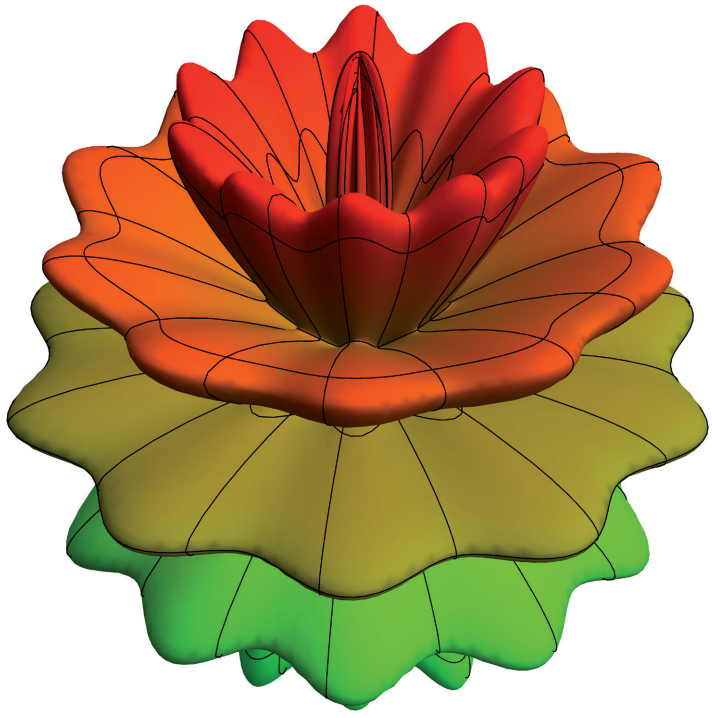
EDÍCIA VYSOKOŠKOLSKÝCH UČEBNÍC

SOFTVÉR MATHEMATICA: PRVÝ DIEL

Oľga Stašová, Matej Medľa



STU SvF
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STAVEBNÁ FAKULTA

SPEKTRUM STU

SOFTVÉR MATHEMATICA:
PRVÝ DIEL

SOFTVÉR MATHEMATICA: PRVÝ DIEL

Oľga Stašová, Matej Medľa

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
2021

Táto učebnica oboznámi čitateľov so základným používaním softvéru „Mathematica“. Jej súčasťou sú aj kapitoly, v ktorých sú s použitím tohto softvéru spracované témy z matematickej analýzy a lineárnej algebry. Je primárne určená pre predmet „Softvér (Mathematica 1)“. Poslúži však aj pre ďalších 13 predmetov vyučovaných pomocou tohto softvéru a tiež pre študentov, ktorí vypracovávajú svoje záverečné práce v Mathematice.

Táto učebnica bola podporená grantmi APVV-19-0460 a VEGA 1/0709/19.

Všetky práva vyhradené. Nijaká časť textu nesmie byť použitá na ďalšie šírenie akoukoľvek formou bez predchádzajúceho súhlasu autorov alebo vydavateľstva.

© Mgr. Oľga Stašová, PhD., Ing. Matej Medľa, PhD.

Recenzenti: doc. RNDr. Daniela Velichová, CSc.
Ing. Viera Kleinová, PhD.
Ing. Mária Boossaert Tješšová, PhD.

Schválila Edičná rada Stavebnej fakulty STU v Bratislave.

ISBN 978-80-227-5138-4

Predslov

Táto učebnica oboznámi čitateľov so základmi používania softvéru Mathematica. Jej súčasťou sú aj kapitoly, v ktorých sú s použitím tohto softvéru spracované témy z matematickej analýzy a lineárnej algebry (základná matematika pre študentov technických odborov). Učebnica je určená predovšetkým poslucháčom študijného programu Matematicko-počítačové modelovanie ako základný študijný materiál pre predmet Softvér (Mathematica 1).

Zároveň však poslúži aj ako doplnková študijná literatúra pre ďalšie predmety tohto študijného programu, pri ktorých sa v rámci ich výuky používa softvér Mathematica: Softvér (Mathematica 2), Štatistické metódy, Numerické riešenie diferenciálnych rovníc, Metóda konečných prvkov, Finančná matematika, Prúdenie kvapalín a plynov, Diferenciálna geometria, Fyzikálne princípy v matematickom modelovaní 2, Numerické metódy v prúdení, Biomechanika, Optimalizácia 2 a Spracovanie obrazu. Učebný text je vhodný aj pre všetkých študentov, ktorí pri vypracovávaní svojich bakalárskych, diplomových a dizertačných prác využívajú tento softvér.

Naša učebnica vychádza z veľmi rozsiahleho dokumentového centra softvéru Mathematica, ktoré je jeho súčasťou. V dokumentovom centre sa nachádza množstvo informácií, ktoré stručne aj podrobne popisujú jednotlivé zabudované funkcie. Ich použitie je v ňom demonštrované na príkladoch rôznych úrovní od veľmi jednoduchých až po špecifické aplikácie. Študentom, ktorí by sa chceli vzdelávať nad rámec našej učebnice, odporúčame rozsiahlu knihu od Leonida Shifrina [1], ktorá vysvetľuje princípy softvéru Mathematica v oveľa širšom rozsahu. Naopak, čitateľov, ktorým postačia len nevyhnutné základy, odkazujeme na knihu od Stephena Wolframa [3]. Jej autor je zároveň aj hlavným tvorcom tohto softvéru. Práca so softvérom Mathematica je tam predstavená na jednoduchých príkladoch stručne a pritom veľmi výstižne. Online verzia tejto knihy je zdarma dostupná na https://www.wolfram.com/language/elementary-introduction/2nd-ed/. Na záver sa ešte zmienime o skriptách od Magdy Komorníkovej a Karola Mikulu [2]. Niektoré témy sú podrobnejšie spracované v našej učebnici, iné v učebnici [2]. Avšak od vydania týchto skrípt prešlo už viac ako 20 rokov a za ten čas sa aj vývoj softvéru Mathematica posunul výrazne dopredu. Naša učebnica vychádza z verzie 12.0, ktorá bola uvedená na trh 16.4.2019.

Radi by sme vyjadrili úprimnú vďačnosť recenzentom doc. RNDr. Daniele Velichovej, CSc., Ing. Viere Kleinovej, PhD. a Ing. Márii Bossaert Tješšovej, PhD., za ich podnetné pripomienky, ktoré prispeli k skvalitneniu učebnice.

Obsah

Kapitola 1

Úvod do softvéru Mathematica

Wolfram Mathematica je symbolický výpočtový softvér, ktorý má v sebe zabudovaný rozsiahly matematický aparát, a to v takej forme, že je vhodný aj pre matematických laikov. Využíva sa teda nielen na univerzitách a vo výskumných inštitúciách, ale svoje uplatnenie našiel aj v širokej palete praktických oblastí (inžinierske úlohy, spracovanie dát, štatistika...). Svojím využitím je podobný softvérom Matlab, Maple a tiež programovaciemu jazyku Python. Jeho nevýhodou v porovnaní s klasickými programovacími jazykmi je fakt, že nedokáže vytvoriť výstupný súbor spustiteľný na počítači, v ktorom nie je softvér Mathematica nainštalovaný.

Vývoj softvéru Mathematica začal v roku 1986 Stephen Wolfram. O rok neskôr založil v Champaign (Illinois) firmu Wolfram Research, ktorá neustále vylepšuje a dopĺňa tento softvér. V roku 1988 bola na trh uvedená prvá verzia 1.0. V súčasnosti je najnovšou verziou 12.2 (od 16.12.2020).

Wolfram Mathematica, tak ako aj programovacie jazyky, dokáže vykonávať základné matematické operácie. Podrobnejšie sa im budeme venovať v kapitole 3.

```
In[1]:= Cos[0] + 1

Out[1]= 2
```

Avšak softvér Mathematica na rozdiel od väčšiny softvérov a programovacích jazykov dokáže vykonávať aj symbolické výpočty. Napríklad integrovať, derivovať, ba dokonca aj riešiť diferenciálne rovnice.

```
In[2]:= ∫_a^b z Sin[z] dz

Out[2]= Cos[a] - b Cos[b] - Sin[a] + Sin[b]

In[3]:= DSolve[u'[x] + u[x] == a Sin[x], u[x], x]

Out[3]= {{u[x] → ℯ^-x C[1] + 1/2 a(-Cos[x] + Sin[x])}}
```

Možnosť zadávať kód, ktorý je formátovaný podobne, ako by ho písal matematik na papier, ho robí ľahko čitateľným.

In[4]:=
$$\sum_{i=1}^{\infty} \frac{1}{i^6}$$

Out[4]=
$$\frac{\pi^6}{945}$$

Používanie symbolických výpočtov si detailnejšie predstavíme v kapitolách 6, 10 a 11.

Softvér Mathematica používa ako základnú dátovú štruktúra zoznam (po angl. list), ktorý slúži nielen na reprezentáciu dát, ale taktiež aj na reprezentovanie vektorov a matíc.

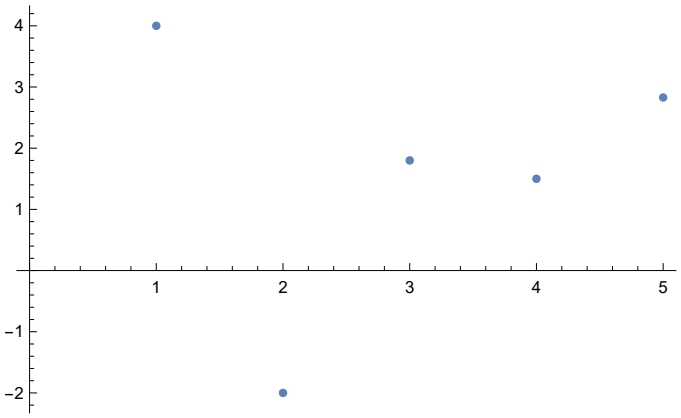
```
In[5]:= {1, 2} + {2, 1}

Out[5]= {3, 3}
```

So zoznamami sa vo všeobecnosti oboznámime v kapitole 4 a ako reprezentantov vektorov a matíc si ich predstavíme v kapitole 12.

Silnou stránkou softvéru Mathematica je aj jeho vizualizačný aparát. Umožňuje nielen výborne vizualizovať zadané dáta,

```
In[6]:= ListPlot[{4, -2, 9/5, 1.5, Sqrt[8]}]
```

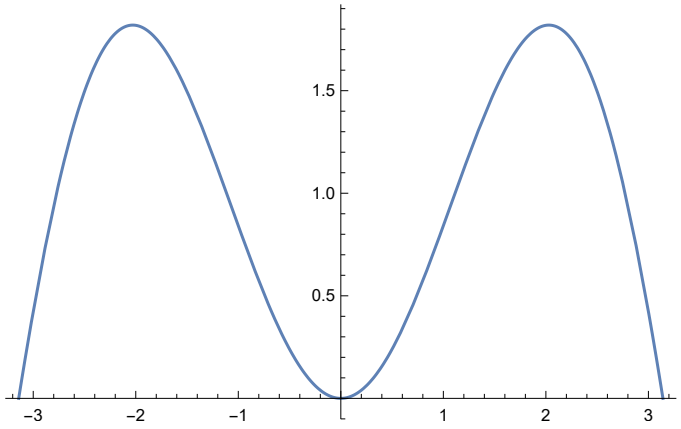
Out[6]=



ale taktiež aj grafy funkcií zadaných symbolicky.

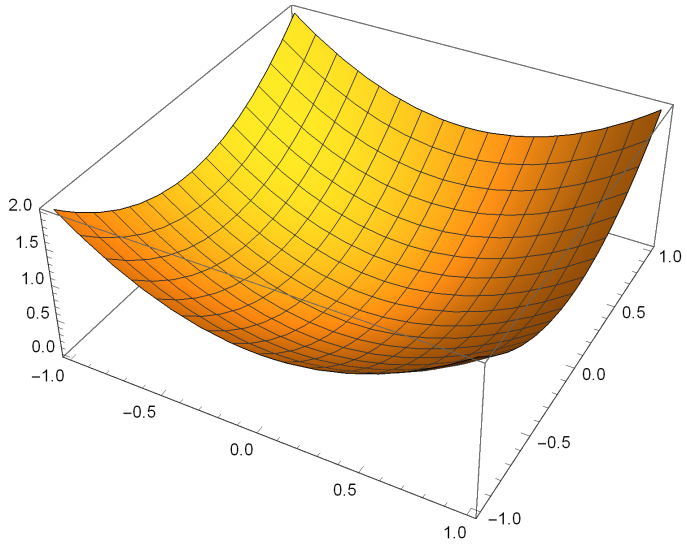
```
In[7]:= Plot[z Sin[z], {z, -π, π}]
```

Out[7]=



In[8]:=
```
Plot3D[x^2 + y^2, {x, -1, 1}, {y, -1, 1}]
```

Out[8]=



V kapitole 9 si predstavíme zobrazovanie základných geometrických útvarov.

In[9]:=
```
Graphics[{Red, Triangle[{{0, 0}, {0, 1}, {3, 0}}]}]
```

Out[9]=

Definovanie funkcií sa dá v softvéri Mathematica zadávať rôznymi spôsobmi. Jeden z nich je veľmi podobný zápisu z klasickej matematiky.

In[10]:=
```
f[x_]:= x^2
f[2] + f[z]
```

Out[10]=
```
4 + z^2
```

Ďalšie spôsoby definovania funkcií vrátane ich benefitov a obmedzení si predstavíme v kapitole 8.

Prednosti softvéru Mathematica ocenia aj tí, ktorí preferujú klasické procedurálne programovanie. Softvér Mathematica totiž umožňuje používať **For**, **Do** cykly, **If**, **Which** podmienky a ďalšie funkcie známe z procedurálneho programovania.

In[11]:=
```
sum = 0;
For[i = 1, i < 999, ++i,
 sum += 1./(i^6)
]
sum
```

Out[11]= 1.01734

V texte za slovenskými termínmi uvádzame v zátvorkách ich anglické ekvivalenty. Upozorňujeme, že nejde o najvýstižnejší preklad, ale o konkrétne anglické termíny používané v softvéri Mathematica.

Kapitola 2

Základná štruktúra

Softvér pozostáva z dvoch základných modulov, z výpočtového jadra (po angl. kernel) a z užívateľského rozhrania (po angl. front end). Rozhranie prijíma od užívateľa príkazy, odovzdáva ich jadru na spracovanie a zobrazuje získané výsledky. Softvér Mathematica využíva vlastný programovací jazyk Wolfram.

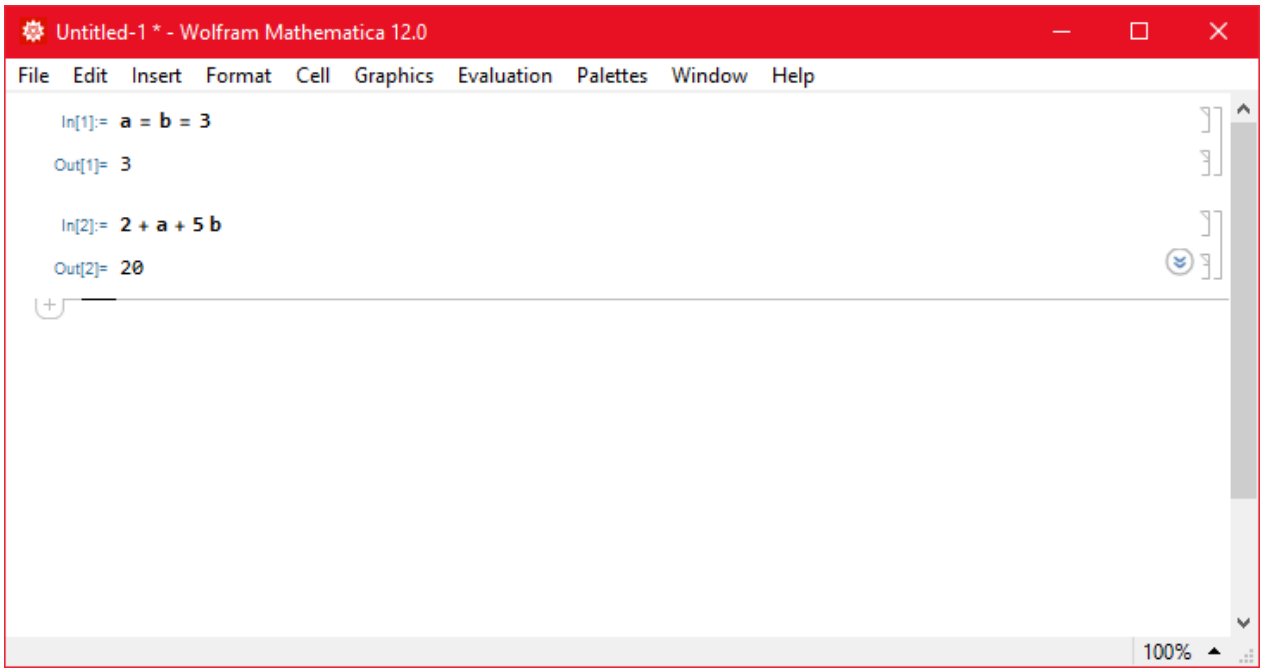
Príkazy sa zadávajú do užívateľského rozhrania tzv. zápisníka (po angl. notebook) pozri Obr. 2.1, ktorý je možné uložiť na disk, znovu načítať a editovať. Keď otvoríme nový zápisník, objaví sa prázdny dokument. Hneď, ako začneme zadávať príkaz alebo text, vytvorí sa bunka (po angl. cell). Bunka sa uzavrie (ukončí) okamžite potom, ako ju (jej zadanie) odošleme na spracovanie do jadra stlačením klávesovej kombinácie **Shift** a **Enter**. Jadro príkaz vykoná a vráti výsledok späť užívateľskému rozhraniu, ktoré ho zobrazí.

```
In[12]:= 5 + 5

Out[12]= 10
```

V tomto prípade jadro vrátilo číslo 10 a užívatelské rozhranie ho zobrazilo. Ak odošleme na spracovanie funkciu zobrazujúcu grafické objekty, jadro vráti späť množinu základných geometrických objektov, ktoré užívateľské rozhranie zobrazí graficky (pre viac informácii pozri kapitolu 9). Hranaté zátvorky na pravej strane zápisníka ohraničujú jednotlivé bunky.

Bunku, do ktorej zadávame príkazy, definície funkcií, premenných a podobne, označujeme ako vstupnú (po angl. input). Vstupná bunka môže obsahovať jeden príkaz alebo aj viac príkazov súčasne. Po jej odoslaní a spracovaní v jadre užívateľské rozhranie zobrazí výsledok vo výstupnej (po angl. output) bunke, ktorá bude v zápisníku umiestnená pod jej prislúchajúcou vstupnou bunkou. Výsledok môže byť číslo, vektor, matica, graf atď.

```
In[13]:= Sin[Pi] + Cos[Pi]

Out[13]= -1
```



Obr. 2.1: Príklad zápisníku vo verzii Wolfram Mathematica 12.0

Každá vstupná bunka je označená ako In[n]:=, kde n je jej poradové číslo v zápisníku a jej zodpovedajúca výstupná bunka ako Out[n]=. Toto sa využíva, keď chceme ďalej pracovať s výsledkami z predchádzajúcich výpočtov. Výsledky vstupných buniek (zobrazené vo výstupných bunkách) sa dajú používať pomocou príkazu `Out[n]`, ktorý je totožný s príkazom `%n`. Ak chceme aplikovať posledný výsledok, tak stačí použiť len príkaz `%` a na predposledný výsledok `%%`.

```
In[14]:= Out[15] + %14

Out[14]= 9
```

Ešte praktickejší spôsob využívania predchádzajúcich výsledkov je uloženie týchto výsledkov do premenných, ktoré si predstavíme v nasledujúcej sekcii.

Softvér Mathematica, na rozdiel od klasických programovacích jazykov, nevykonáva príkazy v takom poradí, ako sú napísané. Príkazy sú vykonané, až keď sa na danú bunku nastavíme kurzorom myši a stlačíme **Shift+Enter**. A keďže sa môžeme myšou nastaviť na jednotlivé bunky v rôznom poradí, môže sa stať, že príkazy boli vykonané v inom poradí, ako sú zapísané. Poradie, v akom boli bunky zavolané, zistíme podľa poradového čísla n v označení bunky In[n].

Textové (po angl. text) bunky obsahujú len text (zvyčajne komentáre k riešeným príkladom).

Poznámka 2.1

Keď do bunky začneme písať text, po pridaní prvého znaku za prvú medzeru sa objaví ponuka Convert to Text Cell na zmenu vstupnej bunky na textovú bunku. Po kliknutí na tento text sa

bunka zmení na textovú. Pri odoslaní textovej bunky do jadra na spracovanie je táto požiadavka ignorovaná. Zadaná textová bunka zostáva nezmenená a nezobrazí sa k nej žiadny výstup. Do jadra na spracovanie je možné odoslať len vstupné bunky. Alternatívne sa štýl bunky môže zmeniť pomocou príkazov v hornom menu v ponuke `Format` → `Style`.

Poznámka 2.2

Niekedy je vhodné mať uložené len údaje vo vstupných bunkách, napr. z dôvodu pamäťovej úspornosti, keď sú výstupné údaje obrovských rozmerov. Všetky výstupné bunky v zápisníku odstránime príkazom v hornom menu `Cell` → `Delete All Output`.

Ak vstupná bunka obsahuje viac príkazov, píšeme každý z nich do samostatného riadku.

```
In[15]:= 5 + 5
         Sin[Pi] + Cos[Pi]

Out[15]= 10

Out[16]= -1
```

Viac príkazov môžeme zadať aj do jedného spoločného riadku, ale musia byť oddelené bodkočiarkou. Bodkočiarka za príkazom (na konci riadku za jediným príkazom a aj medzi príkazmi) zabráni vytvoreniu výstupnej bunky. Príkaz sa síce vykoná, ale jeho výstup sa nezobrazí v zápisníku. Využíva sa to napr. keď nejaké výsledky výpočtov ukladáme do premenných, s ktorými budeme ďalej pracovať, ale nepotrebujeme poznať ich konkrétne hodnoty.

```
In[17]:= 5 + 5;
         Sin[Pi] + Cos[Pi];
```

Do jednej bunky píšeme príkazy, ktoré spolu súvisia a majú byť vykonané v sériovej postupnosti. Nie je vhodné zadávať do spoločnej bunky príkazy, ktoré sa majú vykonať len raz s príkazmi, ktoré chceme opakovať. Ak sa dodatočne rozhodneme rozdeliť príkazy jednej bunky do dvoch samostatných, tak sa nastavíme kurzorom myši na miesto rozdelenia. Potom klikneme pravým tlačidlom myši a po zobrazení ponuky príkazov klikneme ľavým tlačidlom myši na `Divide Cell`.

Bunky môžeme kopírovať, vymazať, nastaviť v nich typ písma a podobne. Bunku, s ktorou chceme pracovať, musíme najskôr vyznačiť – t. j. klikneme ľavým tlačidlom myši na hranatú zátvorku, ktorá ju vpravo označuje. Vyznačiť môžeme súčasne aj viaceré bunky idúce za sebou tak, že sa kurzorom myši nastavíme na zátvorku prvej z nich a držiac ľavé tlačidlo myši ťaháme po všetkých ich zátvorkách. Ak chceme vložiť novú bunku medzi dve už existujúce, nastavíme sa kurzorom myši pod prvú z nich a stlačíme ľavé tlačidlo myši. Keď sa objaví vodorovná čiara cez celý zápisník, môžeme zadávať vstup do novej bunky.

Poznámka 2.3

V prípade, že potrebujeme prerušiť výpočet ešte pred jeho skončením (napr. z dôvodu jeho zacyklenia alebo jeho extrémne dlhého času), klikneme v hornom menu na `Evaluation` → → `Abort Evaluation`.

2.1 Premenné a symboly

Pod premennou rozumieme symbolický názov, ktorý má priradenú hodnotu alebo iné dáta. Za symbol sa považuje hocičo, čo môže fungovať ako premenná, ale nemá priradenú hodnotu. Názov premennej môže obsahovať len písmená a čísla. Nemôže však číslom začínať.

Niektoré názvy premenných, napr. `Pi`, `E`, `I`, sú chránené (po angl. protected), to znamená, že im nemôžeme priradiť hodnotu (keďže ju už majú prednastavenú).

Poznámka 2.4

Softvér Mathematica má preddefinované niektoré matematické konštanty. Predstavíme si len tie najznámejšie.

`Pi`	Ludolfovo číslo $\pi \approx 3.14159$
`E`	Eulerovo číslo $e \approx 2.71828$
`I`	imaginárna jednotka komplexného čísla $i = \sqrt{-1}$
`Infinity`	∞
`Degree`	konštanta na prevod radiánov na stupne: $\frac{\pi}{180}$
`GoldenRatio`	zlatý pomer $\phi = (1 + \sqrt{5})/2 \approx 1.61803$

Tieto konštanty sa okrem klasického zadávania pomocou názvu dajú zadať aj pomocou ich ikon (π, $\mathbb{e}$, $\mathbb{i}$, ∞,...) z matematickej palety, pozri Pozn. 3.2. Informácie o ostatných preddefinovaných konštantách nájdete v dokumentovom centre softvéru Mathematica, pozri Pozn. 2.8.

Ako názov premennej nemôžeme použiť ani názvy funkcií zabudovaných v softvéri Mathematica. Ak chceme zvoliť názov premennej, ktorý ju výstižne vyjadruje, avšak zároveň je totožný s názvom zabudovanej funkcie, môžeme použiť jeho slovenský ekvivalent, alebo anglický variant s malým začiatočným písmenom. Názvy zabudovaných príkazov, funkcií, premenných... v softvéri Mathematica začínajú vždy veľkým písmenom. Premennej sa priraďuje hodnota pomocou `=`

```
In[18]:= a = 1

Out[18]= 1
```

Poznámka 2.5

V softvéri Mathematica je matematická rovnosť definovaná pomocou `==`, zatiaľ čo `=` označuje priradenie – t. j. symbolu naľavo od `=` priradí hodnotu, výraz alebo dáta nachádzajúce sa napravo od `=`.

Dve znamienka "rovná sa" sa používajú napr. pri riešení rovníc (pozri sekciu 11.1) a pri vykresľovaní kontúr funkcií (pozri podsekciu 5.2.2).

Názov premennej môže byť ľubovoľne dlhá postupnosť znakov, nesmie však začínať číslicou. Tú istú hodnotu môžeme priradiť aj viacerým premenným súčasne, napr.

```
In[19]:= a = b = 3

Out[19]= 3

In[20]:= 2 + a + 5b

Out[20]= 20
```

Po priradení hodnoty premennej sa toto priradenie uloží do pamäte jadra. Ak potom otvoríme ďalší zápisník, priradenie premennej zostáva platné aj v ňom. Priradenie premennej sa dá zrušiť nasledujúcimi spôsobmi:

1. Predefinovaním premennej, napr.

```
In[21]:= a = 3; a + 2

Out[21]= 5

In[22]:= a = 11; a - 4

Out[22]= 7
```

2. Uvoľnením, vyčistením (po angl. clear) premennej pomocou `Clear[a]`. Z premennej `a` sa stane symbol `a` (nemá priradenú žiadnu hodnotu). Súčasne môžeme uvoľniť aj viacero premenných, funkcií..., napr. `Clear[k,pom]`.

```
In[23]:= a

Out[23]= 11

In[24]:= Clear[a]

In[25]:= a
```

```
Out[25]= a
```

3. Voľbou z horného menu `Evaluation` → `Quit Kernel` → `Local`. Tento príkaz vymaže z pamäte jadra všetky užívateľské definície a priradenia. Príkaz `Clear[]` je vhodné používať v úlohách s väčším množstvom definícií na miestach, kde už vieme, že ďalej nebudeme používať súčasné priradenie premenných.

```
In[26]:= a + b

Out[26]= a + b
```

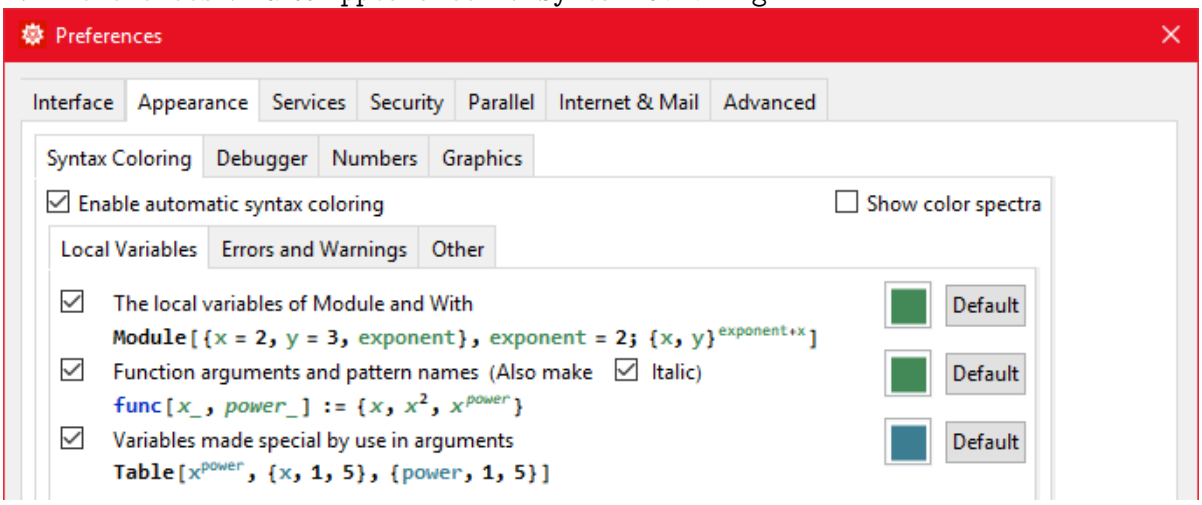
4. Zatvorením softvéru Mathematica,t. j. všetkých zápisníkov. (Zápisník je vhodné predtým uložiť pomocou voľby z horného menu `File` → `Save as`).

Symboly sú v softvéri Mathematica zobrazované **modrou** a premenné a funkcie **čiernou** farbou.

Poznámka 2.6

Softvér Mathematica sa nám snaží pomôcť vykreslením symbolov a premenných rôznymi farbami. Premenné a funkcie s priradenou hodnotou sú vykreslené **čiernou**. Premenná bez priradenia je vykreslená **modrou** farbou a vtedy ju chápeme ako symbol, ktorý sa dá použiť pri symbolických výpočtoch. **Zelenou** farbou sú vykreslené lokálne premenné (pozri sekciu 8.3) a trochu iným odtieňom **zelenej** sú vykreslené niektoré špeciálne vstupné parametre funkcií, ktoré väčšinou slúžia ako lokálne symboly, pozri funkcie `Plot` (v kapitole 5), `D` a `Integrate` (v kapitole 10). Rôznymi odtieňmi **červenej** nás softvér Mathematica upozorňuje na rôzne konflikty a chyby.

Ďalšie informácie o farbách premenných sa dajú nájsť v hornom menu `File` →
→ `Preferences` v karte `Appearance` → `Syntax Coloring`.



2.2 Funkcie

Softvér Mathematica má viac ako 300 000 vstavaných funkcií. Pod pojmom "funkcia" rozumieme v programátorskom prostredí postupnosť na seba nadväzujúcich príkazov tvoriacich jeden logický celok, ktoré sú priradené nejakej premennej. Telo funkcie môže tvoriť aj jediný príkaz. Po zavolaní funkcie sa vykoná postupnosť týchto príkazov, prípadne výstupom bude aj nejaká návratová hodnota.

V niektorej literatúre sú pojmom "funkcia" označované len tie podprogramy, ktoré vracajú nejakú hodnotu a tie, ktoré vykonávajú iba príkazy sú nazývané "procedúry". V našej učebnici však budeme obidve tieto kategórie označovať jednotne ako "funkcie".

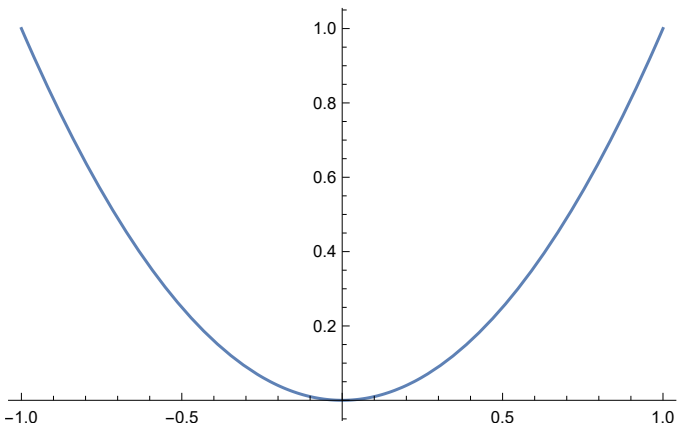
Funkcie môžu vykonávať napríklad vrátenie nejakej výstupnej hodnoty v závislosti od vstupných dát, uloženie vstupných parametrov do súboru, upravenie prvkov nejakého zoznamu, vykreslenie nejakého grafického objektu atď.

Názvy vstavaných funkcií začínajú vždy veľkým písmenom. Ak sú názvy viacslovné, každé ďalšie slovo v názve funkcie začína tiež veľkým písmenom, napr. **`Table`**, **`Solve`**, **`Sum`**, **`PlotStyle`**, **`ListVectorPlot`** atď. V názve nesmú byť medzery. Medzera je totiž jeden zo spôsobov zadania znamienka krát v násobení. Okrem funkcií priamo zabudovaných v softvéri Mathematica si môžeme zadefinovať aj vlastné (tzv. užívateľské), pozri kapitolu 8.

Za funkciou sa píšu hranaté zátvorky **`[]`**. Medzi zátvorky sa píšu vstupné parametre. Funkcie majú dopredu definovaný počet parametrov, ktoré môžu ako vstup dostať. Funkcia môže mať rôzny počet možných vstupných argumentov alebo aj žiaden.

```
In[27]:= Plot[x^2, {x, -1, 1}]
```

Out[27]=



Niektoré však môžu dostať ako vstup ľubovoľný počet argumentov (napr. súčtová funkcia **`Plus`**) alebo zoznam ľubovoľného počtu argumentov (napr. funkcia **`Mean`**, ktorá vracia ako výstup aritmetický priemer všetkých vstupných argumentov).

```
In[28]:= Plus[2, -8, 7.33, 1/7]
         Mean[{2, -8, 7.33, 1/7, 93, -122}]
```

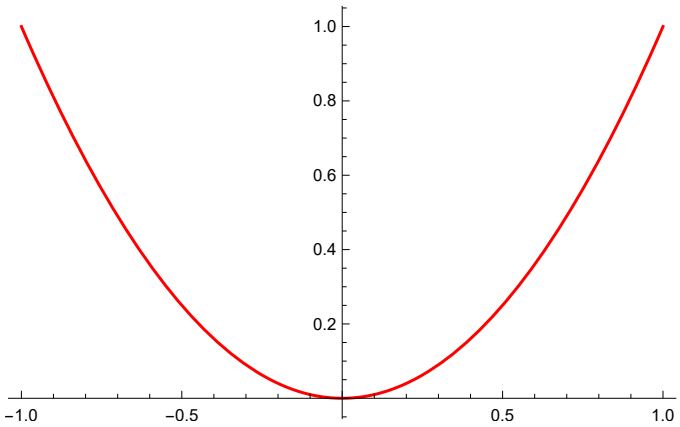
```
Out[28]=  1.47286
         -4.58786
```

V softvéri Mathematica pod pojmom "zoznam" rozumieme objekt, ktorý v sebe drží viaceré prvky a zapisuje sa pomocou krútených zátvoriek **{}**. Viac informácií o zoznamoch nájdete v kapitole 4.

Viacero typov funkcií má aj nepovinné vstupné parametre, tzv. voľby, v tvare: **NazovVolby→NastavenieVolby**.

```
In[29]:= Plot[x^2, {x, -1, 1}, PlotStyle → Red]
```

Out[29]=



Funkcie je možné aj vnárať do seba. Pre väčšinu funkcií platí, že sa najskôr vyhodnotia príkazy v argumentoch funkcie a následne sa výstupné hodnoty pošlú ako vstup do vonkajšej funkcie. Napríklad pre príkaz

```
In[30]:= Log[Sin[Pi/2]]

Out[30]= 0
```

sa najskôr vyhodnotí **Sin[Pi/2]** ako 1 a následne sa funkcia **Log** zavolá s týmto argumentom a vyhodnotí na 0.

Neplatí to pre všetky funkcie. Niektoré funkcie musia pracovať so vstupmi ako s výrazmi, preto ich softvér Mathematica nevyhodnotí, ale pošle funkcii v nezmenenom tvare. Napríklad

```
In[31]:= Solve[y^2 - y == 3, y]
```

$$\text{Out[31]= } \{\{y \to \frac{1}{2}(1 - \sqrt{13})\}, \{y \to \frac{1}{2}(1 + \sqrt{13})\}\}$$

Výstupy funkcií sa dajú ukladať do premenných.

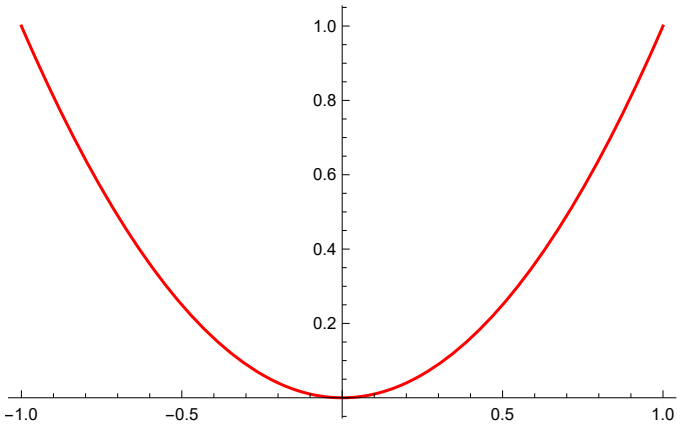
```
In[32]:= riesenie = Solve[y^2 - y == 3, y];
         graf = Plot[x^2, {x, -1, 1}, PlotStyle → Red];
```

S touto premennou môžeme ďalej pracovať.

In[33]:=
```
riesenie
graf
```

Out[33]= $\{\{y \to \frac{1}{2}(1 - \sqrt{13})\}, \{y \to \frac{1}{2}(1 + \sqrt{13})\}\}$

Out[34]=



Poznámka 2.7

Za názvy premenných neodporúčame voliť jedno–dvojznakové skratky, prospešnejšie je používať také pomenovania, ktoré výstižne vyjadrujú podstatu premenných. Ak sa totiž na náš kód pozrie niekto iný, vhodná voľba názvov premenných (ale aj našich funkcií) mu uľahčí porozumenie kódu.

Toto pravidlo zachovávajte aj v prípade, že ide o kód vytvorený len pre vašu vlastnú potrebu. Ak sa naň pozriete o niekoľko týždňov, mesiacov neskôr, neprehľadne napísanému kódu bude náročné porozumieť, napriek tomu, že ste ho písali vy.

2.3 Chybové hlásenia funkcií

Keď odošleme na spracovanie do jadra softvéru funkciu s netriviálnym vstupom (takým, pre ktorý nedokáže nájsť riešenie pomocou pravidiel zabudovaných v nej), vypíše sa pod vstupnú bunku chybové hlásenie.

In[35]:= $\int_{-\infty}^{\infty} x \, dx$

Integrate: Integral of x does not converge on $\{-\infty,\infty\}$.

Out[35]= $\int_{-\infty}^{\infty} x \, dx$

Výstupná bunka sa zobrazí až pod týmto hlásením a v nej sa nachádza naša požiadavka, ktorú sme odoslali vo vstupnej bunke na spracovanie do jadra. Je to zapríčinené tým, že funkcia, ktorú sme použili, nedokáže nájsť riešenie pre zadané vstupné argumenty. V predchádzajúcom príklade chybové hlásenie informuje o tom, že zadaný integrál nekonverguje.

Avšak v niektorých prípadoch s netriviálnym vstupom dostaneme vo výstupnej bunke namiesto zadanej neriešiteľnej požiadavky chybné riešenie.

```
In[36]:= ries = Solve[x < 4, x]

         Solve: The solution set contains a full−dimensional component; use Reduce
         for complete solution information. >>

Out[36]= {{}}
```

V predchádzajúcom príklade hlásenie informuje, že množina riešení obsahuje "full-dimensional component"(t. j. že miera riešenia je rovnaká ako miera priestoru, v ktorom hľadáme riešenie) a z tohoto dôvodu odporúča použiť na riešenie našej požiadavky namiesto funkcie **Solve** funkciu **Reduce**. Napriek tomuto upozorneniu dostaneme ako výstup nejaké (avšak chybné) riešenie. Od funkcie **Solve** sme očakávali, že vráti zoznam **x**, pre ktoré platí **x < 4**. Funkcia nám však vrátila prázdny zoznam, čo nie je správne riešenie. Chybové hlásenie je teda v tomto prípade istým spôsobom upozornením na možnú chybnú interpretáciu výsledku (pre použitie nevhodnej funkcie).

Keďže chybové hlásenie sa nepovažuje za súčasť výstupnej bunky, tak sa do premennej **ries** uložil len prázdny zoznam.

```
In[37]:= ries

Out[37]= {{}}
```

2.4 Dokumentové centrum

Ak chceme zistiť, ako presne funguje nejaká funkcia, ktorej názov poznáme a aké voľby (po angl. options) pre ňu môžeme použiť, máme dve možnosti:

1. Napísať do vstupnej bunky **NazovKonkretnejFunkcie**. Potom sa zobrazí písmeno i v modrom krúžku. Tu máme tri možnosti:

   (a) Kliknúť ľavým tlačidlom myši na modrý krúžok a vtedy sa zobrazí okno s podrobnými informáciami o použití danej funkcie. Nájdeme tam tiež voľby, ktoré sa dajú v rámci funkcie použiť, hypertextové názvy súvisiacich funkcií a základné (po angl. basic) aj zložitejšie príklady (po angl. neat examples) aplikovania funkcie. Príklad, ktorý je vhodný na riešenie našej úlohy, si môžeme skopírovať do zápisníka, zameniť v ňom zadané hodnoty za naše, vymazať alebo pridať doň požadované voľby a podobne.

   (b) Postaviť sa kurzorom myši na **NazovKonkretnejFunkcie**. Vtedy sa okrem písmena i v modrom krúžku zobrazí aj znak dvoch šípok smerujúcich nadol a po kliknutí ľavým tlačidlom myši na ne sa zobrazí stručný návod použitia danej funkcie.

(c) Postaviť sa kurzorom myši na **`NazovKonkretnejFunkcie`** a stlačiť kláves F1. Vtedy sa zobrazí okno s podrobnými informáciami o použití danej funkcie popísané v bode 1a.

2. Napísať do vstupnej bunky **`?NazovKonkretnejFunkcie`** alebo **`??NazovKonkretnejFunkcie`** a odoslať vstupnú bunku (bez kliknutia na modrý krúžok) na spracovanie do jadra súčasným stlačením klávesov **Shift** a **Enter**. Potom sa zobrazí stručný návod použitia danej funkcie ukončený znakmi ». Keď klikneme ľavým tlačidlom myši na », zobrazí sa (resp. zavrie sa) okno s podrobnými informáciami o použití danej funkcie popísané v bode 1a.

Poznámka 2.8

Ak hľadáme nejaké pre nás neznáme zabudované funkcie (prípadne nejaké matematické informácie), ktoré by nám boli nápomocné pri riešení našich úloh, môžeme využiť voľbu z horného menu `Help` → `Find Selected Function`. Potom sa nám zobrazí okno dokumentového centra s názvami rôznych oblastí matematiky. Klikneme ľavým tlačidlom na tú, do ktorej spadá riešenie našej úlohy a zobrazí sa nám zoznam jej podoblastí. Opäť klikneme ľavým tlačidlom, tentokrát na najvhodnejšiu podoblasť našej úlohy. Vtedy sa nám zobrazí okno, v ktorého hornej časti bude stručná charakteristika danej podoblasti a pod ňou hypertextové názvy zabudovaných funkcií rozdelené tematicky do viacerých skupín. Po kliknutí ľavým tlačidlom myši na nejaký z týchto hyperlinkov, dostaneme informácie o tejto funkcii, ako to bolo opísané v predchádzajúcom odstavci.

Poznámka 2.9

Ak ide o nejaké základné zabudované funkcie, môžeme obísť pomoc dokumentového centra a názov priamo „uhádnuť“. Názvy zabudovaných príkazov sú totiž často totožné s ich anglickými názvami. Keď začneme písať anglický názov (začínajúci veľkým písmenom), stačia prvé dve písmená, zobrazí sa nám zoznam zabudovaných príkazov začínajúcich na tieto písmená. Keď o niektorom z nich predpokladáme, že by mohol byť pre nás zaujímavý, nastavíme sa naň kurzorom myši. Vtedy sa vedľa zobrazí písmeno i v modrom krúžku a po kliknutí naň ľavým tlačidlom myši sa zobrazí okno s podrobnými informáciami o použití danej funkcie. Zoznam zabudovaných príkazov začínajúcich na nejaké písmená sa dá zobraziť aj príkazom **`?pismena*`**. Keď sa na niektorý z nich nastavíme kurzorom, zobrazí sa písmeno i v modrom krúžku (na zobrazenie okna s podrobnými informáciami) a znak dvoch šípok smerujúcich nadol (na zobrazenie stručného návodu použitia danej funkcie).

