UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

# BAKALÁŘSKÁ PRÁCE

Animace v Matlabu

**Katedra matematické analýzy a aplikací matematiky**
Vedoucí bakalářské práce: **Mgr. Pavla Kouřilová, Ph.D.**
Vypracovala: **Karolína Markulčeková**
Studijní program: B1103 Aplikovaná matematika
Studijní obor: Matematika–ekonomie se zaměřením na bankovnictví
Forma studia: prezenční
Rok odevzdání: 2015

# BIBLIOGRAFICKÁ IDENTIFIKACE

**Autor:** Karolína Markulčeková

**Název práce:** Animace v Matlabu

**Typ práce:** Bakalářská práce

**Pracoviště:** Katedra matematické analýzy a aplikací matematiky

**Vedoucí práce:** Mgr. Pavla Kouřilová, Ph.D.

**Rok obhajoby práce:** 2015

**Abstrakt:** Cílem této práce je prozkoumat možnosti programu Matlab, které umožňují efektivní vizualizaci dat a tyto možnosti následně aplikovat na animace vhodné pro matematiku. Součástí práce jsou také ilustrativní příklady. Práce se skládá ze tří kapitol. V první a druhé kapitole popíši vybrané příkazy Matlabu včetně jejich základních parametrů, které následně pro vytvoření animace využiji. Ve třetí kapitole se již zaměřím na samotné způsoby vytvoření animace. Poslední sekce třetí kapitoly obsahuje ilustrativní příklady s postupy k jejich vytvoření. Postupy jsou popsány tak, aby jim porozuměli studenti nebo další běžní uživatelé tohoto programu a tak, aby povzbudili i další zájemce o vytvoření podobné animace, která by byla pro ně přínosná.

**Klíčová slova:** Matlab, příkazy, operátor, funkce, vektor, cyklus, for, graf, 2D, movie, animace

**Počet stran:** 48

**Počet příloh:** 2

**Jazyk:** český

# BIBLIOGRAPHICAL IDENTIFICATION

**Author:** Karolína Markulčeková

**Title:** Animation with Matlab

**Type of thesis:** Bachelor's

**Department:** Department of Mathematical Analysis and Application of Mathematics

**Supervisor:** Mgr. Pavla Kouřilová, Ph.D.

**The year of presentation:** 2015

**Abstract:** The objective of this thesis is to explore the possibilities of Matlab program, which allows an effective visualisation of data, and to apply these possibilities into animation appropriate for mathematics. This thesis also contains the examples. Thesis is composed of three chapters. In the first and second chapter I will outline several chosen commands of Matlab including their elementary parameters, which I will then use to create the animation. In the third chapter I will focus on specific ways to create animation. The last section of the third chapter includes the illustrative examples. Each procedure is described properly so the students and basic users of the program could understand it and so it would encourage others to create animations, which they would find useful.

**Key words:** Matlab, commands, operator, functions, vector, cycle, for, chart, 2D, movie, animation

**Number of pages:** 48

**Number of appendices:** 2

**Language:** Czech

**Prohlášení**

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně pod vedením Mgr. Pavli Kouřilové, Ph.D. a všechny použité zdroje jsem uvedla v seznamu literatury.

V Olomouci dne ..... .....

podpis

# Obsah

**Poděkování**

Ráda bych poděkovala své vedoucí bakalářské práce Mgr. Pavle Kouřilové, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

# Úvod

Tématem bakalářské práce je Animace v Matlabu. Cílem práce je popsat příkazy, funkce a postupy vytvoření animace pro matematiku, které mohou pomoci jak studentům, tak i učitelům jako podpůrný prostředek k lepšímu pochopení a představě o geometrických náhledech nebo geometrické interpretaci vybraných matematických objektů, a to s využitím "pohyblivých" obrázků. Má práce obsahuje také ilustrativní příklady, ve kterých je většina popsaných příkazů a funkcí využita. Pro vytvoření animace jsem zvolila program Matlab, který je možné použít jak pro práci s daty, tak pro jejich zobrazení. Využila jsem verzi Matlab R2011b, která je dostupná všem studentům Přírodovědecké fakulty UP Olomouc na pracovních počítačích. Jednou z velkých výhod Matlabu je, že umožňuje pěknou vizualizaci dat jak pomocí 2D a 3D grafiky, tak pomocí animací, na jejichž vytváření se zaměřím. U uživatelů, kteří budou tuto práci používat jako podpůrný prostředek k vytvoření animace se předpokládá základní znalost práce s programem. Práce je rozdělena na tři kapitoly. V první kapitole popisuji způsoby zadávání pravidelných vektorů, vytvoření M-souboru (typu skript) a cyklus `for`. V následující kapitole se již zaměřím na samotnou vizualizaci dat pomocí funkce `plot`. Obsah prvních dvou kapitol je cílený na popis většiny postupů a příkazů, které budeme následně pro vytvoření animace potřebovat. Tyto kapitoly je pak možné vynechat, pokud postupy a příkazy v nich popsané jsou uživateli známé. Třetí kapitola je již zaměřena na dva způsoby, pomocí kterých je možné animace v tomto programu vytvářet. V poslední sekci třetí kapitoly jsou popsány postupy a různé možnosti animování na samotných příkladech. Ilustrativní příklady jsou uloženy ve formě skriptů a jsou přiloženy k mé práci v příloze a na CD.

Jednou z výhod Matlabu je skutečnost, že jeho součástí je také elektronická nápověda (**Help**), ve které je možné dohledat všechny informace potřebné k používání, včetně některých příkladů a videí (demo) postupů. Program Matlab nabízí velké množství příkazů, funkcí a možností úprav, které není v mé práci možné obsáhnout, proto všem zájemcům doporučuji této nápovědy při vytváření vlastních animací také plně využívat.Tato nápověda je v angličtině.

Jelikož teoretická část mé práce má kompilační charakter, neuvádím citace na konci každého odstavce, ale pouze globálně na začátku každé kapitoly.

# Kapitola 1

# Než začneme animovat

První kapitola mé práce je věnovaná a cílená na to, aby uživatel rozšířil své základní znalosti programu Matlab o příkazy a funkce, které bude následně využívat při vytváření animace. Tato kapitola je rozdělena na tři sekce. V první sekci jsou popsány dvě varianty, které lze využít k vytvoření pravidelných vektorů. V druhé sekci se zaměřím na vytváření M-souborů typu skript, které budu využívat pro ukládání ilustrativních příkladů. V poslední sekci popíši cyklus `for`, jenž umožňuje opakování příkazů pro různé proměnné. Většina příkazů a funkcí popsaných v této kapitole bude následně využita pro vytvoření ilustrativních příkladů animace, na které je zaměřena poslední sekce závěrečné kapitoly. Pro uživatele s hlubším zájmem o více možností, které program Matlab nabízí, doporučuji využití nápovědy programu Matlab - **Help**. Při psaní této kapitoly jsem čerpala ze skript Ženčák, P.: Matlab pro začátečníky a mírně pokročilé [1].

## 1.1. Vytváření pravidelných vektorů

Pro jednoduché a rychlé zadávání pravidelných vektorů lze v Matlabu použít operátor : nebo využít funkce `linspace`. Obě tyto možnosti nám pomohou vytvořit libovolně velký řádkový vektor podle zadaných pravidel. Oboje nám bude sloužit k dalšímu použití.

## Operátor :

Operátor : využíváme pro vytváření vektoru, jehož prvky tvoří aritmetická posloupnost. Například pro vytvoření vektoru, který bude obsahovat čísla od 1 do 6, použijeme operátor : tak, že zadáme první prvek (tj. 1) a horní mez (tj. 6). Horní mez pak žádný prvek vektoru nesmí překročit. `>> h = 1:6`

```
h =

    1 2 3 4 5 6
```

Takto zadaný příkaz vygeneruje řádkový vektor s jednotkovým krokem od 1 do 6. Pokud však nechceme, aby čísla rostla postupně o 1, ale o jinou hodnotu, například 2, tak navíc zadáváme velikost kroku doprostřed mezi dva operátory :

```
>> k = 1:2:6
k =

    1 3 5
```

Důležité je také si uvědomit, že poslední číslo vyjadřuje opravdu jen horní mez a nemusí nutně znamenat poslední prvek vektoru. Pokud mezi operátory zadáme číslo se záporným znaménkem, pak vektor bude tvořen klesající posloupností a to vždy o daný počet dolů.

Pro vytvoření vektoru, který začíná číslem 6 a končí číslem 2 použijeme příkaz

```
>> z = 6:-2:1
z =

    6 4 2
```

Pro podobné výsledky jako s operátorem : můžeme použít funkci `linspace`.

