

.

truecolor

lightness

avarage

luminosity

Vytvoření obrázků z předchozího příkladu

algoritmy_rgb2grayscale.m

```
R = imread('flower.png');
[M N K] = size(R);
A = zeros(M,N);   B = A;   C = A;
for r = 1:M
   for c = 1:N
      A(r,c) = (max(R(r,c,:)) + min(R(r,c,:)))/2;
      B(r,c) =  sum(R(r,c,:))/3;
      C(r,c) = 0.2989*R(r,c,1) + 0.5870*R(r,c,2) + 0.1140*R(
   end
end
A = mat2gray(A, [0 255]);
B = mat2gray(B, [0 255]);
C = mat2gray(C, [0 255]);

imwrite(A, 'flower-a.png', 'png');
imwrite(B, 'flower-b.png', 'png');
imwrite(C, 'flower-c.png', 'png');
```

Datové typy a konverzní funkce

Implicitní třídou prvků matic je v Matlabu typ double, při zpracování, resp. uklání, dat je ale někdy nutno explicitně pracovat i s dalšími typy, které uvádí tabulka. Jména typů jsou zároveň i jmény konverzních funkcí.

double	Konverze na double precision
single	Konverze na single precision
int8	Konverze na 8-bit signed integer
int16	Konverze na 16-bit signed integer
int32	Konverze na 32-bit signed integer
int64	Konverze na 64-bit signed integer
uint8	Konverze na 8-bit unsigned integer
uint16	Konverze na 16-bit unsigned integer
uint32	Konverze na 32-bit unsigned integer
uint64	Konverze na 64-bit unsigned integer

Třídu (jméno datového typu) objektu určuje funkce `class(X)`.

Typ objektu můžeme explicitně určit, například jako třetí parametr funkce

```
M = zeros(2, 3, 'uint16').
```

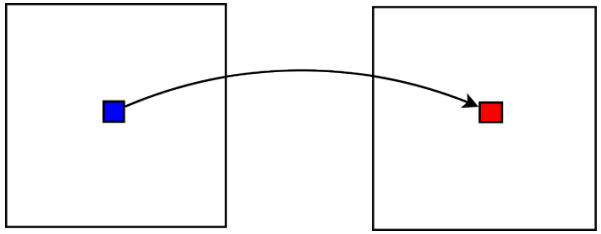
Funkce pro převod obrazových formátů

gray2ind	Konverze obrazů ve stupních šedi na indexovaný soubor.
im2bw	Konverze obrazů ve stupních šedi, indexovaných souborů a truecolor obrazů na binární (černobílé) soubory.
ind2gray	Konverze indexovaných souborů na obrazy ve stupních šedi.
ind2rgb	Konverze indexovaných souborů na truecolor (RGB).
mat2gray	Konverze datové matice na obraz ve stupních šedi, škálování dat.
rgb2gray	Konverze truecolor obrazů (RGB) na obrazy v barvě šedi.
rgb2ind	Konverze truecolor obrazů (RGB) na indexované obrazy.

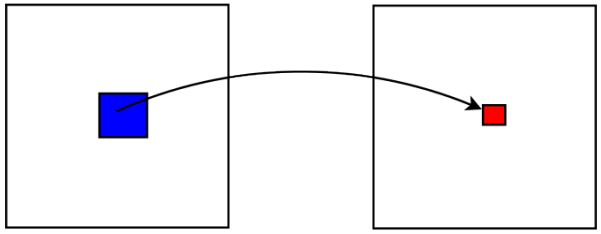
Transformace obrazových dat

Transformace obrazových dat (filtry) můžeme rozdělit do tří základních kategorií (**matematický pohled**, **terminologie obrazové analýzy**).

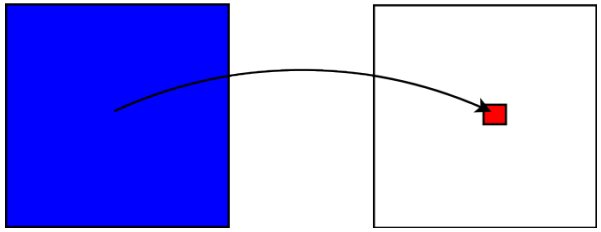
Bodové transformace (**roztažení histogramu**) zobrazují vždy jeden pixel rastru na jeden pixel výstupního rastru o stejných souřadnicích. Časová náročnost je $\mathcal{O}(\mathrm{konst.})$, někdy se jim také říká *lineární filtry.*



Lokální transformace (**filtrace**) určují výslednou hodnotu pixelu jako funkci jistého okolí (blok $k \times \ell$ pixelů); časová náročnost je $\mathcal{O}(p^2)$, kde p je velikost daného okolí.

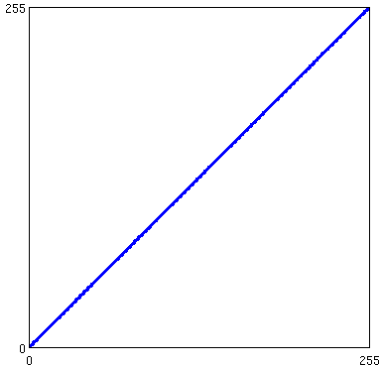


Globální transformace, kde každý výsledný pixel je funkcí celého vstupního rastru a časová náročnost je typicky řádu $\mathcal{O}(N^2)$.