2.5 Zhrnutie

V tejto kapitole sme sa oboznámili s grafickým rozhraním softvéru Mathematica, jeho základným ovládaním, zápisom funkcií a prácou s nimi a s definovaním premenných. Taktiež sme si ukázali,

ako pracovať s dokumentovým centrom, ktoré je výborným prostriedkom na ďalšie štúdium tohto softvéru. V nasledujúcich kapitolách si predstavíme ďalšie základy softvéru Mathematica a vybrané zabudované funkcie. Členenie kapitol bolo zvolené vzhľadom na potreby študentov aplikovanej matematiky.

Kapitola 3

Aritmetické operácie a matematické funkcie

Aj keď softvér Mathematica bol vytvorený predovšetkým na riešenie zložitých komplexných úloh z praxe, môže zároveň slúžiť aj ako vedecká kalkulačka. Jej prednosťami sú: vysoká presnosť (ktorá sa naviac dá rôznymi voľbami optimálne prispôsobovať), užívateľská nenáročnosť a množstvo zabudovaných matematických funkcií.

3.1 Aritmetické operácie

Základné matematické operácie sa v softvéri Mathematica zadávajú pomocou klávesnice nasledovne:

`x+y`	sčítanie	najnižšia priorita
`x-y`	odčítanie	najnižšia priorita
`x*y`	násobenie	stredná priorita
`x/y`	delenie resp. zlomok	vysoká priorita
`x^n`	umocnenie	najvyššia priorita

V prípade, že výraz obsahuje viacero operácií, poradie, v ktorom sa vypočítajú, je dané podľa priority operátora. Prioritu aritmetických operácií meníme použitím okrúhlych zátvoriek. Ich správne používanie je veľmi dôležité. Stačí malá nepozornosť a zmení sa zadanie príkladu. Napr. chceme zadať e^{5x}, avšak ak výraz `5x` nedáme do zátvorky, softvér Mathematica to vyhodnotí ako $e^{5}x$.

```
In[1]:= e^5x
        e^(5x)

Out[1]= e^5 x
        e^(5 x)
```

Ďalšou veľmi častou chybou, ktorej sa študenti dopúšťajú, je zabúdanie zátvoriek pri delení, ak je deliteľom súčin. Napr. ak chceme zadať `Sin[x]/2x` a výraz `2x` nedáme do zátvorky, v dôsledku poradia operácií to softvér Mathematica vyhodnotí ako $\frac{\texttt{Sin[x]}}{2}\texttt{x}$.

Všetky operácie sa dajú zadať aj pomocou zabudovaných funkcií. Napriklad `+` sa dá zadať pomocou funkcie `Plus[x,y]`.

```
In[2]:= 1 + 2;
        Plus[1, 2]

Out[2]= 3
```

Cvičenie 3.1

Sčítajte čísla 5, 7 a π.

Cvičenie 3.2

Od čísla 12 odčítajte súčet čísel 1, $\frac{3}{6}$ a $\frac{\pi}{6}$.

Znak `*` medzi činiteľmi v násobení môžeme nahradiť medzerou alebo na násobenie použijeme zabudovanú funkciu `Times[x,y]`. Ak zadáme medzeru medzi dva činitele, tak ju softvér Mathematica pre lepšiu čitateľnosť nahradí svetlošedým $\times$.

```
In[3]:= 1 * 2;
        1 × 2;
        Times[1, 2]

Out[3]= 2
```

Delenie v tvare podielu dvoch čísel $\frac{\texttt{x}}{\texttt{y}}$ sa dá zadať pomocou klávesovej skratky **Ctrl+/**. Zlomok vieme zadať aj pomocou vstavanej funkcie `Divide[x,y]`.

```
In[4]:= 1/2;
        1
        -
        2
        Divide[1, 2];

Out[4]= 1
        -
        2
```

Viac o tom, prečo softvér Mathematica vracia $\frac{1}{2}$ namiesto `0.5`, sa dozviete v podkapitole 3.3. Ak je výstupom výpočtu zlomok, softvér Mathematica ho vždy vracia upravený v základnom tvare (t. j. najväčší spoločný deliteľ čitateľa a menovateľa je číslo 1).

Poznámka 3.1

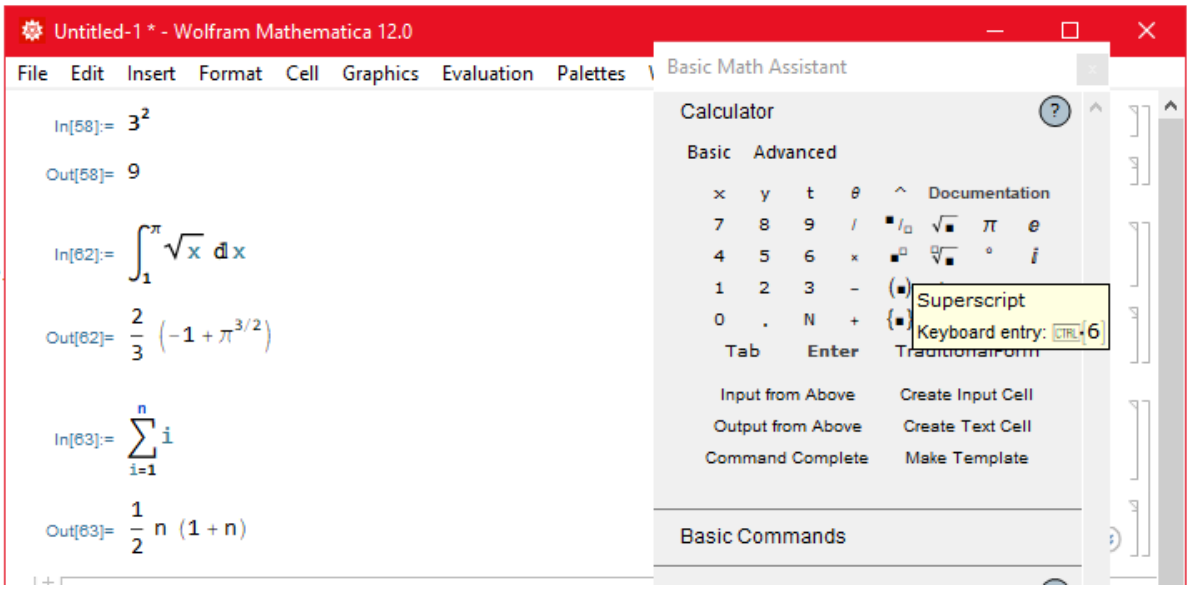
Na začiatku tejto sekcie sme pre zjednodušenie trochu zavádzali. V softvéri Mathematica totiž žiadny operátor na odčítanie dvoch čísel nie je zabudovaný. Na tento účel sa využíva operátor na negáciu čísla `-x`, ktorý sa pomocou funkcie dá zadať ako `Minus[x]`. Tento operátor je dokonca vyhodnotený ako `Times[-1, x]`. Takže keď napíšeme `x-y`, v skutočnosti sa tento výraz vyhodnotí ako `Plus[x,Times[-1, y]]`. Z matematického hľadiska sú tieto operácie ekvivalentné.

Cvičenie 3.3

Aký výsledok vráti softvér Mathematica, ak predelíme číslo 12 číslom 9?

Poznámka 3.2

Softvér Mathematica umožňuje zadávať kód ako štylizovaný text. Takýto kód sa dá zadávať pomocou palety z horného menu `Palletes` → `Basic Math Assistant` alebo pomocou klávesových skratiek. Tie sa dajú nájsť v palete. Keď prejdeme kurzorom myši na konkrétny príkaz, po krátkom čase sa nám zobrazí klávesová skratka pre tento príkaz. Väčšina klávesových skratiek je kombinácia **Ctrl+klávesa** alebo **[Esc]text[Esc]**.



Na umocňovanie môžeme použiť aj zabudovanú funkciu `Power[x,n]` alebo klávesovú skratku **Ctrl+6**, pomocou ktorej zadáme umocňovanie v klasickom tvare $\mathtt{x}^{\mathtt{n}}$.

```
In[5]:= 2^3;
        Power[2, 3];
        2^3
```

Out[5]= 8

V prípade, že je exponent záporne číslo, výsledok bude v tvare zlomku.

In[6]:= $2^{-1/5}$

Out[6]= $\frac{1}{2^{1/5}}$

Cvičenie 3.4

Umocnite 5 na 1/2.

Cvičenie 3.5

Umocnite 5 na -1.

V prípade, že základom mocniny je Eulerovo číslo, môžeme na umocnenie použiť namiesto funkcie **Power[e,n]** aj funkciu **Exp[n]**.

Cvičenie 3.6

Pomocou funkcie **Exp** umocnite e na piatu.

Tieto aritmetické operácie sa dajú vzájomne kombinovať, napr.

In[7]:= $2\left(3 + 5^6 - \frac{7}{3}\right)$

Out[7]= $\frac{93754}{3}$

Poznámka 3.3

Okrem operátorov, ktoré sme si predstavili, existuje ešte veľa ďalších, napr. logické a porovnávacie (||, >, ==,...). Prioritu jednotlivých operátorov v spoločnom výraze zistíme pomocou funkcie **FullForm**, ktorá ako výstup vracia zadaný výraz vyjadrený pomocou funkcií reprezentujúcich operátory.

```
In[8]:= FullForm[a * b + c * d]
        FullForm[a > b || c^d + e * f]

Out[8]= Plus[Times[a, b], Times[c, d]]]
        Or[Greater[a, b], Plus[Power[c, d], Times[e, f]]]
```

Ak si nie ste istí, v akom poradí sa vyhodnocujú operácie, použite radšej nadbytočné okrúhle zátvorky. Tie totiž okrem vynútenia priority operácií v zložitých výrazoch napomáhajú aj čitateľnosti kódu.

3.2 Matematické funkcie

Okrem už vyššie spomenutých funkcií: **Plus**, **Times**, **Power**, **Exp**... má softvér Mathematica v sebe zabudované množstvo ďalších funkcií. Uvádzame len tie najpoužívanejšie.

`Sqrt[x]`	odmocnina
`Abs[x]`	absolútna hodnota
`x!`	faktoriál
`Log[a,x]`	$\log_a x$
`Log[x]`	$\ln(x)$
`Min[x,y,...]`	minimum z čísel x, y,...
`Max[x,y,...]`	maximum z čísel x, y,...
`Divisors[n]`	delitele čísla n
`Limit[f[x],x→x0]`	limita funkcie $f[x]$ v bode x=x0
`Limit[f[x],x→x0,Direction→1]`	limita funkcie $f[x]$ zľava v bode x=x0
`Limit[f[x],x→x0,Direction→-1]`	limita funkcie $f[x]$ sprava v bode x=x0
`Sum[f[i],{i,imax}]`	suma f[i] od i=1 po i=imax
`Sum[f[i],{i,imin,imax}]`	suma f[i] od i=imin po i=imax
`Sum[f[i],{i,imin,imax,krok}]`	suma f[i] od i=imin po i=imax s krokom krok
`Product[f[i],{i,imax}]`	súčin f[i] od i=1 po i=imax
`Product[f[i],{i,imin,imax}]`	súčin f[i] od i=imin po i=imax
`Product[f[i],{i,imin,imax,krok}]`	súčin f[i] od i=imin po i=imax s krokom krok

Na odmocnenie môžeme použiť aj funkciu **Power[x,1/2]** alebo znak odmocniny z palety.

```
In[9]:= Sqrt[9];
        Power[9, 1/2];
        √9

Out[9]= 3
```

Na n-té odmocniny používame funkciu **Power[x,1/n]** alebo znak n-tej odmocniny z palety.

```
In[10]:= Power[27, 1/3];
         27^(1/3)

Out[10]= 3
```

Faktoriál sa dá zadať aj pomocou funkcie **Factorial[x]**.

```
In[11]:= 4!;
         Factorial[4]

Out[11]= 24
```

Funkcie **Sum** a **Product** môžeme zadať z horného menu Palettes → Basic Math Assistant kliknutím na ich ikony v Typesetting.

```
In[12]:= Sum[i, {i, 10}];
         ∑_{i=1}^{10} i

Out[12]= 55
```

Tu sa prvýkrát stretávame s tým, že softvér Mathematica analyzuje zadaný príkaz symbolicky a na základe tejto analýzy ho upraví. Toto je možné vďaka tomu, že na rozdiel od mnohých výpočtových softvérov dokáže vykonávať nielen číselné, ale aj symbolické výpočty. Softvér Mathematica najskôr analyzuje sumu so zadanými vstupnými parametrami a keď ju vie zjednodušiť, tak ju zjednoduší a až potom vypočíta. Tento princíp zabezpečí veľmi rýchle výpočty aj v zdanlivo časovo náročných prípadoch, ako je napr. tento:

```
In[13]:= ∑_{i=1}^{99999999} i

Out[13]= 4 999 999 950 000 000
```

V tomto príklade bol na úpravu zadanej sumy použitý všeobecný vzorec $\sum_{i=1}^{n} i = \frac{1}{2}(n+1)n$ a následne sa **n** nahradilo číslom **99999999**. Názorne si to ukážeme na príklade, kde ako hornú hranicu nezadáme konkrétne číslo ale symbol **n**.

```
In[14]:= ∑_{i=1}^{n} i

Out[14]= 1/2 (n + 1) n
```

Všimnite si, že symbol **n** je vo funkcii **Sum** považovaný za celé číslo (aj keď vo všeobecnosti softvér Mathematica pracuje so symbolmi ako s číslami komplexnými).

Viaceré vstavané funkcie z nasledujúcich kapitol, napr. **Derivative** a **Integrate** z kapitoly 10, majú v sebe zabudované symbolické výpočty.

Cvičenie 3.7

Sčítajte prvých 30 faktoriálov.

Funkcia **Limit[vyraz, x → x0]** vypočíta $\lim_{x->x_0} vyraz$.

```
In[15]:= Limit[Sin[a x]/Sin[b x], x → 0]
```

$$\text{Out[15]=} \quad \frac{a}{b}$$

Za x_0 v limite môžeme dosadiť aj $\pm\infty$ a to buď zadaním z palety (pozri Pozn. 3.2), alebo pomocou preddefinovanej konštanty **Infinity**.

```
In[16]:= Limit[(1 + x/n)^n, n → Infinity]
```

$$\text{Out[16]=} \quad e^x$$

Výstupy niektorých limít môžu vracať aj hodnotu **Indeterminate** (z angl. neurčité), ako napr. v nasledujúcom príklade.

```
In[17]:= Limit[1/x, x → 0]
```

```
Out[17]= Indeterminate
```

Je to zapríčinené faktom, že jednostranné limity pre daný výraz v bode x_0=0 sú rôzne. Limita zľava nadobúda hodnotu $-\infty$, zatiaľ čo limita sprava $+\infty$. Ak chceme počítať jednostrannú limitu, musíme pridať voľbu **Direction → 1** (pre limitu zľava) alebo **Direction → -1** (pre limitu sprava).

```
In[18]:= Limit[1/x, x → 0, Direction → 1]
         Limit[1/x, x → 0, Direction → -1]
```

$$\text{Out[18]=} \quad -\infty$$
$$\infty$$

Naslednujúce funkcie si predstavíme bez obsiahlejšieho komentára, keďže ich použitie je jednoduché.

Zaokrúhlovacie funkcie

Round[x]	najbližšie celé číslo
Floor[x]	najväčšie celé číslo menšie alebo rovnajúce sa x
Ceiling[x]	najmenšie celé číslo väčšie alebo rovnajúce sa x

Goniometrické funkcie

Sin[x]	sínus
Cos[x]	kosínus
Tan[x]	tangens
Cot[x]	kotangens

Cyklometrické funkcie

`ArcSin[x]`	arkussínus
`ArcCos[x]`	arkuskosínus
`ArcTan[x]`	arkustangens
`ArcCot[x]`	arkuskotangens

Hyperbolické funkcie

`Sinh[x]`	hyperbolický sínus
`Cosh[x]`	hyperbolický kosínus
`Tanh[x]`	hyperbolický tangens
`Coth[x]`	hyperbolický kotangens

Cvičenie 3.8

Vypočítajte hodnoty funkcií $f_1(x) = sin(x)$, $f_2(x) = cos(x)$, $f_3(x) = tg(x)$ a $f_4(x) = cotg(x)$ v bode $\pi/2$.

3.3 Presné a približné (numerické) výpočty

Softvér Mathematica rozlišuje nasledujúce typy čísel: celé (po angl. integer), racionálne (po angl. rational), reálne (po angl. real) a komplexné (po angl. complex). Upozorňujeme, že táto klasifikácia nie je úplne totožná s klasifikáciou čísel v klasickej matematike. V nej sa napr. čísla 7 a 7, (t.j. $7, 0$) rovnajú a patria medzi celé aj reálne čísla. Avšak softvér Mathematica tieto dve čísla rozlišuje. Za celé číslo považuje len číslo 7. Všetky čísla obsahujúce desatinnú bodku zaraďuje totiž medzi čísla reálne.

Poznámka 3.4

Keďže Mathematica je americký softvér, namiesto desatinnej čiarky sa v nej používa desatinná bodka.

Aj keď sa za desatinnou bodkou nenachádza žiadne číslo (prípadne len 0), a teda číslo sa javí ako celé, softvér berie do úvahy možnosť, že na nejakom vzdialenom desatinnom mieste sa môže nachádzať nenulová cifra. Takže v softvéri Mathematica si pod pojomom reálne číslo môžeme predstavovať približné číslo, ktorého presnú hodnotu nepoznáme. Podobné je to aj pre dvojicu čísel 7/3 a 7./3 (alebo 7/3.). Prvé číslo je v softvéri Mathematica považované za racionálne a zlomok obsahujúci desatinnú bodku za číslo reálne.

Poznámka 3.5

V softvéri Mathematica je zabudovaná funkcia **Head** (z angl. hlava), ktorá vracia vo výstupe "hlavu" (typ najvrchnejšej funkcie) vstupného argumentu.

```
In[19]:= Head[7]
         Head[7.]
         Head[7/3]
         Head[7./3]

Out[19]= Integer
         Real
         Rational
         Real
```

Poznámka 3.6

V prípade, že vstupným argumentom funkcie **Head** sú zložené výrazy, vo výstupoch dostávame napr. nasledujúce výsledky.

```
In[20]:= Head[Log[2 + 1/2]]
         Head[2 + Log[1/2]]

Out[20]= Log
         Plus
```

Celú reprezentáciu výrazu získame pomocou funkcie **FullForm**.

```
In[21]:= FullForm[Log[2 + 1/2]]
         FullForm[2 + Log[1/2]]

Out[21]= Log[Rational[5, 2]]
         Plus[2, Times[-1, Log[2]]]
```

Komplexné čísla sú čísla obsahujúce imaginárnu jednotku. V softvéri Mathematica ju zadávame z klávesnice pomocou symbolu **I** alebo z palety pomocou symbolu **i**. Reálnu a imaginárnu zložku komplexného čísla získame ako výstup funkcií **Re[komplexne_cislo]** a **Im[komplexne_cislo]**. Môžu nimi byť celé, racionálne alebo reálne čísla.

```
In[22]:= Head[5/7 + 4I]
         Re[5/7 + 4I]
         Im[5/7 + 4I]
```

Out[22]= Complex $\frac{5}{7}$ $\frac{}{4}$

Softvér Mathematica pracuje odlišne s presnými a približnými číslami. Celé a racionálne čísla hodnotí ako presné čísla a reálne čísla (t. j. čísla s desatinnou bodkou) ako čísla približné. Ak je aspoň jedna zo zložiek komplexného čísla reálne číslo, tak aj toto komplexné číslo je klasifikované ako približné číslo. Ak zadáme do nejakého výpočtu všetky operandy ako presné čísla, softvér Mathematica vráti výsledok buď v tvare presného čísla, alebo v tvare výrazu obsahujúceho presné čísla a operácie s nimi. Druhá možnosť nastáva v prípadoch, keď sa výsledok nedá vyjadriť ako presné číslo.

In[23]:=
```
5+5
Sin[Pi/2]
1 + Sqrt[2]/2
2 + Log[1/2]
```

Out[23]=
```
10
1
1 + 1/Sqrt[2]
2 - Log[2]
```

Keďže softvér Mathematica v dvoch posledných prípadoch predchádzajúceho príkladu nedokázal nájsť výsledok v tvare presného čísla, funkcie **Log**, **Sqrt** a **Plus** zostali v nevyčíslenom tvare. Avšak, ak čo i len jeden z mnohých operandov vstupujúcich do výpočtu je približné číslo, tak aj výsledok bude približné číslo.

In[24]:=
```
5. + 5
Sin[Pi/2.]
1. + Sqrt[2]/2
2. - Log[2]
```

Out[24]=
```
10.
1.
1.70711
1.30685
```

Táto vlastnosť sa prejaví aj v prípade, ak do príkladu s komplexným číslom doplníme za imaginárnu zložku desatinnú bodku.

In[25]:=
```
Head[5/7 + 4.I]
Re[5/7 + 4.I]
Im[5/7 + 4.I]
```

```
Out[25]= Complex
         0.714286
         4.
```

Matematické konštanty, ako napr. π a $\mathbb{e}$, sa považujú za presné čísla.

```
In[26]:= 100Pi
         100.Pi

Out[26]= 100π
         314.159
```

Výpočty s približnými číslami sú rýchlejšie. To sa ale výrazne prejaví len pri náročnejších výpočtoch. Ak v takýchto úlohach striktne nepotrebujeme presný výsledok, je veľmi vhodné pracovať s približnými číslami (stačí prvé z presných čísel zmeniť na približné pridaním desatinnej bodky zaň). V prípade, že výsledok je presné číslo v komplikovanom tvare, môžeme ho dať vypísať ako približné pomocou funkcie `N` (z angl. numerical, t. j. numerické, vyčíslené). Táto funkcia sa môže aplikovať dvoma spôsobmi.

```
In[27]:= N[1000Pi]
         N[(1 + √2)/1000]
         1000Pi // N
         (1 + √2)/1000 // N

Out[27]= 3141.59
         0.00241421
         3141.59
         0.00241421
```

Viac o tom, čo presne operátor // znamená, sa dozviete v sekcii 8.2.

Softvér Mathematica má prednastavené zobrazovanie približného čísla na šesť nenulových cifier. V skutočnosti sú tieto čísla uložené v pamäti s viacerými ciframi na desatinných miestach, ale zvyšné pre praktickosť nie sú zobrazené vo výpise. V prípade, že potrebujeme vyčísliť výsledok s presnosťou na `n` cifier, použijeme funkciu `N` v nasledujúcom tvare `N[cislo,n]`.

```
In[28]:= N[100Pi, 2]
         N[100Pi, 3]
         N[100Pi, 30]

Out[28]= 3.1×10^2
         314.
         314.159265358979323846264338328
```

Poznámka 3.7

Ak chceme zmeniť počet zobrazovaných cifier vo všetkých nasledujúcich výstupoch, stačí kliknúť v hornom menu na `Edit` → `Preferences` → `Appearance` → `Formatting` → → `Numbers` a pri vlastnosti `Displayed precision` nastaviť potrebný počet cifier.

3.4 Zhrnutie

V tejto kapitole sme si predstavili základné operácie a funkcie softvéru Mathematica. Vysvetlili sme si, že operácie majú rôznu prioritu (prednosť pred inými), a preto je dôležité pri výpočtoch správne zadať ich poradie (napr. pomocou okrúhlych zátvoriek zmeniť prioritu operácií).

Pri výpočtoch s číslami je dôležité si uvedomiť, s akým typom čísel (presné alebo približné) pracujeme, keďže toto môže mať vplyv nielen na rýchlosť výpočtu, ale aj, ako si ukážeme v kapitole 6, na výsledok.

Kapitola 4

Zoznamy

V tejto kapilote si predstavíme zoznam (po angl. list). Je to jeden zo základných objektov v softvéri Mathematica. Zapisuje sa pomocou krútených zátvoriek **{}**. Z matematického hľadiska sa využíva na definovanie vektorov a matíc.

```
In[1]:= vektor={1, 0, 0};

In[2]:= matica={{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
```

Tento pohľad na zoznamy si bližšie predstavíme v kapitole 12.

Z programátorského hľadiska sa využíva na posielanie viacerých parametrov v jednom argumente funkcie. A keďže funkcia môže vracať len jeden objekt, zoznamy sa používajú ako návratová hodnota aj v situáciách, kde je vo výstupe súčasne viacero objektov (môžu byť aj rôzneho typu). Vždy, keď napíšeme niečo do krútených zátvoriek, ide o zoznam. Podobne, vždy, keď je vo výstupe niečo v krútenych zátvorkách, tak je to zoznam.

Napríklad funkcia na riešenie rovníc **Solve** dostáva parametre pomocou zoznamov a ako výstup vracia zoznam

```
In[3]:= Solve[{x + y == 1, x - y == 2}, {x, y}]

Out[3]= {{x → 3/2, y → -1/2}}
```

V predchádzajúcom príklade je prvým parametrom zoznam dvoch rovníc. Druhý parameter je zoznam dvoch neznámych. Výstup je zoznam s jedným prvkom a tento prvok je opäť zoznam, v tomto prípade s dvoma prvkami. Prvý je riešenie pre prvú neznámu a druhý riešenie pre druhú neznámu. Viac sa o funkcii **Solve** dozviete v kapitole 11.

Na začiatku kapitoly si ukážeme, ako sa zoznamy zapisujú, ako sa pristupuje k jednotlivým prvkom zoznamu a ako sa tieto prvky dajú prepisovať (t. j. nahradiť inými prvkami). V druhej časti si predstavíme základné funkcie na vytváranie zoznamov a ich úpravu.

4.1 Zapisovanie zoznamov a prístup k ich prvkom

Zoznam si môžeme predstavovať ako usporiadanú množinu čísel alebo iných objektov. Zapisuje sa pomocou krútených zátvoriek a prvky zoznamu sú oddelené čiarkami.

```
In[4]:= {1, 3, 3/4, 15}
```

$$\text{Out[4]= } \{1, 3, \frac{3}{4}, 15\}$$

Zoznam môže obsahovať prvky rôzneho typu; čísla, symboly, matematické výrazy alebo iné zoznamy.

```
In[5]:= {5/2, {55, 7, 8}, Log[2], Log[d], b, c, x^5 - 3x + 7, 60, 79};
```

Zoznamy sa dajú ukladať do premenných.

```
In[6]:= list = {5/2, {55, 7, 8}, b, c, 60, 79}
```

$$\text{Out[6]= } \{\frac{5}{2}, \{55, 7, 8\}, b, c, 60, 79\}$$

Cvičenie 4.1

Vytvorte zoznam s prvkami $\cos(0), \cos(\pi), \cos(2\pi), \cos(3\pi), \cos(4\pi)$ a uložte ho do premennej **kosinus**.

K jednotlivým prvkom môžeme pristupovať pomocou dvoch hranatých zátvoriek `[[]]` (na rozdiel od funkcií, pri ktorých používame len jedny hranaté zátvorky). Zoznamy sa indexujú od jednotky.

```
In[7]:= list[[1]]
```

$$\text{Out[7]= } \frac{5}{2}$$

Cvičenie 4.2

```
kosinus = {1, -1, 1, -1, 1}
```

Vyskúšajte, čo sa stane, ak sa budeme snažit pristúpiť k šiestemu prvku zoznamu **kosinus**.

Cvičenie 4.3

```
kosinus = {1, -1, 1, -1, 1}
```

Tretí prvok zoznamu **kosinus** umocnite na druhú.

V prípade záporného indexu `list[[-i]]` vracia `i`-té číslo od konca.

```
In[8]:= list[[-3]]

Out[8]= c
```

Cvičenie 4.4

kosinus = {1, -1, 1, -1, 1}

K tretiemu prvku zoznamu **kosinus** prípočítajte jednotku.

Tento prístup k prvkom môžeme využiť aj na prepísanie hodnoty prvku.

```
In[9]:= list[[1]] = 25;
        list

Out[9]= {25, {55, 7, 8}, b, c, 60, 79}
```

Na prepisovanie hodnoty prvku sa môže použiť aj funkcia **ReplacePart[list,y,i]**. i-tý prvok v zozname list sa nahradí hodnotou y a funkcia vráti nový zoznam s nahradenou hodnotou. Pôvodný zoznam list zostáva nezmenený.

```
In[10]:= list

Out[10]= {25, {55, 7, 8}, b, c, 60, 79}

In[11]:= ReplacePart[list, y, 4]
         list

Out[11]= {25, {55, 7, 8}, b, y, 60, 79}
         {25, {55, 7, 8}, b, c, 60, 79}
```

Cvičenie 4.5

kosinus = {1, -1, 1, -1, 1}

V predchádzajúcich cvičeniach zmena hodnoty prvku zoznamu **kosinus** nezávisela od hodnôt iných prvkov. Teraz nahraďte druhý prvok zoznamu **kosinus** prvým prvkom umocneným na posledný prvok.

Cvičenie 4.6

kosinus = {1, -1, 1, -1, 1}

Vytvorte dvojprvkový zoznam, ktorý bude obsahovať prvý a druhý prvok zoznamu **kosinus**.

Zo zoznamu sa dá vybrať zoznam, ktorý obsahuje i-tý až j-tý prvok pomocou **[[i;;j]]**.

```
In[12]:= list[[2;;4]]

Out[12]= {{55, 7, 8}, b, c}
```

Cvičenie 4.7

kosinus = {1, -1, 1, -1, 1}

Vytvorte dvojprvkový zoznam, ktorý bude obsahovať prvý a druhý prvok zoznamu **kosinus** pomocou značenia, ktoré sme sa práve naučili.

Keď vynecháme druhý index, príkazom `[[i;;]]` dostaneme zoznam od `i`-tého prvku až po posledný.

```
In[13]:= list[[3;;]]

Out[13]= {b, c, 60, 79}
```

Podobne, keď vynecháme prvý index, príkazom **[[;;j]]** dostaneme zoznam od prvého prvku po **j**-tý. Takýto výber sa dá získať aj pomocou funkcií **Take** a **Drop**. Funkcia **Take[list,i]** je ekvivalentom funkcie **list[[;;i]]** a funkcia **Drop[list,i]** ekvivalentom funkcie **list[[i+1;;]]**. Funkcie **Take** a **Drop** naviac fungujú aj pre záporné **i**, t. j. pozícia **i** je tu vyhodnocovaná od konca zoznamu. Pomocou funkcie **Part[list,{i_1,i_2,i_3,i_4,...}]** získame zoznam, ktorého prvkami budú prvky na vymenovaných pozíciách zoznamu **list**.

Príkazy z predchádzajúceho odstavca sa dajú využiť na prepísanie viacerých prvkov zoznamu súčasne.

```
In[14]:= list[[5;;]]={x, y};
         list

Out[14]= {25, {55, 7, 8}, b, c, x, y}
```

Toto sa však nedá využiť pri funkciách **Take** a **Drop**.

```
In[15]:= Drop[list, 4] = {x, y}

         et::write: Tag Drop in Drop[{25,{55,7,8},b,c,x,y},4] is Protected.

Out[15]= {x, y}
```

Cvičenie 4.8

kosinus = {1, -1, 1, -1, 1}

Nahraďte prvé dva prvky zoznamu **kosinus** poslednými dvoma prvkami.

Zoznam na ľavej strane priradenia a zoznam na pravej strane priradenia musia mať rovnaký rozmer (t. j. rovnaký počet prvkov). Ak sú rozmery rôzne, tak sa každý nahrádzaný prvok nahradí celým zoznamom z pravej strany.

```
In[16]:= list[[3;;4]] = {5, 10, 15};
         list
```

```
Out[16]= {25, {55, 7, 8}, {5, 10, 15}, {5, 10, 15}, x, y}
```

Cvičenie 4.9

`kosinus = {1, -1, 1, -1, 1}`

Nahraďte tretí prvok zoznamu **`kosinus`** zoznamom, ktorého prvky sú posledné tri prvky zoznamu **`kosinus`**.

4.1.1 Vnorené zoznamy

Druhý prvok zoznamu **`list`** je taktiež zoznam. A keďže **`list[[2]]`** nám tento zoznam vracia, znovu stačí použiť **`[[i]]`** na získanie jeho i-teho prvku.

```
In[17]:= list[[2]][[3]]

Out[17]= 8
```

Takýto zápis by sa pri viacerých vnorených zoznamoch mohol stať neprehľadným. Z tohoto dôvodu odporúčame používať radšej alternatívny zápis.

```
In[18]:= list[[2, 3]]

Out[18]= 8
```

V ďalšom texte tejto učebnice budeme používať už len tento úspornejší zápis.

Aj v prípade vnorených zoznamov môžeme využiť zápis s dvoma bodkočiarkami `;;`.

```
In[19]:= list = {{11, 12, 13}, {21, 22}, {31, 32, 33, 34}};
         list[[1;;3, 1;;2]]

Out[19]= {{11, 12}, {21, 22}, {31, 32}}
```

Nový zoznam, ktorý obsahuje všetky prvé prvky podzoznamov, môžeme získať nasledovne.

```
In[20]:= list[[;;, 1]]

Out[20]= {{11}, {21}, {31}};
```

A podobne sa dá tento zápis využiť na prepísanie viacerých prvkov súčasne.

```
In[21]:= list[[1;;3, 1;;2]] = {{ax, ay}, {bx, by}, {cx, cy}};
         list

Out[21]= {{ax, ay, 13}, {bx, by}, {cx, cy, 33, 34}}
```

Cvičenie 4.10

`kosinus = {1, -1, {1, -1, 1}, -1, 1}`

Nahraďte v zozname nachádzajúcom sa na tretej pozícii v zozname **kosinus** prvé dva prvky symbolmi **a** a **b**.

Dĺžka (t. j. počet prvkov) zoznamu sa dá zistiť pomocou funkcie **Length[list]**

```
In[22]:= Length[list]

Out[22]= 3
```

alebo pomocou funkcie **Dimensions[list]**.

```
In[23]:= Dimensions[list]

Out[23]= {3}
```

Zoznam, ktorého prvky sú zoznamy rovnakej dĺžky, má v softvéri Mathematica špeciálne postavenie a označujeme ho matica (po angl. matrix) (viac o maticiach v kapitole 12). Ich rozmery sa dajú zistiť pomocou funkcie **Dimensions[list]**, ktorá vracia zoznam rozmerov (t. j. počet riadkov a stĺpcov matice).

```
In[24]:= matica={{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}};
         Dimensions[matica]

Out[24]= {3, 4}
```

Pri vytváraní zoznamov môžeme ísť aj hlbšie a vytvoriť zoznam zoznamov zoznamov.

```
In[25]:= list2x2x3 = {{{111, 112, 113}, {121, 122, 123}},
                      {{211, 212, 213}, {221, 222, 223}}};
         Dimensions[list2x2x3]

Out[25]= {2, 2, 3}
```

4.2 Vytváranie zoznamov úpravou iných zoznamov

Nové zoznamy sa dajú vytvárať základnými matematickými operáciami medzi dvoma zoznamami rovnakej dĺžky

```
In[26]:= {1, 2, 3} + {3, 2, 8}
         {1, 2, 3} - {3, 2, 8}
         {1, 2, 3} * {3, 2, 8}
         {1, 2, 3} / {3, 2, 8}
```

```
Out[26]= {4, 4, 11}

Out[27]= {-2, 0, -5}

Out[28]= {3, 4, 24}

Out[29]= {1/3, 1, 3/8}
```

alebo pomocou zoznamu a čísla (resp. čísla a zoznamu)

```
In[30]:= {1, 2, 3} + 2
         {1, 2, 3} - 2
         {1, 2, 3} * 2
         {1, 2, 3} / 2

Out[30]= {3, 4, 5}

Out[31]= {-1, 0, 1}

Out[32]= {2, 4, 6}

Out[33]= {1/2, 1, 3/2}
```

Cvičenie 4.11

Vytvorte zoznam celých čísel od -3 po 3. Pomocou násobenia zoznamov vytvorte zoznam druhých mocnín čísel od -3 po 3. Uložte ho do premennej **mocniny**.

Dva a viac zoznamov sa dá spojiť do jedného pomocou funkcie **Join**.

```
In[34]:= list = Join[{a, b, c}, {d, e, f}, {g, h, i}]

Out[34]= {a, b, c, d, e, f, g, h, i}
```

Na začiatok zoznamu sa dá pridať ďalší prvok pomocou funkcie **Prepend** a na jeho koniec pomocou funkcie **Append**. Tieto funkcie nezmenia pôvodný zoznam, ale vrátia ako výstup nový zoznam s pridaným prvkom.

```
In[35]:= Prepend[list, 1]
         Append[list, 100]

Out[35]= {1, a, b, c, d, e, f, g, h, i}

Out[36]= {a, b, c, d, e, f, g, h, i, 100}
```

Ak potrebujeme zmeniť vstupný zoznam, uložíme vrátený zoznam do pôvodného zoznamu alebo použijeme namiesto funkcií **Prepend** a **Append** priamo funkcie **PrependTo** a **AppendTo**.

```
In[37]:= list = Prepend[list, 1]
         AppendTo[list, 100]

Out[37]= {1, a, b, c, d, e, f, g, h, i}

Out[38]= {1, a, b, c, d, e, f, g, h, i, 100}
```

Cvičenie 4.12

mocniny = {9, 4, 1, 0, 1, 4, 9}

Pridajte na začiatok a koniec zoznamu **mocniny** druhú mocninu čísla -4 (resp. 4).

V prípade doplnenia nejakého prvku na ľubovoľnú pozíciu v zozname sa dá použiť funkcia **Insert[list, prvok, i]**, ktorá do zoznamu **list** pridá **prvok** na **i**-tú pozíciu a nasledujúce prvky posunie o jednu pozíciu ďalej.

```
In[39]:= list = Insert[list, x, 4]

Out[39]= {1, a, b, x, c, d, e, f, g, h, i, 100}
```

Softvér Mathematica umožňuje viacúrovňové zoznamy spájať do jednoúrovňového zoznamu a naopak. Funkcia **Partition[zoznam, n]** rozdelí **zoznam** na podzoznamy dĺžky **n**.

```
In[40]:= list = Partition[{ax, ay, 13, bx, by, cx, cy, 33, 34}, 3]

Out[40]= {ax, ay, 13}, {bx, by, cx}, {cy, 33, 34}}
```

Funkcia **Flatten** odstráni všetky vnútorné zoznamy a vytvorí jeden dlhý zoznam.

```
In[41]:= Flatten[list]

Out[41]= {ax, ay, 13, bx, by, cx, cy, 33, 34}
```

Pri práci s dátami uloženými v zoznamoch sa často používajú funkcie **Position**, **Count**, **MemberQ**, **Reverse**, **Sort**, **DeleteDuplicates** a aj funkcie **Min** a **Max**, ktoré sme si predstavili v podkapitole 3.2.

```
In[42]:= a = {1, 3, 3/4, 15}

In[43]:= Min[a]

Out[43]= 3/4
```

```
In[44]:= Max[a]

Out[44]= 15
```

V prípade, že zoznam obsahuje aj symbol, tieto funkcie vracajú ako výstup symbol a najmenšie (resp. najväčšie) číslo. O symbole sa totiž nevie, či je väčší alebo menší ako najmenšie (resp. najväčšie) číslo. Ak funkciu **Min** (resp. **Max**) aplikujeme na zoznam obsahujúci podzoznamy, tak výstupom je spoločné minimum (resp. maximum) z hodnôt prvkov zoznamu a prvkov jeho podzoznamov.

```
In[45]:= list = {25, {55, 7, 8}, b, c, 60, 79}

In[46]:= Min[list]

Out[46]= Min[7, b, c]

In[47]:= Max[list]

Out[47]= Max[79, b, c]
```

Cvičenie 4.13

mocniny = {16, 9, 4, 1, 0, 1, 4, 9, 16}

Vytvorte zoznam, v ktorom prvý prvok bude minimálne číslo a druhý prvok maximálne číslo zo zoznamu **mocniny**.

Pomocou funkcie **Reverse** vieme usporiadať prvky zoznamu v opačnom poradí.

```
In[48]:= list = {25, {55, 7, 8}, b, c, 60, 79}

In[49]:= Reverse[list]

Out[49]= {79, 60, c, b, {55, 7, 8}, 25}
```

Všimnite si, že poradie prvkov vnútorného zoznamu zostalo nezmenené. Pôvodný zoznam **list** sa však nemení.

```
In[50]:= list

Out[50]= {25, {55, 7, 8}, b, c, 60, 79}
```

Cvičenie 4.14

kosinus = {1, -1, {a, b, 1}, -1, 1}

Zameňte poradie prvkov podzoznamu na tretej pozícii v zozname **kosinus**, takým spôsobom, aby sa zmenil aj pôvodný zoznam **kosinus**.

Ďalšou funkciou, ktorá sa využíva pri práci so zoznamami, najmä v štatistike, ale aj v iných oblastiach, je funkcia **Sort**. Táto funkcia usporiada prvky zoznamu podľa nasledujúcich pravidiel. Ak sú tieto prvky čísla, tak sú v novovytvorenom zozname usporiadané vždy vzostupne. Ak sú to symboly, tak sú usporiadané v abecednom poradí. A ak sú niektoré prvky čísla a niektoré symboly, tak sú najskôr čísla vo vzostupnom poradí a za nimi symboly v abecednom poradí. Za nimi nasledujú podzoznamy zoradené podľa dĺžky. Ich prvky však zostávajú v pôvodnom poradí.

```
In[51]:= Sort[list]

Out[51]= {25, 60, 79, b, c, {55, 7, 8}}
```

Cvičenie 4.15

kosinus = {1, -1, {a, b, 1}, -1, 1}

Zoraďte prvky v zozname **kosinus**.

Ak sa niektoré hodnoty vyskytujú v zozname opakovane, môžeme duplicitné prvky odstrániť pomocou funkcie **DeleteDuplicates[list]**. Pôvodný zoznam zostáva nezmenený.

```
In[52]:= list3 = {5, up, 2, down, up, down, 2, 3, 4, up, 3, up, 7, 3, 1};
         DeleteDuplicates[list3]
         list3

Out[52]= {5, up, 2, down, 3, 4, 7, 1}
         {5, up, 2, down, up, down, 2, 3, 4, up, 3, up, 7, 3, 1}
```

Cvičenie 4.16

mocniny = {16, 9, 4, 1, 0, 1, 4, 9, 16}

Odstráňte duplicitné prvky zo zoznamu **mocniny**.

Pomocou funkcie **Position[list,y]** vieme zistiť, na ktorých pozíciách sa nachádza v zozname **list** prvok s hodnotou **y**.

```
In[53]:= Position[list3, up]

Out[53]= {{2}, {5}, {10}, {12}}
```

Cvičenie 4.17

mocniny = {16, 9, 4, 1, 0, 1, 4, 9, 16}

Koľké najmenšie číslo v zozname **mocniny** je číslo 9? Čísla s rovnakou hodnotou sú v tomto príklade považované za to isté číslo.

Pomôcka: Použite funkcie **DeleteDuplicates**, **Sort** a **Position**.

Funkcia **Count[list,y]** vracia ako výstup počet pozícií, na ktorých sa nachádza v zozname **list** prvok s hodnotou **y**.

```
In[54]:= Count[list3, up]

Out[54]= 4
```

Funkcia **MemberQ[list,y]** vracia pomocou **True** a **False** informáciu o tom, či sa v zozname **list** nachádza prvok s hodnotou **y**.

```
In[55]:= MemberQ[list3, up]
         MemberQ[list3, upo]

Out[55]= True
         False
```

Na zoznamy sa dá pozerať aj ako na matematické množiny. Softvér Mathematica má vstavané funkcie súvisiace s množinovými operáciami. Funkcia **Union[mnozina$_1$, mnozina$_2$]** (z angl. zjednotenie) vytvorí zoznam obsahujúci zjednotenie prvkov množín **mnozina$_1$** a **mnozina$_2$**. Usporiadanie prvkov novovytvoreného zoznamu je také isté ako pri funkcii **Sort**.

```
In[56]:= mnozina1 = {1, 2, 3, 4};
         mnozina2 = {4, 2, 1, 6, -1};
         Union[mnozina1, mnozina2]

Out[56]= {-1, 1, 2, 3, 4, 6}
```

Funkcia **Intersection[mnozina$_1$, mnozina$_2$]** (z angl. prienik) vytvorí zoznam obsahujúci prienik prvkov množín **mnozina$_1$** a **mnozina$_2$**. Usporiadanie prvkov novovytvoreného zoznamu je také isté ako pri funkcii **Sort**.

```
In[57]:= Intersection[mnozina1, mnozina2]

Out[57]= {1, 2, 4}
```

Cvičenie 4.18

```
mocniny = {16, 9, 4, 1, 0}
kosinus = {1, -1, {a, b, 1}, -1, 1}
```

Vytvorte zoznam obsahujúci prvky, ktoré sú prvkami zoznamu **mocniny** a zároveň aj prvkami zoznamu **kosinus**.

Cvičenie 4.19

Pomocou funkcie **Length** zistite, koľko prvkov má výsledný zoznam z predchádzajúceho cvičenia.