## Funkce `linspace`

Rozdíl mezi operátorem : a funkcí `linspace` je pouze ten, že zde zadáváme první a poslední prvek vektoru a Matlab pak podle toho dopočítá krok sám. Můžeme také doplnit i počet prvků výsledného vektoru. Pokud počet prvků nezadáme sami, vytvoří Matlab vektor o 100 bodech.

Obecně používáme příkaz:

```
>> v = linspace(první prvek,poslední prvek,počet prvků vektoru)
v =
   prvky vektoru
```

Pro vytvoření vektoru, který obsahuje celá čísla od 1 do 5 použijeme příkaz

```
>> v1 = linspace (1,5,5)
v1 =
   1 2 3 4 5
```

Pokud použijeme první číslo větší než druhé, pak Matlab sám rozezná, že se jedná o klesající posloupnost

```
>> v2 = linspace(5,1,5)
v2 =
   5 4 3 2 1
```

Pokud krok nebude jednotkový, dopočítá ho Matlab sám

```
>> v3=linspace(1,4,5)
v3 =
    1 1,75 2,5 3,25 4
```

V tomto případě krok o velikosti 0,75.

## 1.2. M-soubory

Pro provádění většiny jednoduchých příkazů v Matlabu se většinou využívá příkazové okno (**Command Window**). Příkazy provedené v tomto okně avšak není možné zpětně upravovat. Pokud chceme vytvořit animaci, je vhodné zadat více příkazů tak, abychom je mohli případně zpětně upravit, uložit a naráz provést. K tomu, abychom mohli ukládat a upravovat posloupnost příkazů, vytvoříme v Matlabu M-soubor typu skript. M-soubor je textový soubor s příponou .m, který obsahuje kód v programovacím jazyku Matlabu. M-soubor má dva zá-

kladní typy. Prvním typem je skript, jehož vytvoření následně popíši. Druhým typem je funkce, tou se ale v mé práci zabývat nebudu.

## Skripty

Skript je posloupnost příkazů, které uložíme do zvláštního souboru. Ve skriptu můžeme měnit, mazat nebo vytvářet nové proměnné. Umožňují nám také volat jiné skripty nebo funkce a vytvářet nová grafická okna. Při psaní skriptu se příkazy neprovádějí hned, ale až po jeho spuštění v příkazovém okně (viz níže).

## Vytvoření skriptu

M-soubor, který bude obsahovat skript, můžeme vytvořit buď pomocí menu `File-New-M-file` nebo pomocí ikony v nástrojové liště (vlevo nahoře, hned pod `File`). Obě možnosti otevřou nové okno v M-editoru, které slouží pro psaní M-souboru. Pokud chceme otevřít již existující skript, využijeme okna aktuální složky (**Current Folder**), kde soubor najdeme a dvojklikem otevřeme. Druhá možnost je pomocí `File-Open`.

Následuje samotný zápis skriptu. Pro přehlednost skriptu je vhodné dodržet následující strukturu. Do prvního řádku skriptu napíšeme `%` (dále "procento") a za ním výstižnou charakteristiku skriptu. Pomocí procenta pak Matlab oddělí naše poznámky od příkazů, které má provést. Další řádky můžeme použít pro popsání nápovědy nebo postupů skriptu. Nejdůležitější část skriptu je jeho tělo. Tělo skriptu obsahuje posloupnost příkazů, ke kterým můžeme přidávat pomocí procent komentáře. Příkazy nikdy procentem nezačínají.

Po napsání kódu skriptu musíme celý M-soubor uložit. Ukládáme jej do složky, kterou si pro tento účel vytvoříme pod nějakým jménem. Jméno může obsahovat pouze písmena anglické abecedy, podtržítko a číslice. V žádném případě nesmí však číslicí začínat.

Následné spuštění skriptu dává Matlabu pokyn, aby vykonal příkazy, které obsahuje. Vždy před jeho spuštěním je třeba ho uložit. Jeho spuštění můžeme provést například zadáním jeho názvu bez koncovky do příkazového okna nebo v samotném M-editoru stiskem **F5** a nebo použitím ikony (zelená šipka 'play'). Po spuštění skriptu se příkazy provedou v takovém pořadí, v jakém jsou za sebou zadány.

**Poznámka:** Všechny ilustrativní příklady, které tvoří přílohy k mé práci jsou tvořeny pomocí skriptů.

## 1.3. Cyklus `For`

**Funkce:** Opakování příkazu nebo skupiny příkazů.

Cyklus `for` se používá, pokud dopředu víme, kolikrát chceme opakovat zadaný příkaz nebo skupinu příkazů. Někdy je nazýván též jako iterační cyklus nebo cyklus s výčtem hodnot. Cyklus je pro nás zásadní pokud chceme, aby námi zadané příkazy byly provedeny opakovaně pro všechny hodnoty proměnné. Řešení takovéto situace se tak pro nás stává rychlé a přehledné. V případě tvorby animace budeme tohoto cyklu využívat.

### Obecné schéma použití cyklu

Tento cyklus vždy začíná slovem `for` a je ukončen slovem `end`. Většinou se proměnná rovná vektoru hodnot. Obecná syntaxe je pak v následujícím tvaru.

**Syntaxe:**

```
for proměnná = začátek:krok:mez
příkazy = tělo cyklu
end
```

## Průběh provádění cyklu

Proměnná bude postupně nabývat $n$ hodnot, které jsou určeny pomocí zadaného vektoru, případně matice. Kdy $n$ je počet prvků vektoru nebo počet sloupců matice.
Pokud je $n$ nenulové, $n$-krát je provedena posloupnost příkazů v těle cyklu. Po tom, co proběhnou všechny opakování příkazů v těle cyklu, pokračuje Matlab s příkazy následujícími za výrazem `end`, pokud za tímto výrazem ještě nějaké jsou.

**Poznámka:** Pro zájemce doporučuji nastudovat další možnosti (cyklus `while`, příkazy `if` a `break`) v nápovědě programu **Help**.

# Kapitola 2

# Vytváření 2D grafů

V této kapitole si ukážeme jednu z nejsilnějších stránek Matlabu, kterou je vizualizace dat. Proto, abychom mohli data vizualizovat, budeme využívat nejen znalostí z předcházejících kapitol, ale představíme si také nové příkazy a funkce. Ty je potřeba znát proto, abychom efektivně graficky zobrazili naše data. Popíšeme si snadnou cestu jak pomocí programu Matlab vytvořit 2D graf. Naučíme se tento graf pomocí různých příkazů a funkcí upravovat a popisovat.
K tomu, abychom mohli 2D graf vytvořit, jsou nezbytná data ve formě hodnot, které budeme graficky zobrazovat buď spojitě nebo pomocí bodů. Tyto hodnoty mohou být buď námi vypočtené nebo naměřené. Nejčastěji je zadáváme vektorem, a to pomocí operátoru `:` nebo funkce `linspace`, jak bylo popsáno v předchozí kapitole. V Matlabu je několik grafických funkcí, které slouží pro vykreslování dat. Nejpoužívanější z nich je funkce `plot`, kterou si následně představíme. Při psaní této kapitoly jsem čerpala z nápovědy Matlabu **Help** [2].

## 2.1. Funkce `plot`

**Funkce:** Vytvoření 2D grafu.
**Syntaxe:**

```
plot(y)
plot(x,y)
plot(x,y,'LineSpec')
```

Za předpokladu že $x$ a $y$ budou vektory, můžeme jednotlivé příkazy popsat takto:

`plot(y)` Vykresluje do grafu lineárně hodnoty vektoru $y$ v závislosti na jejich pořadí.

`plot(x,y)` Vykresluje graf závislosti vektoru $y$ na vektoru $x$.

`plot(x,y,'LineSpec')` Přidáním argumentu `Linespec` je možné měnit vlastnosti zobrazeného grafu. Mezi tyto vlastnosti patří styl čáry, styl značky a barva pro značku nebo čáru.

Případem, kdy by $x$ nebo $y$ nebyly vektory, se v mé práci nebudu zabývat. Nicméně tyto, i další možnosti, lze prostudovat v nápovědě programu **Help**.