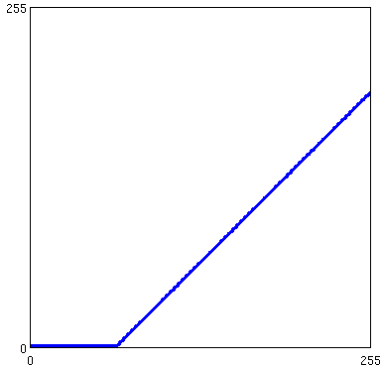
Jednoduché bodové transformace

$y = x$



původní snímek

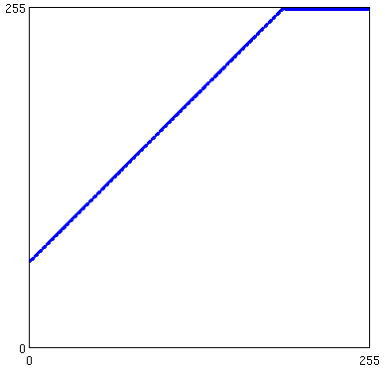
$y = x - K$



`imsubtract(t,64)`

odečteno 64

$y = x + K$

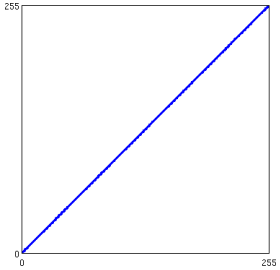


`imadd(t,64)`

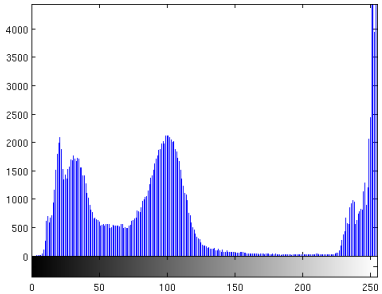
přičteno 64

Jednoduché bodové transformace, histogram

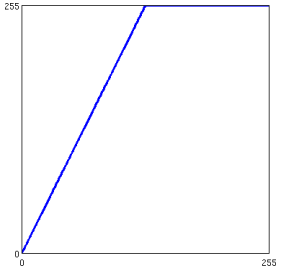
$$y = x$$



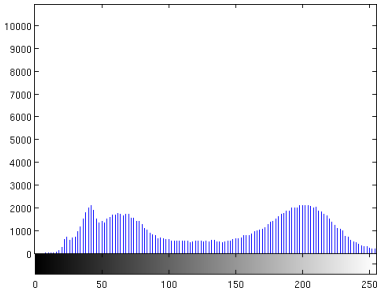
původní snímek



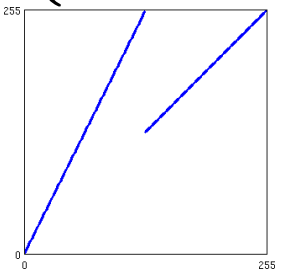
$$y = 2x$$



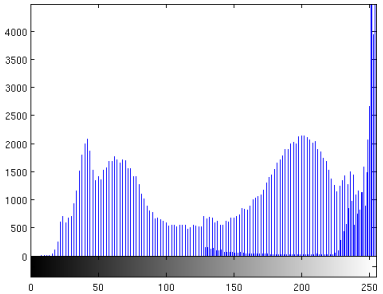
`immultiply(tt,2)`



$$y = \begin{cases} 2x, & x \leq H \\ x, & x > H \end{cases}$$

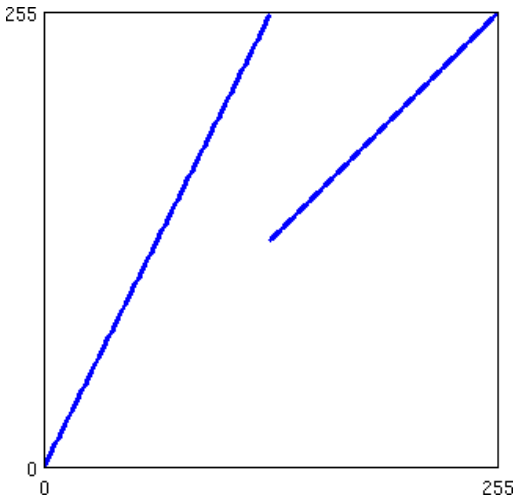


viz další *slide*



Kód pro upravu snímku z předchozího příkladu

$$y = \begin{cases} 2x, & x \leq H \\ x, & x > H \end{cases}$$



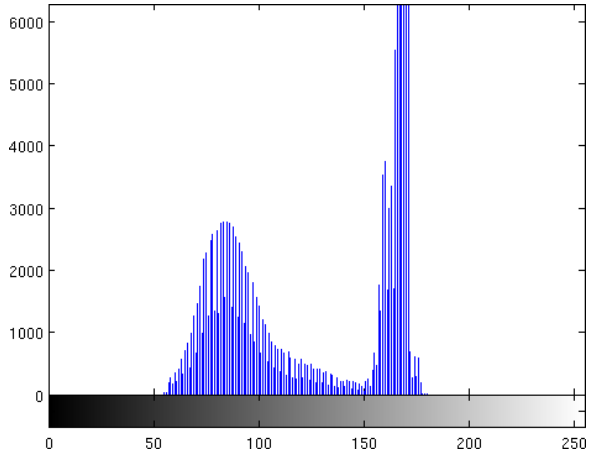
```
t3 = imread('tvrz.png');
H  = 127;
[R S] = size(t3);
for r = 1:R
    for s = 1:S
        pix = t3(r,s);
        if pix <= H
            pix = 2*pix;
        end
        t3(r,s) = (pix);
    end
end
```