Funkcia **Complement[mnozina$_1$, mnozina$_2$]** (z angl. doplnok) vytvorí zoznam obsahujúci prvky množiny **mnozina$_1$**, ktoré sa nenachádzajú v množine **mnozina$_2$**.

```
In[58]:= mnozina1 = {1, 2, 3, 4};
         mnozina2 = {4, 2, 1, 6, -1};
         Complement[mnozina1, mnozina2]
         Complement[mnozina2, mnozina1]

Out[58]= {3}

Out[59]= {-1, 6}
```

4.3 Vytváranie zoznamov pomocou vstavaných funkcií

V tejto časti si ukážeme, ako vytvárať nové zoznamy pomocou vstavaných funkcií. Prvá funkcia, ktorú si predstavíme, je **RandomReal[{x$_{min}$,x$_{max}$},n]** (z. angl. náhodné, reálne), ktorá vygeneruje zoznam **n** náhodných reálnych čísel z intervalu od **x$_{min}$** po **x$_{max}$**.

```
In[60]:= RandomReal[{-3, 2.5}, 4]

Out[60]= {-0.131144, 0.692681, -2.86273, -1.17733}
```

V prípade, že vynecháme druhý vstupný parameter, tak funkcia **RandomReal** vracia jedno reálne číslo. V prípade, že druhým vstupným parametrom je zoznam celých čisel **{n$_1$,n$_2$,...}**, tak funkcia vracia zoznam zoznamov s rozmerom **n$_1$**×**n$_2$**×...

```
In[61]:= RandomReal[{-1, 2}, {3, 2}]

Out[61]= {{1.3625, 1.33724}, {0.90865, 1.6254}, {-0.992239, 1.15147}}
```

Obdobnou funkciou k funkcii **RandomReal** je funkcia **RandomInteger** (z. angl. náhodné, celé), ktorá funguje rovnako, avšak namiesto náhodných reálnych čísel vracia náhodné celé čísla.

Cvičenie 4.20

Vytvorte maticu veľkosti 4×5, ktorej prvkami budú náhodné celé čísla z rozsahu od -10 do 10.

Cvičenie 4.21

Pomocou zabudovanej funkcie zistite maximálny prvok matice z predchádzajúceho cvičenia.

Ďalšou vstavanou funkciou je **Range** (z angl. rozsah), ktorá vytvára zoznamy postupných čísel. Môže dostať tri rôzne vstupy. **Range[i_{max}]** vracia ako výstup zoznam celých čísel od **{1,...,i_{max}]}**; **Range[i_{min},i_{max}]** zoznam celých čísel od **{i_{min},...,i_{max}]}**; a **Range[i_{min},i_{max}, di]** zoznam týchto čísel s krokom **di**.

```
In[62]:= cislaPo = Range[10]

Out[62]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In[63]:= cislaOdPo = Range[-3, 7]

Out[63]= {-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7}

In[64]:= cislaOdPoKrok = Range[-3, 7, 0.5]

Out[64]= {-3., -2.5, -2., -1.5, -1., -0.5, 0., 0.5, 1., 1.5,
          2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7.}
```

Všimnite si, že výstupom je zoznam reálnych čísel a nie zlomkov. Keďže deliaci krok 0.5 je reálne číslo, tak všetky čísla výstupu sú zapísané ako reálne čísla. Ak chceme, aby čísla výstupu boli zapísané ako zlomky, musia byť všetky vstupné parametre buď celé čísla (bez desatinnej bodky), alebo zlomky.

```
In[65]:= cislaOdPoKrok = Range[-3, 7, 1/2]

Out[65]= {-3, -5/2, -2, -3/2, -1, -1/2, 0, 1/2, 1, 3/2,
          2, 5/2, 3, 7/2, 4, 9/2, 5, 11/2, 6, 13/2, 7}
```

Cvičenie 4.22

Pomocou funkcie **Range** a násobenia zoznamov vytvorte zoznam druhých mocnín čísel z rozsahu od -10 do 10 s deliacim krokom 0.5.

Niektoré funkcie dokážu vytvárať nové zoznamy, ak ako vstupný parameter dostanú zoznam (ide väčšinou o matematické funkcie napr. **Sin**, **Log**, **Sqrt**).

```
In[66]:= Sin[Range[0, 5]]

Out[66]= {0, Sin[1], Sin[2], Sin[3], Sin[4], Sin[5]}
```

Výstupom je funkcia aplikovaná na prvky vstupného zoznamu.

Poznámka 4.1

Nie všetky funkcie dokážu ako vstupný argument prijať zoznam a vrátiť zoznam výstupov funkcie aplikovanej na prvky vstupného zoznamu. Funkcie s touto vlastnosťou sa nazývajú `Listable`. Či je nejaká funkcia `Listable` alebo nie, sa dá zistiť pomocou funkcie **`Attributes`**(z angl. atribúty).

```
In[67]:= Attributes[Sin]

Out[67]= {Listable, NumericFunction, Protected}
```

Výstupom funkcie **`Attributes`** je zoznam atribútov (vlastností) vstavanej funkcie, ktorá je jej vstupným argumentom.

Najvšeobecnejšia funkcia na vytváranie zoznamov je funkcia **`Table[vyraz,{i,i_min,i_max,di}]`**, kde **`vyraz`** je predpis, v ktorom sa premenná **`i`** postupne mení od **`i_min`** po **`i_max`** s krokom **`di`**.

```
In[68]:= Table[i^2, {i, -2, 2, 0.5}]

Out[68]= {4., 2.25, 1., 0.25, 0., 0.25, 1., 2.25, 4.}
```

Vstup **`di`** nie je povinný. V prípade, že ho vynecháme, je jeho prednastavená hodnota 1. V prípade, že vynecháme **`i_min`**, jeho hodnota je tiež 1.

Cvičenie 4.23

Pomocou funkcií **`Range`** a **`Table`** vytvorte zoznam dĺžky 10, ktorého prvkami budú zoznamy celých čísel od 1 po `n`, kde `n` je poradie prvku v zozname. Napr. pre zoznam dĺžky 3 dostaneme **`{{1},{1,2},{1,2,3}}`**.

Cvičenie 4.24

Nahraďte prvý prvok v každom podzozname výsledného zoznamu z predchádzajúceho cvičenia číslom 0.

Ak chceme vytvárať zoznam zoznamov, môžeme ako vstupný argument funkcie **`Table`** použiť ďalšiu funkciu **`Table`**.

```
In[69]:= Table[Table[i/j, {j, 1, 2}], {i, 1, 3}]

Out[69]= {{1, 1/2}, {2, 1}, {3, 3/2}}
```

Keďže vytváranie vnorených zoznamov sa často používa, existuje v softvéri Mathematica funkcia **`Table`** aj v tvare **`Table[vyraz,{i,i_min,i_max,di},{j,j_min,j_max,dj},...]`**, ktorá funguje rovnako ako vnorenie **`Table`** do **`Table`**.

```
In[70]:= Table[i/j, {i, 1, 3}, {j, 1, 2}]

Out[70]= {{1, 1/2}, {2, 1}, {3, 3/2}}
```

Všimnite si, že poradie zoznamov reprezentujúcich rozsah indexov `i` a `j` je v 2-indexovej verzii `Table` opačné v porovnaní s vnorenou 1-indexovou verziou `Table` do 1-indexovej verzie `Table`. Vo *viac*-indexovej verzii `Table` zoznam napísaný úplne vpravo prislúcha najvnútornejšiemu zoznamu.

Cvičenie 4.25

Zadajte do vstupnej bunky príkaz pomocou funkcie `Table`, ktorého výstupom bude zoznam `{{1}, {2, 2}, {3, 3, 3}, {4, 4, 4, 4}, {5, 5, 5, 5, 5}}`.

Vo väčšine programovacích jazykov sa najskôr vypočítajú vstupy do funkcie a až potom sa zavolá samotná funkcia už s konkrétnymi číslami. Napr., ak by sme mali zadefinované `i=1,j=1` a potom by sme zavolali predchádzajúcu funkciu `Table`, tak by sme v skutočnosti volali príkaz `Table[1, {1, 1, 3}, {1, 1, 2}]`. Pre vyvarovanie sa takýchto problémov má softvér funkcie, ktorých vstupné parametre sa ponechajú v tvare symbolu (t. j. nenahradia sa hodnotami). Tieto symboly sa potom používaju ako lokálne premenné.

Poznámka 4.2

Softvér Mathematica rozlišuje dva typy premenných: globálne a lokálne. Premenné, s ktorými sme sa doteraz stretávali v tejto učebnici sú nazývané globálne. Ich názov je odvodený z toho, že sú prístupné a použiteľné globálne, t. j. na ľubovoľnom mieste programu počas jeho behu. Pojmom "lokálne premenné" označujeme premenné definované vnútri funkcií, ktoré sa používajú len lokálne, t. j. vnútri funkcií, v ktorých sú definované. Ich zmena neovplyvní globálne premenné s rovnakým názvom a ani hodnoty týchto globálnych premenných nemajú žiadny vplyv na lokálne premenné. S lokálnymi premennými sa stretneme aj pri funkciách `Module` a `Block`, pozri sekciu 8.3.

Vďaka tomu zostane v nasledujúcom príklade hodnota `i` nezmenená.

```
In[71]:= i = 10;
         Table[i, {i, 1, 3}]
         i

Out[71]= {1, 2, 3}
         10
```

V predchádzajúcom príklade je `i` vnútri funkcie `Table` lokálnou premennou. Zatiaľ čo `i` pred funkciou `Table` je globálna premenná. Ak majú lokálna aj globálna premenná rovnaký názov, tak platí, že lokálna premenná má vnútri svojej funkcie prednosť pred globálnou.

Poznámka 4.3

Vlastnosť držať vstupné parametre nevyhodnotené sa v softvéri Mathematica označuje výrazom `Hold` (z angl. držať). Vstavané funkcie môžu nadobúdať vlastnosti: `HoldAll` (z angl. držať všetko), `HoldFirst` (z angl. držať prvý), `HoldRest` (z angl. držať zvyšok). `HoldAll` drží všetky vstupné parametre nevyhodnotené. `HoldFirst` len prvý a `HoldRest` všetky okrem prvého. Vlastnosti jednotlivých funkcií sa dajú zistiť pomocou funkcie **`Attributes`**.

```
In[72]:= Attributes[Table]

Out[72]= {HoldAll, Protected}
```

Poznámka 4.4

Funkcia **`Table`** má vstupné parametre v tvare **`Table[vyraz,{i,`i_{min}`,`i_{max}`}]`**, t. j. prvý parameter je výraz, ktorý funkcia vyhodnocuje. Druhý parameter je zoznam premennej, dolnej hranice a hornej hranice. S takýmto štýlom zadávania vstupných parametrov sa stretneme aj pri iných funkciách, s ktorými sa neskôr oboznámime. Napr. funkcia **`Plot[f,{x,`**x_{min}**`,`**x_{max}**`}]`** slúžiaca na vykreslenie grafu funkcie **`f`** podľa hodnoty premennej **`x`** od x_{min} po x_{max} dostáva parametre v rovnakom tvare (pozri sekciu 5.1), taktiež aj funkcia **`Integrate[f,{x,`**x_{min}**`,`**x_{max}**`}]`**, ktorá vypočíta určitý integrál predpisu **`f`** na danom intervale (pozri podsekciu 10.2.1).

4.4 Zhrnutie

V tejto kapitole sme si predstavili zoznamy, ktoré sa zapisujú pomocou krútených zátvoriek **`{}`**. K ich prvkom sa pristupuje pomocou dvojitých hranatých zátvoriek **`[[]]`**. Tie môžeme použiť aj na prístup k podzoznamu nejakého zoznamu. Rozmery zoznamu vieme zistiť pomocou funkcie **`Length`** alebo **`Dimensions`**.

So zoznamami sa dajú vykonávať základné matematické operácie. Funkcie **`Append`**, **`Prepend`** a **`Insert`** slúžia na vkladanie nových prvkov do zoznamu. Nové zoznamy prvkov sa dajú generovať podľa nejakého predpisu pomocou funkcie **`Table`**. V kapitole 8 si predstavíme ďalší spôsob vytvárania zoznamov pomocou funkcie **`Map`** a v kapitole 7 pomocou nahrádzania a vzorov.

V nasledujúcich kapitolách sa so zoznamami budeme stretávať často. V kapitole 5 si ukážeme, ako zobraziť graf zo zoznamu čísel a v kapitole 12 budeme k zoznamom pristupovať ako k vektorom a maticiam.

Kapitola 5

Grafické zobrazovanie matematických funkcií

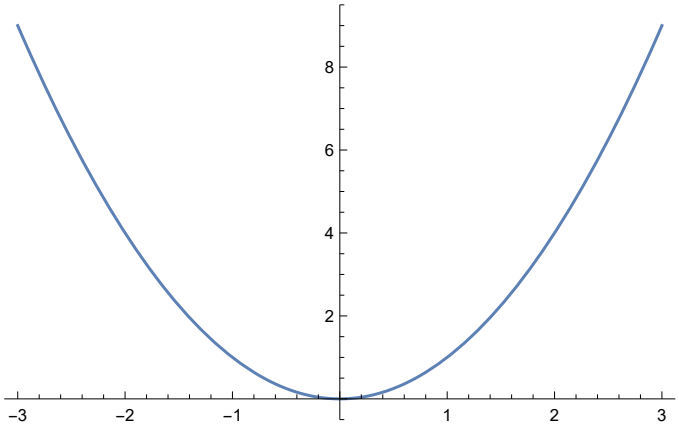
Softvér Mathematica má v sebe zabudovanú grafiku vysokej kvality. Jej ovládanie je jednoduché a značne intuitívne. Vďaka širokej palete grafických volieb umožňuje veľmi precízne vykresľovať požadované vlastnosti grafov. Z týchto dôvodov si študenti často volia práve softvér Mathematica na grafické zobrazovanie výsledkov vo svojich záverečných prácach aj v prípadoch, keď boli riešené v iných programátorských prostrediach. Grafika tohto softvéru je taktiež vhodným zobrazovacím nástrojom pre komerčné úlohy z praxe. Svoje uplatnenie nachádza aj v pedagogickom procese pri osvojovaní si poznatkov o funkciách – zásluhou názornosti pomáha vytvárať u študentov lepšiu predstavu.

5.1 Funkcia Plot

Základnou vstavanou funkciou na vykresľovanie grafov matematických funkcií je funkcia s jednoduchým názvom `Plot` (po angl. zobraz). Jej syntax je nasledovná: `Plot[f,{x,xmin,xmax}]`, kde `f` je predpis funkcie jednej premennej, ktorej graf chceme dať vykresliť, `x` je premenná tejto funkcie a x_{min} a x_{max} sú hranice oblasti vykreslenia. Predpis funkcie môžeme zadať do `Plot` priamo alebo pomocou premennej, ktorej je priradený tento predpis (podrobnejšie o tom v kapitole 8). V obidvoch prípadoch softvér Mathematica zobrazí rovnaký graf.

```
In[1]:= Plot[x^2, {x, -3, 3}];
        kvad[x_]:= x^2
        Plot[kvad[x], {x, -3, 3}]
```

Out[1]=



Rovnako ako funkcia **Table** aj funkcia **Plot** drží vstupy nevyhodnotené, pozri Pozn. 4.3 a text nad ňou. Vstupný parameter **x** je teda lokálna premenná. To znamená, že nezáleží na tom, či sme predtým používali premennú s rovnakým názvom **x**, jej hodnota nijako neovplyvní vstupný parameter funkcie **Plot**.

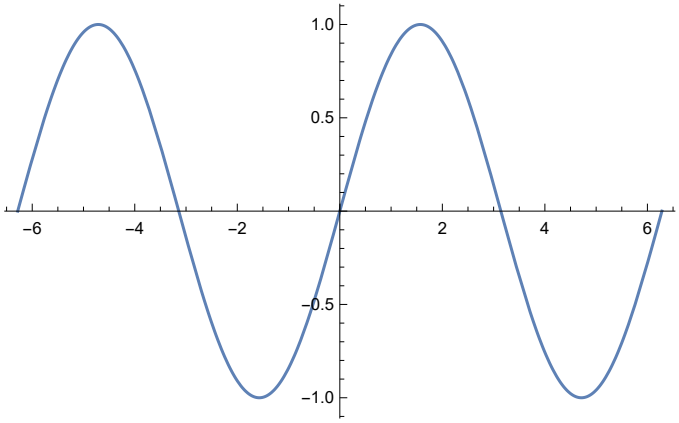
Poznámka 5.1

Upozorňujeme, že pre potreby učebnice sme niektoré grafické výstupy softvéru Mathematica veľkostne zmenšili.

Namiesto symbolu **x** reprezentujúceho premennú funkcie, môžeme použiť ľubovoľný symbol.

```
In[2]:= Plot[Sin[z], {z, -2π, 2π}]
```

Out[2]=

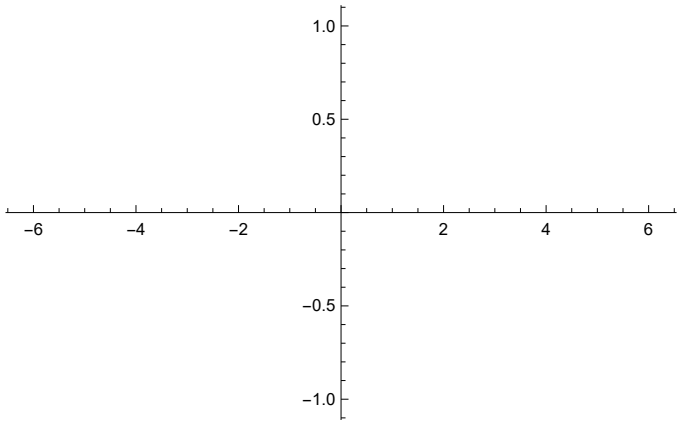


Poznámka 5.2

Je potrebné si uvedomiť, že **f** nie je funkcia z programátorského hľadiska, ale matematický predpis funkcie s premennou **x**. A teda nasledovné volanie funkcie je nesprávne.

```
In[3]:= Plot[Sin, {x, -2π, 2π}]
```

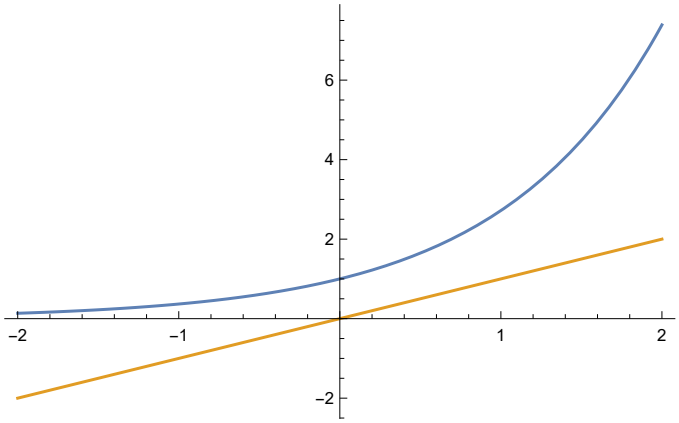
Out[3]=



V ďalšom texte kapitoly budeme občas meniť názov tejto premennej. Aj napriek tomu však horizontálnu os grafu budeme v texte označovať ako os x a vertikálnu ako os y. Prvým vstupným parametrom funkcie **Plot** je predpis funkcie alebo zoznam predpisov funkcií.

In[4]:=
```
Plot[ⅇ^v, {v, -2, 2}];
Plot[{ⅇ^v, v}, {v, -2, 2}]
```

Out[4]=



V druhom prípade sa vykreslia grafy všetkých funkcií zoznamu do jedného obrázku. Každý z nich bude zobrazený inou farbou.

Cvičenie 5.1

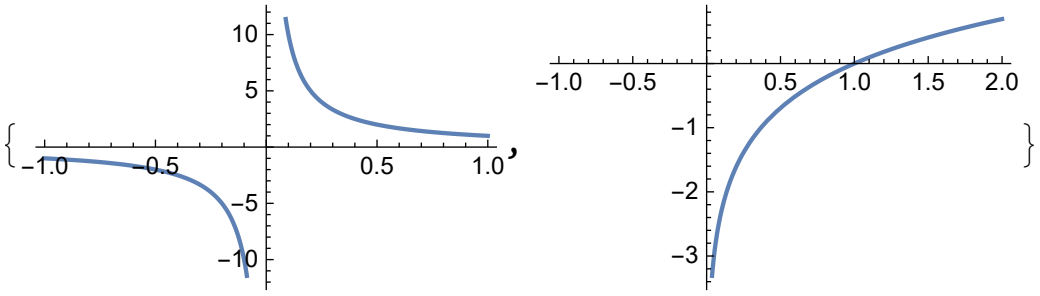
Vykreslite do jedného obrázku grafy funkcií $f(x) = x^i, i = 0, ..., 10$ na oblasti $(-1, 1)$. Vypisovať všetky predpisy funkcií jeden po druhom by bolo veľmi pracné. Využite namiesto toho funkciu **Table**.

Všimnite si, že ak **Table** vložíte priamo do **Plot**, vykreslené grafy funkcií budú mať rovnakú farbu.

Plot vie zobraziť grafy funkcií na daných intervaloch aj v tom prípade, keď funkcie nie sú definované vo všetkých bodoch týchto intervalov.

In[5]:=
```
{Plot[1/x, {x, -1, 1}], Plot[Log[x], {x, -1, 2}]}
```

Out[5]=



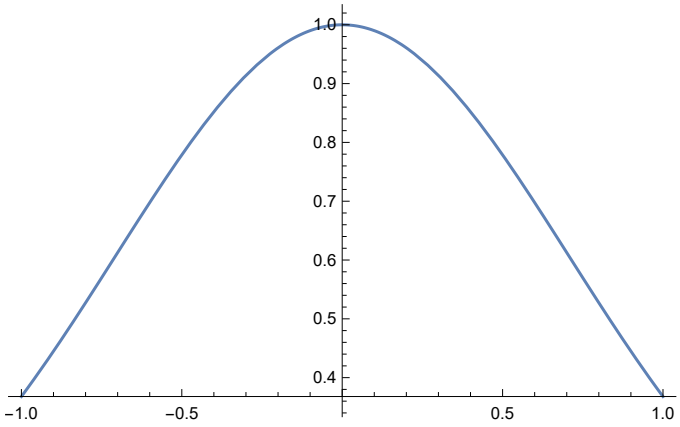
Výstupom funkcie **Plot** je objekt a teda je ho možné uložíť do premennej. Obrázok, ktorý sa zobrazí po zavolaní premennej **plotGaussovaKrivka** je len spôsob, ako softvér Mathematica tento objekt interpretuje.

In[6]:=
```
plotGaussovaKrivka = Plot[ℯ^-x^2, {x, -1, 1}];
```

Keď túto premennú dáme zobraziť, výstupom bude graf funkcie.

In[7]:=
```
plotGaussovaKrivka
```

Out[7]=



5.1.1 Voľby funkcie Plot

V sekcii 2.2 sme sa zmienili, že niektoré vstavané funkcie majú voľby, ktorými môžeme prispôsobiť ich fungovanie.

Povinné vstupné parametre funkcie **Plot** sú len predpis funkcie a ohraničenie oblasti, na ktorej sa má graf funkcie vykresliť. Farba grafu, obor hodnôt, škála a ďalšie charakteristiky boli zvolené automaticky. Tieto vlastnosti sa dajú meniť pomocou tzv. volieb. V tejto podsekcii si niektoré z nich predstavíme, s ďalšími sa oboznámime v kapitole 9.

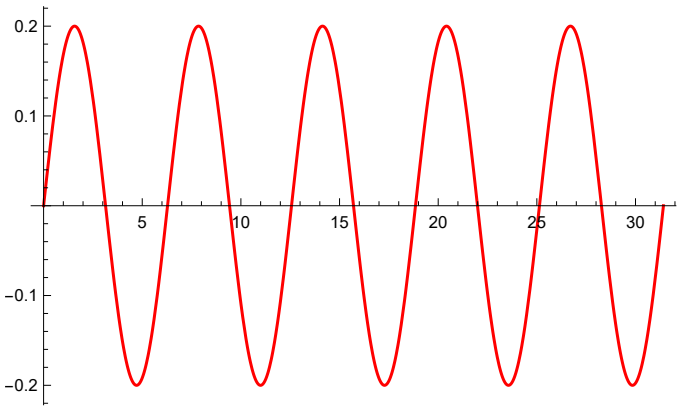
Štýly vykreslenia grafu

Voľba **PlotStyle** slúži na zmenu štýlu vykreslenia grafu funkcie. Umožňuje napr. zadať rôznymi spôsobmi farbu, hrúbku a prerušovanie zobrazovaných grafov funkcií. Oceníme ju najmä pri potrebe rozlišovať viaceré grafy funkcií v jednom obrázku. Farba sa zadáva pomocou **PlotStyle→Farba**.

Najjednoduchší spôsob jej zadania je použitie anglického názvu preddefinovanej farby, napr. červenej (po angl. red).

```
In[8]:= Plot[0.2Sin[u], {u, 0, 10π}, PlotStyle → Red]
```

Out[8]=

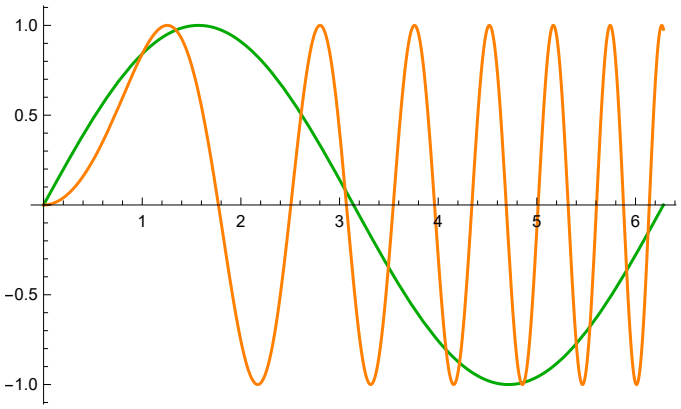


Zoznam všetkých preddefinovaných farieb a charakteristiku jednotlivých vstavaných direktív na vytváranie farieb nájdete v sekcii 9.2.

Ak zobrazujeme zoznam grafov funkcií v jednom obrázku, môžeme ich odlíšiť od seba rôznymi štýlmi, ktoré zadávame ako zoznam štýlov. Prvý štýl prislúcha prvému grafu, druhý druhému,..., posledný poslednému. Nasledujúci príklad reprezentuje rozlíšenie grafov funkcií pomocou farieb.

```
In[9]:= Plot[{Sin[u], Sin[u^2]}, {u, 0, 2π}, PlotStyle→{Darker[Green], Orange}]
```

Out[9]=



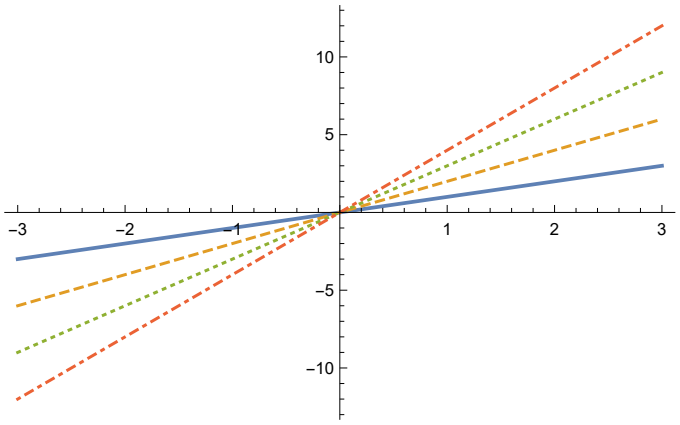
Cvičenie 5.2

Vykreslite do jedného obrázku grafy funkcií $f(x) = \ln(x)$ a $g(x) = \mathrm{e}^x$ na oblasti $(-1, 1)$. Prvý graf vykreslite modrou a druhý zelenou farbou.

V prípade, že pripravujeme obrázok zoznamu grafov funkcií pre čiernobielu tlač, je vhodné voliť štýly, ktoré aj v takejto verzii zabezpečia dostatočnú rozlíšiteľnosť. V nasledujúcom príklade prezentujeme použitie direktív **`Thick`** (z angl. hrubá), **`Dashed`** (z angl. prerušovaná), **`Dotted`** (z angl. bodkovaná) a **`DotDashed`** (z angl. bodkočiarkovaná).

```
In[10]:= Plot[{x, 2x, 3x, 4x}, {x, -3, 3}, PlotStyle → {Thick, Dashed, Dotted,
                                                         DotDashed}]
```

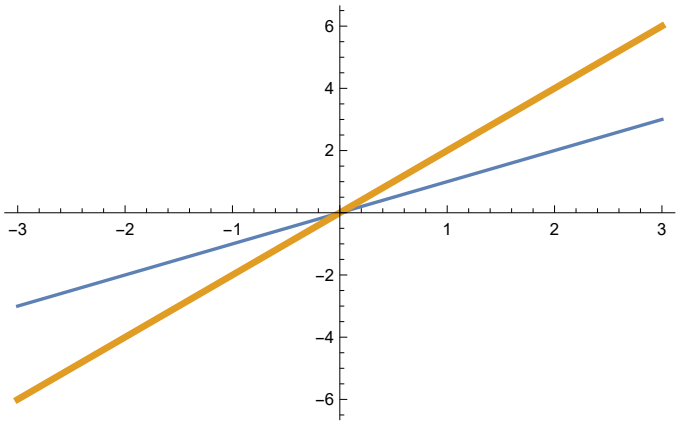
Out[10]=



Rozšírenou verziou direktívy **Thick** je direktíva **Thickness[hrubka]**, ktorá umožňuje pomocou parametra **hrubka** presne nastaviť požadovanú hrúbku čiar.

```
In[11]:= Plot[{x, 2x}, {x, -3, 3}, PlotStyle→{Thickness[0.005], Thickness[0.01]}]
```

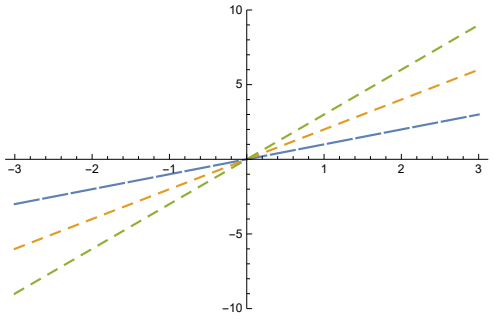
Out[11]=



Podobne, rozšírením direktívy **Dashed** je direktíva **Dashing[{ciarka, medzera}]**, ktorá umožňuje pomocou parametrov **ciarka** a **medzera** nastaviť presne veľkosť čiarky a veľkosť medzery. V prípade, že požadujeme, aby čiarky a medzery boli rovnako dlhé, stačí použiť základnú verziu **Dashing[ciarka]**.

```
In[12]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotStyle → {Dashing[{0.05, 0.01}],
                                                    Dashing[{0.02, 0.02}], Dashing[0.02]}]
```

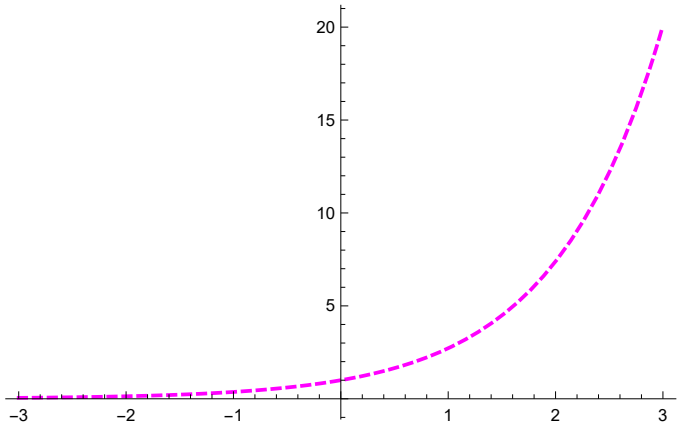
Out[12]=



Ak chceme priradiť jedinému grafu funkcie viacero rôznych štýlov, zadávame ich tiež ako zoznam štýlov.

In[13]:=
```
Plot[Exp[x], {x, -3, 3}, PlotStyle → {Magenta, Thick, Dashed}]
```

Out[13]=



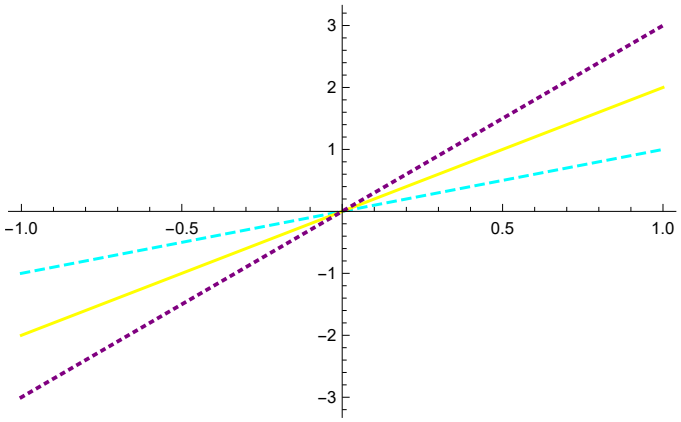
Cvičenie 5.3

Vykreslite graf funkcie $f(x) = cotg(x)$ na oblasti $(-5, 5)$ červenou, čiarkovanou čiarou, ktorá má čiarky dĺžky 0.1 a medzery dĺžky 0.3.

Na poradí jednotlivých direktív v zozname nezáleží. V prípade, že zobrazujeme zoznam grafov funkcií a chceme nastaviť niektorým zobrazovaným grafom viacero štýlov, použijeme zoznam zoznamov direktív.

In[14]:=
```
Plot[{x, 2x, 3x}, {x, -1, 1},
PlotStyle → {{Cyan, Dashed}, Yellow, {Purple, Dotted, Thick}}]
```

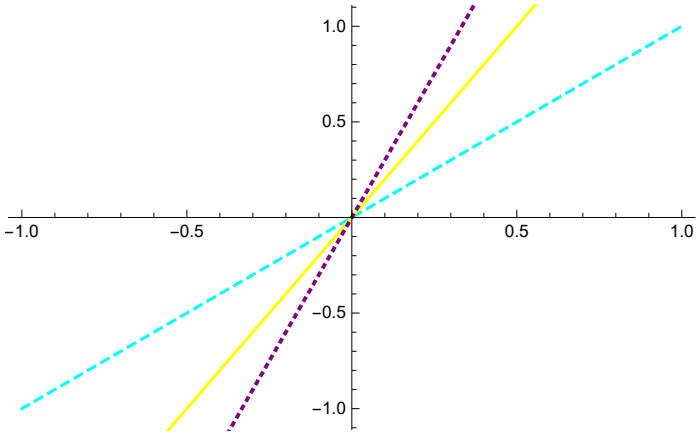
Out[14]=



Viac grafov funkcií vieme zobraziť do jedného obrázku aj pomocou vstavanej funkcie **Show**. Výstup z predchádzajúceho príkladu môžeme vytvoriť aj nasledujúcim spôsobom. Zobrazenia jednotlivých grafov funkcií priradíme do samostatných premenných a zobrazíme ich pomocou funkcie **Show**.

In[15]:=
```
prva = Plot[x, {x, -1, 1}, PlotStyle → {Cyan, Dashed}];
druha = Plot[2x, {x, -1, 1}, PlotStyle → Yellow];
tretia = Plot[3x, {x, -1, 1}, PlotStyle → {Purple, Dotted, Thick}];
Show[tretia, druha, prva]
```

Out[15]=



Na poradí vstupných argumentov funkcie **Show** záleží. Obor hodnôt zobrazenia zoznamu grafov funkcií sa automaticky nastavuje podľa oboru hodnôt grafu prvej funkcie. Pri zobrazovaní zoznamu grafov funkcií do jedného obrázku pomocou funkcie **Plot** sa obor hodnôt zoznamu grafov nastaví podľa grafov všetkých funkcií zoznamu.

Cvičenie 5.4

Vykreslite do jedného obrázku grafy funkcií $f(x) = tg(x)$ a $g(x) = cotg(x)$ na oblasti $(0, 5)$. Nastavte grafom týchto funkcií rôzne farby a rôzne čiarkovanie.

Funkcia **Show** sa používa najmä v situáciách, kde zobrazujeme do jedného obrázku rôzne grafické prvky (výstupy rôznych vstavaných zobrazovacích funkcií). Prospešná je však aj v prípadoch, keď zobrazujeme vyšší počet grafických prvkov pomocou rovnakej zobrazovacej funkcie (**Plot**, **Graphics**).

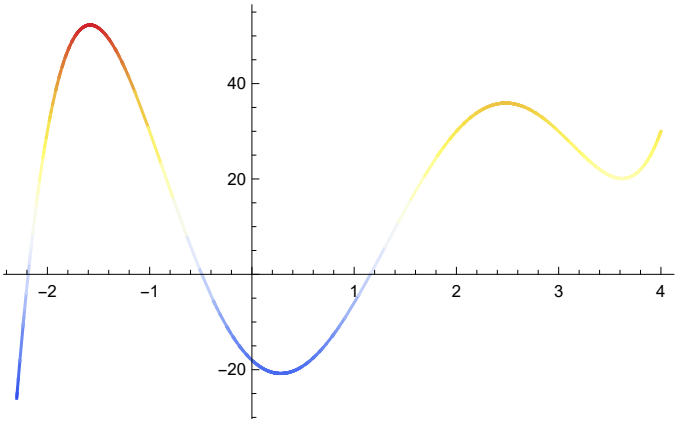
Každý grafický prvok môžeme zadefinovať samostatne a uložiť ho do pomocnej premennej. Potom všetky grafické prvky zobrazíme do jedného obrázku pomocou funkcie **Show**. Získame tak lepšiu prehľadnosť zadania ako v situácii, keď sú všetky grafické prvky zobrazené len pomocou jedinej zobrazovacej funkcie.

Posledná dírektíva voľby **PlotStyle**, s ktorou sa budeme bližšie zaoberať je direktíva **Opacity** (z angl. nepriehľadnosť). Využíva sa najmä v situáciách, keď sa zobrazované grafy funkcií čiastočne prekrývajú (napr. v obrázku zobrazujúcom graf funkcie a graf jej Taylorovho rozvoja, pozri str. 160). Zadáva sa v tvare **Opacity[nepriehladnost]**, kde parameter **nepriehladnost** nadobúda hodnoty z intervalu $(0, 1)$. Hodnota 1 reprezentuje úplnú nepriehľadnosť (rovnaká situácia, ako keby sme vôbec nepoužili direktívu **Opacity**) a hodnota 0 úplnú priehľadnosť (t. j. zobrazovaná funkcia nie je viditeľná).

Graf zobrazovanej funkcie nemusí mať na celom priebehu rovnakú farbu. Pomocou voľby **ColorFunction** vieme rôznymi farbami rozlíšiť jeho jednotlivé časti v závislosti od hodnôt, ktoré funkcia nadobúda.

```
In[16]:= Plot[x^5 - 6x^4 + x^3 + 36x^2 - 20x - 18, {x, -2.3, 4},
          ColorFunction → "TemperatureMap"]
```

Out[16]=



Zoznam nastavení farebných schém voľby **ColorFunction** nájdete, keď v hornom menu kliknete na **Help** → **Find Selected Function** a do okienka na vyhľadávanie zadáte výraz "Color Schemes". Ako si zadefinovať svoje vlastné farebné schémy si ukážeme v Pozn. 9.5 v sekcii 9.2, ktorá sa venuje rôznym farebným modelom.

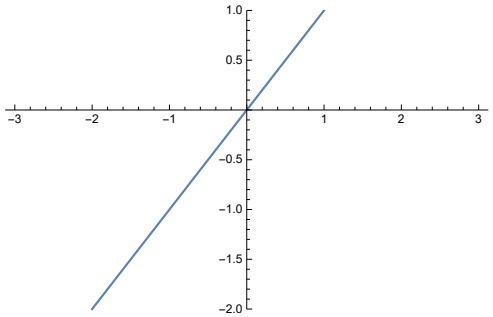
Štýly vykreslenia oblasti

Doteraz sme ani v jednom príklade nedefinovali rozsah vykreslenia grafu funkcie na y-ovej osi. Funkcia **Plot** má v sebe naprogramovaný automatický výber všetkých volieb. Pre prácu užívateľa je väčšinou tento automatický výber postačujúci. Napr., pre funkciu $f(x) = 1/x$ nám nevykreslí všetky hodnoty na intervale $(-1, 1)$, keďže tie by boli z rozsahu $(-\infty, \infty)$.

Ak nám však prednastavený výber nevyhovuje, môžeme funkcii **Plot** zadať hranice zobrazenia pre y-ovú os pomocou voľby **PlotRange→{y$_{min}$, y$_{max}$}**, kde **y$_{min}$** a **y$_{max}$** sú hranice rozsahu vykreslenia na y-ovej osi.

```
In[17]:= Plot[z, {z, -1, 1}, PlotRange → {-2, 1}]
```

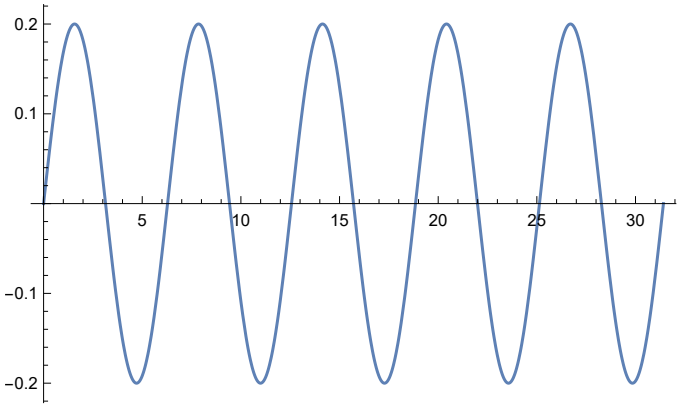
Out[17]=



Pomer veľkosti výšky a šírky obrázku grafu je tiež volený automaticky.

```
In[18]:= Plot[0.2Sin[u], {u, 0, 10π}]
```

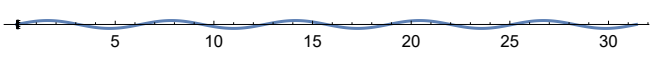
Out[18]=



Takýto graf môže síce delailnejšie geometricky znázorňovať funkciu, ale niekedy podáva mylnú informáciu o hodnotách funkcie a tvare jej grafu, keďže dĺžka dielikov na x-ovej a y-ovej osi môže byť rôzna. V prípade, že požadujeme, aby vykreslený graf nebol zdeformovaný (t. j. aby dieliky na osi x aj osi y boli rovnako veľké), možeme ho zmeniť pridaním voľby **AspectRatio→Automatic**.

```
In[19]:= Plot[0.2Sin[u], {u, 0, 10π}, AspectRatio → Automatic]
```

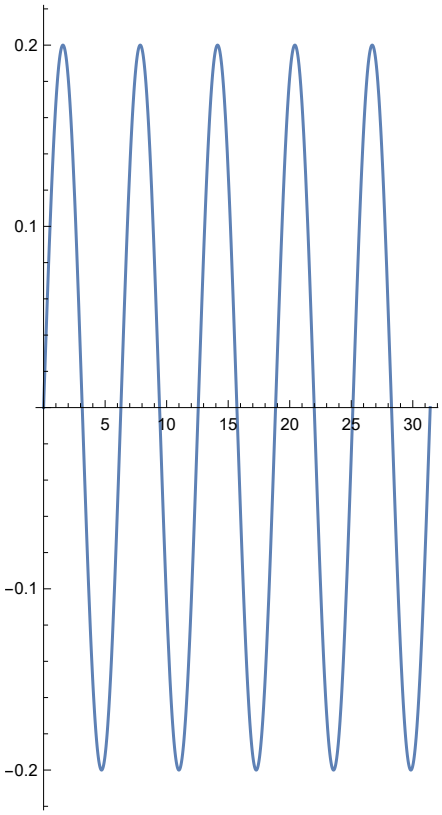
Out[19]=



Ak by bol pre našu prácu vhodnejší iný pomer, možeme nahradiť nastavenie **Automatic** parametrom **k**, ktorý označuje pomer výšky a šírky obrázku.

```
In[20]:= Plot[0.2Sin[u], {u, 0, 10π}, AspectRatio → 2]
```

Out[20]=

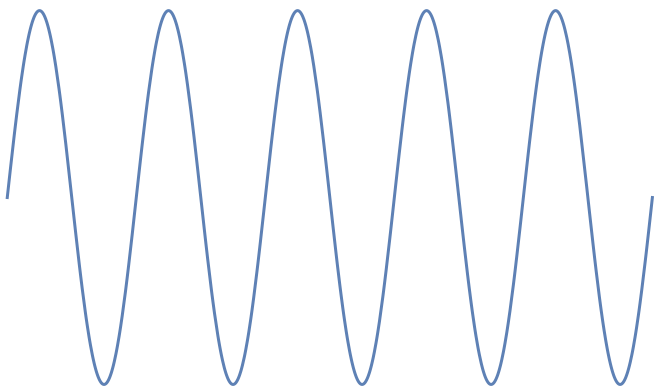


Štýly vykreslenia osí

Vykreslenie osí sa dá vypnúť pomocou voľby **Axes**.