Tabulka 2.1 ukazuje přehled možností úpravy grafu.

| **Symbol** | **Styl čáry** | **Symbol** | **Styl značky** | **Symbol** | **Barva** |
|---|---|---|---|---|---|
| - | plná | . | tečka | b | modrá |
| – | přerušovaná | o | kolečko | g | zelená |
| : | tečkovaná | x | křížek | r | červená |
| -. | čerchovaná | + | křížek | c | tyrkysová |
| | | * | hvězdička | m | fialová |
| | | s | čtverec | y | žlutá |
| | | d | diamant | k | černá |
| | | v | trojúhelník nahoru | | |
| | | ^ | trojúhelník dolů | | |
| | | < | trojúhelník doleva | | |
| | | > | trojúhelník doprava | | |
| | | p | pentagram | | |
| | | h | hexagram | | |

Tabulka 2.1: Vlastnosti pro specifikaci zobrazení (`LineSpec`)

Jednotlivé vlastnosti pro specifikaci zobrazení je možné psát v libovolném pořadí dohromady v 'uvozovkách'. Vlastnosti, které nechceme využít, nepíšeme (vynecháme). Vynecháme-li vlastnost popisující barvu grafu nebo použijeme-li pouze příkaz `plot(x,y)`, graf se vykreslí implicitní barvou, což je modrá. Například

použijeme-li příkaz: `plot(x,y,'-.m')`, graf bude vykreslen čerchovaně fialovou barvou.

Argumenty funkce `plot` lze kombinovat také s dalšími dodatečnými vstupními argumenty v následujícím tvaru:
`plot(...'Vlastnost',Hodnota,...)`
Vlastnosti objektu typu `line` [1], který vytváří tato funkce, je možné přímo nastavovat zadáním názvu vlastnosti a její hodnoty. Následně si uvedeme jen některé z těchto vlastností, které je možno použít a které nelze přímo měnit již pomocí `Linespec`.

| **Vlastnost** | **Popis** |
|---|---|
| `LineWidth` | určuje šířku čáry spojující body, šířka se udává v bodech |
| `MarkerEdgeColor` | určuje barvu okraje značky kreslených bodů |
| `MarkerFaceColor` | určuje, jakou barvou bude vyplněna značka |
| `MarkerSize` | určuje velikost značky pro vykreslované body |

Grafický výstup, který vznikne po zadání příkazu `plot` se implicitně zobrazí v novém grafickém okně s názvem `Figure1`. Pokud bychom chtěli zobrazit další graf v tomto okně pomocí funkce `plot`, tak nový graf přemaže stávající. Chceme-li, aby nový graf nepřemazal ten stávající, použijeme příkaz `hold on`. Pro vypnutí přikreslování grafů použijeme `hold off`.

Pokud před příkazu `plot` zadáme také příkaz `Figure(h)`, pak se grafický výstup zobrazí v takto zadaném okně. Proměnná `h` je identifikátor objektu `figure`. Tímto identifikátorem jsou přirozená čísla (od `1`,`2`...), která jsou zobrazena na horním okraji grafického okna, hned za nápisem `Figure`.

[1] `Line` je grafická funkce nižší úrovně, která vytváří objekty `line`. Objekty `line` jsou dětmi objektů `axes`.

**Příklad 1**

**Cíl:** Vykreslení 2D grafu funkce $x^2$ pomocí funkce `plot`.

**Řešení:**

Z matematického pohledu pomocí 2D grafů obvykle zobrazujeme závislost jedné veličiny na druhé. K popsání této závislosti použijeme vztah

$$y = f(x)$$

$x$ je nezávisle proměnná a $y$ je závisle proměnná. Nyní si ukážeme možnosti Matlabu na konkrétní funkci.

$$y = x^2$$

Tuto funkci budeme vyšetřovat na intervalu od mínus dvou do dvou. Nejdříve tedy vytvoříme vektor hodnot nezávisle proměnné $x$. Bude nám stačit, pokud budou prvky vektoru vzdálené od sebe o krok 0,1.

```
>> x = -2:0.1:2;
```

Nyní můžeme spočítat funkční hodnoty zadané funkce.

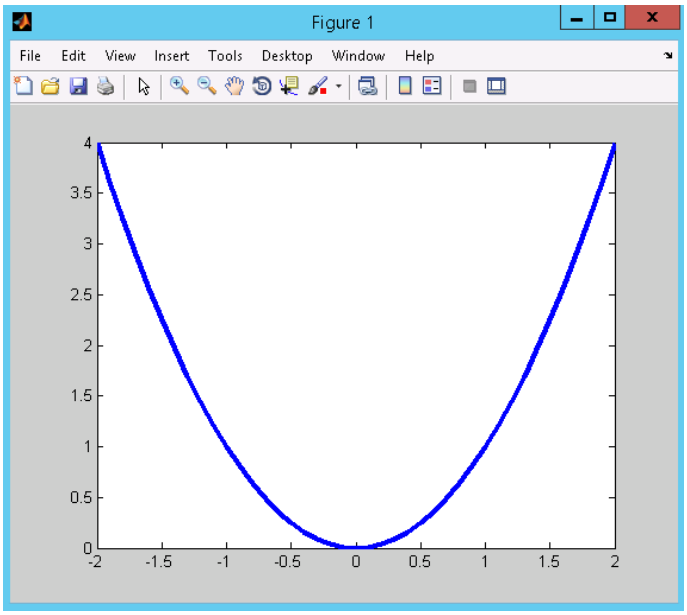
```
>> y = x.^2;
```

Na zobrazení funkční závislosti mezi oběma veličinami využijeme příkaz `plot` v následujícím tvaru.

```
>> plot(x,y,'LineWidth',3)
```

Tento příkaz nám, pokud nenastavíme jinak, v novém okně pojmenovaném Figure1 zobrazí graf této funkce. Zadáním parametru `'LineWidth'` nastavíme tloušťku čáry (Obrázek2.1).

Skript k příkladu je uložen na přiloženém CD pod názvem *Priklad_1* a v příloze pod názvem Příklad 1 (str. 42).




Obrázek 2.1: Graf funkce vykreslený pomocí příkazu plot

## 2.2. Funkce `fplot`

**Funkce:** Vykreslení grafu funkce.

**Syntaxe:**

```
fplot(fun,limits)
fplot(...,'LineSpec')
```

Obdobně jako funkce `plot` můžeme použít funkci `fplot`, abychom vytvořili graf funkce. Funkci `fplot` popíši pouze stručně, pro více informací o dalších argumentech funkce odkazuji opět na nápovědu programu **Help**.

`fplot` Použijeme pro vytvoření grafu funkce označené `fun`. Tato funkce musí být ve tvaru $y =$`fun`$(x)$, kde `fun` je jméno jedné z matematických funkcí Matlabu (`sin`, `cos`, `tan`, `cot`,...) nebo jméno funkce, kterou si sami v Matlabu vytvoříme pomocí anonymní funkce (viz níže) a `x` je vektor hodnot. `Fun` vrací vektor (se stejnou délkou jako vektor `x`) obsahující vypočtené funkční hodnoty v bodech zadaných vektorem $x$.

`fplot(fun,limits)` Vykreslí funkci definovanou proměnnou `fun` v mezích $x$-ové osy, které jsou zadány pomocí `limits=[xmin xmax]`.

`fplot(...,'LineSpec')` Obdobně jako je popsáno u funkce `plot`.

Pokud chceme jako parametr `fun` použít vlastní funkci, která není přímo v Matlabu definována, musíme si tuto funkci vytvořit sami. Funkci v Matlabu můžeme zadat pomocí M-souborů, tímto případem se však zabývat nebudu. Popíši případ zadání jednoduché funkce nevyžadující ke své definici M-soubor. Jednou z možností, jak lze jednoduché funkce v Matlabu vytvořit, je pomocí anonymní funkce. Anonymní funkce jsou jednoduché Matlabovské funkce, které se zadávají pomocí znaku @ takto:

`@(seznam argumentů)výraz`

Pro lepší pochopení uvádím následující příklad (Příklad 2).

**Příklad 2**

**Cíl:** Vykreslení 2D grafu funkce $x^2$ pomocí funkce `fplot`.

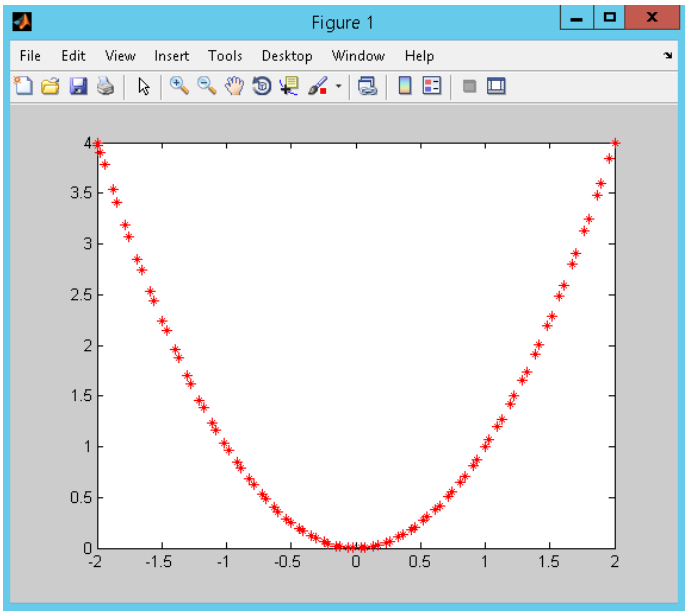
**Řešení:**

Tentokrát zadáme příkazy v následujícím tvaru:

```
>> fn = @(x)x.^2;
>> a = [-2 2 0 4];
>>fplot(fn,a,'*r');
```

V prvním kroku je třeba zapsat funkci, v druhém kroku meze a následně dosadit do funkce **fplot**. Pro ukázku byly také použity jiné vlastnosti grafu. Výsledný graf pak vidíme na Obrázku 2.2.