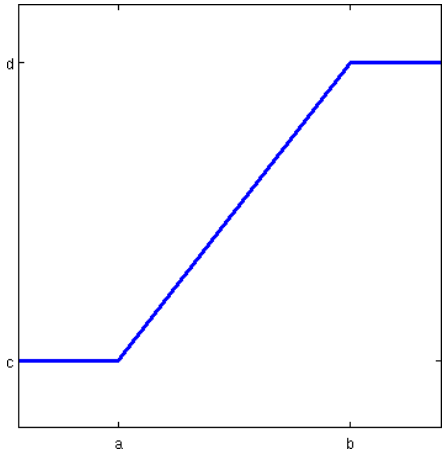
Zobrazeni histogramu

```
figure, imhist(T);
```

Vyrovnání histogramu, imadjust

Vyrovnání histogramu (ekvalizace) je proces, který transformuje hodnoty jasu tak, aby byly rovnoměrně(ji) rozloženy. Histogram na prvním obrázku (vlevo) má hodnoty koncetrovány mimo krajní hodnoty.





Funkce `imadjust(T, [a b] [c d])` vyrovnává hodnoty jasu podle funkce znázorněné na druhém obrázku (vpravo)

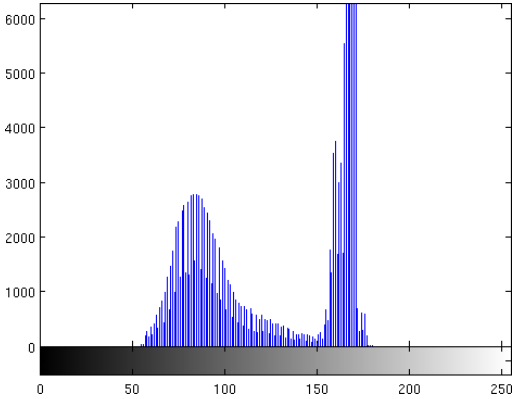
$$y = \left(\frac{x-a}{b-a}\right)(d-c) + c.$$

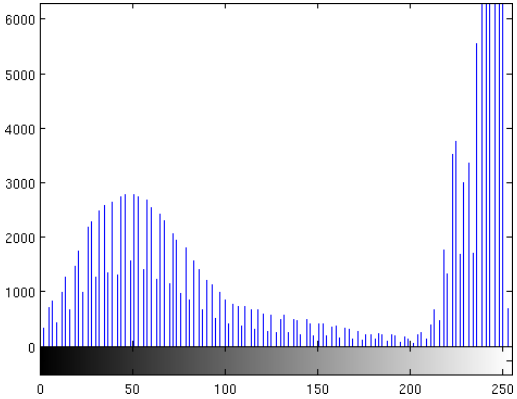
Vyrovnání histogramu, imadjust

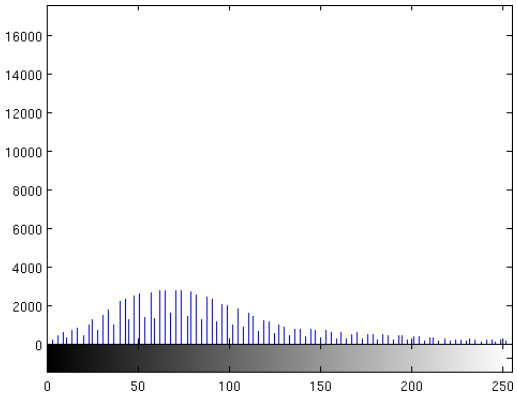
původní snímek

`imadjust(T)`

`imadjust(T,...`
`[60/255; 150/255])`



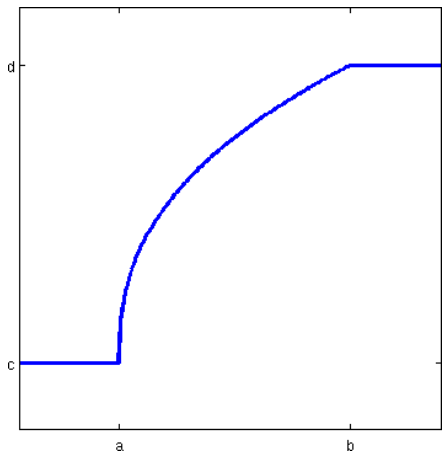




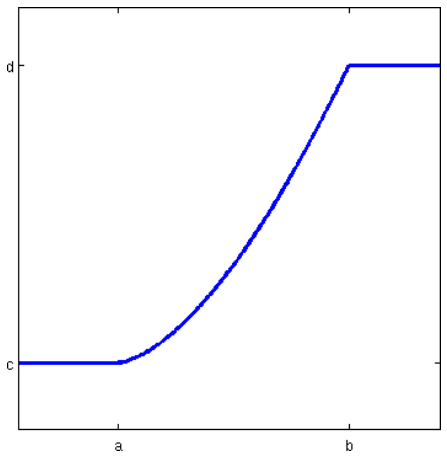
Vyrovnání histogramu, imadjust, gama korekce