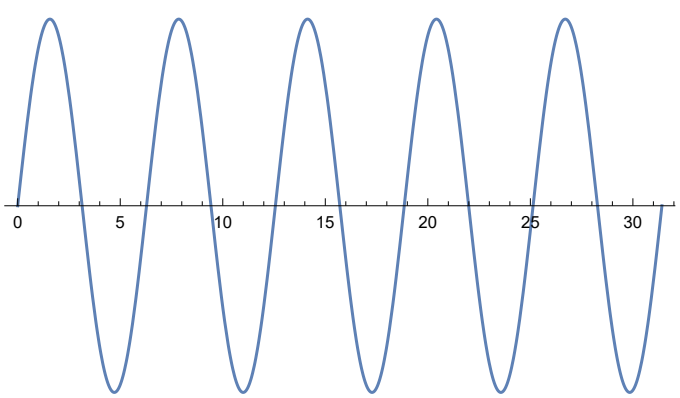
In[21]:=
```
Plot[0.2Sin[u], {u, 0, 10π}, Axes → False]
```

Out[21]=



In[22]:=
```
Plot[0.2Sin[u], {u, 0, 10π}, Axes → {True, False}]
```

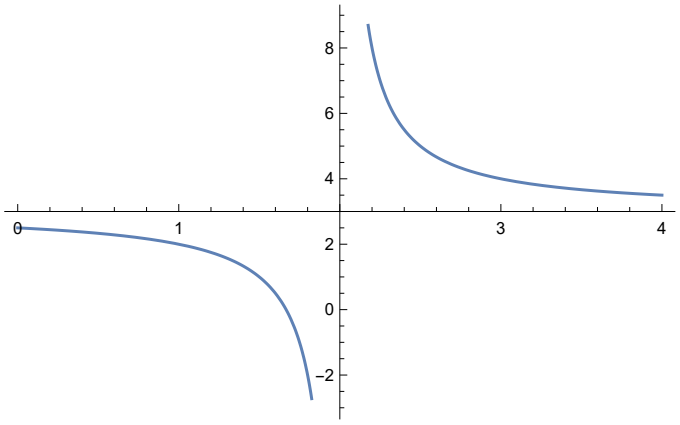
Out[22]=



Niekedy je žiadúce posunúť začiatok súradnicovej sústavy do nejakého iného bodu. Na tento účel sa používa voľba **AxesOrigin**, ktorá napravo od šípky dostáva zadaný nový bod prieniku osí.

In[23]:=
```
Plot[1/(x - 2) + 3, {x, 0, 4}, AxesOrigin → {2, 3}]
```

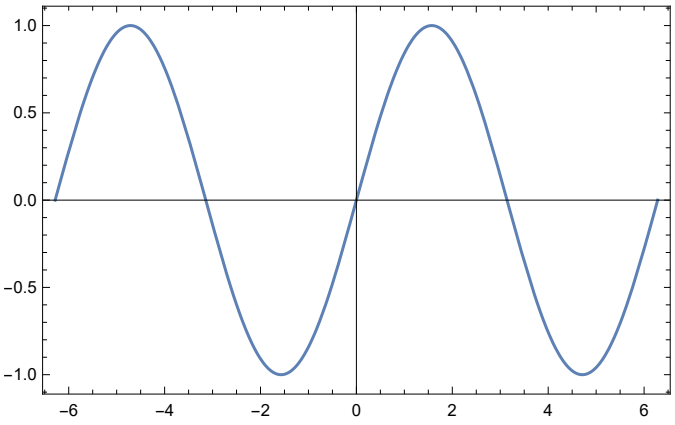
Out[23]=



Okolo grafu môžeme vytvoriť rám (po angl. frame) voľbou **Frame** s nastavením **True**.

In[24]:=
```
Plot[Sin[x], {x, -2Pi, 2Pi}, Frame → True]
```

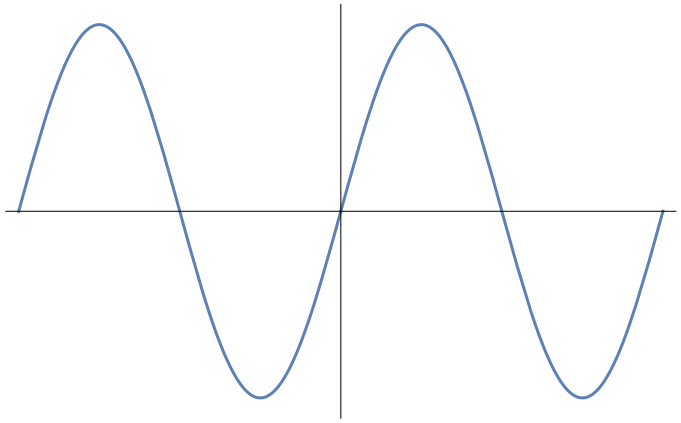
Out[24]=



V prípade, že potrebujeme zobraziť osi súradnicovej sústavy bez čiarok (po angl. ticks) na nich, použijeme voľbu **Ticks → None**.

```
In[25]:= Plot[Sin[x], {x, -2Pi, 2Pi}, Ticks → None]
```

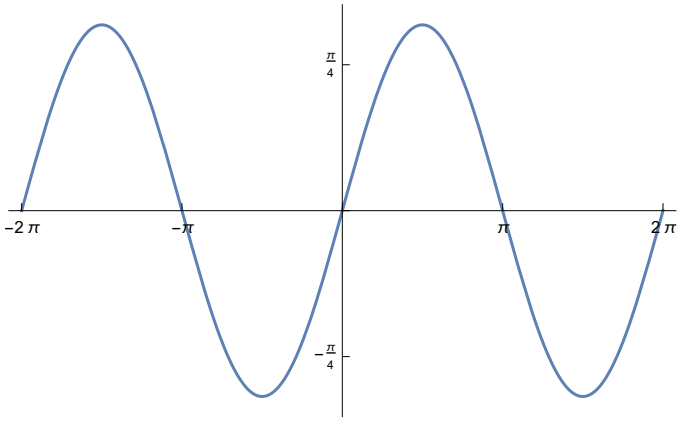
Out[25]=



Čiarky sa na súradnicových osiach zobrazujú automaticky a reprezentujú na nich celé čísla, prípadne ich polovice alebo desatiny. Niekedy je však vhodnejšie mať vyznačené čiarky, ktoré viac zodpovedajú grafu zobrazovanej funkcie, napr. pri goniometrických funkciách vykresliť čiarky prislúchajúce hodnotám čísla π. Nasledujúci príklad znázorňuje zadanie vlastných dielikov na osi x aj osi y.

```
In[26]:= Plot[Sin[x], {x, -2Pi, 2Pi}, Ticks → {{-2Pi, -Pi, 0, Pi, 2Pi},
                                               {-Pi/4, 0, Pi/4}}]
```

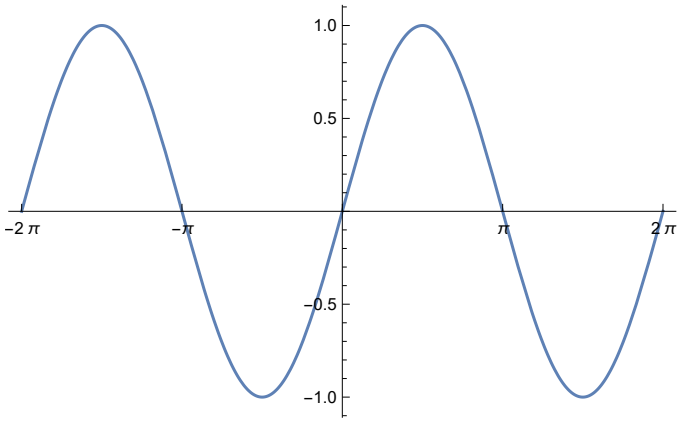
Out[26]=



Prípadne môžeme zadať len čiarky na osi x a na osi y ich zobrazí softvér automaticky.

```
In[27]:= Plot[Sin[x], {x, -2Pi, 2Pi}, Ticks → {{-2Pi, -Pi, 0, Pi, 2Pi},
                                               Automatic}]
```

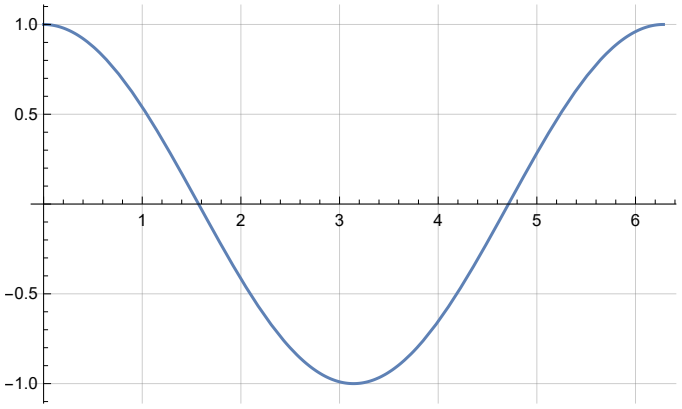
Out[27]=



Pri riešení niektorých úloh je výhodné zobraziť graf s mriežkou. Jednoduchšie sa na ňom sledujú zmeny funkcie, prípadne čítajú z neho hodnoty vo vybraných bodoch. Keď pridáme voľbu **GridLines** (z angl. mriežkové čiary) s nastavením **Automatic**, tak sa na grafe vytvorí mriežka. Pozície jej čiar sú preddefinované v softvéri, zodpovedajú automatickému zobrazovaniu čiarok na osiach grafu.

In[28]:=
```
Plot[Cos[x], {x, 0, 2Pi}, GridLines → Automatic]
```

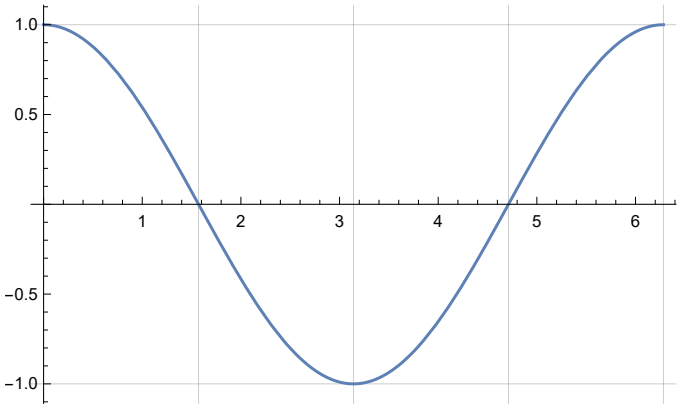
Out[28]=



Vlastné nastavenie pozícií čiar sa zadáva rovnakým spôsobom ako pri voľbe **Ticks**.

In[29]:=
```
Plot[Cos[x], {x, 0, 2Pi}, GridLines → {{0, Pi/2, Pi, 3Pi/2, 2Pi},
                                       {-1, 0, 1}}]
```

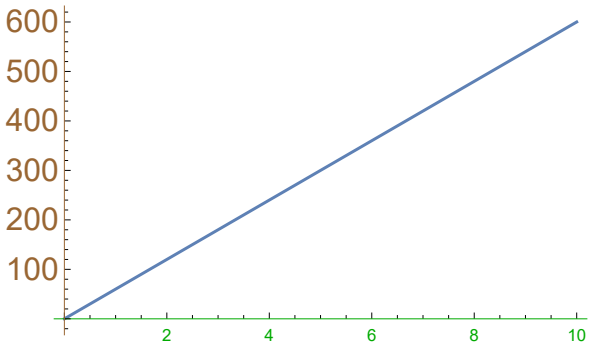
Out[29]=



Keď používame voľby **GridLines**, **Frame** alebo **Axes**, môžeme pomocou volieb **GridLinesStyle**, **FrameStyle** alebo **AxesStyle** zadať štýl zobrazenia mriežky, rámu alebo osí. Tieto voľby využívajú rovnaké direktívy ako voľba **PlotStyle** a naviac umožňujú nastaviť aj veľkosť číslic na osiach a ráme a veľkosť písma v popise osí.

```
In[30]:= Plot[60t, {t, 0, 10},
          AxesStyle → {Darker[Green], {Brown, 17}}]
```

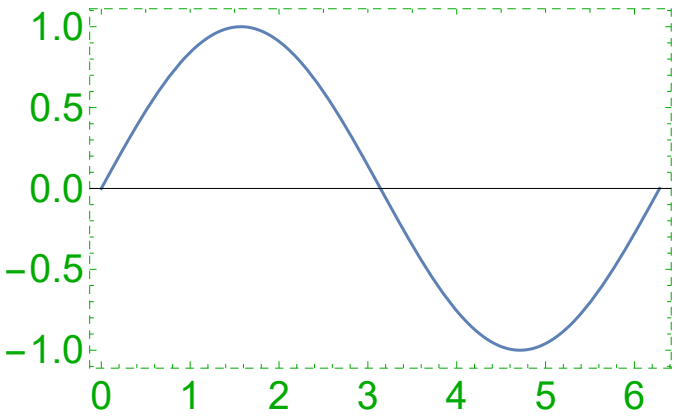
Out[30]=



Zoznam direktív, napr. **{Brown, 17}**, je možné zadať aj pomocou **Directive[Brown, 17]**.

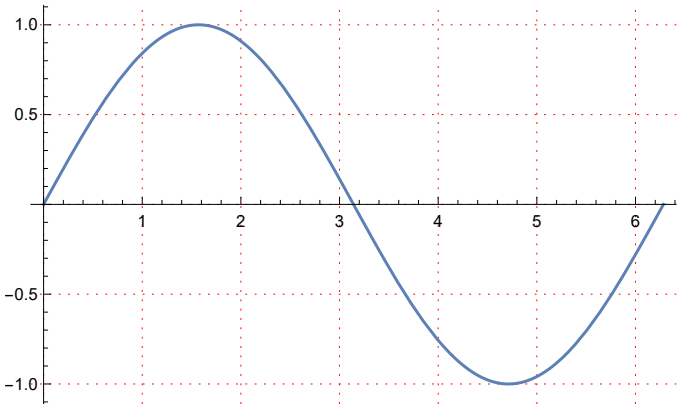
```
In[31]:= Plot[Sin[x], {x, 0, 2Pi}, Frame → True,
          FrameStyle → Directive[Darker[Green], Dashed, 21]]
```

Out[31]=




```
In[32]:= Plot[Sin[x], {x, 0, 2Pi}, GridLines → Automatic,
          GridLinesStyle → Directive[Dotted, Red]]
```

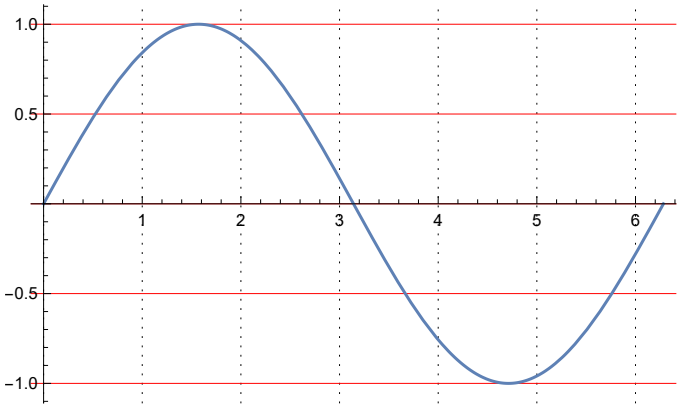
Out[32]=



Upozorňujeme, že ak by sme v predposlednom a v poslednom príklade použili namiesto **Directive** zoznam direktív, neboli by výstupy rovnaké. Ak by bol zadaný zoznam dvoch volieb, prvá by bola priradená zvislým a druhá vodorovným čiaram.

In[33]:=
```
Plot[Sin[x], {x, 0, 2Pi}, GridLines → Automatic,
GridLinesStyle → {Dotted, Red}]
```

Out[33]=

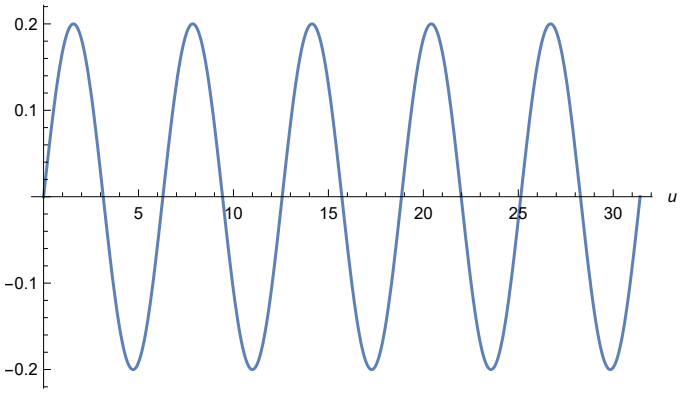


Ak by zadaný zoznam obsahoval tri alebo viac volieb, výstup by bol rovnaký ako v prípade, keby sme nezadali voľbu na úpravu štýlu.

Osi môžeme označiť automaticky (t. j. s použitím premennej zo zadania) pomocou voľby **AxesLabel→Automatic**.

In[34]:=
```
Plot[0.2Sin[u], {u, 0, 10π}, AxesLabel → Automatic]
```

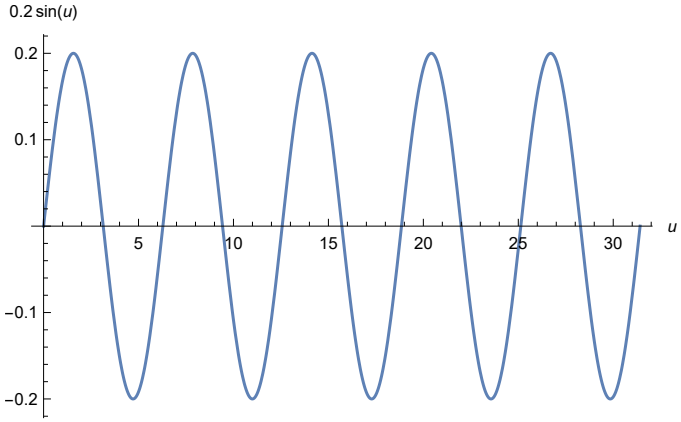
Out[34]=



Ak chceme zadať vlastný popis osí, nahradíme nastavenie **Automatic** zoznamom našich označení.

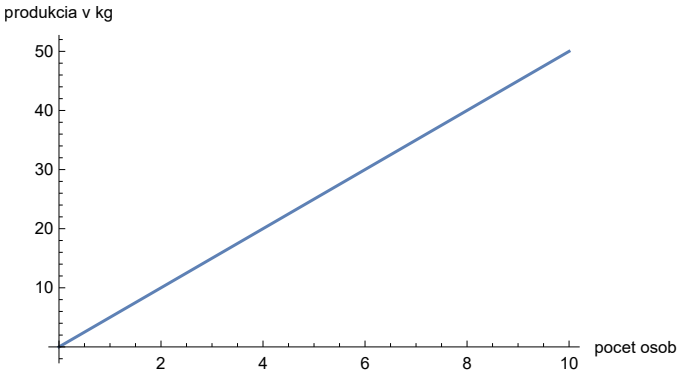
```
In[35]:= Plot[0.2Sin[u], {u, 0, 10π}, AxesLabel → {u, 0.2Sin[u]}]
```

Out[35]=




```
In[36]:= Plot[5t, {t, 0, 10}, AxesLabel → {"pocet osob", "produkcia v kg"}]
```

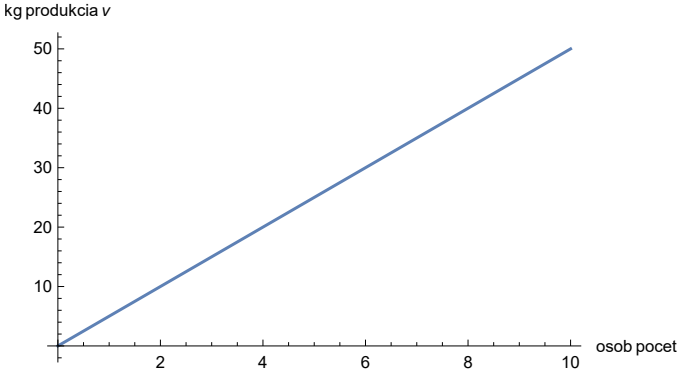
In[37]:=



V druhom príklade sú prvky zoznamu zadané ako text, t. j. reťazec znakov. V softvéri Mathematica je takýto typ vstupného argumentu označovaný ako **String** (z angl. reťazec). Ak by sme nepoužili úvodzovky, dostali by sme nasledujúci výsledok.

```
In[38]:= Plot[5t, {t, 0, 10}, AxesLabel → {pocet osob, produkcia v kg}]
```

Out[38]=

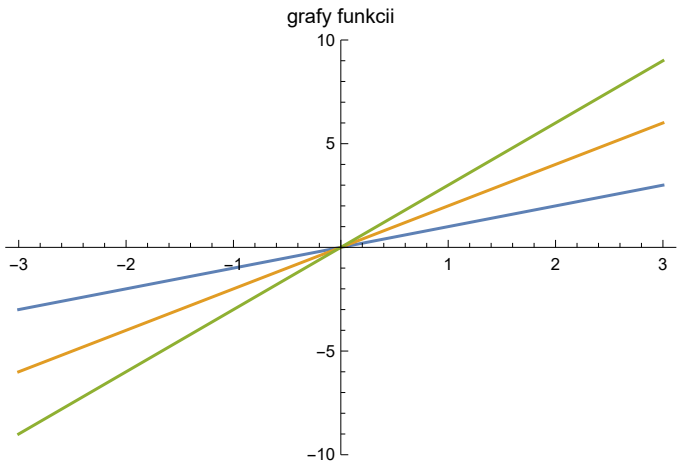


Jednotlivé skupiny znakov oddelené medzerami sú považované za symboly a softvér Mathematica ich usporiadal v abecednom poradí. Ak by sme chceli ako popis zadať nejakú premennú, ktorej už v predchádzajúcich krokoch bola priradená nejaká hodnota, museli by sme ju zadať ako prvok typu **String** (reťazec znakov v úvodzovkách). Ak by sme totiž nepoužili úvodzovky, namiesto názvu premennej by v popise osi bola zobrazená jej hodnota. Toto zadávanie vstupných argumentov ako symboly, premenné alebo prvky typu String je jednotné aj pre voľby **PlotLabel**, **PlotLabels**, **PlotLegends** a **FrameLabel**.

Voľba **PlotLabel** slúži na pridanie popisu celého grafu.

```
In[39]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotLabel → "grafy funkcii"]
```

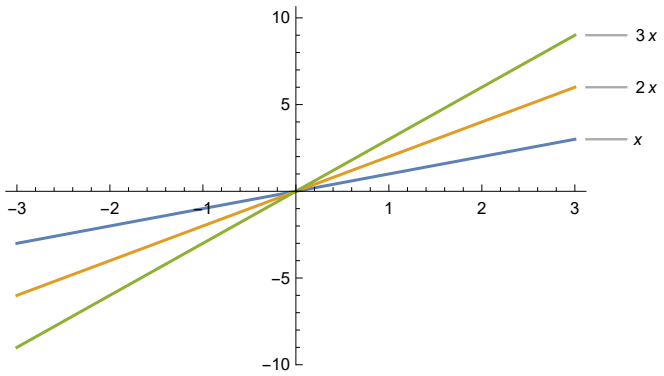
Out[39]=



Voľby **PlotLabels** a **PlotLegends** sa používajú na popis jednotlivých grafov zobrazovaných funkcií. Obidve majé rovnaký spôsob zadávania, ale voľba **PlotLabels** zobrazí popisy vedľa zobrazovaných grafov, zatiaľ čo voľba **PlotLegends** vytvorí v pravej časti obrázku legendu s označením grafov.

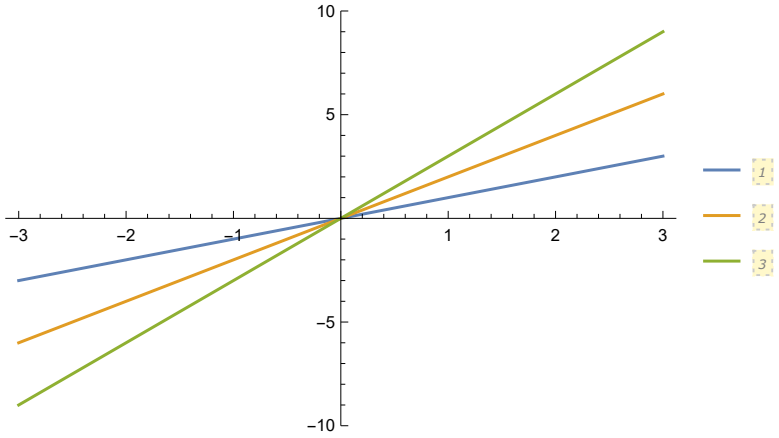
```
In[40]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotLabels → Automatic]
```

Out[40]=




```
In[41]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotLegends → Automatic]
```

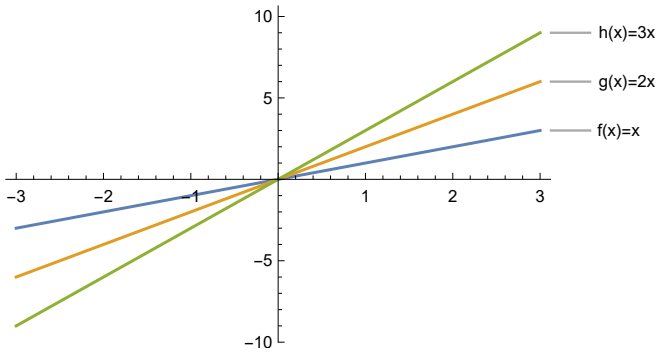
Out[41]=



Zobrazovaným grafom funkcií môžeme priradiť aj vlastné označenia, ktoré zadávame ako zoznam prvkov typu String.

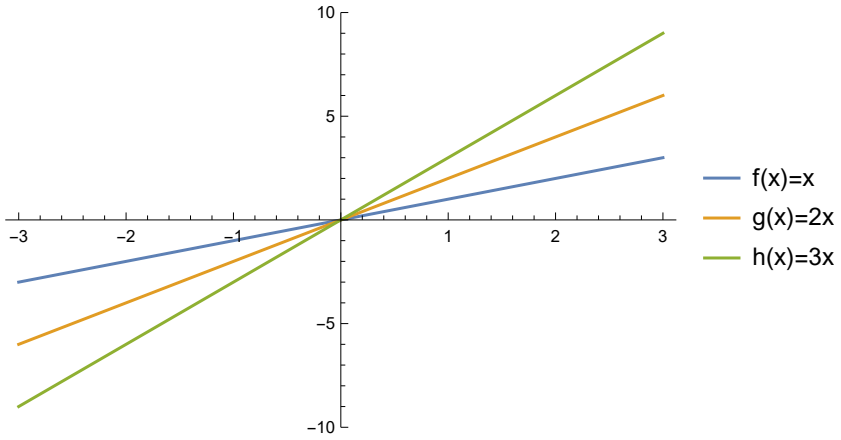
```
In[42]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotLabels → {"f(x)=x", "g(x)=2x",
                                                     "h(x)=3x"}]
```

Out[42]=




```
In[43]:= Plot[{x, 2x, 3x}, {x, -3, 3}, PlotLegends → {"f(x)=x", "g(x)=2x",
                                                     "h(x)=3x"}]
```

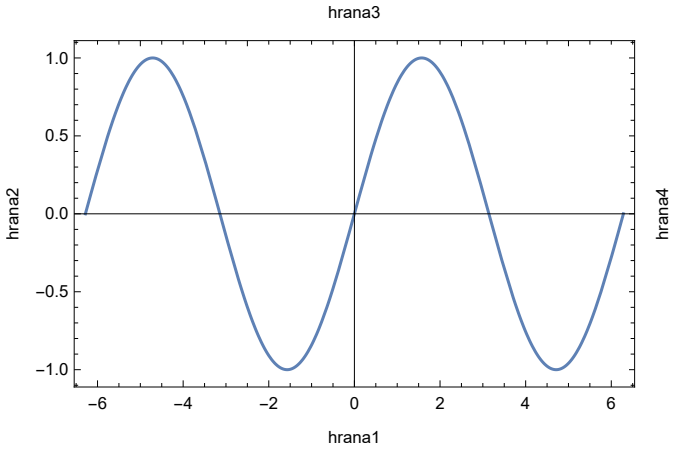
Out[43]=



Rám grafu sa dá označiť pomocou voľby **FrameLabel**.

```
In[44]:= Plot[Sin[x], {x, -2Pi, 2Pi}, Frame → True,
         FrameLabel → {"hrana1", "hrana2", "hrana3", "hrana4"}]
```

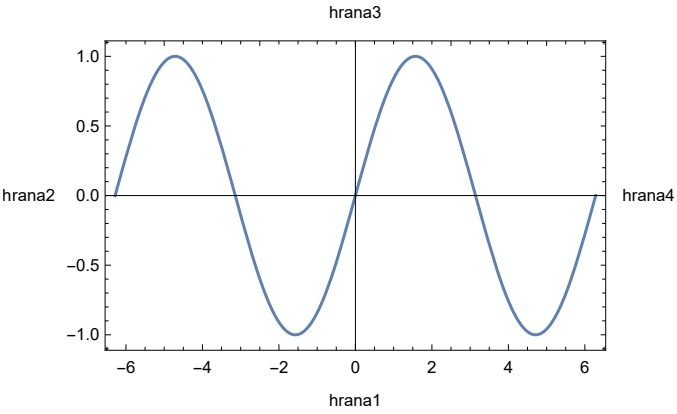
Out[44]=



Ak požadujeme, aby popis bočných hrán nebol kolmý, pridáme naviac voľbu **RotateLabel→False**.

```
In[45]:= Plot[Sin[x], {x, -2Pi, 2Pi}, Frame → True, RotateLabel → False,
         FrameLabel → {"hrana1", "hrana2", "hrana3", "hrana4"}]
```

Out[45]=



Ďalším voľbám a štýlom sa už nebudeme venovať. Čitateľa odkazujeme na dokumentové centrum softvéra Mathematica, pozri sekciu 2.2.

Cvičenie 5.5

Pozrite si v dokumentovom centre funkciu **Plot** a jej voľby (po angl. options). Vyskúšajte si niektoré z nich.

5.2 Funkcie typu Plot

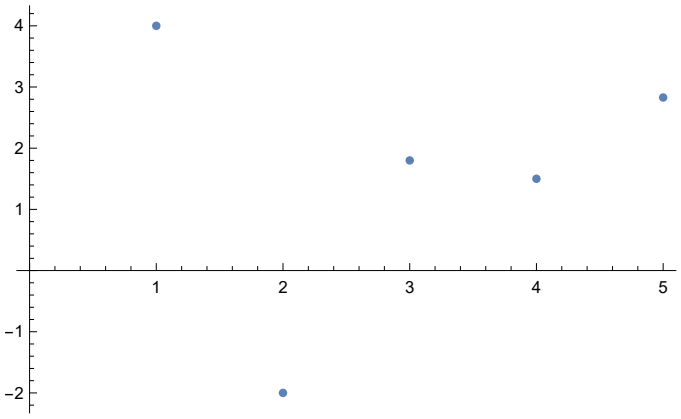
Okrem základnej zobrazovacej funkcie **Plot** sú v softvéri Mathematica zabudované ďalšie funkcie, v ktorých názve sa tiež nachádza slovo **Plot**. Tieto funkcie majú v sebe vo veľkej miere vstavané voľby, ktoré sme si predstavili v predchádzajúcej podsekcii.

5.2.1 Funkcie ListPlot a ListLinePlot

V niektorých situáciách, napr. pri zobrazovaní dát, potrebujeme zobraziť nie spojitú funkciu, ale len diskrétne body reprezentujúce vstupné dáta. Na tento účel slúži vstavaná funkcia **ListPlot** (z angl. zoznam zobraz). Používa sa v dvoch verziách: na zobrazenie zoznamu hodnôt alebo zoznamu dvojhodnotových dát. Zoznam môžeme zadať ako vstup priamo alebo pomocou nejakej premennej.

In[46]:=
```
ListPlot[{4, -2, 9/5, 1.5, Sqrt[8]}];
zozn={4, -2, 9/5, 1.5, Sqrt[8]};
ListPlot[zozn]
```

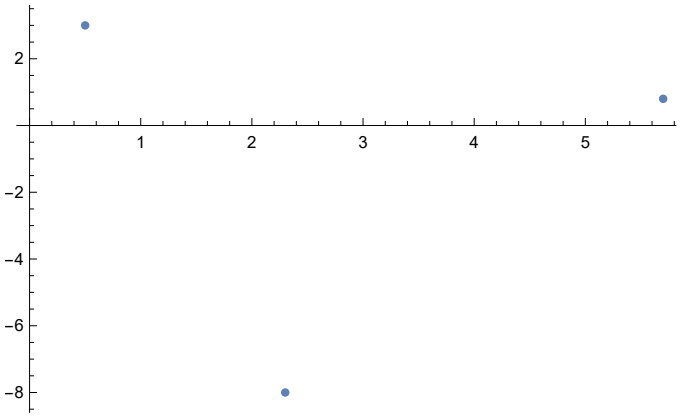
Out[46]=



Pri zobrazovaní dvojhodnotových dát je vstupom funkcie **ListPlot** zoznam dvojprvkových zoznamov.

In[47]:=
```
dvojzozn={{0.5, 3}, {2.3, -8}, {5.7, 0.8}};
ListPlot[dvojzozn]
```

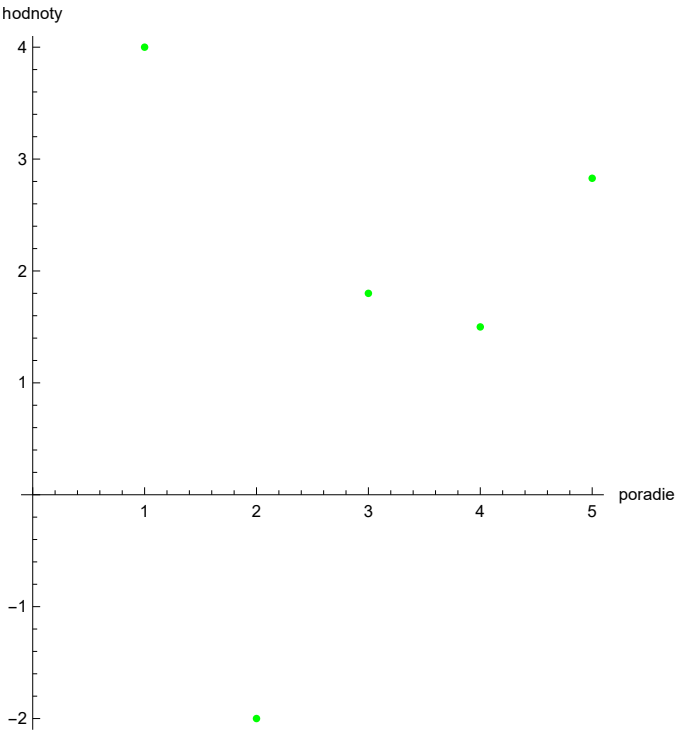
Out[47]=



Vo funkcii **ListPlot** sa používajú na nastavenie štýlu zobrazenia rovnaké voľby a viaceré direktívy ako vo funkcii **Plot**.

In[48]:=
```
zozn={4, -2, 9/5, 1.5, Sqrt[8]};
ListPlot[zozn, PlotStyle → Green, AxesLabel → {"poradie", "hodnoty"},
PlotRange → {-2.1, 4.1}, AspectRatio → Automatic]
```

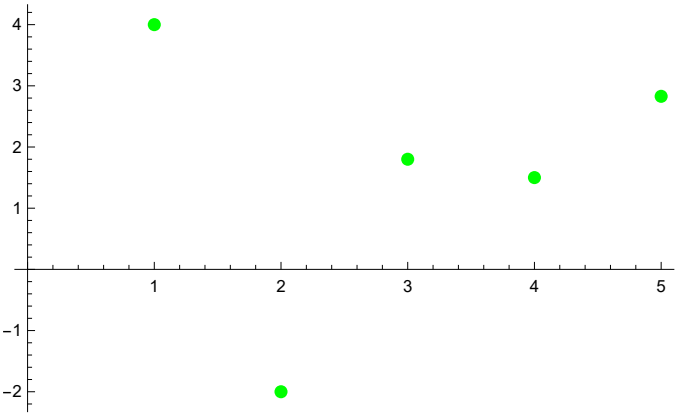
Out[48]=



Okrem direktív, ktoré sme si predstavili pre funkciu **Plot**, sa často používa direktíva **PointSize[velkost]** (z angl. veľkosť bodu), ktorá umožňuje nastaviť požadovanú veľkosť zobrazovaných bodov.

In[49]:=
```
zozn={4, -2, 9/5, 1.5, Sqrt[8]};
ListPlot[zozn, PlotStyle → {Green, PointSize[0.02]}]
```

Out[49]=



Cvičenie 5.6

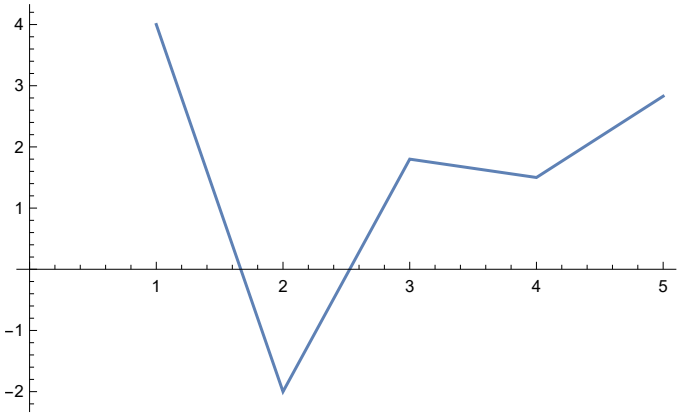
Vytvorte zoznam druhých mocnín čísel od -1 po 1 s krokom 0.1 a zobrazte ich na grafe, kde x-ová os je označená ako x a y-ová os ako x^2.

Pomôcka: využite poznatky z kapitoly 4.

Diskrétne body dokážeme zobraziť pospájané (každý bod spojený lineárne so svojím predchodcom) pomocou voľby **Joined** (z angl. spojené).

```
In[50]:= ListPlot[zozn, Joined → True]
```

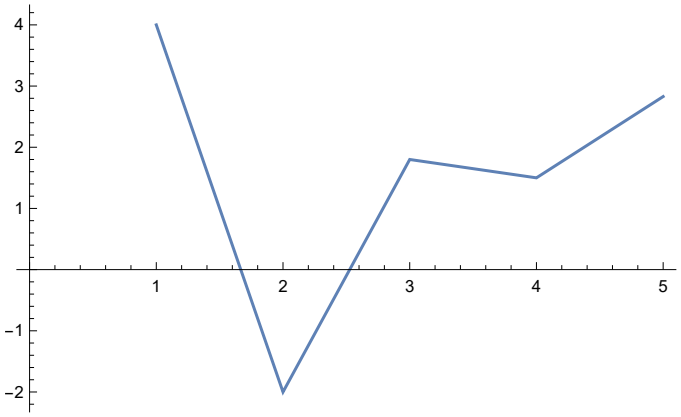
Out[50]=



Príkaz **ListPlot[zozn, Joined→True]** je ekvivalentom vstavanej funkcie **ListLinePlot[zozn]**.

```
In[51]:= ListLinePlot[zozn]
```

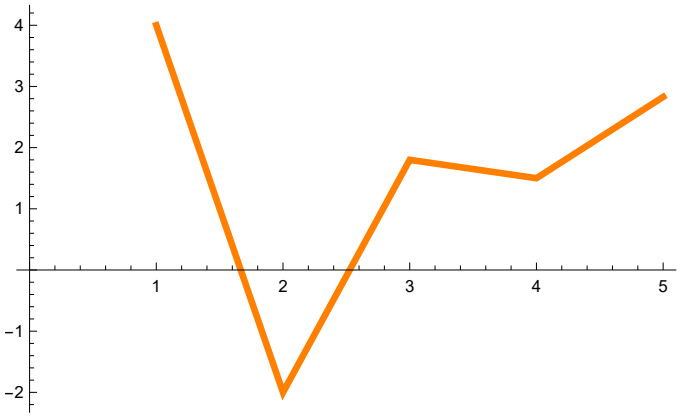
Out[51]=



Štýl pospájania bodov nastavujeme rovnakými voľbami ako štýl zobrazenia matematických funkcií vo vstavanej funkcii **Plot**.

```
In[52]:= ListPlot[zozn, PlotStyle → {Orange, Thickness[0.01]}, Joined → True];
         ListLinePlot[zozn, PlotStyle → {Orange, Thickness[0.01]}]
```

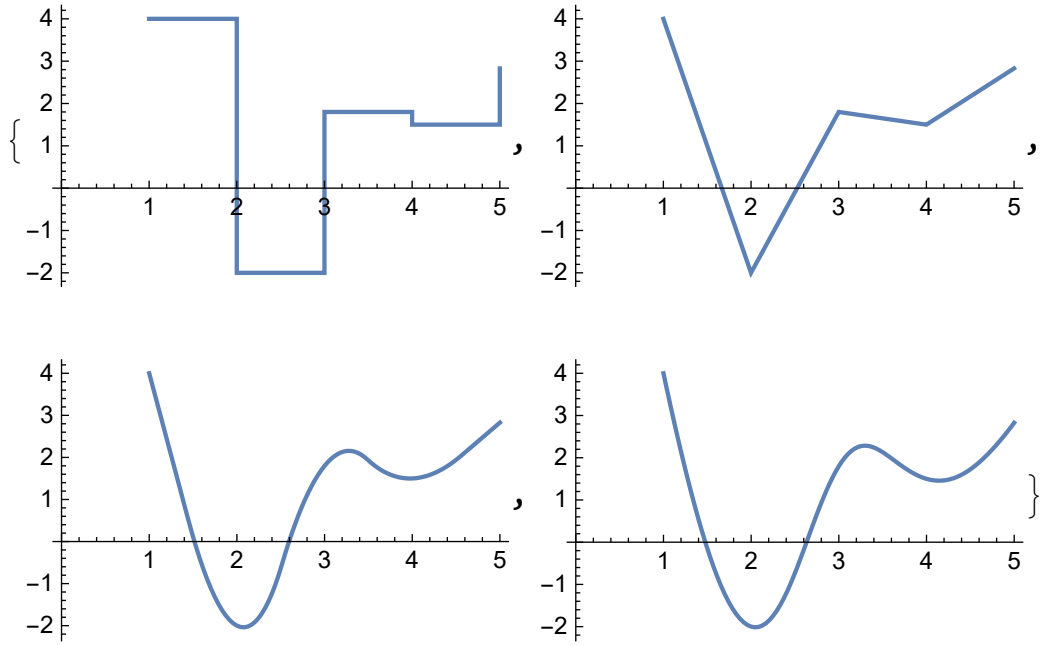
Out[52]=



Lineárne pospájanie bodov zodpovedá interpolácii prvého rádu. Rád interpolácie meníme pomocou voľby `InterpolationOrder → n` (z angl. rád interpolácie).

In[53]:=
```
Table[ListLinePlot[zozn, InterpolationOrder → n], {n, {0, 1, 2, 4}}]
```

Out[53]=



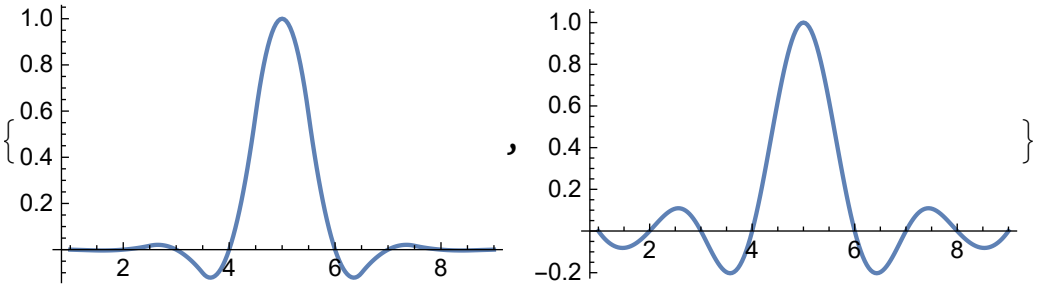
Cvičenie 5.7

Vytvorte zoznam druhých mocnín čísel od -1 po 1 s krokom 0.25 a zobrazte ich ako graf spojitej funkcie interpolovanej 0., 1., 2. a 3. rádom.

Aj keď grafy z predchádzajúceho príkladu pre rád interpolácie 2 a 4 sa vizuálne javia ako rovnaké, sú odlišné. Rozdielnosť grafov s rôznymi rádmi interpolácie (s rádom rovným a väčším ako 2) si ukážeme na nasledujúcich dátach.

In[54]:=
```
z01={0, 0, 0, 0, 1, 0, 0, 0, 0}
Table[ListLinePlot[z01, PlotRange → All, InterpolationOrder → n],
                                                    {n, {2, 8}}]
```

Out[54]=

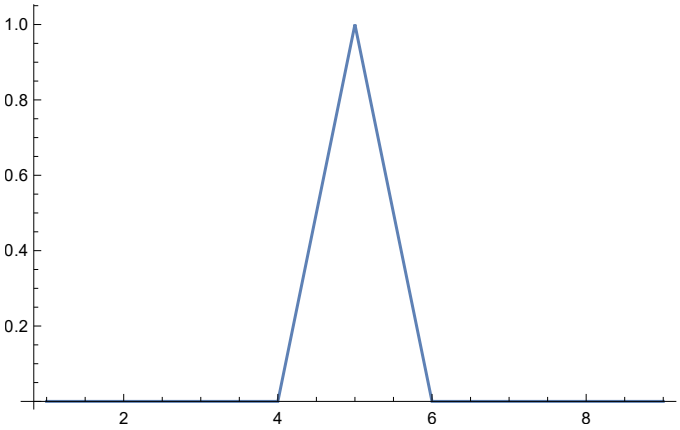


Pripomeňme, že pri interpolovaní zoznamu dát môžeme zadať rád interpolácie maximálne o jednotku nižší, ako je počet prvkov tohto zoznamu. Ak by sme toto pravidlo porušili, softvér by nás na to upozornil nasledovným spôsobom.