Skript k příkladu je uložen v příloze na straně 42 a na přiloženém CD pod názvem Priklad_2.




Obrázek 2.2: Graf funkce vykreslený pomocí příkazu plot

### 2.2.1. Úpravy grafu

Pomocí příkazů pro práci s grafy máme spoustu možností, jak graf upravovat. Můžeme například do grafu přidat mřížku, popsat jeho osy, přidat název nebo měnit další vlastnosti grafu. Jednotlivé příkazy, které nám tyto úpravy umožní, si nyní popíšeme.

#### Zobrazení mřížky do grafu

K zobrazení mřížky do aktuálního grafu slouží příkaz `grid`.

```
>> grid
```

Opětovné použití příkazu následně tuto mřížku zase vymaže. Obdobně můžeme použít příkaz `grid` s parametry `on` pro zobrazení a `off` pro odstranění mřížky.

```
>> grid on
>> grid off
```

## Popis grafu

Pro popis grafu je určená další důležitá skupina příkazů. Pomocí těchto příkazů pak přidáme do aktuálního grafu nadpis (`title`), popisy jeho os (`xlabel` a `ylabel`) nebo legendu (`legend`).

```
>> xlabel ('text')
>> ylabel ('text')
>> title ('text')
>> legend ('text')
```

Pokud bychom chtěli popsat přímo určitá místa grafu, používáme funkci nižší úrovně `text`, která vytvoří textový grafický objekt.
Příkaz

```
>> text(x,y,'string','Vlastnost',Hodnota....)
```

přidá do aktuálního objektu `axes` [2], na pozici určenou bodem `[x,y]` text daný řetězcovým parametrem `string`. Do grafu tedy můžeme na libovolně nadefinované místo přidat popisek. Další parametr `'Vlastnost'` určuje vlastnost textu, která je psaná v uvozovkách a za ní případně její `hodnota` bez uvozovek. Některé z vlastností textu, které lze upravovat, jsou popsány níže.

| **Vlastnost** | **Popis** |
|---|---|
| `FontSize` | nastaví velikost textu |
| `HorizontalAlignment` | zarovná text horizontálně |
| `VerticalAlignment` | zarovná text vertikálně |
| `Color` | barvu textu |

Využití příkazu `grid` a popsání grafu ukazuje Příklad 3.

[2] `axes` je grafická funkce nižší úrovně, která vytváří objekty `axes`. Objekty `axes` jsou 'dětmi' objektů `Figure`.

**Příklad 3**

**Cíl:** Ukázat možnosti úpravy grafu.

**Řešení:**

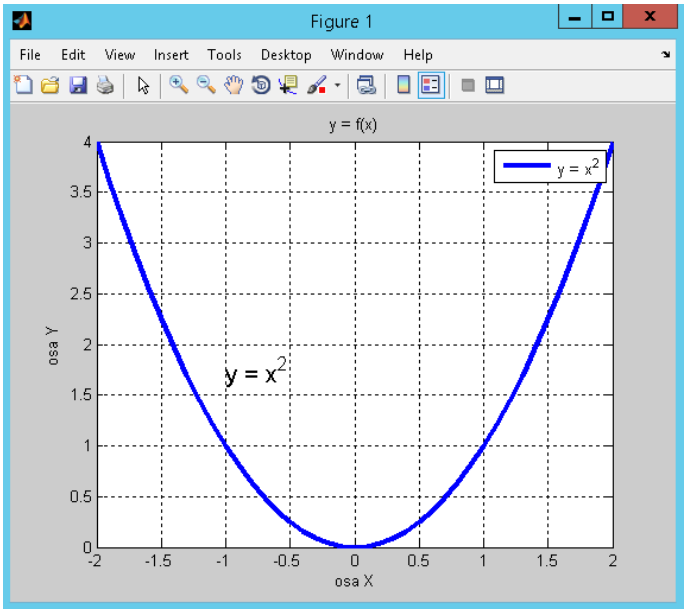
Pro graf z příkladu 1 si ukážeme použití příkazů pro popis grafu a přidání mřížky.

**Skript:**

```
x = -2:0.1:2;
y = x.^2;
plot(x,y,'LineWidth',3)
grid on
xlabel ('osa X')
ylabel ('osa Y')
title ('y = f(x)')
legend ('y = x^2')
text(-1,1.75,'y = x^2','Fontsize',15)
```

Výsledný graf pak vidíme na Obrázku 2.3.

Skript příkladu je uložen v příloze na straně 42 a na přiloženém CD pod názvem Priklad_3.




Obrázek 2.3: Ukázka použití příkazu grid a popsání grafu

### Volitelnost zobrazení os v grafu

Matlab umožňuje také nastavení rozsahu zobrazovaných os. Můžeme předem navolit rozsah osy $x$ a $y$ zvlášť nebo jedním příkazem měnit rozsah obou os současně.

Abychom změnili zobrazovaný rozsah osy $x$, použijeme funkci `xlim([v])`. $V$ je vektor obsahující minimální a maximální hodnotu, kterou chceme zobrazit na ose $x$. Obdobně pak pro změnu rozsahu osy $y$ použijeme funkci `ylim([v])`. Pokud bychom chtěli změnit rozsah obou os současně, použijeme příkaz `axis([v])`, kde $v$ je v tomto případě vektor obsahující postupně minimální hodnotu na ose $x$, maximální hodnotu na ose $x$, minimální hodnotu na ose $y$ a maximální hodnotu na ose $y$.

# Kapitola 3

# Vytvoření animace

V této kapitole si ukážeme dva způsoby, jak můžeme pomocí Matlabu animovat. První způsob spočívá v tom, že si nejprve v předstihu, ne v reálném čase, uložíme snímky. Tyto snímky následně přehrajeme ve formě animace. Tento způsob je vhodnější pro situace, kdy jsou jednotlivé snímky komplexní a navzájem se od sebe dost liší. Vytvoření animace druhým způsobem je postaveno na rychlém překreslování jednoduchých obrázků. Nejprve vytvoříme první snímek a pak už jen měníme příslušná data objektu. Oba tyto způsoby si následně popíšeme. Při psaní této kapitoly jsem použila především nápovědu programu **Help** [2] a skripta Ženčák, P.:Maltab pro začátečníky i mírně pokročilé [1].

## 3.1. Vytvoření animace pomocí funkce `getframe` a `movie`

První způsob tvorby animace nám umožňuje uložit libovolnou posloupnost grafů a zpětně ji přehrát v krátké animaci.

**Tento proces můžeme popsat pomocí dvou kroků:**
Nejprve je třeba generovat jednotlivé snímky, které následně budeme přehrávat v animaci. K tomu použijeme funkci `getframe` (popsáno níže). Při volání této funkce je třeba se ujistit, že počítač není v režimu spořiče obrazovky. Pokud

používáte více virtuálních ploch, je třeba, aby plocha, na které máte spuštěný Matlab, byla na monitoru vidět.

V druhém kroku pak pomocí funkce `movie` spustíme animaci (přehrajeme jednotlivé snímky) pro námi zadaný počet opakování a snímků za sekundu. Počet opakování a snímků za sekundu jsou parametry této funkce, které mohou i nemusí být zadány.

### 3.1.1. Funkce `getframe`

**Funkce:** Získání dat pro animaci.

Tato funkce nám slouží k tomu, abychom si připravili snímky, které následně budeme chtít přehrát. Pomocí `getframe` vytvoříme animační matici `M`, jejíž sloupce jsou tvořeny animačními snímky (bitmapa), kterou následně funkce `movie` použije. Funkci `getframe` obvykle používáme uvnitř cyklu (`for`), který nám pomáhá vytvořit posloupnost snímků pro animaci. Pokud cyklus `for` použít nechceme, ukládáme pomocí `getframe` každý snímek zvlášť (viz Příklad 6 v příloze). `Getframe` vrací strukturu složenou ze složek `cdata` (s polem obrazových dat) a `colormap` (s barevnou mapou).