Gama korekce γ, čtvrtý nepovinný parametr funkce `imadjust`, je zobecněním implicitního lineární vztahu pro vyrovnání histogramu (ekvalizace histogramu)

$$y = \left(\frac{x-a}{b-a}\right)^{\gamma}(d-c)+c$$



$\gamma < 1$



$\gamma > 1$

```
imadjust(T, [], [], 0.4)
imadjust(T, [a b], [c d], 1.6)
```

Lokální transformace - medfilt2

Filtr `medfilt2` je příkladem *lokální transformace,* kdy každý pixel je funkcí (implicitně) 3×3 okolí odpovídajícího pixelu vstupního obrazu (v daném případě je výslednou hodnotou medián příslušné submatice).

Šum ve vstupních datech (vlevo) byl vygenerován funkcí `imnoise(K1, 'salt & pepper')`, viz následující *slide*.

Funkce pro generování šumu - imnoise

```
K1 = imread('kucharky.tiff');
figure(1), imshow(K1);
K2 = imnoise(K1, 'salt & pepper');
figure(2), imshow(K2);
K3 = medfilt2(K2);
figure(3), imshow(K3);
```

Funkce `imnoise(M, ŠUM, ...)` umožňuje generovat *šum* pro následující rozdělení (parametr se zadává jako řetězec)

'gaussian' Gaussovo rozdělení $J = I + N$.

'localvar' Gaussovo rozdělení s nulovou střední hodnotou, jehož variance je definována zvlášť pro každý pixel (pole V).

'poisson' Poissonovo rozdělení.

'salt & pepper' (pepř a sůl) pixely nastavené na bílou a černou hodnotu.

'speckle' (flíčky) multiplikativní šum $J = I(1 + N)$, s implictiní střední hodnou 0 a variancí 0.04.

Lokální transformace - wiener2

```
url = 'http://geo.fsv.cvut.cz/user/gin/git1/matlab/img/';
K1 = imread(strcat(url,'kucharky.tiff'));
figure(1), imshow(K1);
K2 = imnoise(K1, 'gaussian', 0, 0.005);
figure(2), imshow(K2);
K3 = wiener2(K2, [5 5]);
figure(3), imshow(K3);
```

Algoritmus filtrování šumu ve funkci wiener2

Funkce `wiener2` odhaduje pro dané okolí $M \times N$ a pro každy pixel lokální střední hodnotu μ a varianci σ^2.

$$\mu = \frac{1}{MN} \sum_{n_1, n_2 \in \eta} a(n_1, n_2) \qquad \text{a} \qquad \sigma^2 = \frac{1}{MN} \sum_{n_1, n_2 \in \eta} a^2(n_1, n_2) - \mu^2,$$

kde η je $M \times N$ lokální okolí každého pixelu rastru A. Funkce `wiener2` definuje po pixelech Wienerův filter s využitím těchto odhadů

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - \nu^2}{\sigma^2} \left(a(n_1, n_2) - \mu \right),$$

kde ν^2 je variance šumu. Není-li variance šumu známa, použije `wiener2` průměr ze všech lokálních odhadů variance.

Detekce hran

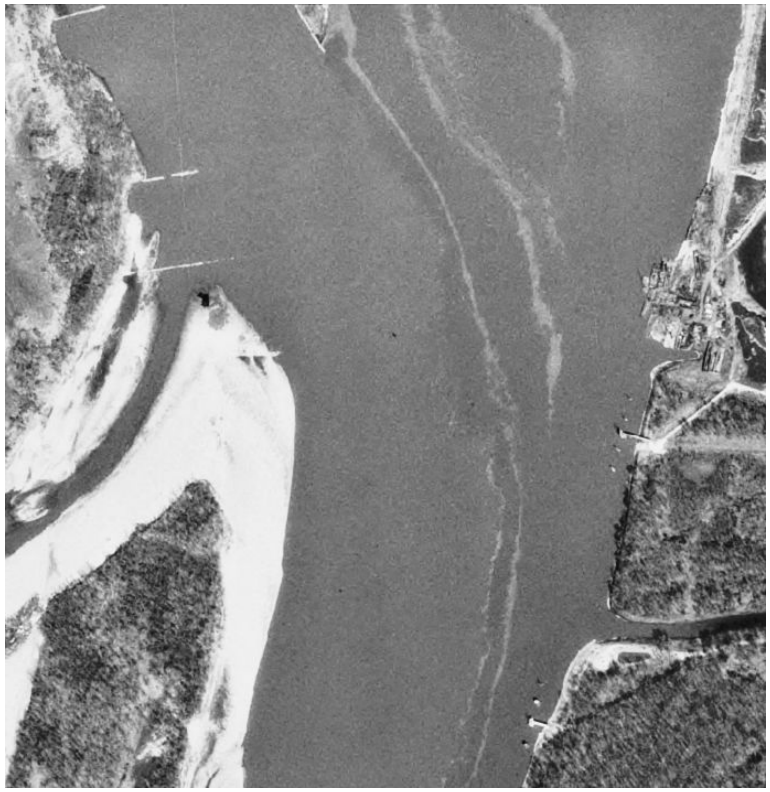
Funkce `edge` poskytuje šest algoritmů pro detekci hran v rastru se stupni šedi (intenzity).