In[55]:=
```
z01={0, 0, 0, 0, 1, 0, 0, 0, 0}
ListLinePlot[z01, PlotRange→All, InterpolationOrder→9]
```

ListLinePlot: 9 is not a valid interpolation order specification, or the given order is too high for the input array {{1.,0.},{2.,0.},{3.,0.},{4.,0.},{5.,0.},{6.,1.},{7.,0.},{8.,0.},{9.,0.}}.

Out[55]=



Keďže deviaty rád interpolácie sa v predchádzajúcom príklade nedal aplikovať, graf vo výstupnej bunke je zobrazený s použitím preddefinovaného prvého rádu interpolácie.

Poznámka 5.3

Pri spracovávaní dát v štatistike sa funkcia **ListPlot** používa niekedy v kombinácii s funkciou **Fit** (z angl. napasovať, prispôsobiť). Jej výstupom je polynomická funkcia, ktorá je pre daný stupeň polynómu najlepšou aproximáciou vstupných dát. Ak je stupeň polynómu menší ako počet prvkov dát o viac ako jednotku, graf polynomickej funkcie (až na ojedinelé výnimky) neprechádza bodmi, ktoré reprezentujú hodnoty dát.

In[56]:=
```
zozn={4, -2, 9/5, 1.5, Sqrt[8]};
f1 = Fit[zozn, {1, x}, x]
f2 = Fit[zozn, {1, x, x^2}, x]
f3 = Fit[zozn, {1, x, x^2, x^3}, x]
```

```
f4 = Fit[zozn, {1, x, x^2, x^3, x^4}, x]
f10 = Fit[zozn, {1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^10}, x]
```

Out[56]= $1.27863 + 0.115685\ x$

Out[57]= $6.55706 - 4.40868\ x + 0.754061\ x^2$

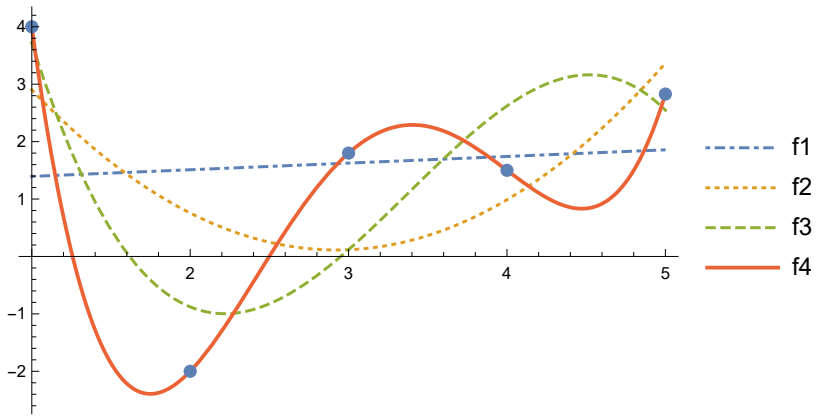
Out[58]= $17.9973 - 20.4794\ x + 6.88274\ x^2 - 0.680964\ x^3$

Out[59]= $53.3284 - 87.0759\ x + 47.4248\ x^2 - 10.4952\ x^3 + 0.817851\ x^4$

Out[60]= $21.137 - 21.3243\ x + 3.32158\ x^2 + 0.862064\ x^3 + 0.0311071\ x^4 - 0.0203706\ x^5 - 0.00611759\ x^6 - 0.00100023\ x^7 - 0.0000725428\ x^8 + 0.0000190369\ x^9 + 0.0000109177\ x^{10}$

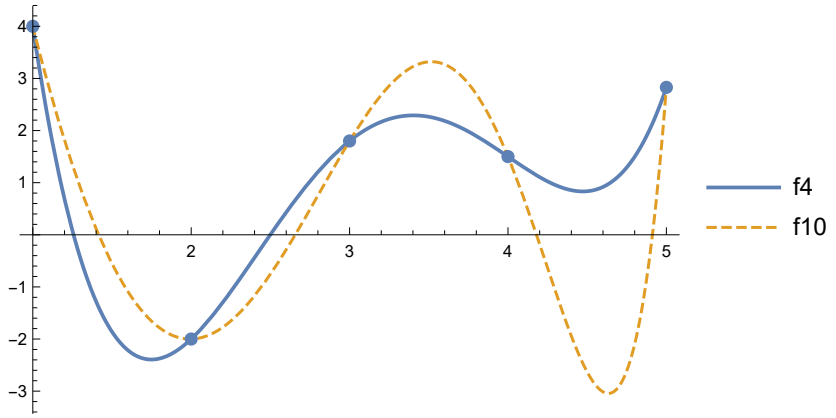
In[61]:=
```
data = ListPlot[zozn, PlotStyle → PointSize[0.02]];
aproximacie = Plot[{f1, f2, f3, f4}, {x, 1, 5}, PlotRange → All,
 PlotStyle → {DotDashed, Dotted, Dashed, Thick},
 PlotLegends → "Expressions"];
Show[aproximacie, data]
```

Out[61]=



In[62]:=
```
data = ListPlot[zozn, PlotStyle → PointSize[0.02]];
aproximacie2 = Plot[{f4, f10}, {x, 1, 5}, PlotRange → All,
 PlotStyle → {Thick, Dashed}, PlotLegends → "Expressions"];
 Show[aproximacie2, data]
```

Out[62]=

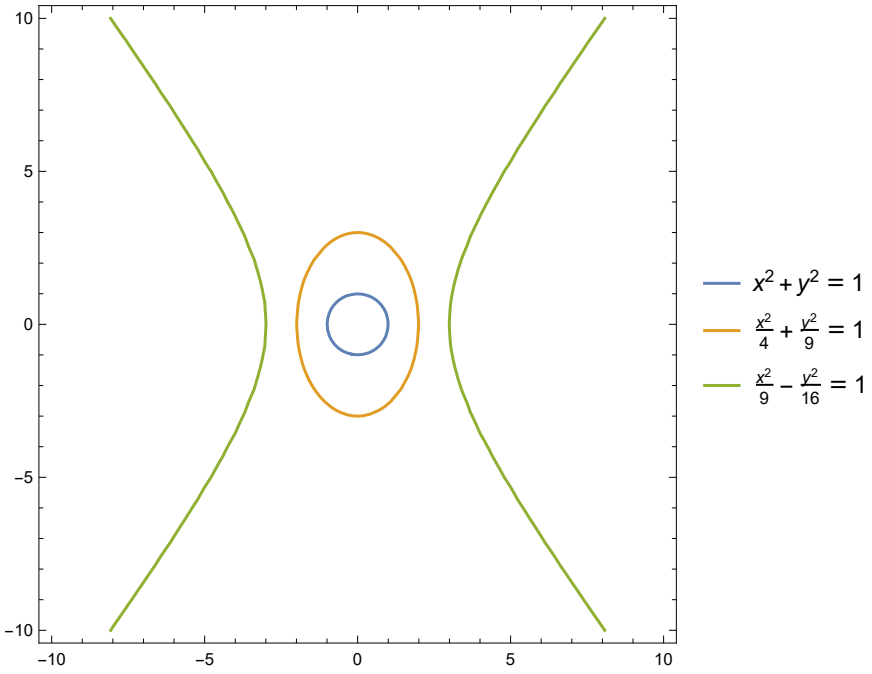


5.2.2 Funkcia ContourPlot

Táto vstavaná funkcia sa používa na zobrazovanie kriviek zadaných implicitne, ako napr. kužeľosečiek. Jej predpis je nasledovný `ContourPlot[lava == prava,{x,xmin,xmax},{y,ymin,ymax}]`, kde `lava` je ľavá a `prava` je pravá strana rovnosti reprezentujúcej množinu bodov, ktorú chceme dať vykresliť, $\mathtt{x_{min}}$ a $\mathtt{x_{max}}$ sú hranice oblasti vykreslenia v horizontálnom smere a $\mathtt{y_{min}}$ a $\mathtt{y_{max}}$ sú hranice oblasti vykreslenia vo vertikálnom smere. V nasledujúcom príklade zobrazíme do jedného obrázku kružnicu, elipsu a hyperbolu.

In[63]:=
```
ContourPlot[{x^2 + y^2 == 1, x^2/4 + y^2/9 == 1,
x^2/9 - y^2/16 == 1}, {x, -10, 10}, {y, -10, 10},
PlotLegends → "Expressions"]
```

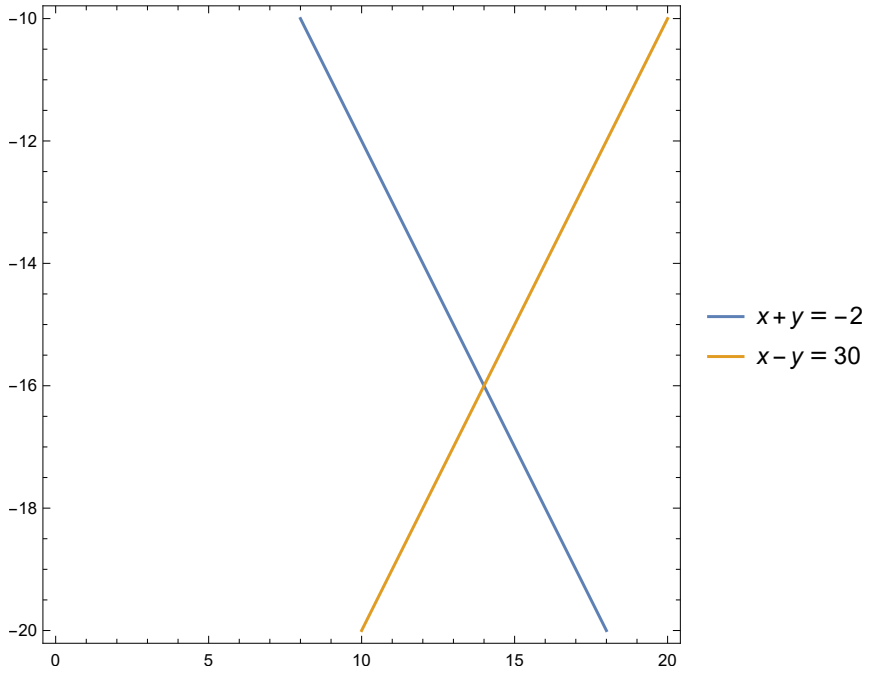
Out[63]=



Pomocou **ContourPlot** vieme graficky riešiť sústavy rovníc. Hranice rozsahu zobrazenia musia byť dostatočne široké, aby sa riešenie nachádzalo na zobrazovanej ploche. Pripomíname, že rovnosť sa zapisuje pomocou dvojice znamienok "rovná sa", pozri Pozn. 2.5.

```
In[64]:= ContourPlot[{x + y == -2, x - y == 30}, {x, 0, 20}, {y, -20, -10},
         PlotLegends → "Expressions"]
```

Out[64]=



Pomocou voľby **ContourStyle** vieme (podobne ako s **PlotStyle**) meniť štýl vykresľovania kontúr.

Funkcia **ContourPlot** taktiež slúži na vykresľovanie izočiar (kontúr, vrstevníc) 3D plôch a jej rozšírená verzia **ContourPlot3D** na vykresľovanie izoplôch – to je však už obsahom druhého dielu tejto učebnice.

5.2.3 Funkcia ParametricPlot

Pomocou tejto vstavanej funkcie sa dajú zobrazovať krivky dané parametricky (po angl. parametric curves). Jej syntax je nasledovná: **ParametricPlot[{f_x,f_y},{t,t_{min},t_{max}}]**, kde **f_x** je x-ová a **f_y** je y-ová zložka parametricky danej krivky, **t** je parameter krivky a **t_{min}** a **t_{max}** sú jeho hranice.

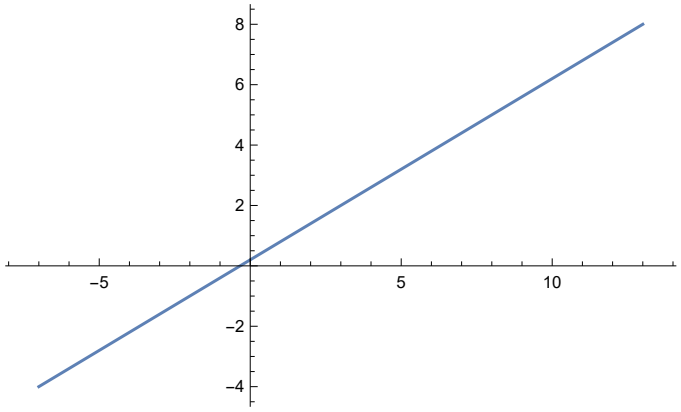
V nasledujúcom príklade vykreslíme priamku prechádzajúcu bodmi $A = [3, 2]$ a $B = [8, 5]$. Táto priamka má smerový vektor $u = B - A = [8 - 3, 5 - 2] = [5, 3]$ a jej parametrické vyjadrenie je:

$$p : x = 3 + 5t$$
$$y = 2 + 3t, t \in R.$$

Za parametre $\texttt{t}_{\texttt{min}}$ a $\texttt{t}_{\texttt{max}}$ dosadíme hodnoty reprezentujúce začiatočný a koncový bod zabrazovaného úseku priamky ($t = 0$ zodpovedá bodu A a $t = 1$ zodpovedá bodu B).

In[65]:=
```
ParametricPlot[{3 + 5t, 2 + 3t}, {t, -2, 2}]
```

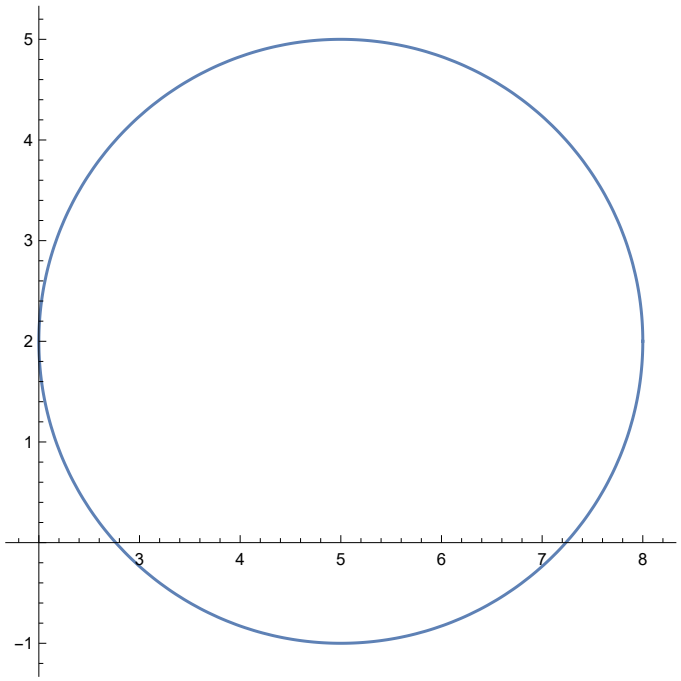
Out[65]=



Ďalej zobrazíme kružnicu so stredom $S = [s1, s2]$ a polomerom r.

In[66]:=
```
r = 3; s1 = 5; s2 = 2;
ParametricPlot[{s1 + r Cos[t], s2 + r Sin[t]}, {t, 0, 2Pi}]
```

Out[66]=



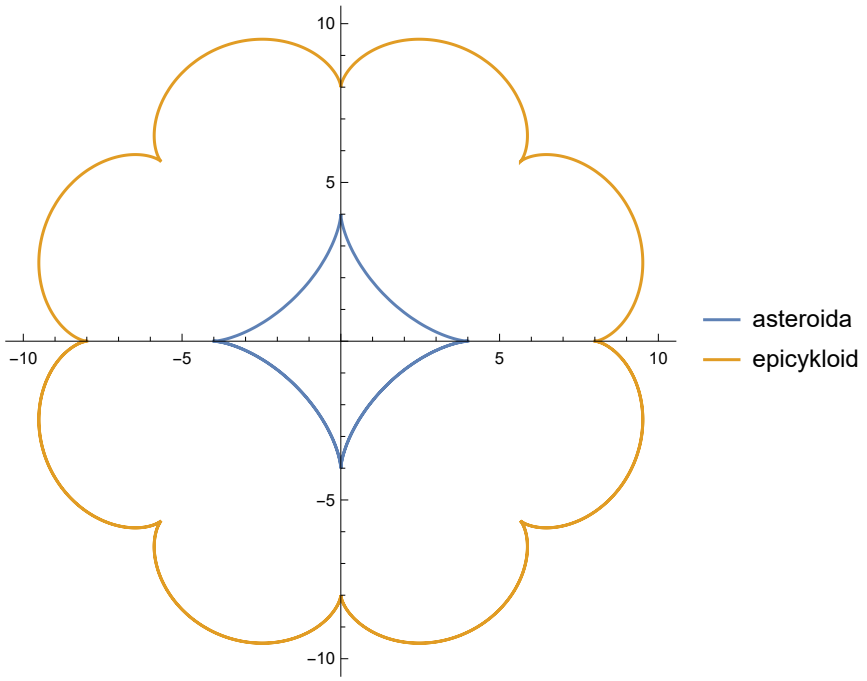
Cvičenie 5.8

Vykreslite elipsu so stredom v bode $[0, 0]$ s hlavnou osou v smere x s dĺžkou 2 a vedľajšou osou v smere y s dĺžkou 1.5.

Nakoniec ukážeme zobrazenie asteroidy a epicykloidu v jednom obrázku.

```
In[67]:= xaster = 3Cos[t] + Cos[3t];
         yaster = 3Sin[t] - Sin[3t];
         xepic = 9Cos[t] - Cos[9t];
         yepic = 9Sin[t] - Sin[9t];
         ParametricPlot[{{xaster, yaster}, {xepic, yepic}}, {t, -Pi, 2Pi},
         PlotLegends → {"asteroida", "epicykloid"}]
```

Out[67]=



Ku funkcii **ParametricPlot** existuje rozšírená verzia **ParametricPlot3D**. Pomocou nej vieme zobrazovať parametricky dané krivky v 3D priestore a taktiež aj 3D plochy.

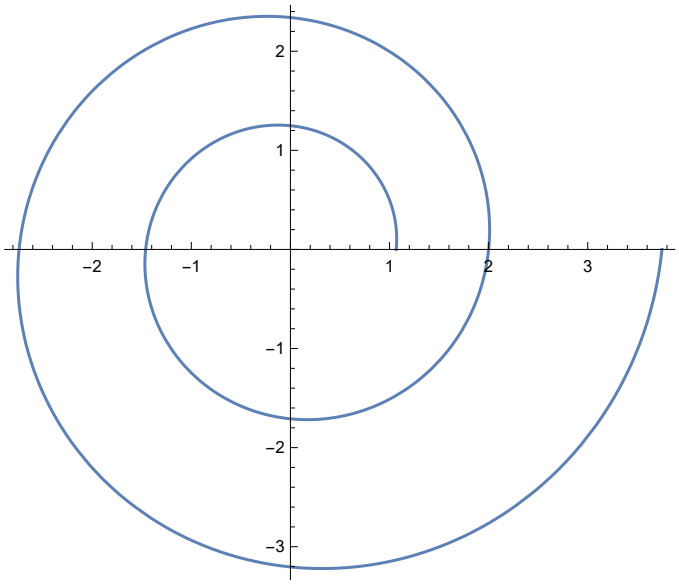
5.2.4 Funkcia PolarPlot

Táto vstavaná funkcia slúži na zobrazovanie kriviek daných v polárnych súradniciach. Jej predpis je nasledovný: **PolarPlot[r,{θ,θmin,θmax}]**, kde polomer **r** a uhol θ sú polárne súradnice zobrazovanej krivky. Softvér Mathematica ju vykresľuje na intervale ($\theta_{\mathtt{min}}$,$\theta_{\mathtt{max}}$). Výstupom je zobrazená krivka $(r\cos(\theta), r\sin(\theta))$.

V nasledujúcom príklade zobrazíme logaritmickú špirálu, ktorej polomer rastie exponenciálne s veľkosťou uhla. Je daná vzťahom: $r = a\mathrm{e}^{b\theta}$.

```
In[68]:= a = 2; b = 0.1;
         PolarPlot[a Exp[b t], {t, -2Pi, 2Pi}]
```

Out[68]=



Cvičenie 5.9

Pomocou funkcie `ParametricPlot` získajte výstup z predchádzajúceho príkladu.

Pomôcka: Prevod z polárnych do kartézskych súradníc má predpis $x = r\cos(\theta)$ a $y = r\sin(\theta)$.

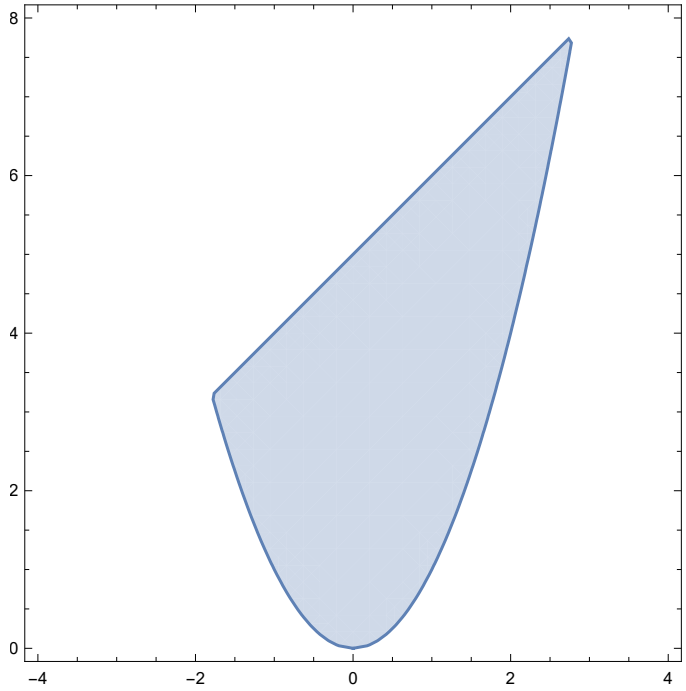
5.2.5 Funkcia RegionPlot

Pomocou tejto vstavanej funkcie vieme vykresľovať oblasti (po angl. regions) roviny. Jej predpis je nasledovný: `RegionPlot[oblast, {x, xmin, xmax}, {y, ymin, ymax}]`, kde `oblast` je oblasť roviny vyjadrená pomocou logických tvrdení, `xmin` a `xmax` sú hranice oblasti zobrazenia v horizontálnom smere a `ymin` a `ymax` sú hranice oblasti zobrazenia vo vertikálnom smere.

V nasledujúcom príklade zobrazíme oblasť ohraničenú parabolou a priamkou. Dve nerovnice ohraničujúce oblasť sú spojené logickým operátorom && (”a zároveň”).

In[69]:=
```
RegionPlot[y > x^2 && y < x + 5, {x, -4, 4}, {y, 0, 8}]
```

Out[69]=



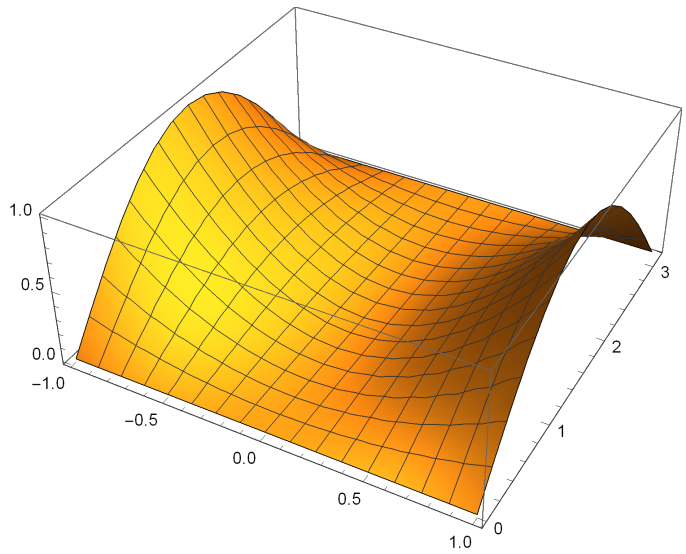
Ďalší príklad použitia funkcie **RegionPlot** nájdete v sekcii 11.2.

5.2.6 Funkcie viacerých premenných

Základnou vstavanou funkciou na zobrazovanie grafov funkcií viacerých premenných je funkcia `Plot3D[f, {x,xmin,xmax}, {y,ymin,ymax}]`. Od funkcie `Plot` sa líši v tom, že `f` je predpis funkcie dvoch premenných a tretí (dodatočný) argument je druhá premenná a jej hranice.

In[70]:=
```
Plot3D[u^2 Sin[v], {u, -1, 1}, {v, 0, π}]
```

Out[70]=



Keďže táto učebnica pokrýva len rozsah učiva prvého semestra predmetu ”softvér Mathematica”, zobrazovanie grafov funkcií viacerých premenných si podrobnejšie predstavíme v jej druhom diele.

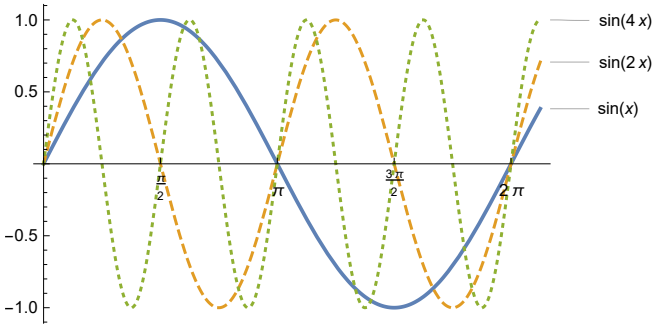
5.3 Funkcia Manipulate

V niektorých situáciách je prospešné poznať, ako vybraný parameter ovplyvňuje graf. Môže ísť o grafy z praxe reprezentujúce napr. ekonomické a hospodárske ukazovatele. Taktiež pri výuke je dôležité názorne prezentovať vplyv jednotlivých parametrov na tvar grafu, napr. ukázať, ako sa mení graf funkcie $f(x) = sin(nx)$ v závislosti od parametra n.

Jedna z možností, ako to dosiahnúť je zobrazenie viacerých grafov funkcií do jedného obrázku.

```
In[71]:= Plot[{Sin[x], Sin[2 x], Sin[4 x]}, {x, 0, 17/8Pi},
         Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic},
         PlotStyle → {Thick, Dashed, Dotted},  PlotLabels → Automatic]
```

Out[71]=

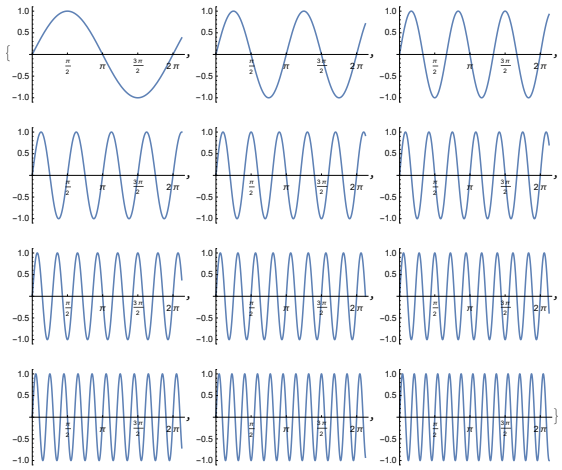


Avšak takéto riešenie sa v prípade väčšieho počtu funkcií stáva neprehľadným.

Ďalšou možnosťou je zobrazenie tabuľky grafov jednotlivých funkcií.

```
In[72]:= Table[Plot[Sin[n x], {x, 0, 17/8Pi},
         Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic}], {n, 1, 12, 1}]
```

Out[72]=

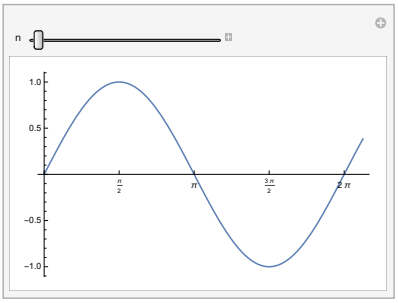


Tento spôsob je síce prehľadnejší, ale v prípade vysokého počtu hodnôt sledovaného parametra je dosť nepraktický. Ako alternatíva k tomuto spôsobu sa ponúka použitie funkcie **Manipulate**, ktorá umožňuje vytvárať interaktívne grafy a objekty. Všimnime si, že jej predpis

Manipulate[vyraz,{n,n_{min},n_{max},dn}] je rovnaký ako predpis funkcie **Table** (pozri str. 52). V prípade funkcie **Manipulate** výraz **vyraz** reprezentuje zobrazovaný graf, objekt alebo výraz obsahujúci parameter **n**, ktorý nadobúda hodnoty z rozsahu **{n_{min},n_{max}}** s krokom **dn**. V nasledujúcom príklade použijeme pre funkciu **Manipulate** rovnaký vstupný argument, aký mala funkcia **Table** v predchádzajúcom príklade.

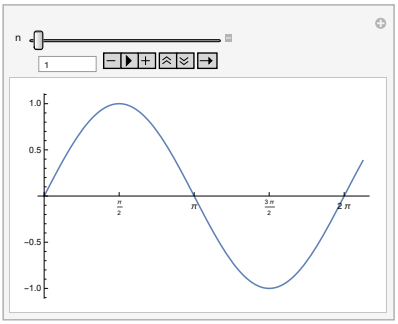
In[73]:=
```
Manipulate[Plot[Sin[n x], {x, 0, 17/8Pi},
Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic}], {n, 1, 12, 1}]
```

Out[73]=



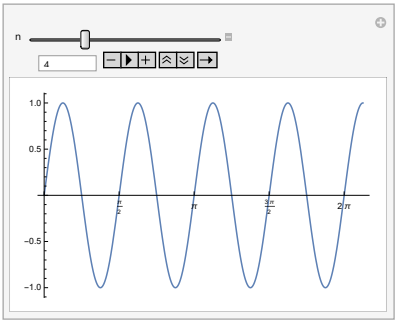
Ako výstup sa nám zobrazil graf s hodnotou meniaceho sa parametra **n** a nad ním lišta s posuvným bežcom. Keď klikneme na malý štvorček so znamienkom "plus" nachádzajúci sa za lištou, zobrazí sa pod lištou menu slúžiace na manipuláciu sledovaného parametra a následne aj zobrazovaného grafu.

Out[74]=



Úplne vľavo v menu sa nachádza okienko, v ktorom sa vypisuje hodnota sledovaného parametra. Na začiatku sa rovná hodnote n_{min}. Túto hodnotu môžeme zmeniť prepísaním alebo posunutím bežca na lište pomocou počítačovej myši.

Out[75]=



Umiestnenie bežca (a teda aj hodnota v prvom okienku) sa dá zmeniť aj pomocou menu pod lištou. Kliknutím na tretie okienko so "šípkou" sa spúšťa a zastavuje animácia. Hodnota v prvom okienku sa postupne mení, prechádza celým rozsahom **{n$_{min}$,n$_{max}$}** s krokom **dn** a príslušne k tomu sa mení aj zobrazovaný graf. Druhé a štvrté okienko slúži na krokovanie animácie dozadu a dopredu. Kliknutím na piate a šieste okienko animáciu zrýchľujeme a spomaľujeme. Posledné okienko slúži na zmenu smeru animácie. Môže nadobúdať stavy: **Forward** (dopredu), **Backward** (dozadu) a **Forward and Backward** (dopredu a dozadu).

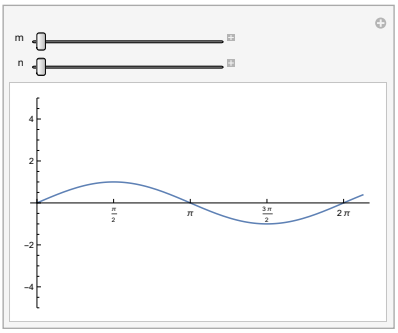
Cvičenie 5.10

Zobrazte kružnicu s polomerom r na oblasti $(-2,2)\times(-2,2)$, kde sa polomer r bude dať meniť pomocou **Manipulate**.

Funkcia **Manipulate** umožňuje manipulovať súčasne s viacerými parametrami. Nasledujúci príklad reprezentuje situáciu s dvoma parametrami.

In[76]:=
```
Manipulate[Plot[m Sin[n x], {x, 0, 17/8Pi},
Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic}],
PlotRange → {-5, 5}], {m, 1, 5, 1}, {n, 1, 12, 1}]
```

Out[76]=



Cvičenie 5.11

Zobrazte elipsu s osami dĺžky a, b na oblasti $(-2,2)\times(-2,2)$, kde sa dĺžky osí a, b dajú meniť pomocou **Manipulate**.

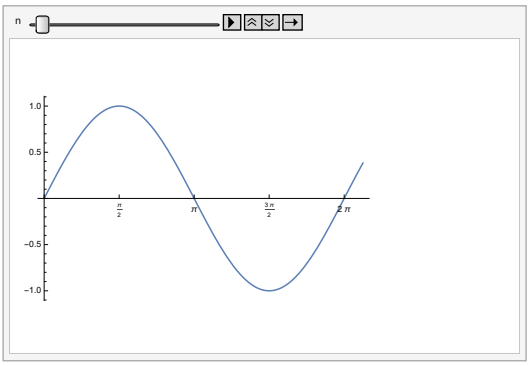
Cvičenie 5.12

Zobrazte kontúru funkcie $\sin(x)\cos(y)=c$ na oblasti $(0,4\pi)\times(0,4\pi)$, kde sa parameter c bude dať meniť pomocou **Manipulate** od -1 po 1.

Okrem funkcie **Manipulate** je v softvéri Mathematica zabudovaná podobná funkcia **Animate**. Obidve funkcie majú rovnaký predpis. Výstupom funkcie **Animate** je už spustená animácia so zobrazeným menu. Dá sa zastavovať a opätovne spúšťať kliknutím na okienko so "šípkou". V menu tejto funkcie sa nenachádzajú okienka na krokovanie.

```
In[77]:= Animate[Plot[Sin[n x], {x, 0, 17/8Pi},
         Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic}], {n, 1, 12, 1}]
```

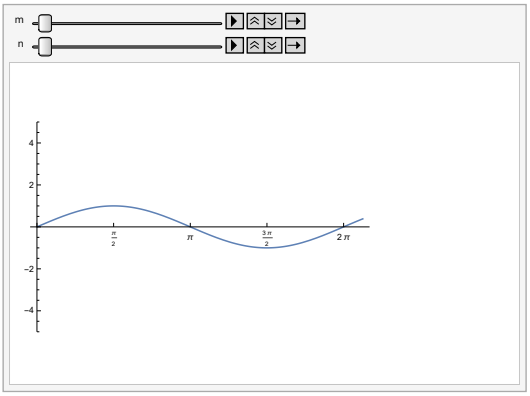
Out[77]=



Funkcia **Animate** podobne ako funkcia **Manipulate** umožňuje manipulovať súčasne viacero parametrov.

```
In[78]:= Animate[Plot[m Sin[n x], {x, 0, 17/8Pi},
         Ticks → {{0, 1/2Pi, Pi, 3/2Pi, 2Pi}, Automatic}],
         PlotRange → {-5, 5}], {m, 1, 5, 1}, {n, 1, 12, 1}]
```

Out[78]=

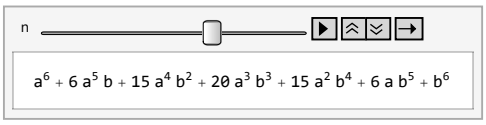


Poznámka 5.4

V tejto kapitole sme si predstavili využitie funkcií **Animate** a **Manipulate** na zobrazovanie grafov. Avšak obidve funkcie môžu vrátiť aj negrafický výstup, napr. výraz s meniacim sa parametrom n.

```
In[79]:= Animate[Expand[(a+b)^n],{n,0,10,1}]
```

Out[79]=



5.4 Aplikácie zobrazovania grafov funkcií – graf funkcie a jeho dotyčnica

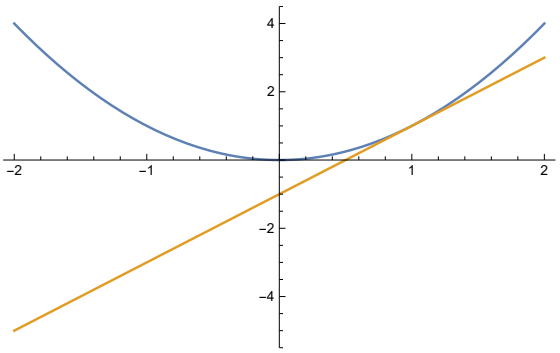
Vykresľovanie grafov funkcií je dobrý spôsob, ako získať prehľad o funkciách: o ich vlastnostiach, o tom, ako jednotlivé parametre ovplyvňujú grafy funkcií, o vzťahoch medzi funkciami... Napr. zobrazenie grafu funkcie a jeho dotyčnice nám pomôže uvedomiť si význam derivácie.

Zadanie: *Vykreslite do jedného obrázku graf funkcie* $f(x) = x^2$ *a jeho dotyčnicu v bode* $[1, f(1)]$.
Funkcia $f(x) = x^2$ má deriváciu $f'(x) = 2x$. V bode $[1, f(1)]$ je $f'(1) = 2$. Dotyčnica grafu funkcie v bode $[1, f(1)]$ je priamka, ktorá prechádza bodom $[1, f(1)]$ a má smernicu 2. Všeobecná rovnica priamky je $d(x) = ax + b$, pričom $[x, y]$ sú súradnice bodov tejto priamky. Smernicou priamky, ktorá je grafom lineárnej funkcie $d(x) = ax + b$ je derivácia tejto funkcie (t. j. konštanta a). Z vyššie uvedeného platí, že $a = 2$. Keďže dotyčnica prechádza bodom grafu funkcie $[1, f(1)]$, platí $f(1) = 2 \cdot 1 + b \implies b = f(1) - 2$. A keďže $f(1) = 1^2 = 1$, tak $b = -1$ a teda rovnica dotyčnice je $d(x) = 2x - 1$.

Zobrazme obidve funkcie do jedného obrázku.

```
In[80]:= Plot[{x^2, 2x - 1}, {x, -2, 2}]
```

Out[80]=



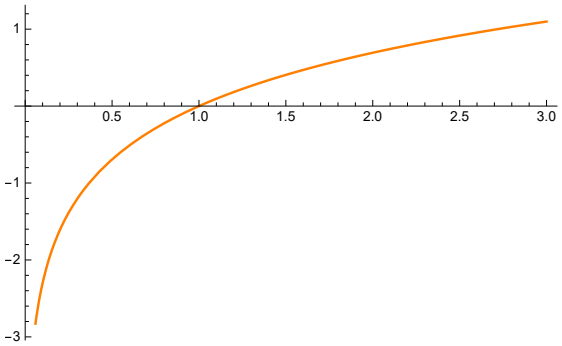
Zadanie: *Vykreslite graf funkcie* $f(x) = \ln(x)$ *spolu s jeho dotyčnicou v bode* $[x_0, f(x_0)]$. *Pomocou* `Manipulate` *budeme meniť hodnotu argumentu* x_0.
Vieme, že dotyčnica v bode $[x_0, f(x_0)]$ je grafom lineárnej funkcie $d(x) = f'(x_0)(x - x_0) + f(x_0)$. Derivácia funkcie $f(x) = \ln(x)$ je $f'(x) = 1/x$. Rovnicu dotyčnice grafu funkcie v bode $[x_0, f(x_0)]$ zadáme nasledovne.

```
In[81]:= 1/x0 (x - x0) + Log[x0]
```

Najskôr si uložme do pomocnej premennej zobrazenie grafu funkcie $f(x) = \ln(x)$.

```
In[82]:= funkciaPlot = Plot[Log[x], {x, 0, 3}, PlotStyle → Orange]
```

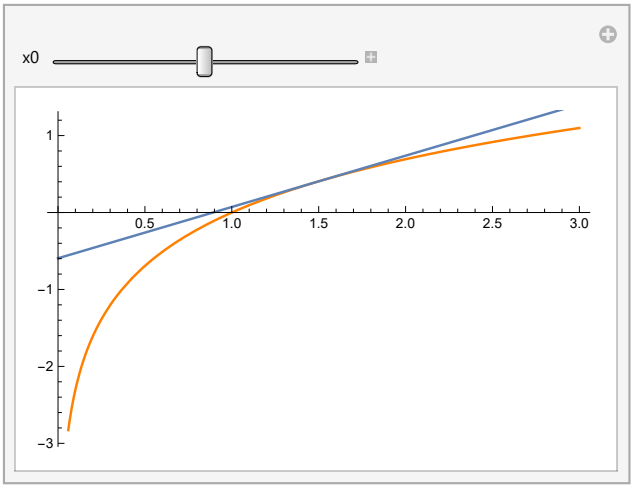
Out[82]=



Potom pomocou **Show** zobrazíme graf funkcie $f(x) = \ln(x)$ a jeho dotyčnicu do jedného obrázku. Bod `x0` budeme meniť pomocou **Manipulate**.

```
In[83]:= Manipulate[Show[funkciaPlot, Plot[1/x0 (x - x0) + Log[x0], {x, 0, 3}]],
                                                                {x0, 0.01, 3}]
```

Out[83]=



5.5 Zhrnutie

V tejto sekcii sme si predstavili zobrazovanie grafov matematických funkcií. Základnou vstavanou funkciou na vykresľovanie grafov funkcií je **Plot**. Má v sebe zabudované množstvo volieb a direktív, ktoré umožňujú detailne nastavovať vlastnosti grafických výstupov. Oboznámili sme sa s tými najčastejšie používanými.

Predstavili sme si aj ďalšie zobrazovacie funkcie určené na vykresľovanie grafov špeciálnych typov funkcií: **ContourPlot** (implicitne dané funkcie), **ParametricPlot** (parametricky dané funkcie), **PolarPlot** (funkcie dané v polárnych súradniciach), **RegionPlot** (rovinné oblasti dané pomocou logických tvrdení). Grafy reprezentujúce diskrétne dáta (zoznamy číselných hodnôt) sa dajú zobrazovať pomocou **ListPlot**.

Vysvetlili sme aj vytváranie interaktívnych výstupov pomocou funkcií **Manipulate** a **Animate**. Tieto funkcie umožňujú interaktívne meniť (nielen) grafy v závislosti od jedného alebo aj viacerých meniacich sa parametrov.

S viacerými z týchto funkcií sa stretneme aj v ďalšom texte tejto učebnice, keďže vizualizácia grafov funkcií a dát poslúži ako výborná pomôcka pri oboznamovaní sa s ďalšími vstavanými funkciami softvéru Mathematica.

Kapitola 6

Úprava výrazov

Veľkou prednosťou softvéru Mathematica je jeho schopnosť prevádzať symbolické výpočty. Keď namiesto čísel alebo konkrétnych funkcií zadáme do výpočtu symboly (premenné bez priradenia), výpočet prebehne a softvér Mathematica vráti výsledok obsahujúci zadané symboly.