**Syntaxe:**

```
M=getframe
M=getframe(h)
```

`M=getframe` Vrací sloupcový vektor s jedním animačním rámečkem (bitmapa). Animační rámeček je snímek aktuálního objektu `axes`.

`M=getframe(h)` Vezme snímek z objektu `figure` nebo `axes`, který má daný identifikátor `h`.

### 3.1.2. Funkce `movie`

**Funkce:** Přehrávání zadané animační matice.

Funkce `movie` je závěrečným krokem pro vytvoření animace. Tato funkce využívá připravených dat (snímků pomocí `getframe`). `Movie` většinou následuje po uzavření cyklu `for` příkazem `end` nebo po tom, co jednotlivé snímky nadefinujeme bez použití cyklu. Obecně tedy tato funkce slouží k přehrání zaznamenané animační matice, jejíž sloupce jsou tvořeny jednotlivými snímky (pomocí `getframe`). Pokud tyto snímky přehrajeme, dojde k vytvoření animace, která se zobrazí v novém grafickém okně Matlabu. Pro tuto animaci si můžeme volit například počet opakování nebo snímků za sekundu (rychlost).

**Syntaxe:**

```
movie(M)
movie(M,n)
movie(M,n,fps)
M=moviein(n)
```

`movie(M)` Tento příkaz přehraje jedenkrát animaci z matice `M`. `M` pak musí být matice, jejíž sloupce jsou animační snímky.

`movie(M,n)` Pokud přidáme parametr `n`, pak funkce `movie` nepřehraje animaci jedenkrát, ale `n` krát po sobě. Pokud pak `n` bude záporné, tak každé přehrání je provedeno jednou vpřed a jednou vzad. V případě že `n` bude vektor, pak prvky od druhého výše určují pořadí, ve kterém jsou snímky přehrány. První prvek vektoru udává počet, kolikrát se animace přehraje. Například pro matici M o třech sloupcích, vektor `n = [20 3 2 1]`, přehraje animaci dvacetkrát ve zpětném pořadí.

`movie(M,n,fps)` Další parametr `fps` udává rychlost snímků za sekundu. Pokud není zadán, implicitně bude jeho hodnota `fps = 12`.

`M=moviein(n)` Vytváří dopředu dostatečné místo v paměti pro uchování $n$ snímků, které následně vytvoříme pomocí `getframe`.

## 3.2. Vytvoření animace pomocí funkce `set` a `drawnow`

Vytvoření animace druhým způsobem je postaveno na rychlém překreslování jednoduchých obrázků, což je založeno na tom, že vytvoříme první snímek a pak už jen měníme příslušná data. Neukládáme tedy posloupnost snímků, jako v předchozím případě, ale pouze vytvoříme sekvenci obrázků. Tato sekvence se pak jeví v grafickém okně jako animace. Nejdříve tedy pomocí funkce `plot` vytvoříme objekt u kterého následně měníme data. Tuto změnu dat nám umožňuje funkce `set`. Po nastavení změny dat použijeme funkci `drawnow`, která umožňuje bezprostředně aktualizovat obrazovku a vytvořit tak sekvenci obrázků.

### 3.2.1. Funkce `set`

**Funkce:** Umožňuje nastavovat vlastnosti objektu.

**Syntaxe:**

```
set(h,'vlastnost',změna,...)
```

`set(h,'vlastnost',změna,...)` Použijeme k nastavení změn vlastností objektu s identifikátorem `h`.

Tuto funkci budeme pro animaci využívat hlavně pro změnu `Xdata` a `Ydata`.

### 3.2.2. Funkce `drawnow`

**Funkce:** Umožňuje bezprostředně aktualizovat obrazovku (provést příkazy).

**Syntaxe:**

```
drawnow
```

`drawnow` Přinutí Matlab, aby bezprostředně aktualizoval obrazovku. Pokud máme posloupnost příkazů v M-souboru, tak po jeho spuštění jsou příkazy prováděny postupně, jak jsou za sebou ve skriptu napsány. Tato funkce nám pomůže aktualizovat obrazovku dříve, než budou provedeny všechny příkazy souboru. Nastavíme-li pak pomocí funkce `set` změny, jsou tyto změny bezprostředně vykresleny.

### 3.2.3. Ukládání animace

Po tom, co vytvoříme animaci, je možné ji spouštět přímo v programu Matlab. Ne vždy však máme program Matlab k dispozici nebo animace chceme vložit například do prezentace. Pro tyto případy si ukážeme dvě funkce, které umožní ukládání animace do formátu avi.

První funkce je `movie2avi`, která má obecný tvar:
`movie2avi(M,'nazev.avi','vlastnost',hodnotavlastnosti)`
`M` je animační matice a `nazev.avi` je název animace s příponou. Zadáním dalších vstupních parametrů funkce můžeme ovlivnit například kvalitu ('quality'), jejíž hodnoty se uvádí v procentech. Dalším parametrem je například rychlost ('fps'), jejíž hodnoty udáváme jako počty snímků za sekundu. Implicitní hodnota pro kvalitu je 75 procent a pro rychlost 15 snímků za sekundu. Tuto funkci zadáváme na konec kódu skriptu (obvykle za funkcí movie).

Druhá funkce sloužící pro ukládání animací je `writevideo`. Obecný postup pro tento způsob je následující:
1.Připravíme nový soubor

```
vid = VideoWriter('nazev.avi');
open(vid);
```

2.Vytvoříme jednotlivé snímky a uložíme je do souboru

```
writeVideo(vid,frame);
```

Pokud při ukládání snímků použijeme `getframe(f)`, kde se proměnná `f` je dána jako `f = figure()`, uloží se snímek celého grafického okna. Animace je pak zobrazena i s popisem os a titulkem grafu, což je pro geometrickou interpretaci matematických dat mnohdy nutné. Pokud bychom ukládali snímky pomocí `getframe` bez identifikátoru, výsledná animace bude zobrazena bez popisků os a bez titulku (viz Příklad 4.2). Samotné `getframe` snímá pouze obsah bílé plochy grafického okna.
Animace ve formátu avi pak najdeme uložené ve stejné složce, ve které jsou uloženy M-soubory s kódem po jejich vytvoření.

## 3.3. Ilustrativní příklady

V poslední sekci se zaměřím na popis ilustrativních příkladů, které jsou přiloženy k mé práci. Příklady byly vytvořeny tak, aby ukázaly funkčnost většiny příkazů, které má práce obsahuje. Uživatel programu si tak může udělat lepší představu o možnostech geometrické interpretace, vizualizace a především animování v tomto programu. Příklady ukazují konkrétní postupy, které uživatelům programu usnadní cestu k vytvoření jejich vlastní animace. Jednotlivé příklady jsou přiloženy v podobě M-souborů typu skript na přiloženém CD nebo jsou vypsány a vloženy do práce v závěrečné příloze. Ve skriptech jsou pomocí procent popsány postupy a funkce jednotlivých příkazů. Na přiloženém CD jsou také uloženy jednotlivé animace ve formátu avi.

Pro všechny příklady se předpokládá znalost vysokoškolské matematiky v

rozsahu předmětů Matematika 1,2. Proto tedy z matematického hlediska popíši příklady jen velice stručně.

## Příklady s použitím funkce `getframe` a `movie`

Vytvoření animace prvním způsobem umožňuje ukládání jednotlivých snímků. Díky tomu lze následně animace ukládat do formátu avi, který je pro nás vhodný pro spuštění i mimo prostředí Matlabu. Většina příkladů je zaměřena na postupy vytvoření animace právě tímto způsobem.

**Příklad 4**

**Cíl:** Postupné vykreslování tečen v různých bodech grafu funkce $f(x) = x^2$.

**Řešení:**

Proto, abychom zobrazili tečny v jednotlivých bodech grafu funkce, je důležité si připomenout obecnou rovnici tečny v bodě grafu. Má-li funkce $f$ v bodě $x_0$ derivaci, pak lze zapsat rovnici tečny v bodě $[x_0; f(x_0)]$ takto:

$$y = f'(x_0) * (x - x_0) + f(x_0) \ ,$$

kde hodnota $f'(x_0)$ je směrnicí této tečny. Tuto rovnici jsem využila pro zobrazení tečen v konkrétních bodech tohoto grafu.