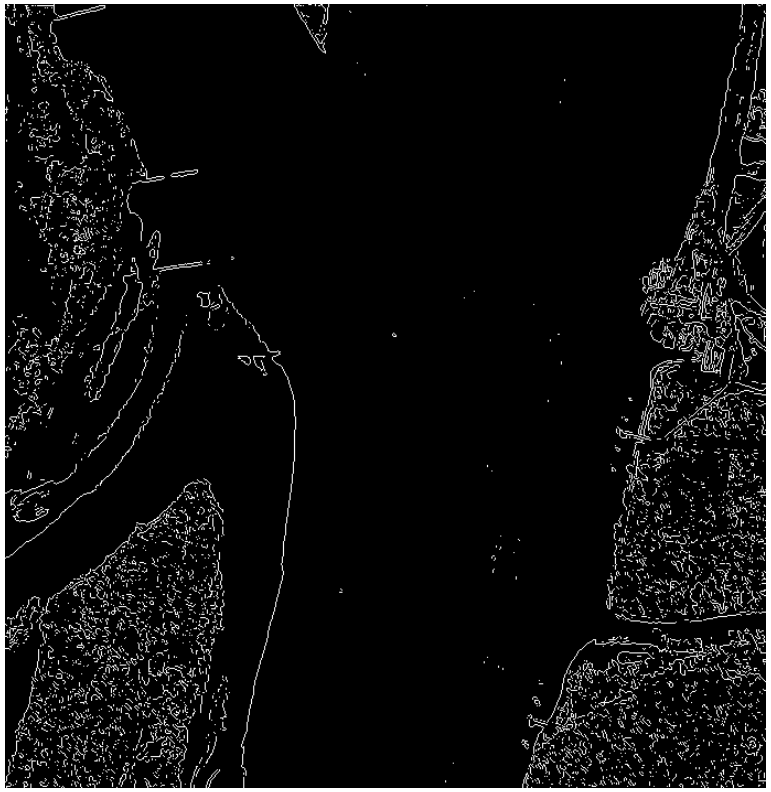
'log'	Laplacian of Gaussian filter
'prewitt'	Prewitt horizontal edge-emphasizing filter
'sobel'	Sobel horizontal edge-emphasizing filter
'roberts'	specifies the Roberts method.
'zerocross'	specifies the zero-cross method.
'canny'	Canny method

```
url = 'http://geo.fsv.cvut.cz/user/gin/git1/matlab/img/';
K1 = imread(strcat(url,'soutok.jpg'));
K2 = edge(K1, 'sobel');
figure(2), imshow(K2);
```

Detekce hran pomocí filtru ’sobel’

Soutok řek Missouri a Missisipi





Detekce hran - různé algoritmy

log

prewitt

sobel

roberts

zerocross

canny

Zaostření snímku - fspecial('unsharp')

Jeden z algoritmů pro zaostření snímků je *Laplacian Edge enhacement*, který vede na následující 3×3 kernel (resp. verzi s volitelným parametrem α)

$$\Delta h = \frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \quad \rightarrow \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \frac{1}{1+\alpha}\begin{bmatrix} \alpha & 1-\alpha & \alpha \\ 1-\alpha & -4 & 1-\alpha \\ \alpha & 1-\alpha & \alpha \end{bmatrix}$$

Laplacián se liší od ostatních filtrů v tom, že uvažuje i druhé diference.

Neformální zavedení konvoluce pro 2D rastry

Diskrétní konvoluce je proces (filtr), který převádí vstupní pixely rastru f na výstup g jako funkci vstupního rastru a *kernelu* h podle vztahu

$$g(i,j) = f * h = \sum_m \sum_n f(i+m, j+n)\, h(p+m, q+n)$$

kde f je vstupní rastr a matice h je kernel (filtru), typicky dimenze 3×3 (v tom případě $p = q = 2$ a indexy m, n nabývají hodnot -1, 0 a 1) nebo 5×5.

Příklad:

$$\begin{bmatrix} 6 & 8 & 22 & 29 & 28 \\ 24 & \mathbf{16} & \mathbf{10} & \mathbf{16} & 10 \\ 45 & \mathbf{23} & \mathbf{31} & \mathbf{31} & 34 \\ 36 & \mathbf{58} & \mathbf{50} & \mathbf{68} & 48 \\ 30 & 41 & 50 & 62 & 57 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix} = (10 \times -1) + (\mathbf{31} \times \mathbf{1}) = \mathbf{21}$$

Sčítáme součiny pozičně odpovídajících si prvků (nenásobíme submatice).