Naviac, ak zadáme vo výraze všetky čísla ako celé čísla (čísla typu Integer, bez desatinnej bodky) alebo podiel celých čísel, softvér Mathematica k takémuto výrazu pristupuje inak ako väčšina programovacích jazykov. Tento prístup je podobný tomu, ako by sme s takýmito číslami pracovali na papieri. Namiesto toho, aby sme napr. zlomok 1/3 alebo číslo $\sqrt{2}$ vyčíslili, tak ho ponecháme v zadanom tvare a pokračujeme vo výpočtoch. Výsledkom teda nebude jedno reálne číslo (číslo typu Real, s desatinnou bodkou), ale výraz obsahujúci zlomky a odmocniny, ktorý to číslo reprezentuje.

```
In[1]:= 1/3 + √2x

Out[1]=  1/3 + √2 x
```

6.1 Funkcia na zjednodušenie výrazov – Simplify

V mnohých prípadoch sú však takéto výstupy zapísané vo veľmi komplikovaných tvaroch. Pomocou vstavaných funkcií ich vieme previesť na jednoduchší tvar alebo na tvar, ktorý je pre našu ďalšiu prácu vhodnejší. Najvyužívanejšou funkciou na úpravu výrazov je funkcia **Simplify** (z. angl. zjednoduš), ktorá prevedie vstupný výraz (ak je to možné) na jednoduchší tvar. Dá sa používať dvoma spôsobmi (viac sa o operátore // dozviete v sekcii 8.2).

```
In[2]:= Sin[x]^2 + Cos[x]^2
        Simplify[Sin[x]^2 + Cos[x]^2]
        Sin[x]^2 + Cos[x]^2 // Simplify

Out[2]= Cos[x]^2 + Sin[x]^2
        1
        1
```

Táto funkcia neslúži len na úpravu algebraických výrazov (obsahujúcich symboly), ale používa sa aj na výrazy číselné.

```
In[3]:= 3 - 1/(3 + Sqrt[11]) - 2(4 + Sqrt[11])/(3 + Sqrt[11])
```

$$\text{Out[3]=}\quad \frac{1}{2}(11 - 3\sqrt{11})$$

Niektoré výrazy sa nedajú zjednodušiť bez toho, aby softvér dostal zadané informácie o vstupnom výraze. V softvéri Mathematica sú všetky nezadefinované symboly považované za komplexné čísla.

```
In[4]:= Simplify[Sqrt[x^2]]
```

$$\text{Out[4]=}\quad \sqrt{x^2}$$

Predpoklady (po angl. assumptions) týkajúce sa vstupného výrazu môžeme zadať do funkcie **Simplify** pomocou voľby **Assumptions**.

```
In[5]:= Simplify[Sqrt[x^2], Assumptions → x>0]
```

```
Out[5]= x
```

Bez predpokladu, že premenná **x** je reálne číslo, neplatí rovnosť $\sqrt{x^2} = |x|, \forall x$, a teda softvér Mathematica vracia výraz v neupravenom tvare. Ak v našich výpočtoch pracujeme so symbolmi, o ktorých vieme, že sú prvkami (po angl. elements) množiny reálnych čísel alebo množiny celých čísel, môžeme túto informáciu zadať. Dá sa to urobiť nasledujúcimi spôsobmi.

```
In[6]:= Simplify[Sqrt[x^2], Assumptions → Element[x, Reals]]
        Simplify[Sqrt[x^2], Assumptions → x ∈ Reals]
```

```
Out[6]= Abs[x]
        Abs[x]
```

V softvéri Mathematica sú zabudované aj skrátené verzie príkazov pre zadávanie predpokladov (vynechanie **Assumptions->**).

```
In[7]:= Simplify[Sqrt[x^2], x>0];
        Simplify[Sqrt[x^2], Element[x, Reals]];
        Simplify[Sqrt[x^2], x ∈ Reals];
```

Upozorňujeme ešte na existenciu funkcie **Assuming**, pomocou ktorej sa tiež dajú zadávať predpoklady. Jej predpis je nasledovný: **Assuming[predpoklady,vyraz]**. Príkazy z predchádzajúcich príkladov by sa pomocou nej zapísali nasledovne.

```
In[8]:= Assuming[x>0, Simplify[Sqrt[x^2]]]
        Assuming[x ∈ Reals, Simplify[Sqrt[x^2]]]

Out[8]= x
        Abs[x]
```

Zápis predpokladov pomocou funkcie **Assuming** je síce komplikovanejší, ale dá sa aplikovať aj v situáciách, kde nie je možné použiť voľbu **Assumptions**, napr. zadanie predpokladov pre integrál zadaný pomocou symbolu $\int$, pozri str. 165.

Cvičenie 6.1

Symbol $\in$ sa dá zadať pomocou palety, pozri Pozn. 3.2 v podkapitole 3.1. Vyskúšajte zadať predchádzajúci príklad pomocou palety a pomocou klávesovej skratky.

Cvičenie 6.2

Funkcie $f(x) = ln(x)$ a $g(x) = e^x$ sú inverzné funkcie, a teda výraz $ln(e^x)$ sa dá napísať za určitých podmienok ako x. Použite funkciu **Simplify** na zjednodušenie výrazu $ln(e^x)$ s takým predpokladom na x, aby sme dostali výsledok x.

Funkcia **Simplify** má aj svoju rozšírenú verziu funkciu **FullSimplify** (z angl. plne zjednoduš). Tá je síce pomalšia, ale má v sebe zabudované ďalšie dodatočné úpravy výrazov súvisiace hlavne so špeciálnymi funkciami.

```
In[9]:= Simplify[x Gamma[x]]
        FullSimplify[x Gamma[x]]

Out[9]= x Gamma[x]
        Gamma[1 + x]
```

Funkcia **FullSimplify** vráti vždy aspoň taký jednoduchý výraz, aký by vrátila funkcia **Simplify**.

Cvičenie 6.3

To, že sú funkcie `x Gamma[x]` a `Gamma[1 + x]` totožné, sa dá overiť aj pomocou funkcie **Plot**. Zobrazte obidve funkcie na rovnakom grafe na definičnom obore $x \in (0, 1)$.

Ďalšie funkcie na úpravu výrazov môžeme podľa typu výrazov, na ktoré sa aplikujú, rozdeliť do troch skupín: funkcie slúžiace na prácu s polynómami, trigonometrickými výrazmi a zlomkami. Niektoré funkcie, napr. **Factor**, **ExpandAll**..., patria súčasne do dvoch kategórií.

6.2 Funkcie pre polynómy

Pri práci s polynómami sa najčastejšie používajú dve navzájom opačne fungujúce funkcie **Expand** (z angl. rozložiť) a **Factor**(z angl. činiteľ). Funkcia **Expand** rozloží zadaný výraz na polynóm.

```
In[10]:= Expand[(a + b)^4]

Out[10]= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4
```

Cvičenie 6.4

Rozložte výraz $(x-y)(x+y)$ na polynóm.

Funkcia **Factor** upraví zadaný polynóm na súčin polynómov čo najnižšieho stupňa.

```
In[11]:= Factor[a^4 + 4a^3 * b + 6a^2 * b^2 + 4a * b^3 + b^4]

Out[11]= (a + b)^4

In[12]:= Factor[x^10-1]

Out[12]= (-1 + x)(1 + x)(1 - x + x^2 - x^3 + x^4)(1 + x + x^2 + x^3 + x^4)
```

Cvičenie 6.5

Upravte výraz $a^3 + 3a^2b + 3ab^2 + b^3$ na súčin polynómov čo najnižšieho stupňa.

Cvičenie 6.6

Nájdite riešenia rovnice $x^2 + 5x - 84 = 0$.

Pomôcka: Ľavá strana rovnice $a\ x^2 + b\ x + c = 0$ sa dá napísať v tvare $a\ (x - r_1)(x - r_2)$, kde r_1 a r_2 sú riešenia rovnice.

Poznámka: V softvéri Mathematica existuje funkcia **Solve**, ktorá sa používa na riešenie takýchto rovníc (pozri kapitolu 11). Spôsob riešenia použitý v tomto cvičení je zameraný na precvičenie funkcie **Factor**.

Ak máme polynóm, ktorý obsahuje viacero členov s rovnakými **x^n**, môžeme členy s rovnakými mocninami sčítať pomocou funkcie **Collect[vstup,x]** (z angl. zhromaždi).

```
In[13]:= Collect[a^2 x^2 + a x^2 + a x, x]

Out[13]= a x + (a + a^2) x^2
```

V prípade, že potrebujeme zistiť len koeficient pred nejakým konkrétnym členom, nemusíme prevádzať celý výraz na súčet polynómov pomocou funkcie **Collect** alebo **Expand**, stačí použiť funkciu **Coefficient[vyraz,Y]**. Táto funkcia vracia koeficient pred členom **Y** v zadanom výraze.

```
In[14]:= Coefficient[(x + 1)^8, x^2]
         Coefficient[(x + 1)^8, x^5]
         Expand[(x + 1)^8]

Out[14]= 28
         56
```
$1 + 8x + 28x^2 + 56x^3 + 70x^4 + 56x^5 + 28x^6 + 8x^7 + x^8$

Cvičenie 6.7

Aký koeficient sa nachádza vo výraze $(a+b+c)^2$ pred členom $a+b$?

Funkciu **Expand** sme v predchádzajúcom príklade použili len pre overenie správnosti fungovania funkcie **Coefficient**. Všimnite si, že funkcia **Coefficient** nevracia len číselný koeficient, ale celú zvyšnú časť člena polynómu, z ktorého bol vybraný **Y**.

```
In[15]:= Coefficient[(a + b)^4, b^2]
         Expand[(a + b)^4]
```
Out[15]= $6a^2$

$a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$

Ak aplikujeme funkciu **Coefficient[vstup,Y]** na polynóm, v ktorom sa nachádza viacero členov obsahujúcich **Y**, tak výstupom funkcie bude súčet koeficientov týchto členov.

```
In[16]:= Coefficient[9x^2 + 4x^3 + 8x - 5x^2 - x^3 + 7x^2, x^2]
         Collect[9x^2 + 4x^3 + 8x - 5x^2 - x^3 + 7x^2, x]

Out[16]= 11
```
$8x + 11x^2 + 3x^3$

Ak potrebujeme previesť na tvar polynómu nielen čitateľ, ale aj menovateľ, použijeme namiesto funkcie **Expand** funkciu **ExpandAll**.

```
In[17]:= Expand[(a + b)^2 / (c + d)^2]
         ExpandAll[(a + b)^2 / (c + d)^2]
```
Out[17]= $\frac{a^2}{(c+d)^2} + \frac{2ab}{(c+d)^2} + \frac{b^2}{(c+d)^2}$

$\frac{a^2}{c^2 + 2cd + d^2} + \frac{2ab}{c^2 + 2cd + d^2} + \frac{b^2}{c^2 + 2cd + d^2}$

Na delenie polynóma polynómom sa používajú funkcie **PolynomialQuotient[p,q,x]** (z angl. polynomický podiel) a **PolynomialRemainder[p,q,x]** (z angl. polynomický zvyšok). Prvá z nich vracia polynóm **p** vydelený polynómom **q** (**x** je premenná týchto polynómov), druhá zvyšok po tomto delení.

```
In[18]:= p = 2x^3 + x^2 - 9x - 10;
         q = x + 1;
         PolynomialQuotient[p, q, x]
         PolynomialRemainder[p, q, x]

Out[18]= -8 - x + 2x^2
         -2
```

6.3 Funkcie pre trigonometrické výrazy

Na úpravu trigonometrických výrazov sa používa skupina funkcií začínajúcich slovom **Trig**. Zmienime sa len o tých najdôležitejších. Funkcia **TrigExpand** je rozšírením funkcie **Expand**, má v sebe naviac zabudované aj rozklady trigonometrických výrazov.

```
In[19]:= Expand[Sin[2x]]
         TrigExpand[Sin[2x]]

Out[19]= Sin[2x]
         2Cos[x]Sin[x]
```

Opačnou funkciou ku funkcii **TrigExpand** nie je funkcia **TrigFactor**, ale funkcia **TrigReduce**.

```
In[20]:= TrigFactor[2Cos[x]Sin[x]]
         TrigReduce[2Cos[x]Sin[x]]

Out[20]= 2Cos[x]Sin[x]
         Sin[2x]
```

Cvičenie 6.8

Rozložte výraz $\sin(\alpha + \beta)$.

Poznámka 6.1

V softvéri Mathematica je zabudovaná aj funkcia **Reduce**. Tá ale upravuje len rovnice a nerovnice (nie polynomické výrazy). Viac si o nej povieme v kapitole 11.

Funkcia **TrigFactor** prevádza trigonometrické výrazy na tvar súčinu.

```
In[21]:= TrigFactor[Sin[x] + Sin[y]]
```

$$\text{Out[21]= } 2\text{Cos}\left[\frac{x}{2} - \frac{y}{2}\right]\text{Sin}\left[\frac{x}{2} + \frac{y}{2}\right]$$

6.4 Funkcie pre zlomky

V softvéri Mathematica je zabudovaných viacero funkcií, ktoré nám uľahčia prácu so zlomkami. Predstavíme si len tie najpoužívanejšie.

Funkcia **Cancel** (z. angl. vykrátiť) upraví zlomok na základný tvar (vydelí čitateľa aj menovateľa najväčším spoločným deliteľom).

```
In[22]:= Cancel[(a - b) / (a^2 - 2a * b + b^2)]
```

$$\text{Out[22]= } \frac{1}{a - b}$$

Avšak, ak dostane funkcia **Cancel** ako vstup súčet zlomkov, upraví na základný tvar len tieto zlomky (samostatne, nesčíta ich).

```
In[23]:= Cancel[(x + 1) / (x^2 - 1) + (x - 1) / (x^2 - 1)]
```

$$\text{Out[23]= } \frac{1}{-1 + x} + \frac{1}{1 + x}$$

Cvičenie 6.9

Vykráťte zlomok $\frac{a^2-b^2}{a-b}$.

Funkcia **Together** (z angl. spoločne) upraví súčet zlomkov na jeden zlomok pomocou jednoduchého pravidla.

```
In[24]:= Together[a/b + c/d]
```

$$\text{Out[24]= } \frac{b\ c + a\ d}{b\ d}$$

Na výsledný zlomok sa naviac automaticky aplikuje krátenie (ako keby sme použili funkciu **Cancel**).

```
In[25]:= Together[x^2/(x^2 - 1) + x/(x^2 - 1)]
```

$$\text{Out[25]= } \frac{x}{-x + 1}$$

Podobne funguje aj funkcia **Factor**. Ak namiesto polynómu dostane ako vstup súčet zlomkov, sčíta ich a výsledný zlomok vráti upravený v základnom tvare. Táto funkcia na rozdiel od funkcie **Together** naviac čitateľa aj menovateľa rozloží na súčin polynómov čo najnižšieho stupňa.

```
In[26]:= Together[x^2 / (x^2 - 2x - 15) -1 / (x^2 - 2x - 15)]
         Factor[x^2 / (x^2 - 2x - 15) -1 / (x^2 - 2x - 15)]
```

$$\text{Out[26]=}\quad \frac{\frac{-1 + x^2}{-15 - 2x + x^2}}{\frac{(-1 + x)(1 + x)}{(-5 + x)(3 + x)}}$$

Cvičenie 6.10

Upravte súčet $\frac{a}{a+b} + \frac{b}{a-b}$ na jeden zlomok.

Posledná funkcia, s ktorou sa bližšie oboznámime, je funkcia **Apart**. Táto funkcia upraví vstupujúci zlomok na súčet parciálnych zlomkov.

```
In[27]:= Apart[(9x^2 - 2x - 8 )/(x^3 -4x)]
```

$$\text{Out[27]=}\quad \frac{3}{-2 + x} + \frac{2}{x} + \frac{4}{2 + x}$$

```
In[28]:= Apart[(x - 2) / (x^4 + 3x^3 + 4x + 12)]
```

$$\text{Out[28]=}\quad \frac{5}{23(3 + x)} + \frac{-22 + 15x - 5x^2}{23(4 + x^3)}$$

Môžeme ju teda považovať za opačnú funkciu k funkcii **Together**.

Prelínanie vlastností vstavaných funkcií softvéru Mathematica je spôsobené postupným vývojom tohto softvéru. Ten sa stále zdokonaľuje, doplňujú sa doň nové funkcie a aj do tých už existujúcich nové vlastnosti, preto sa funkčnosť vstavaných funkcií môže líšiť v závislosti od konkrétnej verzie.

6.5 Zhrnutie

V tejto kapitole sme si predstavili najdôležitejšie vstavané funkcie používané na upravovanie výrazov. Pomocou nich sa dajú upravovať nielen číselné, ale aj algebrické (obsahujúce symboly) výrazy. Upozornili sme, že softvér Mathematica implicitne považuje všetky symboly za komplexné čísla a ukázali sme, ako sa dajú zadať podmienky (predpoklady) na množinu, do ktorej symbol patrí. Základnou vstavanou funkciou na úpravu výrazov je **Simplify**, existuje k nej aj rozšírená verzia **FullSimplify**. Oboznámili sme sa aj s ďalšími vstavanými funkciami, ktoré sa používajú pri práci s polynómami, trigonometrickými výrazmi a zlomkami.

S výrazmi sa budeme stretávať aj v nasledujúcich kapitolách, napr. pri deriváciách a integráloch. Výraz totiž môže slúžiť aj ako predpis matematickej funkcie. Pri používaní **Plot** sme tiež zobrazovanú funkciu zadávali ako výraz.

Kapitola 7

Nahrádzanie a vzory

Dôležitou súčasťou softvéru Mathematica je používanie nahrádzania a vzorov. Nahradenie sa zadáva pomocou tzv. transformačného pravidla: **vzor→nahradenie**. Dosadzovací operátor **/.** pozostáva z dvoch znakov: lomítka a bodky bez medzery medzi nimi.

```
In[1]:= Sin[x] + x /. x→2

Out[1]= 2 + Sin[2]
```

V klasickej matematike existuje ekvivalentný zápis $\sin(x)+x\big|_{x\to 2}$. Využíva sa hlavne pri deriváciach (napr. $\frac{\mathrm{d}\sin(x)}{\mathrm{d}x}\big|_{x\to 2}$) z dôvodu, že najskôr sa výraz musí podľa premennej x zderivovať a až potom sa vo výslednom výraze nahrádza x hodnotou 2.

Dosadzovací operátor **/.** sa využíva aj následne po príkazoch, ktoré vo svojich výstupoch vracajú výsledky v tvare transformačného pravidla, napr. po funkciách **Solve** a **FindRoot**.

```
In[2]:= dosadenie = FindRoot[{x + 2y == 7, 3x - y == 1}, {{x, 1}, {y, 1}}]

Out[2]= {x→1.28571, y→2.85714}

In[3]:= x^2 + y^2 /. dosadenie

Out[3]= 9.81633
```

7.1 Dosadzovanie hodnôt za symboly vo výrazoch

Pri komplexnejších úlohách častokrát potrebujeme odvodiť jednotlivé na seba nadväzujúce kroky všeobecne (t. j. so symbolmi, bez číselných hodnôt) a až do výsledku chceme dosadiť číselné hodnoty. Toto sa využíva napr. ak chceme riešiť viacero úloh rovnakého typu s rôznymi vstupnými parametrami.

```
In[1]:= y + x /. x→b
```

```
Out[1]= b + y
```

Cvičenie 7.1

Nahraďte vo výraze $ln(y) + y$ symbol y číslom e.

Symbol sa dá nahradiť aj priamo v premennej, v ktorej je uložený výraz so symbolom, ktorý chceme nahradiť.

```
In[2]:= vysl = x^2 - 5x
        vysl /. x→2
        vysl

Out[2]= -5x + x^2
        -6
        -5x + x^2
```

Ak chceme výraz po nahradení uložiť do premennej, môžeme to urobiť nasledovne.

```
In[3]:= vysl = x^2 - 5x /. x→2
        vysl

Out[3]= -6
        -6
```

Keď dosádzame súčasne viacero hodnôt, môžeme ich zadať ako zoznam nahradení (alebo pomocou premennej urobiť postupne viacero nahradení za sebou).

```
In[4]:= 2x + 3y - x * y
        2x + 3y - x * y /. {x→2, y→-1}
        pom = 2x + 3y - x * y /. x→2
        pom /. y→-1

Out[4]= 2x + 3y - xy
        3
        4 + y
        3
```

Za symboly sa dajú dosádzať nielen číselné hodnoty, ale aj iné výrazy. Na poradí, v akom sú hodnoty dosadzované, nezáleží.

```
In[5]:= x + y
        x + y /. {x→2, y→x+3}
        x + y /. {y→x+3, x→2}
```

```
Out[5]= x + y
        5 + x
        5 + x
```

Cvičenie 7.2

Vo výraze $a^2 + b^2$ nahraďte symbol a výrazom $\sin(x)$ a symbol b výrazom $\cos(x)$.

Všimnite si, že v poslednom príklade pri dosadzovaní za `y` pomocou dosadzovacieho operátora je premenná `x` považovaná za symbol aj napriek tomu, že sme súčasne za `x` dosadili číselnú hodnotu `2`. Ak však namiesto dosadzovacieho operátora použijeme priraďovací príkaz, výsledok bude číselná hodnota.

```
In[6]:= y = x + 3; x = 2;
        x + y

Out[6]= 7
```

Dosadzovací operátor `/.` dosadí za symbol (symboly) len v jednom konkrétnom výraze (má lokálny charakter) na rozdiel od priraďovacieho príkazu s globálnym charakterom. Po priradení sa zo symbolov stávajú premenné a svoju hodnotu si uchovávajú aj v nasledujúcich krokoch. Priradenie sa dá zmeniť predefinovaním pomocou nového priraďovacieho príkazu alebo ho môžeme odstrániť príkazom `Clear` (z premennej sa opäť stane symbol).

Softvér Mathematica umožňuje nahradiť vo výraze nielen symboly, ale aj zložitejšie výrazy, ba dokonca aj nejaké konkrétne číslo iným číslom.

```
In[7]:= s^2 + 1/2 + s^2 /. {s^2→2, 1/2→100}

Out[7]= 104
```

V takýchto situáciach je však potrebné dávať pozor na to, ako softvér Mathematica upravuje výrazy. Pred nahradením totiž softvér Mathematica najskôr upraví výraz, v ktorom nahrádzame. V prípade nepozornosti (alebo neznalosti) môžeme dostať prekvapujúce výsledky.

```
In[8]:= (s^2 + 1/2)/s^2 /. {s^2→2, 1/2→100}

Out[8]= 102/s^2
```

V predchádzajúcom príklade softvér Mathematica považuje výraz $\frac{102}{s^2}$ za súčin `102 s^-2` a teda `s^-2` sa nezhoduje s členmi, ktoré majú byť nahradené. Toto ale nie je zrejmé z výstupu, ktorý nám softvér Mathematica vracia. Ako je výraz interne reprezentovaný vieme zistiť pomocou funkcie `FullForm`. Jej výstupom je tvar vstupného výrazu, v ktorom je tento výraz uložený v pamäti softvéru a s ktorým softvér Mathematica priamo pracuje.

```
In[9]:= % // FullForm

Out[9]= Times[102, Power[s, -2]]
```

Cvičenie 7.3

Ak vo výraze $(x^2 + 1/2)/2 + x^2$ nahradíme x^2 číslom 2 a $1/2$ číslom 100 dostaneme nečakaný výsledok. Vysvetlite prečo.

Ak potrebujeme používať to isté transformačné pravidlo opakovane, je výhodné uložiť ho do nejakej premennej.

```
In[10]:= dosadenie = {s→3, t→5};
         s + t /. dosadenie
         s - t /. dosadenie
         s * t /. dosadenie

Out[10]= 8
         -2
         15
```

Ak aplikujeme na výraz zoznam zoznamov transformačných pravidiel, výstupom je zoznam výrazov s nahradeniami.

```
In[11]:= vaha/objem /. {{vaha→20.3, objem→1.1}, {vaha→31.2, objem→1.5}}

Out[11]= {18.4545, 20.8}
```

Cvičenie 7.4

Vo výraze $x^2 + x - 12$ nahraďte symbol x číslom 3 a číslom -4 tak, aby výsledok bol zoznam nahradení.

Výstup v tvare zoznam zoznamov nám vracia napr. funkcia **Solve**, keď má rovnica alebo sústava rovníc viac riešení. Viac si o tejto funkcii povieme v kapitole 11.

Cvičenie 7.5

Príkaz `Solve[x^2 + x - 12 == 0, x]` vracia riešenie rovnice $x^2 + x - 12 = 0$ ako zoznam nahradení. Nahraďte vo výraze $x^2 + x - 12$ všetky x riešeniami rovnice pomocou výstupu funkcie **Solve**.

Na záver si predstavíme ešte jeden spôsob nahrádzania pomocou dosadzovacieho operátora a to prípad, keď nahrádzame nie konkrétny výraz, ale všeobecný vzor (po angl. pattern). Všeobecný vzor sa zadáva pomocou podčiarkovníka _ (v softvéri Mathematica sa označuje pojmom "blank", t. j. prázdne miesto). Reprezentuje ľubovoľný výraz nachádzajúci sa v nejakom konkrétnom zložitejšom výraze.

```
In[12]:= x^2 + y^2 /. _^2→a

Out[12]= 2a
```

V predchádzajúcom príklade je napravo od dosadzovacieho operátora zadané pravidlo, že "*čokoľvek* na druhú" nachádzajúce sa vo výraze naľavo od dosadzovacieho operátora sa nahradí symbolom a.

Toto *čokoľvek* (zadané znakom podčiarkovníka) je dokonca možné pomenovať (jeho názov zadávame tesne pred znak podčiarkovníka) a to nám umožňuje nahrádzať naľavo od dosadzovacieho operátora *čokoľvek* výrazom obsahujúcim toto *čokoľvek* v nejakom presne predpísanom tvare. Táto vlastnosť je prezentovaná na nasledujúcom príklade.

```
In[13]:= a^b + c^d /. x_^y_→y^x

Out[13]= b^a + d^c
```

Poznámka 7.1

Existuje aj oneskorená verzia nahradenia. Zadáva sa pomocou znakov :>, ktoré sa po stlačení medzerovníka automaticky skonvertujú na symbol ↦. Na rozdiel od klasického nahrádzania tu dochádza k vyčísleniu až po nahradení. Tento rozdiel výstižne ilustruje nasledujúci príklad.

```
In[14]:= {x, x} /. x→RandomReal[]

Out[14]= {0.365978, 0.365978}

In[15]:= {x, x} /. x:→RandomReal[]

Out[15]= {0.711075, 0.883911}
```

V prvom prípade boli vo výstupe obidve čísla rovnaké, keďže najskôr prebehlo vyčíslenie (vygenerovanie náhodného čísla) a až potom nahradenie. V druhom prípade najskôr prebehlo nahradenie a až potom vyčíslenie (vygenerovanie náhodných čísel).

7.2 Funkcie využívajúce vzory

Používanie vzorov predstavuje silný nástroj softvéru. So vzormi sa stretávame nielen pri nahradzovaní, ale využívajú sa aj vo viacerých vstavaných funkciách. S niektorými z nich sa teraz oboznámime.

Funkcia **Cases[zoznam, vzor]** vracia výstupný zoznam prvkov, ktoré sa zhodujú zo vzorom.

```
In[16]:= Cases[{a + b, a + a, c + Sin[x], 5 + 7, 4}, x_ + y_]

Out[16]= {a + b, c + Sin[x]}
```

Všimnite si, že vo výstupe nie je **a+a** a tak isto ani **5+7**, keďže najskôr sa výrazy vyhodnotili na **2a** a **12** a až potom sa overovalo, či sa zhodujú so vzorom. **Cases[zoznam, vzor→nahradenie]** umožňuje využiť nahradzovacie pravidlo na nahradenie prvkov, ktoré spĺňajú vzor.

```
In[17]:= Cases[{a + b, a + a, c + Sin[x], 5 + 7, 4}, x_ + y_→1 + y]

Out[17]= {1 + b, 1 + Sin[x]}
```

Príkaz **Position[zoznam, vzor]** vracia indexy prvkov, ktoré sa zhodujú so vzorom.

```
In[18]:= Position[{a + b, a + a, c + Sin[x], 5 + 7, 4}, x_ + y_]

Out[18]= {{1}, {3}}
```

Táto funkcia sa dá aplikovať aj na vnorené zoznamy.

```
In[19]:= Position[{{a, a, a}, a, a}, a]

Out[19]= {{1, 1}, {1, 2}, {1, 3}, {2}, {3}}
```

Prvá dvojica výstupu reprezentuje informáciu, že v prvom prvku zoznamu sa nachádza ďalší zoznam a jeho prvý prvok sa zhoduje so vzorom, druhá a tretia dvojica informáciu, že druhý aj tretí prvok tohto podzoznamu sa tiež zhodujú so vzorom a štvrté a piate číslo informáciu, že aj druhý a tretí prvok zoznamu sa tiež zhodujú so vzorom.

Pomocou funkcie **DeleteCases[zoznam, vzor]** môžeme zo zoznamu odstrániť prvky zhodujúce sa so vzorom.

```
In[20]:= DeleteCases[{a^2, b + c, Sin[x^2], 5^E}, x_^y_]

Out[20]= {b + c, Sin[x^2]}
```

Všimnite si, že **x^2** nebolo odstránené (keďže je len argumentom funkcie a nie samostatným prvkom) a naopak prvok **5^E** bol odstránený (keďže ho softvér Mathematica nevie presne vyčísliť a tak zostal v nevyčíslenom tvare zhodujúcom sa so vzorom).

7.3 Zhrnutie

V tejto kapitole sme si predstavili dosadzovací operátor `/.`, ktorý umožňuje nahrádzanie symbolov a výrazov číselnými hodnotami, symbolmi a výrazmi. Upozornili sme, že pri dosadzovaní do výrazov je dôležité dávať pozor, v akom tvare s týmto výrazom softvér Mathematica pracuje. Táto informácia je výstupom funkcie **`FullForm`**. Oboznámili sme sa aj s oneskoreným nahrádzaním, ktoré sa používa pri nahrádzaní výstupmi niektorých funkcií (keď potrebujeme, aby vyčíslenie prebehlo až po dosadení).

Ďalej sme predstavili vzory. Zadávajú sa pomocou symbolu `_` a reprezentujú ľubovoľný výraz nachádzajúci sa v nejakom konkrétnom zložitejšom výraze. Vysvetlili sme ich úlohu nielen pri nahrádzaní, ale aj v rámci viacerých funkcií, v ktorých sa používajú. So vzormi sa stretneme v kapitole 8 o vytváraní užívateľských funkcií a stretli sme sa s nimi aj pri zobrazovaní funkcií zadaných ako `f[x_]:= predpis funkcie`. Svoje miesto zaujímajú vzory aj pri definovaní tzv. riedkych matíc, ktoré si predstavíme v kapitole 12.

Kapitola 8

Vytváranie užívateľských funkcií

Na vývoji softvéru Mathematica pracujú stovky programátorov. Do každej jeho novej verzie sú dopĺňané ďalšie vstavané funkcie a v súčasnosti ich je už viac ako 300 000, pozri podsekciu. 2.2. Avšak aj napriek tomuto obrovskému množstvu zabudovaných funkcií je pri riešení niektorých úloh veľmi prospešné používať vlastné (užívateľské) funkcie. Týmto termínom sú označované funkcie, ktoré užívateľ naprogramuje sám. Využívajú sa zvyčajne v úlohach, v ktorých sa niektoré príkazy (alebo skupiny príkazov) vyskytujú opakovane. Uplatnenie nachádzajú aj vo veľmi dlhých programoch, ktoré sa použitím užívateľských funkcií stávajú prehľadnejšími.

V softvéri Mathematica existuje viacero spôsobov, ako definovať užívateľské funkcie. Prvý sa podobá na matematický zápis funkcie.

```
In[1]:= f[x_]:= x^2
```

Druhý spôsob je pomocou **Function** (z angl. funkcia).

```
In[2]:= g = Function[x, x^2]
```

Obidve takto zadané funkcie sa volajú rovnakým spôsobom ako vstavané funkcie.

```
In[3]:= f[3]
        g[3]

Out[3]= 9
        9
```

V nasledujúcich sekciách si podrobnejšie predstavíme obidva prístupy. Popíšeme ich spoločné črty a odlišnosti; a taktiež výhody a nevýhody ich použitia v rôznych aplikáciách.

8.1 Funkcie matematického štýlu

Ide o najbežnejší spôsob definovania funkcií v softvéri Mathematica. Ako príklad uvádzame nasledujúce definície funkcií.

```
In[4]:= f[x_]:= x + Cos[x]
        vysledok[x_]:= 2x - 3Exp[x]
```

Za povšimnutie stojí, že sa nevytvorila výstupná bunka. Je to spôsobené tým, že pri definovaní funkcií sa nepoužíva klasické priradenie (pomocou znaku **=**), ale tzv. oneskorené priradenie pomocou znaku **:=**. Podrobnejšie vysvetlenie tohto znaku sa nachádza v podsekcii 8.1.1.

Výrazy na ľavej strane **f** a **vysledok** sú názvy užívateľských funkcií. **x** je názov lokálnej premennej, pozri Pozn. 4.2. Vo vstupe funkcie musí byť nutne nasledovaný podčiarkovníkom **_**. V softvéri Mathematica je tento znak označovaný angl. termínom "blank". Zabezpečí, že vstupom užívateľskej funkcie môže byť akýkoľvek výraz. Na pravej strane definície funkcie sa podčiarkovníky za premennými nepíšu. Podrobnejšie sme sa s podčiarkovníkmi oboznámili v kapitole 7.

Cvičenie 8.1

Definujte funkciu, ktorá má matematický predpis $f(x) = x^2 + e^x$. Akú hodnotu nadobúda táto funkcia v bode $x = 3$?

Cvičenie 8.2

Zobrazte graf tejto funkcie na definičnom obore (-1,1), pozri kapitolu 5.

Funkcie **f** a **vysledok** boli definované pre jeden vstupný argument. Keď sú tieto funkcie zavolané s dvoma (alebo viacerými) vstupnými argumentami, softvér Mathematica vráti zadaný výraz späť.

```
In[5]:= f[1, 1]

Out[5]= f[1, 1]
```

Vo väčšine programovacích jazykoch by sa v takejto situácii zobrazilo chybové hlásenie. Softvér Mathematica je v tomto odlišný. Keď dostane ako vstup niečo, čo nie je definované (napr. **f[1,1]** z predchádzajúceho príkladu), tak vo výpočtoch pracuje s takýmto výrazom, ako keby to bol symbol.

Táto definícia funkcie má však jednu nevýhodu. V prípade, že v premennej, ktorá bude použitá ako názov funkcie, je uložená nejaká hodnota, softvér zobrazí chybové hlásenie.

```
In[6]:= g = 5;
        g[x_]:= x

        SetDelayed: Tag Integer in 5[x_] is Protected. ≫

Out[6]= $Failed
```

Z tohto hlásenia je zrejmé, že softvér vyhodnotil naše zadané príkazy ako žiadosť o zadefinovanie funkcie s názvom **5**. Táto nezrovnalosť je zapríčinená faktom, že premennej **g** sme priradili hodnotu 5 ešte pred definovaním rovnomennej funkcie.

Takže, ak má byť funkcia zadefinovaná názvom premennej, ktorá bola pri predchádzajúcej práci použitá, musíme túto premennú najskôr uvoľniť (odstrániť jej priradenie v pamäti) pomocou vstavanej funkcie **Clear**, pozri str. 19.

```
In[7]:= Clear[g]
        g[x_]:= x
```

Funkcie s viacerými vstupnými argumentami sa definujú obdobne, pridaním ďalších lokálnych premenných.

```
In[8]:= u[x_,y_,z_]:= x^2 + y^2 + z
        u[3, 3, 2]

Out[8]= 20
```

Vstup funkcie môže byť zložitejší ako len n-tica vstupných parametrov. Napr. v nasledujúcom príklade je vstupom zoznam dĺžky dva (t. j. dvojprvkový zoznam). Lokálny názov prvého prvku vstupného zoznamu je **x** a druhého **y**.

```
In[9]:= v[{x_,y_}]:= x^2 + y^2

In[10]:= x = {3, 3};
         v[x]

Out[10]= 18
```

Cvičenie 8.3

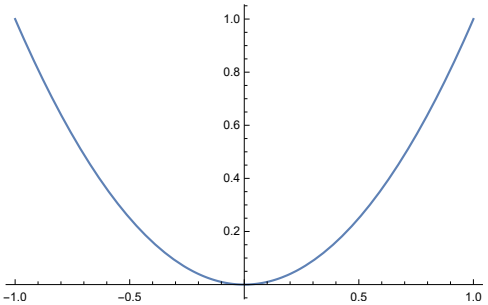
Vytvorte funkciu, ktorej vstupným argumentom bude zoznam prvkov a jej výstupom náhodný prvok tohto zoznamu.

Vstupnými argumentmi nasledujúcej užívateľskej funkcie sú samostatná lokálna premenná a zoznam lokálnych premenných. Pripomeňme, že takýto typ vstupu má aj vstavaná funkcia **Table**, pozri Pozn. 4.4.

```
In[11]:= integralPlot[vyraz_, {x_, a_, b_}]:= {Integrate[vyraz, {x, a, b}],
                                                Plot[vyraz, {x, a, b}]}
```

Výstupom tejto užívateľskej funkcie bude teda zoznam obsahujúci dva prvky. Prvým prvkom je číslo – výstup vstavanej funkcie **Integrate** (pozri podsekciu 10.2.1), ktorá slúži na výpočet určitého integrálu. Druhým prvkom je graf zadaného argumentu **vyraz**.

```
In[12]:= Clear[x];
         integralPlot[x^2, {x, -1, 1}]
```

Out[12]= $\{\frac{2}{3},$  $\}$

Pred použitím funkcie **integralPlot** so vstupným parametrom **x** musí byť najskôr premenná **x** uvoľnená. Ak by sa to neurobilo, tak by do funkcie **integralPlot** nebol dosadený symbol **x**, ale hodnota v ňom uložená. Následkom toho by funkcia **Integrate** dostala ako vstupný parameter číslo a nie symbol, podľa ktorého má integrovať. To by zapríčinilo prerušenie výpočtu a zobrazenie chybového hlásenia upozorňujúceho na konflikt vo vstupnom argumente.

V niektorých vstavaných funkciách, napr. **Plot** alebo **Table** je tento problém prekonaný. Vstupné premenné sú zabudované ako lokálne, takže hodnota rovnomenných globálnych premenných ich neovplyvňuje.

Cvičenie 8.4

Uložte do premennej **x** nejakú hodnotu a zavolajte funkciu **integralPlot[x^2, {x, -1, 1}]**. Zistite, aké chybové hlásenie softvér Mathematica vypíše a aký výstup táto funkcia vráti.

Je dobré oboznámiť sa s možnými chybami, aby ste v prípade vlastných chýb v zložitejších funkciách vedeli, v ktorej časti kódu ich hľadať.

8.1.1 Oneskorené priradenie :=

Premenným môžu byť priradené hodnoty alebo dáta pomocou tzv. klasického priradenia. Výraz na pravej strane od znamienka **=** sa vyhodnotí, uloží do premennej na ľavej strane a zobrazí vo výstupnej bunke.

```
In[13]:= x = 3;
         b = x + 1

Out[13]= 4
```

Ak je pri priradení použité namiesto klasického priradenia tzv. oneskorené priradenie (t. j. namiesto znamienka **=** znamienko **:=**), výraz na pravej strane sa bez vyhodnotenia uloží do premennej na ľavej strane. Výstupná bunka sa nezobrazí.

```
In[14]:= c:= x + 1
```

Výraz sa vyčísli (a jeho hodnota zobrazí vo výstupnej bunke) až po zavolaní premennej.

```
In[15]:= c

Out[15]= 4
```

Keďže predpis uložený vo funkcii **c** je závislý na **x**, v prípade zmeny **x** sa zmení aj hodnota, ktorú vráti funkcia **c**.

```
In[16]:= x = 8;
         c

Out[16]= 9
```

Cvičenie 8.5

Vytvorte príkaz, ktorý po zavolaní vykreslí graf vždy inej náhodne generovanej postupnosti 10 čísel.

Užívateľské funkcie sa definujú pomocou oneskoreného priradenia. Na nasledujúcom príklade je ukázané, čo by sa stalo, ak by v nich bolo použité klasické priradenie.

```
In[17]:= h[x_] = x + Cos[x]

Out[17]= 8 + Cos[8]
```

Rozdiel medzi obidvoma typmi definícií je zrejmý už v ich predpisoch. V prípade použitia oneskoreného priradenia je premenná **x** na pravej strane zobrazená zelenou farbou (pozri napr. prvý príklad v sekcii 8.1), zatiaľ čo pri použití klasického – čiernou. To vyjadruje, že v prvom prípade ide o lokálnu premennú a v druhom o premennú, ktorej je niečo priradené. Pri volaní funkcie **h** (s klasickým priradením) dostávame vždy rovnaký výsledok bez ohľadu na vstupný argument.

```
In[18]:= h[5]

Out[18]= 8 + Cos[8]
```

Aj pomocou klasického priradenia vieme zadefinovať užívateľskú funkciu tak, aby fungovala ako klasická matematická funkcia, ale názov jej vstupného argumentu nesmie byť totožný s názvom nejakej globálnej premennej. Ak taká premenná už existuje, alebo si nie sme istí, či sme premennú s rovnakým názvom nepoužívali pri predchádzajúcej práci, stačí na jej uvoľnenie použiť funkciu **Clear**.

```
In[19]:= Clear[x]
         h[5]

Out[19]= 8 + Cos[8]
```

```
In[20]:= h[x_] = x + Cos[x]
         h[5]

Out[20]= x + Cos[x]
         5 + Cos[5]
```

V niektorých užívateľských funkciách však oneskorené priradenie spôsobuje problémy (vzhľadom na absenciu vyhodnotenia výrazu), zatiaľ čo rovnaké funkcie s klasickým priradením fungujú správne. Ako príklad uvádzame užívateľskú funkciu, v ktorej definícii sa využíva vstavaná funkcia **D** slúžiaca na derivovanie (pozri kapitolu 10).

```
In[21]:= Clear[x]
         h1[x_] = D[x + Cos[x], x]
         h1[4]

Out[21]= 1 - Sin[x]
         1 - Sin[4]
```

Pri použití klasického priradenia je výraz najskôr vyhodnotený (t. j. v predchádzajúcom príklade zderivovaný) a potom priradený premennej na ľavej strane. Takže funkcii **h1** bol priradený výsledok vyhodnocovania t. j. **1 - Sin[x]**.

Pri použití oneskoreného priradenia je výraz priradený premennej bez vyhodnotenia

```
In[22]:= h2[x_]:= D[x + Cos[x], x]
```

Takže funkcii **h2** bola priradená funkcia derivujúca výraz **x + Cos[x]** podľa **x**. Ak zavoláme funkciu **h2** so vstupným argumentom **4**, softvér Mathematica to vyhodnotí ako požiadavku zderivovať **4 + Cos[4]** podľa čísla **4** a vypíše chybové hlásenie.

```
In[23]:= h2[4]

         General::ivar: 4 is not a valid variable.

Out[23]= ∂4(4 + Cos[4])
```

Užívateľská funkcia z predchádzajúceho príkladu bude fungovať správne aj s oneskoreným priradením, ak na pravú stranu doplníme vstavanú funkciu **Evaluate** (z angl. vyhodnoť). Táto funkcia zabezpečí, že výraz sa pred priradením do premennej vyhodnotí.

```
In[24]:= h3[x_]:= Evaluate[D[x + Cos[x], x]]
         h3[4]

Out[24]= 1 - Sin[4]
```

Rovnako ako pri vstavaných funkciách, môžeme aj pri nami definovanej funkcii použiť **?** pred jej názvom (pozri str. 25), aby sme získali definíciu tejto funkcie.

```
In[25]:= ?h3
```

```
Out[25]=
Symbol
Global`h3
Definitions  h3[x_] := 1 – Sin[x]
Full Name    Global`h3
```

Cvičenie 8.6

Pomocou **?** zistite, aké informácie Vám vypíše softvér Mathematica o jednotlivých užívateľských funkciách zadefinovaných v tejto podsekcii.

Cvičenie 8.7

Výraz **Sin[a]Cos[b] - Cos[a]Sin[b]** sa dá upraviť na jednoduchší tvar. Ak by sme pomocou tohto výrazu definovali funkciu dvoch premenných **a** a **b**, vykonalo by sa zbytočne veľa operácií. Zadefinujte funkciu **sinamb**, ktorá bude ako výstup vracať zjednodušený výraz z výrazu **Sin[a]Cos[b] - Cos[a]Sin[b]**. Použite funkciu **Simplify** zo sekcie 6.1.

Pomocou príkazu **?sinamb** overte, či je skutočne vo funkcii **sinamb** definovaná zjednodušená forma výrazu **Sin[a]Cos[b] - Cos[a]Sin[b]**.

8.1.2 Podčiarkovník _

Podčiarkovník za lokálnou premennou vo vstupe užívateľskej funkcie umožní, aby vstupným argumentom tejto funkcie mohol byť akýkoľvek výraz. Ak by sme ho vynechali, vytvorilo by sa len jediné priradenie (pre **x**). Takže užívateľská funkcia by fungovala iba v prípade, ak by sme jej ako vstupný parameter zadali **x**.

```
In[26]:= k[x]:= x + 1
         k[7]

Out[26]= k[7]
```

Z výstupu predchádzajúceho príkladu by sa mohlo zdať, že funkcia **k** nebola definovaná. V skutočnosti však bola definovaná len pre jediný vstupný argument **x**.

```
In[27]:= k[x]
```

```
Out[27]= x + 1
```

Ak by premennej **x** bola ešte pred definovaním funkcie priradená nejaká hodnota, napr. číslo 7, tak by funkcia bola definovaná pre vstupný argument, ktorým by bola táto priradená hodnota (a zároveň aj pre **x** alebo inú premennú, ktorá má priradenú hodnotu 7).

```
In[28]:= Clear[k]
         x = 7;
         k[x]:= x + 1
         k[7]
         k[x]

Out[28]= 8
         8
```

Keď si dáme vypísať definíciu funkcie **k**, softvér Mathematica nám zobrazí informáciu, že táto funkcia je definovaná pre hodnotu 7.

```
In[29]:= ?k
```

```
Out[29]=
Symbol
Global`k
Definitions  k[7] = 8
Full Name  Global`k
```

Funkcie (nielen tie v softvéri Mathematica) sú v skutočnosti len predpisy pre nahradzovanie. Pre definovanú funkci **f[x_]:=x^2** volanie **f[2]** znamená: nahraď čokoľvek (reprezentované vstupným argumentom pomenovaným **x**) hodnotou 2 a vypočítaj výraz.