**Skript:**

`clear all;` % smaže všechny proměnné

`clc;` % smaže obsah Command Window

`close all;` % zavře všechny okna

`f = figure();` % pojmenuje nový grafický objekt

`x = linspace(-100,100);` % vytvoří pravidelný vektor o 100 bodech (čím více bodů, tím hladší křivka grafu funkce)

`plot(x,x.^2,'-.');` % vykreslí graf funkce f(x) = x^2 čerchovaně

`title('Přikreslování tečen do grafu funkce f(x) = x^2');` % přidá titulek

`hold on` % zabrání odmazání grafu

`axis([-30 30 -30 300]);` % nastavení rozsahu osy $x$ a $y$

`for i = 1:1:20` % cyklus umožní opakované provedení příkazů pro různé hodnoty proměnné

`plot(x,-2*i*x-i^2,'m');` % vykreslí tečny pro zápornou část osy $x$ fialově

`plot(x,2*i*x-i^2,'y');` % vykreslí tečny pro kladnou část osy $x$ žlutě

`M(i) = getframe(f);` % vytvoření animační matice

`end` % ukončení cyklu `for`

`movie(M)` % přehraje animační matici jako animaci

`movie2avi(M,'Priklad_4','fps',1)` % uloží animaci pod názvem - Priklad_4

Všechny uvedené příkazy jsou napsány do skriptu a následně uloženy. Na začátku jsem použila trojici příkazů pro obnovení prostředí Matlabu, které umožní smazat všechny uložené proměnné v paměti programu, smazat obsah příkazového okna a zavřít všechny otevřené okna. Tyto příkazy budu používat také ve všech dalších příkladech animací (dále tři tečky). Všechny funkce příkazů jsem popsala pomocí %.

Skript k tomuto příkladu lze prostudovat i v příloze (str. 42). Na CD je skript uložen jako Priklad_4.m a animace pod stejným názvem s příponou avi.

**Příklad 4.1**

**Cíl:** Postupné vykreslování a odmazávání tečen v různých bodech grafu funkce $f(x) = x^2$.

**Řešení:**

Tento příklad je svou podstatou stejný jako příklad předchozí, je určen pro ukázku toho, jak ovlivní přidání příkazu `hold off` výslednou animaci. Příkaz `hold off` jsem vepsala za příkazy `plot`, které umožňují vykreslení tečen. Použité příkazy z příkladu 4 zůstaly jinak nezměněny, proto je zde již znovu neuvádím.

**Skript:**

```
...
plot(x,-2*i*x-i^2,'m'); % vykreslí tečny pro zápornou část osy x fialově
plot(x,2*i*x-i^2,'y'); % vykreslí tečny pro kladnou část osy x žlutě
hold off
M(i) = getframe(f); % vytvoření animační matice
...
```

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 43). Na CD je skript uložen jako Priklad_4_1.m a animace pod stejným názvem s příponou avi.

**Příklad 4.2**

**Cíl:** Postupné vykreslování tečen v různých bodech grafu funkce $f(x) = \sin(x)$.

**Řešení:**

V tomto příkladě jsem zvolila jako základ graf funkce $f(x) = \sin(x)$, do kterého budu opět vykreslovat tečny. Příklad je svou podstatou stejný jako příklad 4 a 4.1 s tím rozdílem, že na něm ukážu změnu vlastností grafu, konkrétně tloušťku čáry. Zároveň také nastavím rozsah osy $x$, ve kterém budou jednotlivé tečny zobrazeny. Tečny pak budou mít délku omezenou tímto rozsahem a budou tvořit úsečky. Ukážu také co se stane, když při ukládání snímku pomocí `getframe` vynechám identifikátor objektu `figure`.

**Skript:**

```
...
f = figure();
% nadefinujeme argumenty funkce movie
reruns = 1; % počet, kolikrát se animace přehraje
fps = .2; % snímky za sekundu (rychlost přehrávání)
x = linspace(0,10,1000);
plot(x,sin(x),'b','Linewidth',3); % Linewidth nastaví tloušťku čáry
title('Funkce f(x) = sin(x) a tečny v 10 bodech této funkce')
```

```
hold on
axis([ 0 10 -2 2 ])
for i = 1:1:10
x2 = linspace((i)-2,(i)+2,1000); % zadání linspace v tomto tvaru umožní
```
vykreslit tečny jako úsečky
```
plot(x2,x2*cos(i)- i*cos(i)+sin(i),'r','Linewidth',2) %vykreslí tečny
M(i) = getframe; % uloží snímky bez popisků grafu (pouze bílou plochu, ve
```
které je graf vykreslen)
```
end
movie(M,reruns,fps) % přehraje jednou animaci s rychlostí 0,2 snímky za
```
sekundu
```
movie2avi(M,'Priklad_4_2.avi','fps',1) % uloží animaci
```

Tečny ke grafu této funkce jsem omezila pomocí funkce `linspace`, kterou jsem zadala tak, aby byl dán rozsah osy `x`, ve kterém se tečny zobrazí. V tomto příkladě jsem také nepoužila identifikátor u příkazu `getframe`. Následkem toho jsou pak jednotlivé snímky animace uloženy bez popisků os. Pro případ využití animací pro matematiku je vhodné identifikátor použít.

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 43). Na CD je skript uložen jako Priklad_4_2.m a animace pod stejným názvem s příponou avi.

**Příklad 5**

**Cíl:** Animace aproximace funkce $f(x) = \sin(x)$ Taylorovým polynomem.

**Řešení:**

Taylorův polynom lze v Matlabu vypočítat jednoduše pomocí funkce `taylor`.

**Skript:**

```
...
f = figure();
reruns = 2;
fps = .2;
syms x % x definuje jako symbol
a = [-2*pi 2*pi -5 5]; % meze
fplot('sin(x)',a,'k','Linewidth',2); % využíváme fplot pro vykreslení
grafu funkce f(x) = sin(x)
hold on % zabraní přepisování jednoho snímku druhým
title('Graf funkce f(x) = sin(x) a její aproximace Taylorovým polynomem')
for i = 1:10 % použití cyklu for pro výpočet Taylorova polynomu i-tého stupně
T_i = taylor(sin(x),i+1); % funkce taylor, která vypočítá Taylorův poly-
nom i-tého stupně
fplot(char(T_i),a) % char označuje znakovou proměnnou
M(i) = getframe(f); % uloží jednotlivé snímky
end
movie(M,reruns,fps) % přehraje snímky jako animaci
movie2avi(M,'Priklad_5.avi','fps',1) % uloží animaci
```

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 44). Na CD je skript uložen jako Priklad_5.m a animace pod stejným názvem s příponou avi.

**Příklad 5.1**

**Cíl:** Animace aproximace funkce $f(x) = \sin(x)$ Taylorovým polynomem.

**Řešení:**

Cíl tohoto příkladu je z matematického hlediska stejný jako u příkladu 5. Nyní však budu ukládat každý snímek zvlášť. Zobrazím pouze Taylorovy polynomy s lichými mocninami, které jsou pro tuto funkci totožné s polynomy sudého stupně.

**Skript:**

...

```
f = figure();
reruns = 1;
fps = .2;
nframes = 6; % počet rámců (snímků)
Frames = moviein(nframes); % nastavení matice Frames
syms x
a = [-2*pi 2*pi -5 5];
fplot('sin(x)',a,'-k')
text(-5.5,1.5,'sin(x)') % přidáme popis grafu (umístění podle zadaných souřadnic)
title('aproximace funkce f(x) = sin(x) Taylorovým polynomem')
Frames(:,1) = getframe(f); % první sloupec matice Frames
hold on % zabraňuje postupnému překreslování jednoho grafu druhým
T_1 = taylor(sin(x),2); % nadefinujeme jednotlivé snímky Frames
fplot(char(T_1),a,'m'); % vykreslí Taylorův polynom prvního stupně modrou barvou
text(-3,-2,'T_1') % přidá popisek
Frames(:,2) = getframe(f); % druhý sloupec matice Frames
T_3 = taylor(sin(x),4);
fplot(char(T_3),a,'r');
text(-3,4,'T_3')
Frames(:,3) = getframe(f);
T_5 = taylor(sin(x),6);
fplot(char(T_5),a,'g');
text(-5,-2,'T_5')
Frames(:,4) = getframe(f);
T_7 = taylor(sin(x),8);
fplot(char(T_7),a,'b');
```

```
text(-4,4,'T_7')
Frames(:,5) = getframe(f);
T_9 = taylor(sin(x),10);
fplot(char(T_9),a,'y');
text(-5.75,-0.5,'T_9')
Frames(:,6) = getframe(f);
movie(Frames,reruns,fps)% Frames přehrajeme jako animaci
legend('sin(x)','T_1','T_3','T_5','T_7','T_9') % přidá legendu
movie2avi(Frames,'Priklad_5_1.avi','fps',.2)
```

Tento příklad ukazuje jak lze ukládat každý snímek zvlášť. Tato možnost je sice trochu zdlouhavější, ale umožní změnit vlastnosti pro každý snímek. Pro přehlednost výsledné animace jsem zvolila pro každý graf jinou barvu a přidala jsem jeho popis. Na závěr je přidána legenda.