Příklad — výpočet zaostření

```
url = 'http://geo.fsv.cvut.cz/user/gin/git1/matlab/img/';
D = double(imread(strcat(url,'kucharky.tiff')));

[R,C] = size(D);
S = zeros([R,C],'uint8');
a = 0.2;
for r=2:R-1        % pro zjednuseni ignorujeme okraje snimku
    for c=2:C-1    % a vstupni rastr ulozime jako tridu doubl
        x = 0;
        if c >= C/2
            x = 1/(1+a)*( ...
            D(r-1,c-1)*a + D(r-1,c)*(1-a) + D(r-1,c+1)*a + .
            D(r,c-1)*(1-a) - D(r,c)*4 + D(r,c+1)*(1-a) + ...
            D(r+1,c-1)*a + D(r+1,c)*(1-a) + D(r+1,c+1)*a ...
        );
        end
        S(r,c) = uint8(D(r,c) - x);
    end
end
S(:,C/2:C/2) = 180;  % explicitne oddelime zaostrenou cast
```

Příklad - vygenerovaný obrázek

Příklady několika obvyklých kernelů

average

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

gaussian

$$\begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$

log

$$\begin{bmatrix} 0.4038 & 0.8021 & 0.4038 \\ 0.8021 & -4.8233 & 0.8021 \\ 0.4038 & 0.8021 & 0.4038 \end{bmatrix}$$

laplacian

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

laplacian α

$$\frac{1}{1+\alpha}\begin{bmatrix} \alpha & 1-\alpha & \alpha \\ 1-\alpha & -4 & 1-\alpha \\ \alpha & 1-\alpha & \alpha \end{bmatrix}$$

prewitt

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

sobel

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

zvýraznění hran

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

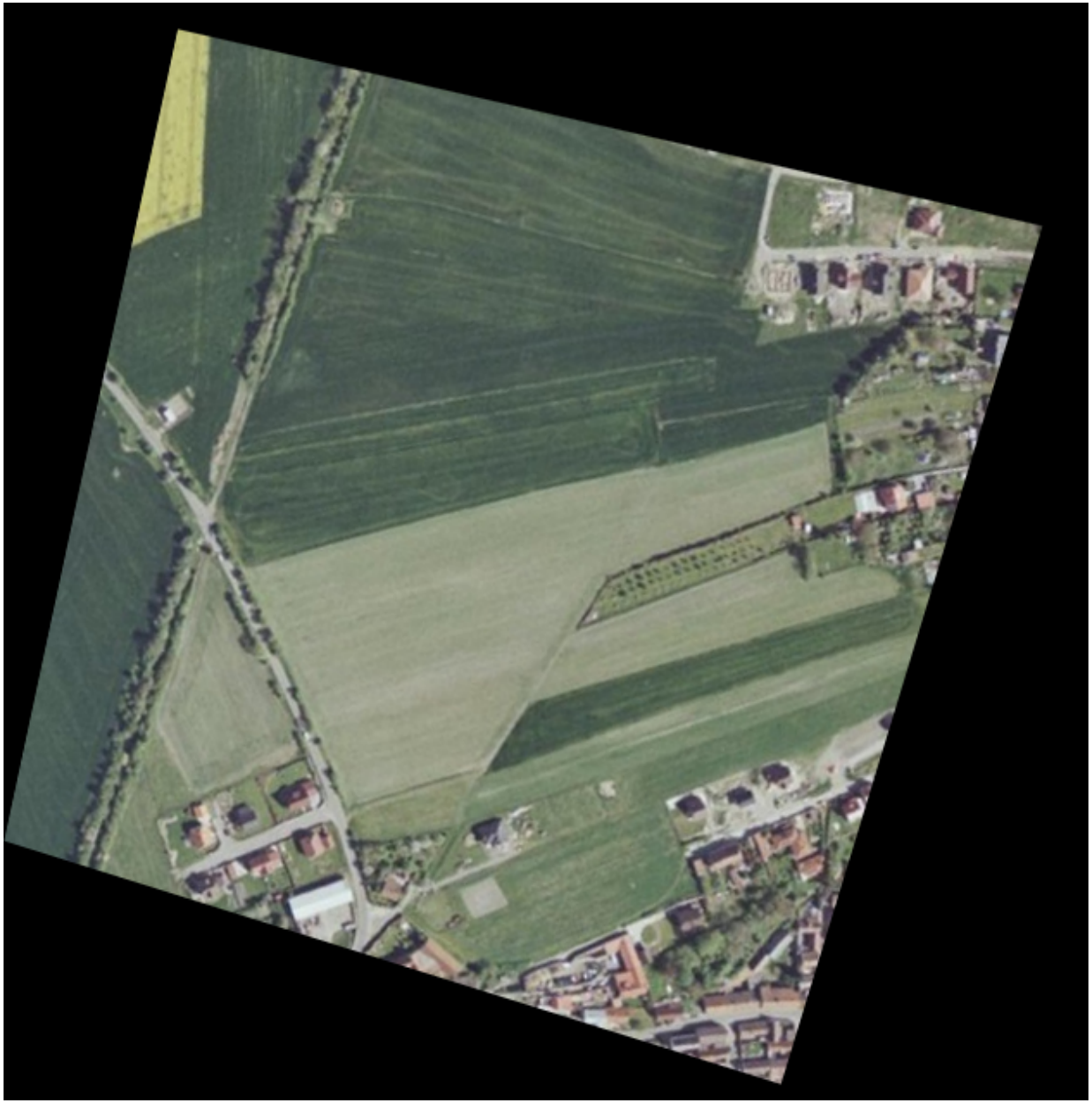
Ortofoto a projektivní transformace

- Naším úkolem je natransformovat *pohledový* snímek vpravo (perspektiva) do ortofota. Vhodnou metodou je nelineární projektivní transformace.
- Pro výpočet parametrů projektivní transformace použijeme obecnou funkci `cp2tform`, kde parametry jsou seznamy identických bodů. Transformaci snímku zajistí funkce `imtransform`, pro společné zobrazení obou snímků použijeme funkci `imshowpair`.