Poznámka 8.1

Ak vo vstupe funkcie nahradíme podčiarkovník výrazom **_head**, kde namiesto **head** napíšeme konkrétny návratový typ funkcie **Head** (pozri Pozn. 3.5), množina možných vstupných výrazov sa zúži len na tento konkrétny návratový typ funkcie **Head**.

```
In[30]:= f1[x_]:= 10x + 3
         f2[x_Integer]:= 10x + 3
         f1[2]
         f1[2.5]
         f2[2]
         f2[2.5]
```

```
Out[30]= 23
         28.
         23
         f2[2.5]
```

8.1.3 Definovanie funkcie pre rôzne vstupné argumenty

Keďže užívateľské funkcie sa dajú definovať pre rôzne číselné vstupy, môže vyvstať otázka, čo sa stane, ak zadefinujeme funkciu pre nejaký číselný vstup a potom funkciu s rovnakým menom pre iný číselný vstup. Premaže druhá definícia prvú, alebo budú súčasne platné obidve? Ukážeme si to názorne na nasledujúcom príklade.

```
In[31]:= m[1]:= 7
         m[2]:= 15

In[32]:= m[1]
         m[2]

Out[32]= 7
         15
```

Z výstupov je zrejmé, že v platnosti zostala aj prvá definícia. Potvrdíme si to aj zobrazením definície funkcie `m`.

```
In[33]:= ?m
```

```
Out[33]= Symbol
         Global`m
         Definitions  m[1] := 7
                      m[2] := 15
         Full Name    Global`m
```

Funkcie definované pre konkrétne čísla sa dajú používať ako indexované premenné. Môžeme ich považovať za alternatívnu verziu ku zoznamom.

Poznámka 8.2

Vždy, keď zavoláme funkciu `m[vstup]`, softvér Mathematica vyhodnotí, či sa `vstup` zhoduje so vzorom `m[1]` a ak áno, výstupom bude číslo 7. Ak sa nezhoduje, postupuje vyhodnocovanie ďalej až dovtedy, kým sa nenájde taký vzor, s ktorým sa `vstup` zhoduje. Ak skontroluje všetky vzory (definície) a `vstup` sa nezhoduje so žiadnym z nich, tak vráti ako výstup `m[vstup]`.

Poznámka 8.3

Definovanie funkcie pre rôzne vstupné parametre sa označuje termínom ”preťaženie funkcie” (po angl. function overloading). S týmto pojmom sa často stretávame aj v iných programovacích jazykoch.

Funkciu je možné dodefinovať aj pre všeobecný vstup.

```
In[34]:= m[x_]:= Cos[x]
```

V takomto prípade aj napriek tomu, že by všeobecná definícia mohla platiť aj pre predtým zadefinované číselné vstupy, majú vyššiu prioritu definície pre konkrétne vstupy.

```
In[35]:= m[1]
         m[2]
         m[4]

Out[35]= 7
         15
         Cos[4]
```

Potvrdíme si to opäť zobrazením definície funkcie `m`.

```
In[36]:= ?m
```

Out[36]=

Symbol	
Global`m	
Definitions	m[1] := 7 m[2] := 15 m[x_] := Cos[x]
Full Name	Global`m

V softvéri Mathematica rovnako ako vo väčšine moderných programovacích jazykov sú podporované rekurzívne funkcie. Pod pojmom ”rekurzívne funkcie” rozumieme v programovacom jazyku funkcie, ktoré odkazujú sami na seba. To znamená, že obsahujú vo svojom tele príkaz na volanie samej seba. Výborným príkladom rekurzívnej funkcie je zadefinovanie faktoriálu.

```
In[37]:= faktorial[n_]:= n faktorial[n-1]
```

Pozorný čitateľ si všimne, že takto definovaná funkcia bude volať samú seba nekonečnokrát. Tento problém je zapríčinený tým, že funkcia nemá definovaný tzv. základný prípad, teda vstup, pre ktorý vie vypočítať hodnotu bez rekurzívneho volania. Pre faktoriál je to hodnota 1 pre vstup $n = 0$. Prirodzenou voľbou pre doplnenie základného prípadu je použitie klasickej programátorskej

podmienky **If**. Avšak s programátorskými podmienkami sa oboznámime až v druhom diele tejto učebnice a keďže v programátorskom jazyku je dobré držať sa štýlu, ktorý je pre jazyk prirodzený, základný prípad pridáme pomocou definície funkcie pre vstup 0.

```
In[38]:= faktorial[0] = 1;
```

Nakoniec overíme, či naša funkcia funguje správne.

```
In[39]:= faktorial[5]

Out[39]= 120
```

Pripomeňme, že v softvéri Mathematica existuje aj vstavaná funkcia **Factorial**, pozri sekciu 3.2.

Poznámka 8.4

V skutočnosti rekurzívna funkcia nemôže volať samú seba až nekonečnokrát. Každé volanie funkcie sa totiž ukladá v pamäti počítača a kapacita pamäte je obmedzená. Okrem toho je v softvéri Mathematica zadefinovaná horná hranica, ktorá určuje koľkokrát maximálne sa rekurzívna funkcia môže volať. Táto premenná sa nazýva **$RecursionLimit** a jej preddefinovaná hodnota je 1024. Túto hranicu môžeme jednoducho zmeniť pomocou nasledujúceho príkazu.

```
In[40]:= $RecursionLimit = 2048;
```

Je možné nastaviť ju aj na hodnotu ∞.

```
In[41]:= $RecursionLimit = Infinity;
```

Upozorňujeme, že aj v takomto prípade je počet volaní limitovaný veľkosťou pamäte.

Cvičenie 8.8

Zadefinujte užívateľskú funkciu, ktorá vracia prvky Fibonacciho postupnosti. Pripomeňme, že Fibonacciho postupnosť má nasledovný predpis: $fibonacci(0) = 0$, $fibonacci(1) = 1$ a $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$.

Poznámka: V softvéri Mathematica existuje vstavaná funkcia **Fibonacci**.

Cvičenie 8.9

Zobrazte prvých 10 prvkov Fibonacciho postupnosti na grafe.

Pomôcka: Použite funkciu **ListPlot**.

Pozorný čitateľ si určite všimne, že na výpočet $fibonacci(n)$ potrebujeme vypočítať $fibonacci(n-1)$ a $fibonacci(n-2)$; a na výpočet $fibonacci(n-1)$ potrebujeme $fibonacci(n-2)$ a $fibonacci(n-3)$. Takže $fibonacci(n-2)$ sa počíta zbytočne dvakrát. Keby sme túto úlohu riešili

na papieri, bolo by na prvý pohľad zrejmé, že výsledok pre $fibonacci(n-2)$ si po prvom vypočítaní stačí niekde zapísať a pri druhom použití ho namiesto počítania stačí prečítať. Softvéru však túto súvislosť musíme zadať my, napr. nasledovne.

```
In[42]:= fibonacci[0] = 0; fibonacci[1] = 1;
         fibonacci[n_]:= fibonacci[n] = fibonacci[n - 1] + fibonacci[n - 2]
```

Vždy, keď zavoláme funkciu **fibonacci** pre nejaké **n**, tak sa jej výsledok uloží do definície **fibonacci[n]** a pri jej ďalšom volaní so vstupom **n** sa vráti priamo táto uložená hodnota (bez opätovného výpočtu).

Cvičenie 8.10

Zadefinujte funkciu **fibonacci** takým spôsobom, ako v predchádzajúcom príklade. Zavolajte funkciu **fibonacci** pre vstup 7. Pomocou **?** zistite, ako je zadefinovaná funkcia **fibonacci**. Všimnite si, pre aké rôzne vstupy je funkcia zadefinovaná.

8.2 Anonymné funkcie

Ďalší spôsob, ako zadefinovať užívateľskú funkciu, je použitie **Function**. V softvéri Mathematica sa takto definované funkcie označujú termínom "pure function", v slovenskej literatúre pojmom "anonymná funkcia", prípadne aj "lambda funkcia". V tejto učebnici budeme používať termín "anonymná funkcia". Ako príklad takto definovanej funkcie uvádzame nasledujúcu definíciu.

```
In[43]:= f = Function[x, x + Cos[x]]

Out[43]= Function[x, x + Cos[x]]
```

Predpis takto definovanej funkcie je nasledovný: **Function[parameter, telo]**, kde **parameter** označuje lokálny názov vstupnej premennej a **telo** je definícia funkcie. Takto definovanú funkciu voláme klasickým spôsobom.

```
In[44]:= f[1]

Out[44]= 1 + Cos[1]
```

Cvičenie 8.11

Zadefinujte nejakú matematickú funkciu pomocou **Function** a potom dajte zobraziť jej graf.

Výhoda takejto definície spočíva v tom, že jej nemusíme dať meno kvôli tomu, aby sme ju mohli aplikovať (podľa toho názov anonymná funkcia). Anonymnú funkciu pre vstupný argument 1 zadávame nasledovne.

```
In[45]:= Function[x, Sin[x] + 2][1]
```

```
Out[45]= 2 + Sin[1]
```

Vstupom funkcie viacerých parametrov je zoznam týchto vstupných argumentov.

```
In[46]:= f = Function[{x, y}, x^2 + y^2]
         f[2, 3]

Out[46]= Function[{x, y}, x^2 + y^2]
         13

In[47]:= Function[{x, y}, x^2 + y^2][2, 3]

Out[47]= 13
```

Vstupné parametre môžeme ľubovoľne pomenovať. Ich názvy môžu byť jednoznakové alebo viacznakové. Definícia anonymnej funkcie dokonca umožňuje používať vstupné parametre bez názvu, a to takým spôsobom, že **Function[]** obsahuje iba **telo** (bez časti **parameter**) a v časti **telo** sú názvy parametrov nahradené pomocou symbolu #.

```
In[48]:= Function[Sin[#] + Cos[#]][3]

Out[48]= Sin[3] + Cos[3]
```

Ak ide o funkciu viacerých parametrov, prvý parameter označujeme ako # alebo #1 a ďalšie pomocou #n, kde n je poradové číslo parametra.

```
In[49]:= Function[Cos[#] + Sin[#2]]
         Function[Cos[#] + Sin[#2]][3, 8]
         Function[Cos[#1] + Sin[#2]][3, 8]

Out[49]= Function[Cos[#1] + Sin[#2]]
         Cos[3] + Sin[8]
         Cos[3] + Sin[8]
```

Všimnite si, že aj keď sme v prvom príkaze predchádzajúceho príkladu použili ako prvý parameter #, v jeho výstupe bol prvý parameter zobrazený ako #1.

Cvičenie 8.12

Zadefinujte nejakú matematickú funkciu pomocou **Function** (bez toho, aby vstupný parameter mal svoj názov) a potom dajte zobraziť jej graf.

Keďže týmto spôsobom nedefinujeme, koľko vstupných parametrov funkcia má, ale len koľko ich používa, takto definované funkcie môžu dostať ľubovoľný počet vstupných parametrov. Nadbytočné parametre sú ignorované.

```
In[50]:= Function[Cos[#1] + Sin[#2]][3, 8, 9]

Out[50]= Cos[3] + Sin[8]
```

Posledný, ale v programátorskej praxi najvyužívanejší spôsob je definícia anonymnej funkcie bez použitia **Function**. Ukončenie definície funkcie je dané pomocou znaku **&**(ampersand). V tomto prípade sa dá funkcia z predchádzajúceho príkladu zapísať nasledovne.

```
In[51]:= g = Cos[#] + Sin[#2]&
         g[3, 8]

Out[51]= Cos[#1] + Sin[#2]&
         Cos[3] + Sin[8]

In[52]:= Cos[#] + Sin[#2]&[3, 8]

Out[52]= Cos[3] + Sin[8]
```

Nevýhodou tohto zápisu funkcie je fakt, že neumožňuje definovať komplikovanejšie vstupy ako napr. vstup funkcie **integralPlot** zo začiatku tejto kapitoly.

8.3 Užívateľské funkcie s využitím Block, Module a With

Ak chceme naprogramovať funkciu, ktorej **telo** bude obsahovať viacero príkazov, tak tieto príkazy umiestnime medzi okrúhle zátvorky a oddelíme ich od seba bodkočiarkami.

```
In[53]:= f[x_]:= (pom = 7; x + pom)
```

Modrá farba premennej **pom** vyjadruje, že nejde o lokálnu premennú ale o symbol (t. j. globálnu premennú bez priradenia). Po zavolaní funkcie sa farba **pom** zmení na čiernu (zo symbolu sa stane premenná).

```
In[54]:= f[2]
         pom

Out[54]= 9
         7
```

Je zrejmé, že premenné v tele funkcie, ktoré neboli explicitne definované ako lokálne premenné, sú globálne premenné. Takto definovaná funkcia sa stáva problematickou, ak je volaná v programe, v ktorom sa nachádza globálna premenná s rovnakým menom ako nejaká premenná tejto funkcie. Po zavolaní funkcie totiž dôjde k prepísaniu hodnoty premennej nielen vnútri funkcie ale v celom programe.

Z tohto dôvodu je prospešné používať v užívateľských funkciách lokálne premenné. Definujeme ich pomocou vstavaných funkcií **Module** a **Block**. Obidve fungujú podobne, preto si ich použitie podrobne vysvetlíme len na funkcii **Module**. Nakoniec ukážeme, aký je rozdiel v ich aplikovaní. Funkcia **With** nepoužíva lokálne premenné, ale nahrádzania pomocou nich. Podrobnejšie to vysvetlíme na konci tejto sekcie.

Funkcia **Module[lokalne premenne, vyraz]** (a tak isto aj **Block**) má dva vstupné parametre; **lokalne premenne** je zoznam lokálnych premenných a **vyraz** je príkaz (alebo skupina príkazov), ktorý sa má vykonať a v ktorom vystupujú premenné zo vstupného parametra **lokalne premenne**. Predchádzajúcu funkciu zadefinujeme pomocou **Module** nasledovne.

```
In[55]:= f[x_]:= Module[{pom}, pom = 7; x + pom]
```

Všimnite si, že v prípade použitia **Module** nemusí (ale môže) byť skupina príkazov umiestnená v okrúhlych zátvorkách. Takto definovaná funkcia neprepisuje v programe hodnotu globálnej premennej **pom**.

```
In[56]:= Clear[pom]
         f[2]
         pom

Out[56]= 9
         pom
```

V prípade, že chceme definovať premennú s nejakou počiatočnou hodnotou, môžeme to urobiť priamo v zozname lokálnych premenných.

```
In[57]:= f[x_]:= Module[{pom = 7}, x + pom]
```

Rovnakým spôsobom definujeme funkciu pomocou **Block**.

```
In[58]:= f[x_]:= Block[{pom = 7}, x + pom]
         f[2]
         pom

Out[58]= 9
         pom
```

Module a **Block** fungujú veľmi podobne, odlišujú sa však v spôsobe, ako pracujú s lokálnymi premennými.

Keby sme nemali možnosť používať lokálne premenné, mohli by sme sa konfliktom v názvoch premenných vyhnúť tak, že by sme na koniec názvu premennej pridali nejaké dlhé náhodné číslo. Podobným spôsobom funguje funkcia **Module**. K názvu lokálnej premennej softvér Mathematica pridá číslo, aby nedošlo ku konfiktu s globálnou premennou rovnakého mena.

Existuje aj druhý spôsob, ako sa vyhnúť konfliktu pri rovnomenných premenných. V prípade, že chceme zadefinovať lokálnu premennú s rovnakým názvom, aký má už predtým používaná globálna premenná, mohli by sme hodnotu uloženú v globálnej premennej prekopírovať do dočasnej premennej. Následne globálnu premennú uvoľniť a ďalej s ňou pracovať ako s lokálnou premennou. Keď platnosť lokálnej premennej skončí, mohli by sme prekopírovať do globálnej premennej pôvodnú hodnotu z dočasnej premennej. Na takomto princípe funguje funkcia **Block**.

Náznak týchto princípov fungovania je vidieť, keď zavoláme lokálnu premennú s nepriradenou hodnotou vnútri **Module** a **Block**. Vtedy dostaneme ako výstup len názov premennej.

```
In[59]:= f1[x_]:= Module[{pom}, pom]
         f2[x_]:= Block[[{pom}, pom]
         f1[2]
         f2[2]

Out[59]= pom$11585
         pom
```

Tento postup pri použití **Module** zapríčiní, že sa za parameter **x** v nasledujúcom príklade dosadí len $\mathbf{pom}^2$ a nie číslo **4**, hoci **pom=2**.

```
In[60]:= x = pom^2
         Module[{pom = 2}, pom + x]
         Block[{pom = 2}, pom + x]

Out[60]= pom^2
         2 + pom^2
         6
```

Ďalší spôsob, ako sa vyhnúť konfliktu rovnomenných premenných (v programe a v uživateľskej funkcii) je použitie vstavanej funkcie **With**. V tomto prípade sa v tele funkcie nepoužívajú premenné, ale nahradenia. Predpis funkcie **With** je nasledovný `With[{x=x0, y=y0,...}, vyraz]`. Vo výraze `vyraz` sa všetky symboly `x, y,...` nahradia hodnotami `x0, y0,...`. To znamená, že každé `x, y,...` bude inicializované nejakou hodnotou, ktorá sa ale nemôže už potom ďalej meniť. Ako príklad funkcie využívajúcej **With** uvádzame funkciu `f[x_]:= With[{pom = 7}, x + pom]`, ktorá funguje rovnako ako funkcia **f** zo začiatku tejto sekcie. V tomto prípade, samozrejme nemôžeme **pom** používať ako premennú a po jej inicializácii už nemôžeme meniť jej hodnotu. Išlo by totiž o priraďovanie do čísla **7** a nie do premennej **pom**. Ak viete, že hodnota premennej sa po jej inicializácii už nebude meniť, použite radšej funkciu **With** (namiesto **Module**), keďže je rýchlejšia.

Na záver porovnáme použitie **Module**, **Block** a **With**. Ak vieme, že lokálne premenné nebudú meniť hodnotu, je najvýhodnejšie použiť **With**. Keď lokálne premenné budú meniť hodnotu, tak sa používa **Module** alebo **Block**. Funkciu **Block** aplikujte v situáciach, keď budete používať globálne

premenné, ktorých hodnota sa má obnoviť (pozri predchádzajúci príklad – vo výstupe je výsledkom číslo **6**, keďže v **pom**^{2} sa obnovila hodnota). Ak si nie ste istí výberom funkcie, použite **Module**. Je síce najpomalšia z týchto 3 funkcií, ale je aj najvšeobecnejšia.

8.4 Rôzne spôsoby aplikovania funkcií

Základný spôsob volania funkcií pomocou `NazovFunkcie[]` sme si pripomenuli na začiatku tejto kapitoly. V tejto sekcii si ukážeme ďalšie možnosti, ktoré nám uľahčia prácu s funkciami. Viaceré z týchto spôsobov sú inšpirované klasickou matematikou.

V matematike sa špeciálny symbol $\circ$ používa na zápis skladania funkcií $(f \circ g)(x) = f(g(x))$. V softvéri Mathematica sa skladanie funkcií zadáva pomocou znaku `@` (zavináč).

```
In[61]:= g[x_]:= Cos[x]
         m[x_]:= Cos[x] + Sin[Cos[x]]
         g@m[x]

Out[61]= Cos[Cos[x] + Sin[Cos[x]]]
```

Tento operátor zabezpečí, že sa výraz napravo od neho aplikuje ako vstup do výrazu naľavo.

```
In[62]:= a@b@c@d

Out[62]= a[b[c[d]]]
```

V opačnom poradí sa dajú funkcie skladať pomocou operátora `//` (dva znaky lomítka).

```
In[63]:= a//b//c//d

Out[63]= d[c[b[a]]]
```

Pripomeňme si, že tento spôsob sme používali na aplikovanie funkcií **N** a **Simplify**.

```
In[64]:= π // N
         Sin[x]^2 + Cos[x]^2 // Simplify

Out[64]= 3.14159
         1
```

Funkcia sa dá hromadne aplikovať aj na zoznam vstupov, a to pomocou funkcie **Map**.

```
In[65]:= Map[a, {5, 1/2, 1.1, π}]

Out[65]= {a[5], a[1/2], a[1.1], a[π]}
```

Keďže tento spôsob aplikovania funkcie na zoznam je v praxi veľmi často používaný, bola do softvéru Mathematica zabudovaná aj jeho skrátená verzia `/@`.

```
In[66]:= Sin/@{0, π/2, 3}

Out[66]= {0, 1, Sin[3]}
```

Cvičenie 8.13

Uložte zoznam z predchádzajúceho príkladu do premennej `a`. Pomocou funkcií `Table` a `Length` naprogramujte funkciu, ktorá vráti výstup z predchádzajúceho príkladu.

V prípade, že máme vnorené zoznamy, t. j. napr. zoznam, ktorého prvky sú zoznamy čísel, môžeme aplikovať funkciu na jednotlivé prvky vnútorného zoznamu pomocou špecifikovania úrovne, na ktorej sa má funkcia aplikovať. Úroveň sa špecifikuje pomocou tretieho parametra. V prípade, že úroveň je zapísaná v krútených zátvorkách, funkcia sa aplikuje iba na prvky danej úrovne.

```
In[67]:= Map[a, {{11, 12}, {21}}, {2}]

Out[67]= {{a[11], a[12]}, {a[21]}}
```

V prípade, že úroveň je zapísaná bez krútených zátvoriek, funkcia sa aplikuje rekurzívne na prvky po danú úroveň.

```
In[68]:= Map[a, {{11, 12}, {21}}, 2]

Out[68]= {a[{a[11],a[12]}],a[{a[21]}]}
```

Cvičenie 8.14

Pomocou anonymnej funkcie (`#` a `&`), `Range` a `Map` (`/@`) vytvorte zoznam druhých mocnín prvých desať prirodzených čísel.

8.5 Aplikácie užívateľských funkcií

V tejto kapitole si ukážeme, ako vyriešiť príklad z kapitoly 5 pomocou definovania vlastných funkcií. Takto riešený príklad je oveľa prehľadnejší.

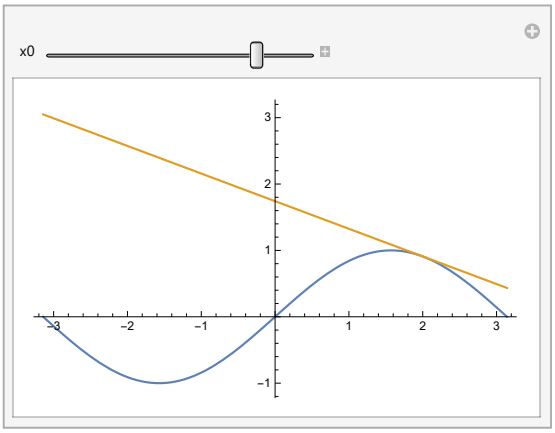
Zadanie: *Vykreslite do jedného obrázku graf funkcie* $f(x) = \sin(x)$ *a jeho dotyčnicu v bode* $[x_0, f(x_0)]$*, ktorý sa bude meniť pomocou* `Manipulate`*. Využite definovanie užívateľských funkcií.* Dotyčnica v bode $[x_0, f(x_0)]$ má rovnicu $d(x) = f'(x_0)(x - x_0) + f(x_0)$. Derivácia funkcie $f(x) = \sin(x)$ je $f'(x) = \cos(x)$. Zadefinujeme funkciu na vytvorenie tejto dotyčnice. Bude mať dva vstupné argumenty, aby sme mohli meniť dotykový bod x_0.

```
In[69]:= dotycnica[x0_, x_] := Cos[x0] (x - x0) + Sin[x0]
```

Graf tejto funkcie zobrazíme s grafom pôvodnej funkcie do jedného obrázku. Premenná `x0` sa bude meniť pomocou **Manipulate**.

```
In[70]:= Manipulate[Plot[{Sin[x], dotycnica[x0, x]}, {x, -π, π}], {x0, -π, π}]
```

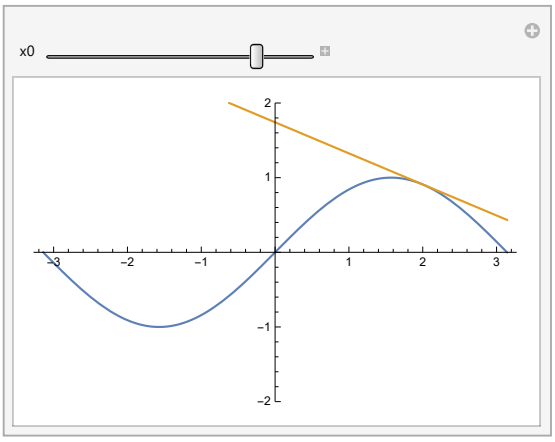
Out[70]=



Keď meníme premennú `x0`, obor zobrazených hodnôt sa mení súčasne podľa grafu dotyčnice. V tomto prípade to nie je veľmi žiadúce. Z tohto dôvodu zvolíme jednotný obor hodnôt pomocou **PlotRange**.

```
In[71]:= Manipulate[Plot[{Sin[x], dotycnica[x0, x]}, {x, -π, π},
                                PlotRange → {-2,2}], {x0, -π, π}]
```

Out[71]=



8.6 Zhrnutie

V tejto kapitole sme sa naučili definovať funkcie matematického štýlu, anonymné funkcie a funkcie využívajúce lokálne premenné s pomocou vstavaných funkcií **Module**, **Block** a **With**. Prvý spôsob umožňuje vytvárať funkcie aj pre veľmi komplikované vstupy. Výhodou anonymnej funkcie je fakt, že ona a aj jej parametre môžu byť bezmenné (prospešné v niektorých typoch úloh). Posledný spôsob

zadania funkcie bráni konfliktom rovnomenných premenných (v programe a v užívateľskej funkcii). V závere kapitoly sme predstavili rôzne spôsoby aplikovania užívateľskej funkcie.

Kapitola 9

Zobrazovanie grafických objektov v rovine

V kapitole 5 sme si predstavili zobrazovanie matematických funkcií. Výstupmi boli objekty, ktoré sa javili ako súvislé krivky. Jeden zo spôsobov, ako grafické softvéry (nielen Mathematica) vykresľujú zložitejšie objekty (t. j. aj grafy funkcií), je ich zobrazenie pomocou množiny jednoduchých geometrických útvarov. V tejto kapitole sa budeme venovať zobrazovaniu základných rovinných grafických objektov. Softvér Mathematica disponuje aj kvalitnou grafikou trojrozmerných útvarov, avšak 3D grafika je mimo rozsahu tejto učebnice. Trojrozmernej grafike sa budeme venovať v jej druhom diele.

Na zobrazenie rovinných objektov sa v softvéri Mathematica používa vstavaná funkcia **Graphics**. Jej predpis je nasledujúci **Graphics[{direktivy, objekt}]**, kde **objekt** je zobrazovaný objekt (alebo zoznam objektov) a **direktivy** sú direktívy určujúce jeho vlastnosti (farbu, priesvitnosť...), pozri podsekciu 5.1.1. Nasledujúci príkaz reprezentuje zobrazenie červeného trojuholníka, ktorého vrcholmi sú body $(0,0)$, $(0,1)$ a $(3,0)$.

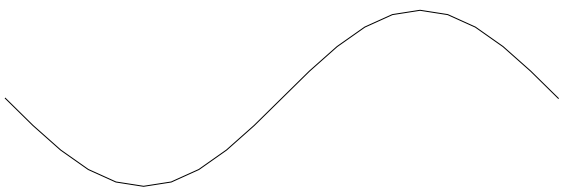
```
In[1]:= Graphics[{Red, Triangle[{{0, 0}, {0, 1}, {3, 0}}]}]
```

Out[1]=

V prvej sekcii si ukážeme, ako sa zobrazujú základné rovinné objekty a ako pomocou nich môžeme vytvárať zložitejšie geometrické útvary, napr. nasledujúci objekt vytvorený z množiny úsečiek.

```
In[2]:= Graphics[Line[Map[{#, Sin[#]}&, Range[-π, π, 2π/20]]]]
```

Out[2]=



V tejto sekcii sa oboznámime aj s najčastejšie používanými direktívami, ktorými nastavujeme vlastnosti zobrazovaných objektov. V sekcii 9.2 podrobne popíšeme rôzne spôsoby vytvárania farieb v softvéri Mathematica.

9.1 Základné grafické objekty a ich direktívy

Softvér Mathematica umožňuje zobraziť veľký počet rôznych grafických útvarov. V tejto sekcii predstavíme len tie najpoužívanejšie. V prípade záujmu o tie menej známe odkazujeme čitateľa na dokumentové centrum, v ktorom sú tieto objekty označované pojmom "graphics primitives". Hoci majú rovnaký predpis zadania ako funkcie, nie sú funkciami. Nevykonávajú žiadne príkazy a po odoslaní ich zadania do výpočtového jadra na spracovanie sa vytvorí výstupná bunka obsahujúca presne ten istý text, ktorý bol odoslaný na spracovanie. Pripomeňme si, že rovnaký výstup vracia softvér Mathematica v prípade, keď odošleme na spracovanie symbol.

Objekty, ktoré chceme zobraziť, použijeme ako vstupný argument funkcie `Graphics`. Po jej zavolaní užívateľské rozhranie (pozri kapitolu 2) vykreslí tvar, ktorý reprezentuje tento objekt (alebo zoznam objektov).

9.1.1 Grafický object Point

Najjednoduchším geometrickým objektom je bod (po angl. Point). Jeho vstupom je zoznam jeho súradníc.

```
In[3]:= Graphics[Point[{178, -37}]]
```

Out[3]=

Ak zobrazujeme jediný bod (alebo ľubovoľný jediný objekt), tak je tento zadaný bod (alebo objekt) vždy (bez ohľadu na jeho konkrétne súradnice) zobrazený v strede zobrazovacej plochy a má prednastavenú veľkosť. Umiestnenie objektu na zobrazovacej ploche v závislosti od jeho súradníc sa prejaví až v prípade zobrazovania minimálne dvoch objektov na jednom obrázku. V situáciách, keď zobrazujeme súčasne viac ako jeden objekt, platia pre zobrazovanie týchto objektov dve nasledujúce pravidlá. Pomery vzdialenosti medzi bodmi objektov v `x`-ovom a `y`-ovom smere musia byť zachované. Vzdialenosti medzi jednotlivými bodmi sú maximálne (t. j. najväčšie možné, aké povolí zobrazovacia plocha). To znamená, že napr. ak zobrazujeme dvojicu bodov $(0, 0)$ a $(5, 1)$ a dvojicu $(0, 0)$ a $(-100, -20)$ (ktoré majú rovnaké pomery vzdialenosti bodov), ich výstupy budú rovnaké.

```
In[4]:= Graphics[{Point[{0, 0}], Point[{5, 1}]}];
        Graphics[{Point[{0, 0}], Point[{-100, -20}]}]
```

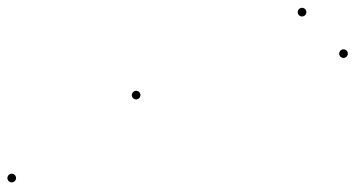
Out[4]=

Z vyššie uvedeného je zrejmé, že zobrazovacia plocha je obdĺžnik, ktorého pomer strán závisí od súradníc zobrazovaných bodov. Všimnite si tiež, že ak funkcia **Graphics** dostane ako vstup zoznam objektov, vykreslí ich všetky súčasne do jedného obrázku. Táto dôležitá vlastnosť funkcie **Graphics** platí aj v prípadoch, keď sa vo vstupnom zozname nachádzajú rôzne typy objektov (bod, kruh, obdĺžnik...).

Zobrazenie viacerých bodov súčasne sa dá zadať rôznymi spôsobmi. Jednou z možností je použiť ako vstup funkcie **Graphics** zoznam objektov **Point**. Tento spôsob sme si ukázali v predchádzajúcom príklade. Ďalšou možnosťou je použiť ako vstup funkcie **Graphics** objekt **Point**, ktorého vstupom bude zoznam zoznamov súradníc jednotlivých bodov.

In[5]:=
```
Graphics[{Point[{1, 0}], Point[{4, 2}], Point[{9, 3}], Point[{8, 4}]}];
Graphics[Point[{{1, 0}, {4, 2}, {9, 3}, {8, 4}}]]
```

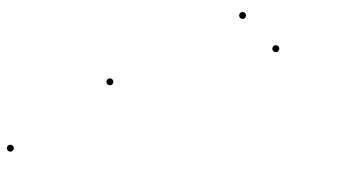
Out[5]=



V komplikovanejších úlohách (najmä ak zobrazujeme väčší počet objektov alebo rôzne typy objektov) je výhodné zadefinovať najskôr jednotlivé objekty samostatne a potom ich dať spolu zobraziť pomocou funkcie **Show** (z angl. ukáž).

In[6]:=
```
bodA = Graphics[{Point[{1, 0}]];
bodB = Graphics[Point[{4, 2}]];
bodC = Graphics[Point[{9, 3}]];
bodD = Graphics[Point[{8, 4}]];
Show[bodA, bodB, bodC, bodD]
```

Out[6]=



Vlastnosti zobrazovaných objektov (napr. farba, veľkosť...) sa nastavujú pomocou direktív. Pripomeňme si, že direktívy sme si predstavili v podsekcii 5.1.1. Väčšina direktív sa dá aplikovať aj do funkcií typu **Plot** a aj do funkcie **Graphics**. Niektoré sú však špecifické a je ich možné použiť len v konkrétnom type funkcie. Spôsob zadávania direktív do funkcií typu **Plot** a do funkcie **Graphics** je však odlišný (a to aj v prípade, že ide o rovnaké direktívy). Všeobecne ich predpisy zadávania môžeme zapísať nasledovne **FunkciaTypuPlot[objekt, Plot- Style → {direktiva$_1$,..., direktiva$_n$}]**,

Graphics[{direktiva$_1$,...,direktiva$_n$, objekt}] Napr. direktívu pre veľkosť bodu (pozri str. 76) zadávame takto.

```
In[7]:= ListPlot[{{0, 0}}, PlotStyle → PointSize[0.1]];
        Graphics[{PointSize[0.1], Point[{0, 0}]}];
```

Funkcia **ListPlot** zobrazuje množinu bodov a umožňuje k nej zadať direktívu alebo zoznam direktív. Avšak na rozdiel od funkcie **Graphics** všetky zobrazované body majú rovnaké vlastnosti. Funkcia **Graphics** umožňuje zadať každému objektu vlastný zoznam direktív. Direktívy vo funkcii **Graphics** musia byť uvedené vždy pred objektom, ktorému určujú vlastnosti.

V nasledujúcom príklade zobrazíme dva body rôznej veľkosti.

```
In[8]:= Graphics[{PointSize[0.05], {PointSize[0.1], Point[{0, 0}]}, Point[{0, 1}]}];
        Graphics[{PointSize[0.1], Point[{0, 0}], PointSize[0.05], Point[{0, 1}]}]
```

Out[8]=

Obidva príkazy z predchádzajúceho príkladu dávajú rovnaký výstup. V prvom príkaze zoznam obsahuje direktívu pre veľkosť bodu, potom zoznam a nakoniec bod. Vo vnútornom zozname meníme veľkosť bodu. Avšak, keďže je táto zmena veľkosti vnútri zoznamu, tak platí len pre objekty tohto zoznamu. V druhom príkaze sú vynechané zátvorky vnútorných zoznamov. Umožňuje nám to fakt, že direktívy vo funkcii **Graphics** sa aplikujú na všetky objekty v zozname za touto direktívou. Ak sú v jednom zozname pred objektom dve alebo viac direktív rovnakého typu, tak sa uplatní tá posledná.

Cvičenie 9.1

Pomocou funkcie **Graphics** zobrazte body $(0,0)$, $(1,1)$, $(2,4)$, $(3,9)$, $(4,16)$ a $(5,25)$.
Poznámka: Je zrejmé, že v tomto cvičení a aj vo viacerých ďalších cvičeniach v tejto kapitole by bolo praktickejšie využiť funkcie z kapitoly 5. Napr. tieto body by sa dali vykresliť príkazom **ListPlot[{{0, 0}, {1, 1}, {2, 4}, {3, 9}, {4, 16}, {5, 25}}]**. Avšak kvôli lepšiemu osvojeniu poznatkov o funkcii **Graphics** a o geometrických objektoch, ktoré zobrazuje, budeme používať v týchto cvičeniach túto funkciu.

Farba grafických útvarov sa dá nastaviť pomocou viacerých druhov direktív. Najjednoduchší spôsob je použitie prednastavených farieb, napr. **Red**, **Green**, **Blue**. Kompletný zoznam týchto farieb nájdete v prvom odstavci sekcie 9.2. Farbu pozadia (po angl. background) nastavujeme príkazom **Background→Farba**.

In[9]:=
```
Graphics[{Red, PointSize[0.1], Point[{0, 0}], Blue, Point[{4, 1}]},
Background → Green]
```

Out[9]=

Podrobnejšie si nastavovanie farieb vysvetlíme v sekcii 9.2.

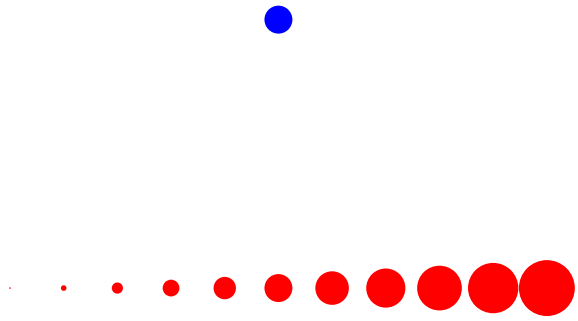
Poznámka 9.1

Na rozdiel od direktív, ktoré sa zadávajú pred geometrický objekt do spoločného zoznamu, príkaz na nastavenie farby pozadia (keďže je to voľba a nie direktíva) píšeme až za objekt alebo ak sa objekt nachádza v zozname, tak za tento zoznam. Viac si o voľbách vo funkcii Graphics povieme o stranu ďalej.

V predchádzajúcom texte sme sa zmienili o tom, že funkcia **Graphics** môže dostať ako vstup vnorené zoznamy. Táto vlastnosť má dve výhody. Grafické objekty môžeme vytvárať aj pomocou funkcie **Table** bez ohľadu na to, že vo vstupnom argumente budú nejaké zátvorky naviac. Požadované vlastnosti priraďujeme objektom jednoducho, keďže direktívy sú aplikované iba na objekty, ktoré sa nachádzajú za danou direktívou v spoločnom najvnútornejšom zozname.

In[10]:=
```
Graphics[{Blue,
  {Red, Table[{PointSize[x/10], Point[{x, 0}]}, {x, 0, 1, 0.1}]},
  PointSize[0.05], Point[{0.5, 0.5}]}]
```

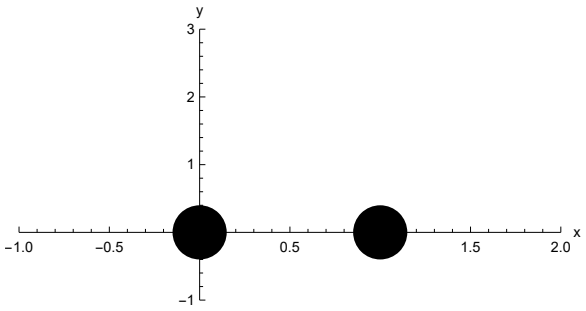
Out[10]=



Pri zobrazovaní geometrických útvarov môžeme podobne ako vo funkciách typu **Plot** používať okrem direktív aj voľby.

In[11]:=
```
Graphics[
  {PointSize[0.1], Point[{0, 0}], Point[{1, 0}]},
  PlotRange → {{-1, 2}, {-1, 3}}, Axes → True,
  AxesLabel → {"x", "y"}, AspectRatio → 1/2]
```

Out[11]=



Voľby sa zadávajú za zoznamom direktív a objektov a ovplyvňujú celý graf. Jednotlivé voľby sme si podrobne vysvetlili v podsekcii 5.1.1.

Poznámka 9.2

Direktívy si vo funkcií **Graphics** môžeme predstavovať ako príkazy, ktoré určujú, aké pero má kreslič použiť na vykreslenie objektov. V prípade **PointSize** kreslič použije na vykreslenie bodov pero s hrúbkou hrotu zadanej veľkosti. V prípade nastavovania farieb použije kreslič pero zadanej farby. Kreslí daným perom dovtedy, kým mu inou direktívou neprikážeme zmeniť pero, alebo pokiaľ sa neskončí najvnútornejší zoznam, v ktorom sa táto direktíva nachádza. Ak sa zoznam skončí, aktivuje sa opäť pero platné pre zoznam o úroveň vyššie.

Cvičenie 9.2

Pomocou **Graphics** vykreslite body funkcie $f(x) = x^2$ pre $x \in \{0, 0.05, 0.1, 0.15, 0.2, ..., 1\}$.

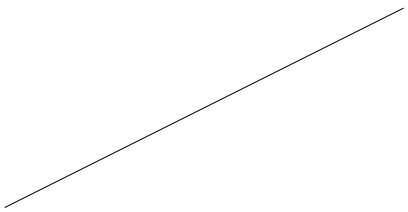
Pomôcka: Vypisovať súradnice všetkých bodov by bolo veľmi pracné. Využite namiesto toho funkciu **Table** (pozri str. 52) alebo funkciu **Map** (pozri str. 125) spolu s definíciou vlastnej (kvadratickej) funkcie (pozri kapitolu 8) a funkciou **Range** (pozri str. 51).

9.1.2 Grafický object Line

Ďalší geometrický objekt, ktorý si predstavíme je čiara s predpisom **Line[{{x$_0$, y$_0$}, {x$_1$, y$_1$}}]**, kde **{x$_0$, y$_0$}** je začiatočný bod čiary a **{x$_1$, y$_1$}** je koncový bod čiary.

In[12]:=
```
Graphics[Line[{{0, 0}, {2, 1}}]]
```

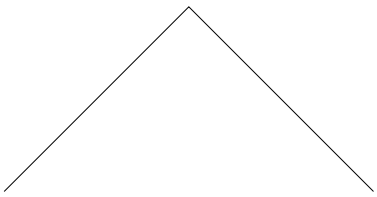
Out[12]=



Ak je vstupným parametrom funkcie **Line** zoznam s viac ako dvoma bodmi, zobrazí sa lomená čiara. Body sú pospájané v takom poradí, ako boli zadané vo vstupnom zozname.

In[13]:=
```
Graphics[Line[{{0, 0}, {1, 1}, {2, 0}}]]
```

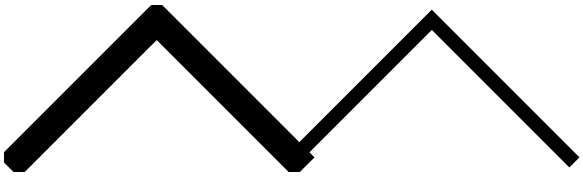
Out[13]=



Hrúbka čiar sa dá meniť pomocou direktívy **Thickness[r]** alebo pomocou prednastavených hrúbok **Thick** a **Thin**.

In[14]:=
```
Graphics[{
  Thickness[0.05], Line[{{0, 0}, {1, 1},{2, 0}}],
  Thickness[0.025], Line[{{2, 0}, {3, 1}, {4, 0}}]
}]
```

Out[14]=



Cvičenie 9.3

Pomocou **Graphics** a **Line** zobrazte graf funkcie $f(x) = x^2$ pre body $x \in \{0, 0.05, 0.1, 0.15, , ..., 1\}$ (lomenú čiaru prechádzajúcu týmito bodmi).

Štýl vykreslenia čiar zadávame pomocou direktív **Dotted** (bodkovane), **Dashed** (čiarkovane) a **DotDashed** (bodkočiarkovane).

In[15]:=
```
Graphics[{Thick,
  Dotted, Line[{{0, 0}, {1, 1}, {2, 0}}],
  Dashed, Line[{{2, 0}, {3, 1}, {4, 0}}],
  DotDashed, Line[{{4, 0}, {5, 1}, {6, 0}}]
}]
```

Out[15]=

Variantom geometrického útvaru **Line** je objekt **Arrow** (orientovaná čiara). Funguje rovnako ako **Line** len v poslednom bode čiary zobrazuje naviac hrot šípky. Používa rovnaké direktívy ako **Line**.

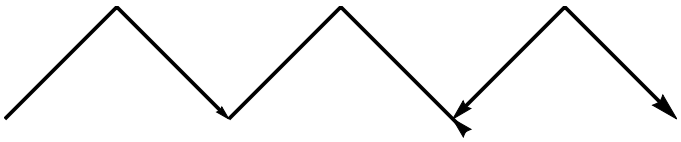
```
In[16]:= Graphics[{Thick,
           Red, Dotted, Arrow[{{0, 0}, {1, 1}, {2, 0}}],
           Green, Dashed, Arrow[{{2, 0}, {3, 1}, {4, 0}}],
           Blue, DotDashed, Arrow[{{4, 0}, {5, 1}, {6, 0}}]
         }]
```

Out[16]=

V tomto objekte je naviac zabudovaná direktíva **Arrowheads**, ktorá umožňuje zmeniť veľkosť hrotu a jeho smer a taktiež pridať aj ďalšie hroty.

```
In[17]:= Graphics[{Thick,
           Arrowheads[0.02], Arrow[{{0, 0}, {1, 1}, {2, 0}}],
           Arrowheads[-0.03], Arrow[{{2, 0}, {3, 1}, {4, 0}}],
           Arrowheads[{-0.03, 0.04}], Arrow[{{4, 0}, {5, 1}, {6, 0}}]
           }, PlotRange → {{0, 6}, {-0.2, 1}}]
```

Out[17]=



Záporné znamienko pred číslom vo vstupnom parametri direktívy **Arrowheads** zmení smer hrotu na opačný. Keď zadáme do **Arrowheads** zoznam **n** čísel, na čiare sa zobrazí **n** hrotov s danými veľkosťami a smermi. Pozície hrotov sú na čiare rovnomerne rozdelené.