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 45). Na CD je skript uložen jako Priklad_5_1.m a animace pod stejným názvem s příponou avi.

## Příklad s použitím funkce `set` a `drawnow`

Tento způsob neumožňuje ukládání samotných snímků, ale pouze změnu nastavení dat a bezprostřední vykreslení této změny. Vytvoříme tak sekvenci obrázků, kterou lze přehrát v prostředí programu Matlab.

**Příklad 6**

**Cíl:** Pohyb bodu po grafu funkce $f(x) = \cos(x)$ a zobrazení změny funkční hodnoty.

**Řešení:**

Vykreslím graf funkce $f(x) = \sin(x)$ a následně bod, jehož polohu budu měnit pomocí funkce `set`. Přidám do grafu také text a pod ním zobrazím změnu funkční hodnoty. Abych zobrazila změnu funkční hodnoty použiji funkci `num2srt`, která

slouží k transformaci čísla na řetězec.

**Skript:**

...

```
f = figure();
x = linspace(0,10,1000);
y = cos(x);
plot(x,y)
hold on
p = plot(x(1),y(1),'o','MarkerFaceColor','m');
```

% argument `MarkerFaceColor` nastavíme barvu bodu

```
hold off
text(5,3,'Změna funkční hodnoty:','FontSize',14)
```

% přidá popisek na dané souřadnice

```
t = text(5,2.5,num2str(y(1)),'VerticalAlignment','top','FontSize',14);
```

% použijeme funkci `num2srt`, která slouží k transformaci čísla na řetězec - umožní zobrazení změny funkční hodnoty

```
axis('equal')
```

% automatické nastavení poměru stran os

```
title('pohyb bodu po grafu')
for i = 2:length(x)
```

%`length(x)` = délka vektoru `x`

```
set(p,'xdata',x(i),'ydata',y(i));
```

% `set` umožňuje nastavit vlastnosti objektu `p` - posouvá bod

```
set(t,'String',num2str(y(i)));
```

% `set` umožňuje nastavit vlastnosti objektu `t` - umožní zobrazení změny funkční hodnoty

```
drawnow
```

% umožní bezprostředně aktualizovat obrazovku

```
end
```

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 46). Na CD je skript uložen jako Priklad_6.m.

**Příklad 6.1**

**Cíl:** Pohyb bodu po grafu funkce $f(x) = \cos(x)$ a zobrazení změny funkční hodnoty.

**Řešení:**

Pokud chci příklad 6 uložit jako animaci, musím uložit snímek každé změny obrazovky pomocí `getframe`. Animace je tentokrát uložena pomocí funkce `writeVideo`.

**Skript:**

```
...
f = figure();
vid = VideoWriter('Priklad_6_1.avi'); % vytvoří soubor
vid.Quality = 100; % kvalita
vid.FrameRate = 50; % snímky za sekundu
open(vid); % otevře soubor
...
drawnow
getframe(f)
writeVideo(vid, getframe(f)); % uloží snímky do souboru
end
```

Protože použité příkazy jsou z velké části stejné jako v příkladu 6, jsou vypsány pouze části skriptu sloužící k ukládání animace.

Celý skript k tomuto příkladu lze prostudovat v příloze (str. 47). Na CD je skript uložen jako Priklad_6_1.m a animace pod stejným názvem s příponou avi. Dále je přiložena tatáž animace pod názvem Priklad_6_2, která je záměrně uložena v horší kvalitě (45%). Uložená animace k příklad 6.1 má velikost přibližně 41kB, zatímco animace k příkladu 6.2 má velikost 14kB. Na těchto dvou animacích lze vidět, že horší kvalita sice ovlivní velikost animace, ale zároveň tuto animaci značně znekvalitní. Doporučuji tedy ukládat animace ve vysoké kvalitě.

# Závěr

Ve své práci jsem se nejprve zaměřila na příkazy potřebné pro vizualizaci dat a vytvoření animace. Tyto příkazy jsem popsala tak, abych běžnému uživateli tohoto programu rozšířila jeho dosavadní znalosti o nové, které jsou potřebné pro vytváření animací. Následně jsem popsala možnosti, které využívají těchto příkazů pro vytvoření animace. Uživatel tak získá představu nejen o geometrických náhledech, ale především o geometrické interpretaci vybraných matematických objektů ve formě animace. Dospěla jsem k názoru, že pro tvorbu animace v tomto programu je nejlepší využít postupné ukládání snímků a jejich následné přehrání pomocí funkce `movie`, protože tyto uložené snímky lze pak i ukládat ve formátu avi. Takto uložené animace může uživatel přehrát i mimo prostředí Matlab a vložit je tak například do prezentace.

Práce je doplněna o příklady, které jsem samostatně zpracovala. Tyto příklady mohou sloužit jako podpůrný prostředek pro uživatele Matlabu, kteří se rozhodnou pro tvorbu vlastní animace.

V neposlední řadě bych si velmi přála, aby v matematice našli zalíbení hlavně mladí lidé, pro které se výuka matematiky doplněná o animace stane pochopitelnější a zajímavější.

# Literatura

[1] Zencak, P.: *Matlab pro začátečníky i mírně pokročilé.* Olomouc, 2013.

[2] The Mathworks, Inc.: MATLAB *Documentation*, Mathworks.com, 2011, [online]. Publikováno: 1994, poslední změna: 2015, [citováno 3.5.2015]. dostupné z: http://www.mathworks.com/help/matlab/index.html.

# Příloha

## Příklad 1

```
% Příklad 1: Použití funkce plot
x = -2:0.1:2;
y = x.^2;
plot(x,y,'LineWidth',3)
```

## Příklad 2

```
% Příklad 2: Použití funkce fplot
fn = @(x)x.^2;
a = [-2 2 0 4];
fplot(fn,a,'*r');
```

## Příklad 3

```
% Příklad 3: Popis grafu a zobrazení mřížky
x = -2:0.1:2;
y = x.^2;
plot(x,y,'LineWidth',3)
grid on
xlabel ('osa X')
ylabel ('osa Y')
title ('y = f(x)')
legend ('y = x^2')
text(-1,1.75,'y = x^2','Fontsize',15)
```

# Příklad 4

% Příklad 4: Postupné vykreslování tečen v různých bodech grafu funkce f(x) = x^2.
`clear all;` % smaže všechny proměnné
`clc;` % smaže obsah Command Window
`close all;` % zavře všechny okna
`f = figure();` % pojmenuje nový grafický objekt
`x = linspace(-100,100);` % vytvoří pravidelný vektor o 100 bodech (čím více bodů, tím hladší křivka grafu funkce)
`plot(x,x.^2,'-.');` % vykreslí graf funkce x^2 čerchovaně
`title('Přikreslování tečen do grafu funkce f(x) = x^2');` % přidá titulek
`hold on` % zabrání odmazání grafu
`axis([-30 30 -30 300]);` % nastavení rozsahu osy x a y
`for i = 1:1:20` % cyklus umožní opakované provedení příkazů pro různé hodnoty proměnné
`plot(x,-2*i*x-i^2,'m');` % vykreslí tečny pro zápornou část osy x fialově
`plot(x,2*i*x-i^2,'y');` % vykreslí tečny pro kladnou část osy x žlutě
`M(i) = getframe(f);` % vytvoření animační matice
`end` % ukončení cyklu for
`movie(M)` % přehraje animační matici jako animaci
`movie2avi(M,'Priklad_4','fps',1)` % uloží animaci pod názvem - Priklad_4

# Příklad 4.1

% Příklad 4.1: Přikreslování a odmazávání tečen ke grafu funkce f(x) = x^2.
`clear all;` % smaže všechny proměnné
`clc;` % smaže obsah Command Window
`close all;` % zavře všechny okna
`f = figure();` % pojmenuje nový grafický objekt
`x = linspace(-100,100);` % vektor pro x
`for i = 1:1:20` % cyklus umožní opakované provedení příkazů pro různé hodnoty proměnné
`plot(x,x.^2,'-.');` % vykreslí graf funkce f(x) = x^2 čerchovaně
`title('Přikreslování tečen do grafu funkce f(x) = x^2');` % přidá titulek
`hold on` % zabrání odmazání grafu
`axis([-30 30 -30 300]);` % nastavení rozsahu osy x a y
`plot(x,-2*i*x-i^2,'m');` % vykreslí tečny pro zápornou část osy x fialově
`plot(x,2*i*x-i^2,'y');` % vykreslí tečny pro kladnou část osy x žlutě

```
hold off
M(i) = getframe(f); % vytvoření animační matice
end % ukončí cyklu for
movie(M) % přehraje animační matici jako animaci
movie2avi(M,'Priklad_4','fps',1) % uloží animaci pod názvem - Priklad_4
```