Ortofoto a projektivní transformace - výpočet

```
% ortofoto demo - data upravena z klad55.jpg
% http://geoportal.cuzk.cz/UKAZKOVA_DATA/ORTOFOTO_JPG.zip

% nacteni snimku
url = 'http://geo.fsv.cvut.cz/user/gin/git1/matlab/img/';
A = imread(strcat(url, 'klad55-ortofoto.jpg'));
B = imread(strcat(url, 'klad55-perspektiva.jpg'));

% identicke body (x,y)
A_ib = [115 123; 445 177; 467 302; 332 362;
        279 446;  95 234; 389 465; 170 533];
B_ib = [ 42  93; 409  70; 467 174; 345 260;
        312 353;  48 200; 436 337; 214 467];

% obraz B je trucolor, tj. pole o trech dimenzich,
% viz parametry XData a YData
P = cp2tform(B_ib, A_ib, 'projective');
Q = imtransform(B, P, 'XData',[1 size(B,2)], ...
                'YData',[1 size(B,1)]);
```

Transformovaný snímek (perspektiva → ortofoto)



Výsledné ortofoto (falsecolor)

Výsledné ortofoto (blend)

Vložení transformovaného rastru po pixelech

```
P = cp2tform(B_ib, A_ib, 'projective');

R = size(A,1);
S = size(A,2);

for r = 1:size(B,1)
    for s = 1:size(B,2)
        [y, x] = tformfwd(P, s, r);
        x = round(x);
        y = round(y);
        if x < 1 || x > S || y < 1 || y > R
            continue;
        end
        A(x,y,1) = B(r,s,1);
        A(x,y,2) = B(r,s,2);
        A(x,y,3) = B(r,s,3);
    end
end
```

Výsledné ortofoto vložené po pixlech

Příklad přímého výpočtu projektivní transformace

$$X = \frac{ax + by + c}{gx + hy + 1}, \qquad Y = \frac{dx + ey + f}{gx + hy + 1}$$

Koeficienty rovnic oprav získáme vynásobením transformačních rovnic společným jmenovatelem pravých stran

a	b	c	d	e	f	g	h	r
x	y	1				$-xX$	$-yX$	X
			x	y	1	$-xY$	$-yY$	Y

V maticovém zápisu

$$\mathbf{Mq} = \mathbf{r}, \qquad \text{kde} \qquad \mathbf{q} = [a\ b\ c\ d\ e\ f\ g\ h]'$$

S využitím singulárního rozkladu

$$[\mathbf{U}, \mathbf{W}, \mathbf{V}] = \mathrm{svd}(\mathbf{M}), \qquad \mathbf{UWV}' = \mathbf{M}, \qquad \mathbf{q} = \mathbf{VW}^{-1}\mathbf{U}'\mathbf{r}$$

kde inverzi diagonální matice $\mathbf{W}^{-1}$ vypočítame jako převrácené hodnoty nenulových prvků diagonály.