# Příklad 4.2

```
% Příklad 4.2: Přikreslování tečen do grafu funkce f(x) = sin(x).
% vyčistíme prostředí Matlabu
clear all;
clc;
close all;
f = figure(); % nadefinujeme argumenty funkce movie
reruns = 1; % počet, kolikrát se animace přehraje
fps = .2; % snímky za sekundu (rychlost přehrávání)
x = linspace(0,10,1000);
plot(x,sin(x),'b','Linewidth',3); % Linewidth nastaví tloušťku čáry
title('Funkce f(x) = sin(x) a tečny v 10 bodech této funkce')
hold on
axis([ 0 10 -2 2 ])
for i = 1:1:10
x2 = linspace((i)-2,(i)+2,1000); % zadání linspace v tomto tvaru umožní
vykreslit tečny jako úsečky
plot(x2,x2*cos(i)- i*cos(i)+sin(i),'r','Linewidth',2) % vykreslí tečny
M(i) = getframe; % uloží snímky bez popisků grafu (pouze bílou plochu, ve
které je graf vykreslen)
end
movie(M,reruns,fps) % přehraje jednou animaci s rychlostí 0,2 snímky za
sekundu
movie2avi(M,'Priklad_4_2.avi','fps',1) % uloží animaci
```

# Příklad 5

```
% Příklad 5: Animace aproximace funkce f(x) = sin(x) Taylorovým polynomem.
% vyčistíme prostředí Matlabu
clear all;
clc;
close all;
```

```
f = figure();
reruns = 2;
fps = .2;
syms x % definuje x jako symbol
a = [-2*pi 2*pi -5 5]; % meze
fplot('sin(x)',a,'k','Linewidth',2); % využíváme fplot pro vykreslení grafu
funkce f(x) = sin(x)
hold on % zabraní přepisování jednoho snímku druhým
title('Graf funkce f(x) = sin(x) a její aproximace Taylorovým polynomem')
for i = 1:10 % použití cyklu for pro výpočet Taylorova polynomu i-tého stupně
T_i = taylor(sin(x),i+1); % funkce taylor, která vypočítá Taylorův polynom
i-tého stupně
fplot(char(T_i),a) % char označuje znakovou proměnnou
M(i) = getframe(f); % uloží jednotlivé snímky
end
movie(M,reruns,fps) % přehraje snímky jako animaci
movie2avi(M,'Priklad_5.avi','fps',1) % uloží animaci
```

# Příklad 5.1

```
% Příklad 5.1: Animace aproximace funkce f(x) = sin(x) Taylorovým polynomem
bez použití smyčky for.
% postupné ukládání jednotlivých snímků (Frame)
% následné spuštění animace
% obnovení protředí Matlab
close all;
clear all;
clc;
f = figure();
reruns = 1; % počet, kolikrát se animace přehraje
fps = .2; % snímky za sekundu (rychlost přehrávání)
nframes = 6; % počet rámců (snímků)
Frames = moviein(nframes); % nastavení matice 'Frames'
syms x
a = [-2*pi 2*pi -5 5]; % nastavení meze
fplot('sin(x)',a,'-k') % graf funkce f(x) = sin(x)
text(-5.5,1.5,'sin(x)') % přidáme popis grafu (umístění podle zadaných
souřadnic)
title('aproximace funkce f(x) = sin(x) Taylorovým polynomem')
% nadefinujeme jednotlivé snímky 'Frames'
```

% ke každému přidáme text
% pro každý graf nastavena jiná barva
`Frames(:,1) = getframe(f);` % první sloupec matice Frames
`hold on` % zabraňuje postupnému překreslování jednoho grafu druhým
`T_1 = taylor(sin(x),2);` % nadefinujeme jednotlivé snímky Frames
`fplot(char(T_1),a,'m');` % vykreslí graf funkce taylor modrou barvou
`text(-3,-2,'T_1')` % přidá popisek
`Frames(:,2) = getframe(f);` % druhý sloupec matice Frames
```
T_3 = taylor(sin(x),4);
fplot(char(T_3),a,'r');
text(-3,4,'T_3')
Frames(:,3) = getframe(f);
T_5 = taylor(sin(x),6);
fplot(char(T_5),a,'g');
text(-5,-2,'T_5')
Frames(:,4) = getframe(f);
T_7 = taylor(sin(x),8);
fplot(char(T_7),a,'b');
text(-4,4,'T_7')
Frames(:,5) = getframe(f);
T_9=taylor(sin(x),10);
fplot(char(T_9),a,'y');
text(-5.75,-0.5,'T_9')
Frames(:,6) = getframe(f);
```
`movie(Frames,reruns,fps)`% Frames přehrajeme jako animaci
`legend('sin(x)','T_1','T_3','T_5','T_7','T_9')` % přidá legendu
`movie2avi(Frames,'Priklad_5_1.avi','fps',.2)` % uloží ve formátu avi a pomocí argumentu fps je nastavena rychlost, jakou se přehraje animace

# Příklad 6

% Příklad 6: Pohyb bodu po grafu funkce f(x) = cos(x) a zobrazení změny funkční hodnoty.
% vykreslíme graf funkce f(x) = cos(x)
% fialový bod na začátek grafu
% použijeme příkaz axis k zafixování os
% posouváme bod pomocí aktualizace vlastností xdata a ydata v cyklu for
% použijeme drawnow abychom okamžitě aktualizovali změny dat
% tímto docílíme k vytvoření animaci
```
close all;
clear all;
```

```
clc;
f = figure();
x = linspace(0,10,1000);
y = cos(x);
plot(x,y)
hold on
p = plot(x(1),y(1),'o','MarkerFaceColor','m');% argument MarkerFace-
Color nastavíme barvu bodu
hold off
text(5,3,'Změna funkční hodnoty:','FontSize',14) % přidá popisek na dané
souřadnice
t = text(5,2.5,num2str(y(1)),'VerticalAlignment','top','FontSize',14);
% použijeme funkci num2srt, která slouží k transformaci čísla na řetězec - umožní
zobrazení změny funkční hodnoty
axis('equal') % automatické nastavení poměru stran os
title('Pohyb bodu po grafu funkce f(x) = sin(x)')
for i = 2:length(x) % length(x) je funkce pro zjištění délky vektoru x
set(p,'xdata',x(i),'ydata',y(i)); % set umožňuje nastavit vlastnosti ob-
jektu p - posouvá bod
set(t,'String',num2str(y(i))); % set umožňuje nastavit vlastnosti objektu
t - umožní zobrazení změny funkční hodnoty
drawnow % umožní bezprostředně aktualizovat obrazovku
end
```

# Příklad 6.1

```
% Příklad 6.1: Pohyb bodu po grafu funkce f(x) = cos(x) a uložení animace.
close all;
clear all;
clc;
f = figure();
vid = VideoWriter('Priklad_6_1.avi'); % vytvoří soubor
vid.Quality = 100; % kvalita
vid.FrameRate = 50; % snímky za sekundu
open(vid); % otevře soubor
x = linspace(0,10,1000);
y = cos(x);
plot(x,y)
hold on
p = plot(x(1),y(1),'o','MarkerFaceColor','m');% argument MarkerFace-
Color nastavíme barvu bodu
```

```
hold off
text(5,3,'Změna funkční hodnoty:','FontSize',14) % přidá popisek na dané souřadnice
t = text(5,2.5,num2str(y(1)),'VerticalAlignment','top','FontSize',14);
% použijeme funkci num2srt, která slouží k transformaci čísla na řetězec - umožní zobrazení změny funkční hodnoty
axis('equal') % automatické nastavení poměru stran os
title('Pohyb bodu po grafu funkce f(x) = sin(x)')
for i = 2:length(x)) % length(x) je funkce pro zjištění délky vektoru x
set(p,'xdata',x(i),'ydata',y(i)); % set umožňuje nastavit vlastnosti objektu p - posouvá bod
set(t,'String',num2str(y(i))); % set umožňuje nastavit vlastnosti objektu t - umožní zobrazení změny funkční hodnoty
drawnow % umožní bezprostředně aktualizovat obrazovku
getframe(f)
writeVideo(vid, getframe(f)); % uloží snímky do souboru
end
```