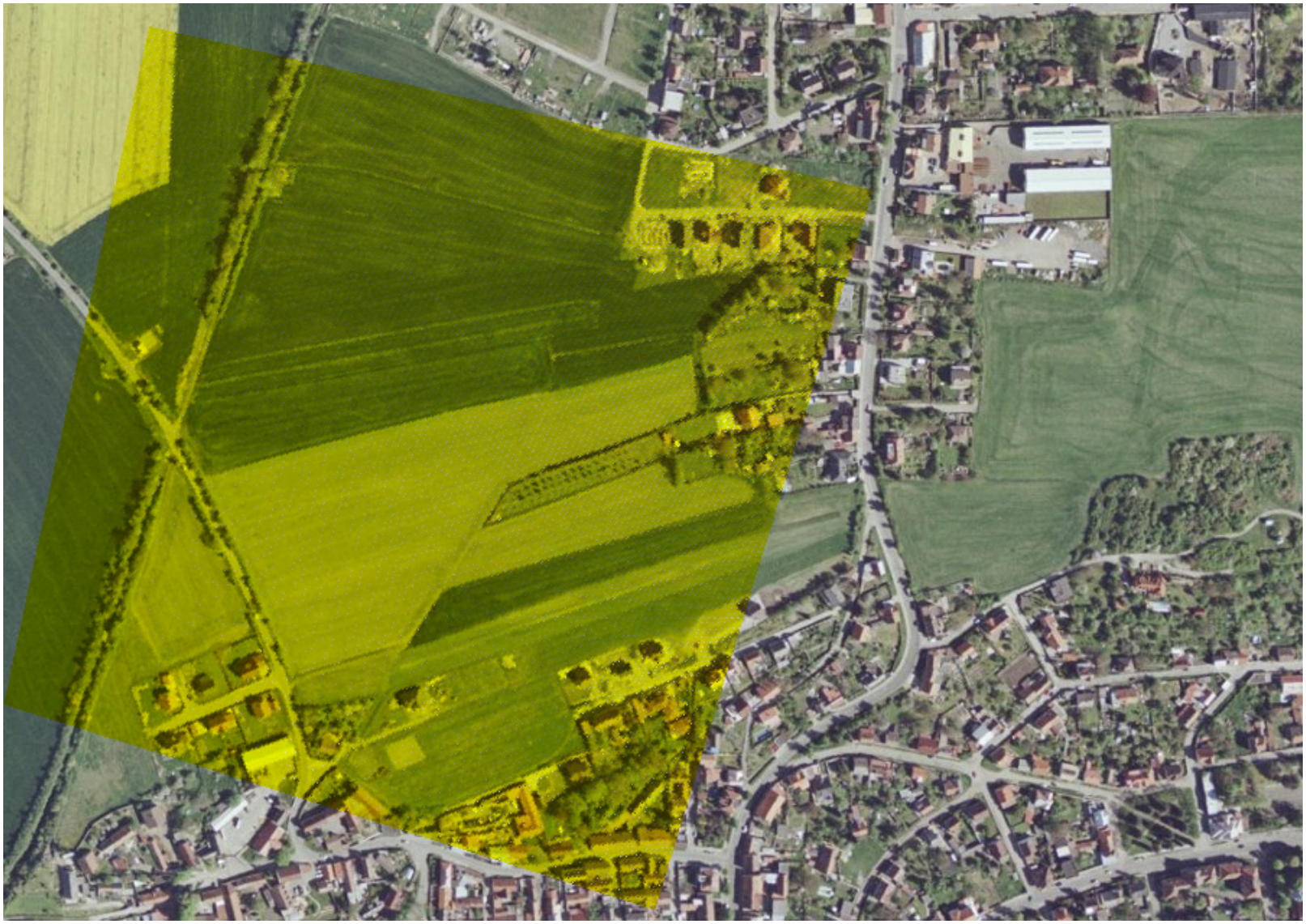
Sestavení rovnic oprav z identických bodů

```
A_ib = [115 123; 445 177; 467 302; 332 362;
        279 446;  95 234; 389 465; 170 533];
B_ib = [ 42  93; 409  70; 467 174; 345 260;
        312 353;  48 200; 436 337; 214 467];

M = zeros(size(A_ib, 1)*2, 8);    % rovnice oprav
r = zeros(size(M, 1),1);          % prava strana
j = 1;
for i = 1:size(A_ib,1)
    X = A_ib(i,1);   Y = A_ib(i,2);
    x = B_ib(i,1);   y = B_ib(i,2);
    M(j,1)=x;  M(j,2)=y;  M(j,3)=1;  M(j,7)=-x*X; M(j,8)=-y*
    r(j)  =X;  j = j + 1;
    M(j,4)=x;  M(j,5)=y;  M(j,6)=1;  M(j,7)=-x*Y; M(j,8)=-y*
    r(j)  =Y;  j = j + 1;
end
scale = max(abs(M));   scale(eq(scale,0)) = 1;
for j = 1:size(M,2)                  % zmena meritka pro zvysen
    M(:,j) = M(:,j)./scale(j);       % numerické stability rese
end;
```

Výpočet parametrů (MNČ) a transformace pixelů

```
[U, W, V] = svd(M, 'econ');        % reseni MNC
w = diag(W);    w(eq(w,0)) = 1;    w = 1./w;
q = V*diag(w)*U'*r;  % mat nasobeni pouze pro strucnejsi zap
a = q(1)/scale(1); b = q(2)/scale(2); c = q(3)/scale(3);
d = q(4)/scale(4); e = q(5)/scale(5); f = q(6)/scale(6);
g = q(7)/scale(7); h = q(8)/scale(8);

R = size(A,1);     S = size(A,2);
for r = 1:size(B,1)
    for s = 1:size(B,2)
        q =  g*s + h*r + 1;
        x = round( (a*s + b*r + c)/q );
        y = round( (d*s + e*r + f)/q );
        if x < 1 || x > S || y < 1 || y > R
            continue;
        end
        A(y,x,1) = B(r,s,1);     % tri slozky RGB
        A(y,x,2) = B(r,s,2);
        A(y,x,3) = B(r,s,3)*0;   % ### zvyrazneni vysledku ###
    end
end
```

Perspektiva transformovaná do ortofota (zvýrazněno)



Obsah

První seznámení

Čtení a zápis textových souborů

Speciální operace

Algoritmizace

Grafy funkcí a plochy

Funkce

Uložení čísel v počítači

Obrazová data
- Transformace obrazových dat

Rozdíly mezi Octave a Matlab

Rozdíly mezi Octave a Matlab

- Narozdíl od Matlabu je Octave svobodný software.
- Matlab nabízí nesrovnatelně mohutnější nabídku specializovaných softwarových řešení (toolboxes).
- Octave rozšiřuje možnosti jazyka o některé konstrukce, které mohou být z praktického pohledu programátora užitečné, představuji ale zdroj nekompatibility, více například zde.

******

Dále uvádíme rozdíly mezi oběma systémy, které se vyskytují v předchozích příkladech.