Poznámka 9.3

V predchádzajúcom príklade sme použili direktívu **PlotRange** pre zväčšenie vykreslenej oblasti. Inak by otočený hrot druhej šípky nebol rozoznateľný. **Graphics** síce automaticky nastavuje vykreslenú oblasť tak, aby boli zobrazené všetky vykresľované objekty, ale hroty šípok do tohto kontrolovaného zoznamu nezahŕňa.

Cvičenie 9.4

Dajte zobraziť orientovanú čiaru prechádzajúcu bodmi $(0,0)$, $(8,1)$ a $(12,0)$. Direktíve **Arrowheads** zadajte ako vstup zoznam siedmich rôznych kladných aj záporných čísel.

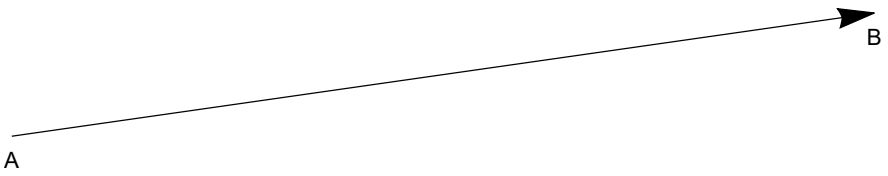
9.1.3 Grafický object Text

Softvér Mathematica umožňuje ku zobrazovaným objektom doplniť aj textový popis. Používa sa na to grafický objekt **Text["textovy popis",{x,y}]**, kde "textovy popis" je text, ktorý zobrazujeme a (x,y) je bod, do ktorého umiestníme stred textu. Ak tento bod pri zadávaní vstupného

argumentu objektu **Text** vynecháme, tak bude stred textu umiestnený do implicitne nastaveného bodu $(0, 0)$. V nasledujúcom príklade zobrazíme vektor AB s popisom jeho koncových bodov.

```
In[18]:= vektor = Graphics[Arrow[{{0, 0}, {7, 1}}]];
         textA = Graphics[Text["A", {0, -0.2}]];
         textB = Graphics[Text["B", {7, 0.8}]];
         Show[vektor, textA, textB]
```

Out[18]=

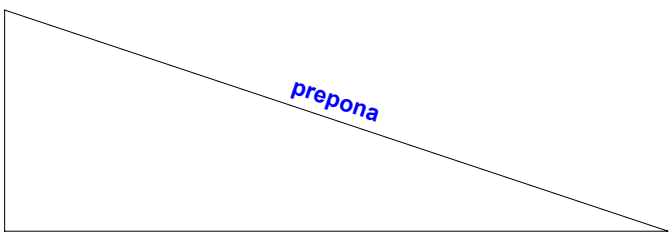


textovy popis nie je vždy nutné zadávať pomocou úvodzoviek. Avšak, ak ich v niektorých konkrétnych prípadoch vynecháme, spôsobí to problém. Text v úvodzovkách je v softvéri Mathematica považovaný za tzv. reťazec znakov, t. j. prvok typu **String**. Ak by sme nepoužili úvodzovky, jednotlivé skupiny znakov textu oddelené medzerami by boli považované za symboly a softvér Mathematica by ich usporiadal v abecednom poradí. Ak by sme zadali ako **textovy popis** nejakú premennú, ktorej už pri predchádzajúcej práci bola priradená nejaká hodnota, tak by sa bez použitia úvodzoviek zobrazila táto hodnota a nie zadaný názov premennej. Pripomeňme si, že tento princíp pre vstupné argumenty textového popisu, ktoré nie sú typu **String**, platí aj pre voľby **PlotLabel**, **PlotLabels**, **PlotLegends** a **FrameLabel** (pozri str. 71).

Textovému popisu môžeme pomocou **Style** nastaviť vlastnosti ako napr. veľkosť, farbu, typ a sklon písma.

```
In[19]:= trojuholnik = Graphics[Line[{{0, 0}, {3, 0}, {0, 1}, {0, 0}}]];
         prepona = Graphics[Text[Style["prepona", Medium, Blue, Bold],
                   {1.5, 0.6}, Automatic, {3, -1}]];
         Show[trojuholnik, prepona]
```

Out[19]=



Voľba `Automatic, {3, -1}` zapríčinila sklon textového popisu v smere vektora `(3, -1)`.

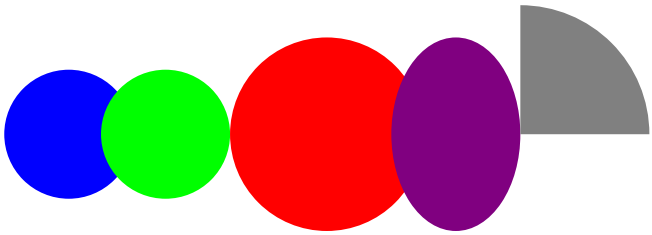
9.1.4 Dvojrozmerné grafické objekty

Z dvojrozmerných útvarov, ktoré dokáže softvér Mathematica zobrazovať, podrobnejšie popíšeme dva základné objekty `Disk` (z angl. kruh) a `Polygon` (z angl. mnohouholník). `Disk[{x, y},`

`{rx, ry}, {us, ue}]` reprezentuje plnú elipsu (t. j. útvar, ktorého hranice sú dané elipsou), prípadne kruh (špeciálny typ plnej elipsy), kde bod `{x, y}` je stred elipsy, `{rx, ry}` sú dĺžky polosí elipsy (ak zadáme namiesto tohto zoznamu len jedno číslo, bude to číslo reprezentovať polomer kruhu) a `{us, ue}` určujú uhol výseku, ktorý sa má zobraziť. Ak nezadáme tretí zoznam, vykreslí sa celá elipsa (kruh). Ak nezadáme dĺžky polosí ani polomer, vykreslí sa kruh s polomerom 1; a ak nezadáme ani stred, vykreslí sa kruh s polomerom 1 so stredom v bode $(0, 0)$.

In[20]:=
```
Graphics[{
  Blue, Disk[],
  Green, Disk[{1.5, 0}],
  Red, Disk[{4, 0}, 1.5],
  Purple, Disk[{6, 0}, {1, 1.5}],
  Gray, Disk[{7, 0}, 2, {0, π/2}]
}]
```

Out[20]=

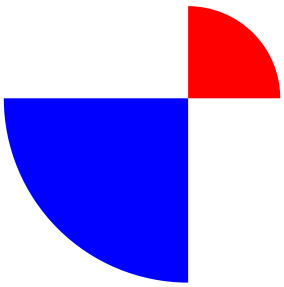


Z predchádzajúceho príkladu (z čiastočného prekrývania sa objektov) je zrejmé, že objekty sa vykresľujú v rovnakom poradí, ako sú zadané v zozname.

Cvičenie 9.5

Zadajte príkaz, ktorým zobrazíte nasledujúci obrázok, kde červený štvrťkruh má polomer 1 a modrý polomer 2.

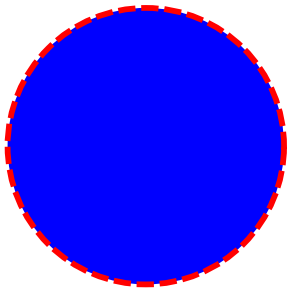
Out[21]=



Okrem direktív na nastavovanie farieb môžeme použiť aj direktívu `EdgeForm`, ktorá nastavuje vlastnosti hraníc objektu. Vstupným parametrom funkcie `EdgeForm` môžu byť tie isté direktívy, ktorými sa zadával štýl vykreslenia objektu `Line`.

In[22]:=
```
Graphics[{Blue, EdgeForm[{Thickness[0.02], Red, Dashing[0.04]}], Disk[]}]
```

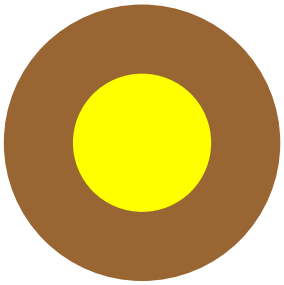
Out[22]=



Cvičenie 9.6

Zadajte príkaz, ktorým zobrazíte nasledujúci obrázok.

Out[23]=



Vykreslenie hnedého kruhu uložte do premennej s názvom `hnedy` a vykreslenie žltého kruhu do premennej `zlty`. Potom zobrazte obidva kruhy pomocou funkcie `Show`.

Zistite, aký je rozdiel medzi príkazmi `Show[hnedy, zlty]` a `Show[zlty, hnedy]`.

Variantom geometrického objektu `Disk` je objekt `Circle`, ktorý vykresľuje hranice objektu `Disk`. Používa rovnaké direktívy ako objekt `Line`.

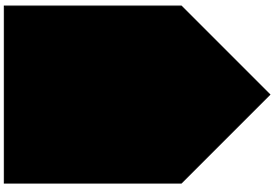
In[24]:=
```
Graphics[{Orange, Dashed, Thick, Circle[{0, 0}, {7, 2}, {π/2, 2π}}]
```

Out[24]=

Na záver si predstavíme geometrický objekt `Polygon` (z angl. mnohouholník). Jeho vstupom je zoznam bodov, ktoré reprezentujú vrcholy mnohouholníka.

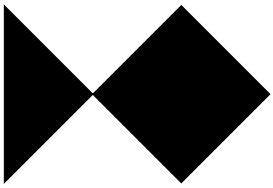
In[25]:=
```
Graphics[Polygon[{{0, 0}, {0, 2}, {2, 2}, {3, 1}, {2, 0}}]]
```

Out[25]=



In[26]:=
```
Graphics[Polygon[{{0, 0}, {2, 2}, {3, 1}, {2, 0}, {0, 2}}]]
```

Out[26]=



Všimnite si, že geometrické útvary z dvoch posledných príkladov majú síce rovnaké vrcholy, ale odlišný tvar. Je zrejmé, že ich tvar závisí od poradia bodov vo vstupnom zozname objektu `Polygon`.

Cvičenie 9.7

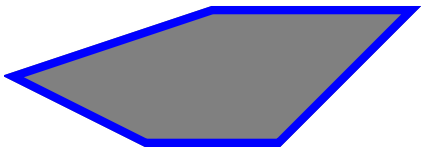
Kruh sa dá aproximovať pomocou pravidelného mnohouholníka. Čím viac hrán mnohouholník má, tým presnejšie aproximuje kruh. Parametrický predpis jednotkovej kružnice je $kruznica(u) = (\cos(u), \sin(u)),\ u \in (0, 2\pi)$. Keď do funkcie *kruznica* zadáme ako vstup číslo z intervalu $(0, 2\pi)$, výstupom bude bod kružnice.

Pomocou tejto funkcie vytvorte zoznam 50-tich rovnomerne rozdelených bodov na kružnici. Pomocou `Graphics` a `Polygon` zobrazte mnohouholník, ktorého vrcholy sú body z tohto zoznamu.

V objekte `Polygon` sa používajú rovnaké direktívy ako v objekte `Disk`.

In[27]:=
```
Graphics[{EdgeForm[{Thickness[0.02], Blue}], Gray,
  Polygon[{{0, 0}, {1, 0}, {2, 1}, {0.5, 1}, {-1, 0.5}}]}]
```

Out[27]=



V softvéri Mathematica je zabudovaných viacero objektov, ktoré sú špeciálnymi typmi objektu `Polygon`. Z nich si predstavíme len obdĺžnik (po angl. Rectangle). Jeho predpis je daný nasledovne: `Rectangle[{xmin, ymin}, {xmax, ymax}]`, kde bod `{xmin, ymin}` je ľavý dolný vrchol obdĺžnika a `{xmax, ymax}` je pravý horný vrchol. Ak vo vstupe objektu `Rectangle` nezadáme druhý bod, vykreslí sa štvorec so stranou dĺžky 1, ktorého ľavý, dolný vrchol je bod zo vstupu; a ak nezadáme žiadny vstup, vykreslí sa štvorec so stranou veľkosti 1, ktorého ľavý, dolný vrchol je bod $(0, 0)$.

V objekte `Rectangle` sa okrem už predstavených direktív a volieb dá ako v jedinom grafickom objekte používať aj voľba `RoundingRadius → n`. Použitie tejto voľby spôsobí zaoblenie vrcholov obdĺžnika, pričom číslo `n` udáva veľkosť polomeru zaoblenia vrcholov.

In[28]:=
```
Graphics[Rectangle[{0, 0}, {3, 1}, RoundingRadius → 0.25]]
```

Out[28]=

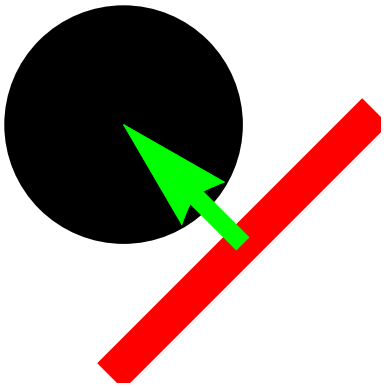
Cvičenie 9.8

Pozrite si v dokumentovom centre zoznam všetkých grafických objektov, ktoré sa dajú zobraziť pomocou funkcie `Graphics`. V softvéri Mathematica sú tieto objekty označené pojmom "graphics primitives". Objekty `Triangle` (z angl. trojuholník) a `Parallelogram` (z angl. rovnobežník) sú špeciálne typy objektu `Polygon`. Pomocou týchto objektov dajte vykresliť trojuholník a rovnobežník s ľubovoľnými vstupnými parametrami.

Grafické objekty rovnakých aj rôznych typov môžeme súčasne zobraziť pomocou funkcie **`Graphics`**, keď ich zadáme do jej vstupu ako zoznam objektov

In[29]:=
```
a = Graphics[{
  Disk[{0, 1}, 0.5],
  Red, Thickness[0.1], Line[{{0, 0}, {1, 1}}],
  Green, Thickness[0.05], Arrowheads[0.3], Arrow[{{0.5, 0.5}, {0, 1}}]
}]
```

Out[29]=

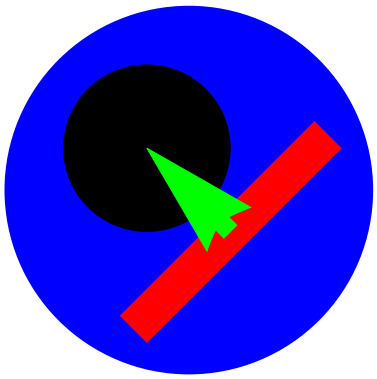


9.1.5 Funkcia Show

Objekty, ktoré sú vstupmi viacerých funkcií **`Graphics`**, zobrazíme spoločne do jedného obrázku pomocou funkcie **`Show`**.

In[30]:=
```
b = Graphics[{Blue, Disk[{0.25, 0.75}, 1.1]}];
Show[b, a]
```

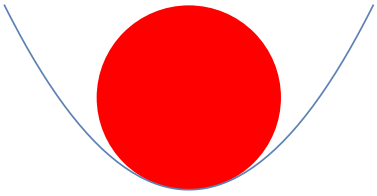
Out[30]=



Vstupnými argumentmi funkcie **Show** môžu byť súčasne výstupy funkcie **Graphics** a výstupy funkcií typu **Plot**.

In[31]:=
```
Show[Graphics[{Red, Disk[{0, 0.5}, 0.5]}], Plot[x^2, {x, -1, 1}]]
```

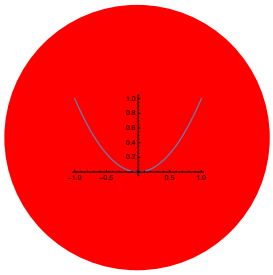
Out[31]=



Výstupy funkcií typu **Plot** môžeme vložiť do výstupov funkcie **Graphics** aj pomocou grafického objektu **Inset** (z angl. vlož). Avšak v tomto prípade nebudú grafické rozsahy výstupov obidvoch typov funkcií rovnaké (ako to bolo pri použití funkcie **Show**), ale výstup objektu, ktorý je vstupom objektu **Inset**, bude proporčne zmenšený. Nasledujúci príklad je verzia predchádzajúceho príkladu, v ktorom je namiesto funkcie **Show** použitý objekt **Inset**.

In[32]:=
```
Graphics[{Red, Disk[{0, 0.5}, 0.5], Inset[Plot[x^2, {x, -1, 1}]]}]
```

Out[32]=



Poznámka 9.4

V tejto kapitole sme často poukazovali na podobnosť funkcie **Graphics** a funkcií typu **Plot**. Nie je to náhoda, kedže funkcie typu **Plot** vykresľujú svoje výstupy pomocou funkcie **Graphics** ako množinu grafických objektov. Ukážeme si to aplikovaním funkcie **InputForm** na výstup funkcie **Plot**. Funkcia **InputForm** zabráni, aby sa graf vykreslil a vypíše jeho kód.

```
In[33]:= Plot[x^2, {x, -1, 1}, PlotStyle → {Blue, Thickness[0.01]}] //
         // InputForm

Out[33]= Graphics[{{{{}, {}, Annotation[{Directive[Opacity[1.],
           AbsoluteThickness[1.6], RGBColor[0, 0, 1], Thickness[0.01]],
           Line[{{-0.9999, 0.9999}, {-0.999, 0.9987}, {-0.9987, 0.9975},
           ..., {0.9993, 0.9986}, {0.9999, 0.9999}}]
                }, ... }}]
```

Pre lepšiu čitateľnosť výstupu sme väčšinu z 274 bodov nahradili troma bodkami a tak isto aj všetky voľby. Z výstupu funkcie **InputForm** je zrejmé, že výstup funkcie **Plot** je vytvorený pomocou funkcie **Graphics**. Zobrazovaným útvarom je lomená čiara vykreslená pomocou **Line**, ktorá aproximuje funkciu $f(x) = x^2$. Na nastavenie hrúbky čiary sa tu používa direktíva **Thickness** a na nastavenie farby direktíva **RGBColor** (pre viac informácií o tejto direktíve pozri nasledujúcu sekciu).

9.2 Farby

Softvér Mathematica má v sebe zabudovaných viacero možností, ako zadávať farby v grafike. Najjednoduchší spôsob je použitie anglického názvu nejakej preddefinovanej farby. Zoznam základných preddefinovaných farieb je nasledovný.

```
In[34]:= {Red, Green, Blue, Black, White, Gray, Cyan,
          Magenta, Yellow, Brown, Orange, Pink, Purple}

Out[34]= {■, ■, ■, ■, □, ■, ■, ■, ■, ■, ■, ■, ■}
```

Okrem týchto základných farieb sú vstavané aj ich svetlé (po angl. light) verzie.

```
In[35]:= {LightRed, LightGreen, LightBlue, LightGray, LightCyan, LightMagenta,
          LightYellow, LightBrown, LightOrange, LightPink, LightPurple}

Out[35]= {■, ■, ■, ■, ■, ■, ■, ■, ■, ■, ■}
```

Tmavé verzie nie sú zabudované v softvéri. Svetlejšie aj tmavšie verzie preddefinovaných farieb vieme vytvoriť pomocou direktív **Lighter[farba]** (z angl. svetlejšia) a **Darker[farba]** (z angl. tmavšia).

```
In[36]:= Map[Lighter, {Red, Green, Blue, Black, White, Gray, Cyan,
           Magenta, Yellow, Brown, Orange, Pink, Purple}]
                     {Red, Green, Blue, Black, White, Gray, Cyan,
         Magenta, Yellow, Brown, Orange, Pink, Purple}
         Map[Darker, {Red, Green, Blue, Black, White, Gray, Cyan,
           Magenta, Yellow, Brown, Orange, Pink, Purple}]
```

Out[36]= {, , , , , , , , , , , , }

Out[37]= {, , , , , , , , , , , , }

Out[38]= {, , , , , , , , , , , , }

Všimnite si v predchádzajúcich príkladoch, že zodpovedajúce si dvojice **LightFarba** a **Lighter[Farba]** zobrazujú rôzne odtiene farieb. Verzia **LightFarba** je výrazne bledšia a často sa v grafike používa ako farba pozadia (po angl. background) obrázkov.

Ak pre našu prácu nie je postačujúca ani táto bohatá paleta farieb, softvér Mathematica nám umožňuje vytvoriť si vlastnú farbu pomocou početných direktív. Najpoužívanejšou z nich je direktíva **RGBColor[red,green,blue]**, kde parametre **red**, **green** a **blue** reprezentujú červený, zelený a modrý kanál. Každý z nich môže nadobúdať hodnoty z intervalu $(0, 1)$. Nula znamená, že konkrétny farebný kanál je úplne vypnutý a jednotka, že je plne zapnutý. **RGBColor[0,0,0]** vytvorí čiernu farbu, **RGBColor[1,1,1]** bielu a napr. **RGBColor[0.5,0.9,0.5]** odtieň zelenej farby.

```
In[39]:= {RGBColor[0, 0, 0], RGBColor[1, 0, 0], RGBColor[0, 1, 0],
           RGBColor[0, 0, 1], RGBColor[1, 1, 1], RGBColor[0.5,0.9,0.5]}
```

Out[39]= {, , , , , }

V tomto prípade sa farby miešajú na rovnakom princípe, ako sa miešajú farby svetla. Ak sú vo svetle zastúpené všetky farby (frekvencie), svetlo má bielu farbu.

```
In[40]:= Table[RGBColor[b, b, b], {b, 0, 1, 0.1}]
```

Out[40]= {, , , , , , , , , , }

```
In[41]:= Table[RGBColor[r, 0, 0], {r, 0, 1, 0.1}]
```

Out[41]= {, , , , , , , , , , }

```
In[42]:= Table[RGBColor[0, g, 0], {g, 0, 1, 0.1}]
```

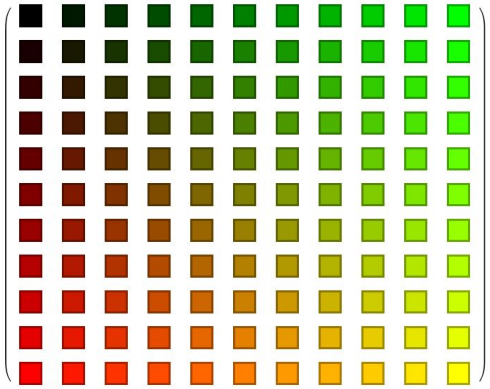
Out[42]= {, , , , , , , , , , }

```
In[43]:= Table[RGBColor[0, 0, b], {b, 0, 1, 0.1}]
```

Out[43]= {, , , , , , , , , , }

```
In[44]:= Table[RGBColor[r, g, 0], {r, 0, 1, 0.1}, {g, 0, 1, 0.1}] // MatrixForm
```

Out[44]=



Cvičenie 9.9

Vytvorte takéto "farebné matice" pre ďalšie dvojice vstupných parametrov direktívy `RGBColor`.

Na podobnom princípe funguje aj direktíva `CMYKColor[cyan,magenta,yellow,karbon]`. Výsledná farba je vytvorená ako kombinácia štyroch farebných kanálov, ktorých parametre opäť môžu nadobúdať hodnoty z intervalu $(0, 1)$. Farby v direktíve `CMYKColor` sa miešajú rovnakým spôsobom, ako sa miešajú farby v tlačiarni alebo ako ich miešajú maliari. Ak zmiešame všetky zložky, dostaneme čiernu farbu (na rozdiel od direktívy `RGBColor`, ktorá by v takomto prípade vytvorila bielu farbu).

```
In[45]:= Table[CMYKColor[b, b, b, b], {b, 0, 1, 0.1}]
```

Out[45]= {□, □, □, □, □, □, □, □, □, □, □}

```
In[46]:= Table[CMYKColor[c, 0, 0, 0], {c, 0, 1, 0.1}]
```

Out[46]= {□, □, □, □, □, □, □, □, □, □, □}

```
In[47]:= Table[CMYKColor[0, m, 0, 0], {m, 0, 1, 0.1}]
```

Out[47]= {□, □, □, □, □, □, □, □, □, □, □}

```
In[48]:= Table[CMYKColor[0, 0, y, 0], {y, 0, 1, 0.1}]
```

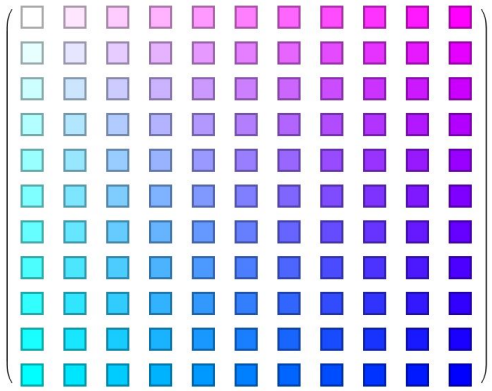
Out[48]= {□, □, □, □, □, □, □, □, □, □, □}

```
In[49]:= Table[CMYKColor[0, 0, 0, k], {k, 0, 1, 0.1}]
```

Out[49]= {□, □, □, □, □, □, □, □, □, □, □}

```
In[50]:= Table[CMYKColor[c, m, 0, 0], {c, 0, 1, 0.1}, {m, 0, 1, 0.1}] // MatrixForm
```

Out[50]=



Cvičenie 9.10

Vytvorte takéto "farebné matice" pre ďalšie dvojice vstupných parametrov direktívy `CMYKColor`.

Čiernobielou verziou k týmto dvom direktívam je `GrayLevel[level]`. Jej vstupný argument označuje úroveň (po angl. level) intenzity šede. Nadobúda hodnoty z intervalu $(0, 1)$, pričom príkaz `GrayLevel[0]` zobrazí čiernu farbu a `GrayLevel[1]` bielu farbu.

```
In[51]:= Table[GrayLevel[b], {b, 0, 1, 0.1}]
```

Out[51]= {■, ■, ■, ■, ■, ■, ■, ■, ■, □, □}

Pomocou funkcie `Blend[{farba1,farba2},podiel]` (z angl. miešaj) vieme namiešať farbu z `farba1` a `farba2`, kde parameter `podiel` udáva podiel druhej farby. Je to číslo z intervalu $(0, 1)$ a `farba1` sa teda na výslednej farbe podiela hodnotou `1-podiel`.

```
In[52]:= Blend[{Red, Yellow}, 1/4]
```

Out[52]= ■

Za parametre `farba1` a `farba2` nemusíme dosádzať len preddefinované farby, môžeme tam vložiť ľubovoľný predpis na vytvorenie farby v softvéri Mathematica.

```
In[53]:= {Blend[{RGBColor[0.8, 0.1, 0.5],
                 CMYKColor[1/2, 0.712, 0.5, 0.2]}, 0.83]
```

Out[53]= ■

Posledný spôsob vytvorenia vlastnej farby, ktorý si predstavíme, je použitie direktívy `Hue` (po angl. odtieň). Tá môže byť zadaná s jedným alebo troma vstupnými parametrami. Prvý povinný parameter (číslo z intervalu $(0, 1)$) je odtieň farebnej škály, kde červená farba je reprezentovaná číslom 0, postupne prechádza cez žltú, zelenú, bledomodrú, modrú, fialovú opäť do červenej reprezentovanej číslom 1.

```
In[54]:= Table[Hue[h], {h, 0, 1, 0.1}]
```

Out[54]= {■, ■, ■, ■, ■, ■, ■, ■, ■, ■, ■}

Druhý parameter direktívy **Hue** reprezentuje sýtosť (po angl. saturation) a tretí jas (po angl. brightness), obidva nadobúdajú hodnoty z intervalu $(0, 1)$. Číslo 1 označuje maximálnu sýtosť a jas a číslo 0 úplne vypnutú sýtosť a jas. Príkazy **Hue[h]** a **Hue[h,1,1]** vytvoria rovnakú výslednú farbu.

Na nasledujúcom príklade si ukážeme, aký vplyv má na výslednú farbu parameter reprezentujúci sýtosť. Ten bude postupne nadobúdať hodnoty: 0, 0.1,..., 1. Ostatné dva parametre budú zafixované, parameter pre odtieň farby bude nadobúdať hodnotu 0.3 a jas bude maximálny.

```
In[55]:= Table[Hue[0.3, s, 1], {s, 0, 1, 0.1}]
```

Out[55]= {□, □, □, □, □, □, □, □, □, □, □}

Obdobne si ukážeme vplyv parametra reprezentujúceho jas. Odtieň farby bude rovnaký ako v predchádzajúcom príklade, sýtosť bude maximálna a jas bude postupne nadobúdať hodnoty: 0, 0.1,..., 1.

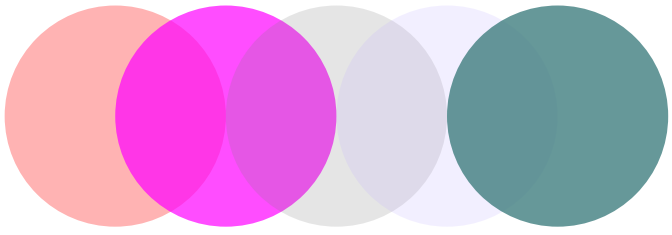
```
In[56]:= Table[Hue[0.3, 1, b], {b, 0, 1, 0.1}]
```

Out[56]= {■, ■, ■, ■, ■, ■, ■, ■, ■, ■, ■}

Do direktív: **RGBColor**, **CMYKColor**, **GrayLevel** a **Hue** môžeme zadať ešte jeden nepovinný vstupný parameter nepriehľadnosť (po angl. opacity), ktorý opäť nadobúda hodnoty z intervalu $(0, 1)$. Hodnota 1 znamená, že zobrazovaný objekt bude úplne nepriehľadný a 0 označuje úplnú priehľadnosť (t. j. objekt bude neviditeľný).

```
In[57]:= Graphics[{RGBColor[1, 0, 0, 0.3], Disk[],
         CMYKColor[0, 1, 0, 0, 0.7], Disk[{1, 0}],
         GrayLevel[0.5, 0.2], Disk[{2, 0}],
         Hue[0.7, 0.3], Disk[{3, 0}],
         Hue[0.5, 0.5, 0.5, 0.8], Disk[{4, 0}]}]
```

Out[57]=



Farebné modely, ktoré sme si v tejto sekcii predstavili, môžeme používať napr. ako direktívy pre funkciu **Graphics** a funkcie typu **Plot**.

```
In[58]:= Graphics[{RGBColor[0.7, 0.3, 0.7], Disk[], CMYKColor[0.2, 0.4, 0, 0],
         Rectangle[{2, -1}, {3, 1}], Hue[0.3, 0.6, 0.9], Disk[{5, 0}]}]
```

Out[58]=

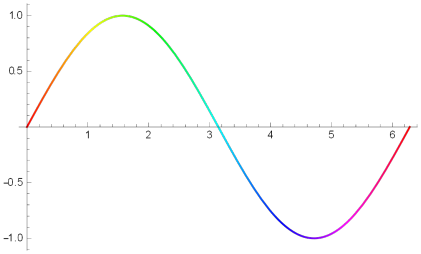
Poznámka 9.5

V sekcii 5.1.1 sme si predstavili voľbu `ColorFunction → "TemperatureMap"`. Namiesto výrazu `"TemperatureMap"` však môžeme použiť nami zadefinovanú funkciu, ktorá dostáva ako vstup preškálované súradnice bodov a jej výstupom je farba, ktorej odtieň závisí od vstupných súradníc. Preškálovanie súradníc znamená, že ak vykresľujeme funkciu na ľubovoľnom definičnom obore (napr. $(1, 3)$), tak sa tento vstup preškáluje na obor $(0, 1)$ a tak isto sa preškáluje aj obor hodnôt. Farbu môžeme vygenerovať pomocou niektorého z farebných modelov, ktoré sme si predstavili v tejto kapitole.

V nasledujúcom príklade zobrazíme graf funkcie $f(x) = sin(x)$, kde sa farba bodov bude meniť v závislosti od x-ovej súradnice pomocou direktívy `Hue`.

```
In[59]:= Plot[Sin[x], {x, 0, 2π}, ColorFunction→Function[{x, y}, Hue[x]]]
```

Out[59]=



Na definovanie užívateľskej funkcie v predchádzajúcom príklade sme použili `Function`, pozri sekciu 8.2. Z výstupu je zrejmé, že zobrazovaný graf funkcie prechádza celým farebným spektrom (od červenej, cez zelenú, modrú opäť do červenej) iba raz, čo potvrdzuje preškálovanie x-ovej súradnice z jej pôvodného rozsahu $(0, 2\pi)$ na rozsah $(0, 1)$.

9.3 Aplikácie Graphics

Zadanie: *Bez použitia funkcie* `Plot` *zobrazte graf funkcie* $f(x) = \sin(x)$ *na intervale* $(0, 2\pi)$ *tak, že jeho farba sa bude meniť v závislosti od y-ovej súradnice a postupne prechádzať pásmom farieb funkcie* `Hue`.

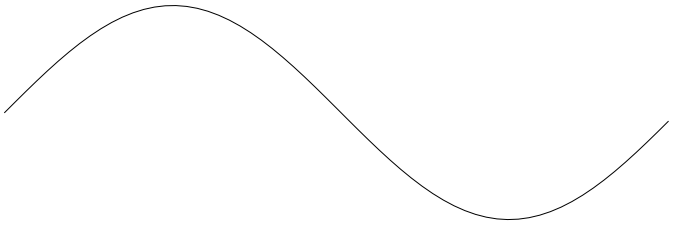
Najskôr si vytvoríme zoznam bodov grafu funkcie $f(x) = \sin(x)$ na intervale $(0, 2\pi)$ s nejakým (dostatočne malým) deliacim krokom dx.

In[60]:=
```
a = 0;
b = 2π;
dx = 0.1;
body = Table[{x, Sin[x]}, {x, a, b, dx}];
```

Lomenú čiaru prechádzajúcu týmito bodmi vieme zobraziť pomocou nasledujúceho príkazu.

In[61]:=
```
Graphics[Line[body]]
```

Out[61]=



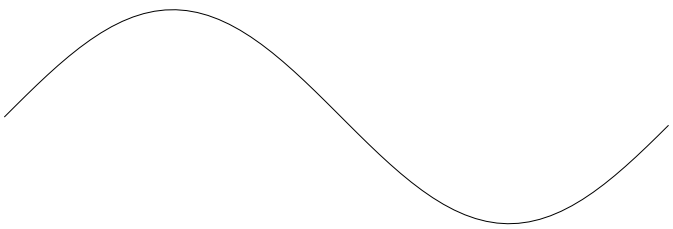
Avšak, keďže každá úsečka má mať vlastný odtieň farby, musia byť jednotlivé úsečky zadefinované samostatne. Zoznam úsečiek vytvoríme podobne ako zoznam bodov pomocou **Table**.

In[62]:=
```
ciary = Table[Line[{{x, Sin[x]}, {x + dx, Sin[x + dx]}}],
                                          {x, a, b - dx, dx}];
```

Tento zoznam uložený v premennej **ciary** zobrazíme pomocou **Graphics** a dostaneme rovnaký výstup ako v predchádzajúcom príklade.

In[63]:=
```
Graphics[ciary]
```

Out[63]=



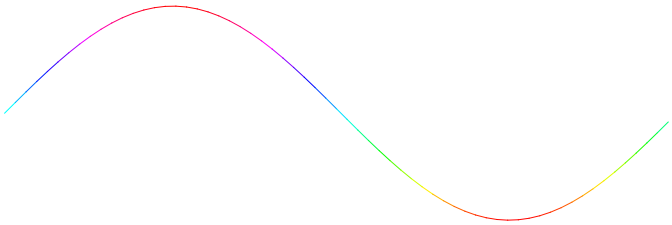
Do tohto príkazu je však možné dodefinovať farbu pre jednotlivé úsečky. Nasledujúcim príkazom namiesto zoznamu úsečiek vytvoríme zoznam dvojprvkových zoznamov, kde prvý prvok n-tého podzoznamu bude zadanie farby pre n-tú úsečku, ktorá bude druhým prvkom tohto podzoznamu. Farbu zadáme pomocou direktívy **Hue**. Keďže jej vstupom môže byť len číslo z intervalu $(0, 1)$ a obor hodnôt funkcie $f(x) = \sin(x)$ je interval $(-1, 1)$, preškálujeme hodnoty funkcie z rozsahu $(-1, 1)$ na $(0, 1)$.

```
In[64]:= ciary = Table[{Hue[Sin[x] 0.5 + 0.5], Line[{{x, Sin[x]},
                        {x + dx, Sin[x + dx]}}]}, {x, a, b - dx, dx}];
```

Na záver dáme zobraziť tento zoznam grafických objektov a ako výstup dostaneme graf funkcie $f(x) = \sin(x)$ spĺňajúci podmienky zadania.

```
In[65]:= Graphics[ciary]
```

Out[65]=



9.4 Zhrnutie

V tejto kapitole sme si predstavili funkciu **Graphics**, ktorá slúži na vykreslenie základných grafických objektov. Jej vstupom je zoznam objektov (**Point**, **Line**, **Polygon**, **Text**,...) a ich direktív (**Green**, **Dashed**, **PointSize[0.1]**,...). Vstupné grafické objekty sú zobrazené farbou a štýlom definovanými pomocou direktív. Upozornili sme, že na rozdiel od funkcií typu **Plot**, kde sú direktívy zadávané pomocou voľby **PlotStyle** za zobrazované funkcie, vo funkcii **Graphics** sa direktívy zadávajú pred zobrazované objekty do spoločného zoznamu s týmito objektmi.

V sekcii 9.2 sme si predstavili najpoužívanejšie spôsoby na vytváranie farieb: preddefinované farby, ich svetlejšie a tmavšie verzie, direktívy **RGBColor**, **CMYKColor**, **GrayLevel**, **Hue** a funkciu **Blend**.

Kapitola 10

Diferenciálny a integrálny počet

Mathematica je žiadaným softvérom aj vďaka kvalitnému aparátu symbolických výpočtov. Disponuje početnými funkciami súvisiacimi s diferenciálnym a integrálnym počtom. Prepojenie medzi zabudovanými funkciami derivovania a integrovania je úzke, keďže sú to navzájom inverzné operácie. Softvér Mathematica umožňuje derivovať a integrovať nielen konkrétne ale aj všeobecné funkcie.

```
In[1]:= D[f[x] * g[x],x]

Out[1]= g[x] f'[x] + f[x] g'[x]

In[2]:= ∫ c f[x] dx

Out[2]= c ∫ f[x] dx
```

V tejto kapitole si predstavíme rôzne spôsoby derivovania a integrovania v softvéri Mathematica. Upozorníme aj na problémy, ktoré môžu v jednotlivých situáciach vyvstať. Taktiež ukážeme aj aplikácie derivácií a integrálov v praktických príkladoch.

10.1 Derivovanie

V odbornej literatúre sa stretávame s rôznymi zápismi derivácie. Táto nejednotnosť je daná historickým vývojom. Najfrekventovanejšími sú Leibnitzov a Lagrangeov zápis. Vo fyzike, geometrii a v oblastiach popisujúcich fyzikálne javy sa používa historický zápis vyjadrujúci funkčný vzťah medzi závislou premennou `y` a nezávislou premennou `x`. Túto formu zápisu zaviedol v roku 1675 nemecký matematik Leibnitz. V klasickej matematike sa zaužíval najstručnejší zápis derivácie pomocou apostrofu, ktorý zaviedol o storočie neskôr taliansko-francúzsky matematik Lagrange.

V softvéri Mathematica sú pre výpočet derivácie zabudované funkcie, ktorých zápis zodpovedá dvom vyššie spomenutým matematickým zápisom. Leibnizovmu zápisu $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ prislúcha funkcia `D[f,x]`, kde `f` je funkcia alebo funkčný predpis, ktorý derivujeme a `x` je premenná, podľa ktorej derivujeme.

```
In[3]:= D[Sin[x],x]

Out[3]= Cos[x]
```

Lagrangeov zápis $f'(x)$ sa v softvéri Mathematica zadáva pomocou apostrofu.

```
In[4]:= Sin'[x]

Out[4]= Cos[x]
```

Obidva vyššie uvedené zápisy majú svoje klady aj zápory. Podrobnejšie ich popíšeme v nasledujúcich podsekciách.

10.1.1 Funkcia Derivative

Funkcia **Derivative** je alternatívnym variantom vstavanej funkcie na derivovanie v Lagrangeovskom štýle pomocou apostrofu.

```
In[5]:= f[x_]:= x^2
        f'[y]
        f'[3]

Out[5]= 2 y
        6
```

Cvičenie 10.1

Vypočítajte hodnotu derivácie funkcie $f(x) = ln(x)$ v bode $[3, ?]$.

Derivovanie pomocou apostrofu je najjednoduchším spôsobom, ako zadať deriváciu. Keďže ide o derivovanie funkcie jednej premennej, nie je potrebné určovať (a zadávať), podľa ktorej premennej sa derivuje a táto vstavaná funkcia takýto vstupný argument ani nemá v sebe zabudovaný. "Programátorsky" môžeme zadať takúto deriváciu pomocou funkcie **Derivative**. Je to síce komplikovanejšie, avšak v niektorých situáciách výhodnejšie, napr. prehľadnejší zápis pri deriváciách vyšších rádov (pozri str. 153). Informácie o derivovaní v Lagrangeovskom štýle nájdeme v dokumentovom centre (pozri Pozn. 2.8) pod pojmom "Derivative".

Výstupom funkcie **Derivative[n][f][z]** je **n**-tá derivácia z funkcie **f(z)**. Ak ako tretí vstupný argument zadáme nejakú číselnu hodnotu,výstupom bude vyčíslená hodnota tejto derivácie v danom bode.

Predchádzajúci príklad pomocou funkcie **Derivative** zádame nasledovne.

```
In[6]:= f[x_]:= x^2
        Derivative[1][f][y]
        Derivative[1][f][3]
```

```
Out[6]= 2 y
        6
```

Takýmto zápisom derivujeme priamo funkciu a výstupom je tiež funkcia; nielen funkcia z matematického hľadiska, ale aj z programátorského hľadiska. Takže derivovanie funkcie bez pomenovania jej premennej je tiež možné.

```
In[7]:= f[x_]:= x^2
        f'

Out[7]= 2 #1 &
```

Výstupom je anonymná funkcia (pozri sekciu 8.2). Takúto funkciu môžeme uložiť do premennej a ďalej s ňou pracovať.

```
In[8]:= derf = f';
        derf[5]

Out[8]= 10
```

Pomocou funkcie **Derivative** zadáme predchádzajúce príklady nasledovne.

```
In[9]:= f[x_]:= x^2
        Derivative[1][f]
        Derivative[1][f][5]

Out[9]= 2 #1 &
        10
```

Cvičenie 10.2

Zobrazte graf funkcie $f(x) = ln(x)$ a deriváciu jej grafu do jedného obrázku.

V prípade Lagrangeovského štýlu zadania **n**-tej derivácie používame **n** apostrofov.

```
In[10]:= h = 2 Cos[#]&;
         h''''''''''''''

Out[10]= -2 Sin[#1] &
```

Avšak v príkladoch s deriváciami vysokých rádov odporúčame použiť radšej zadanie pomocou **Derivative**, ktoré je v takýchto prípadoch oveľa prehľadnejšie.

```
In[11]:= Derivative[13][h]

Out[11]= -2 Sin[#1] &
```

Cvičenie 10.3

Zobrazte graf funkcie $f(x) = x^7 + 3x^4 + 7$ a grafy jej prvých 7 derivácií do jedného obrázku.

Pomôcka: Vypisovať zoznam 7 derivácii by bolo zbytočne zdĺhavé. Využite namiesto toho funkciu **Table** (pozri str. 52).

V derivovaných funkciách môžu vystupovať aj symboly.

```
In[12]:= h = Cos[c * #] &;
         h'

Out[12]= -c Sin[c #1] &
```

Tieto symboly sú tu považované za konštanty.

Nevýhodou tohto zápisu derivácie je fakt, že ak potrebujeme derivovať funkciu závislú od dvoch alebo viacerých zadefinovaných funkcií (napr. súčet funkcií **h** a **f**), musíme zadefinovať tretiu funkciu vyjadrujúcu ich vzťah (napr. v tomto prípade ich súčet) a derivovať tú.

```
In[13]:= g[x_]:= h[x] + f[x]
         g'[y]

Out[13]= 2y - 2Sin[y]
```

Tento zápis je však trochu zdĺhavý. Môžeme ho zadať aj bez toho, aby sme túto novú funkciu priamo pomenovali.

```
In[14]:= (h[#] + f[#] &)'[y]

Out[14]= 2y - 2Sin[y]
```

Tento zápis je síce kratší, ale horšie čitateľný.

Cvičenie 10.4

Zderivujte súčin funkcií **f** a **h**.

Podobným nedostatkom trpí Lagrangeov zápis aj v klasickej matematike. Zápis $(x^2 + \sin(x))'$ nedáva zmysel. Väčšina čitateľov by síce pochopila, čo autor týmto zápisom myslel a derivovali by ho podľa premennej x; ale čo v prípadoch, keď by v derivovanom výraze vystupovalo viac neznámych, napr. $(cx^2 + \sin(x))'$? Podľa ktorej premennej by mal byť derivovaný, podľa x alebo c?

V nasledujúcej podsekcii si predstavíme funkciu, ktorá umožňuje derivovať funkciu závislú od dvoch alebo viacerých zadefinovaných funkcií bez zadefinovania novej funkcie vyjadrujúcej tento vzťah. Avšak jej použitie má iné nevýhody.

Cvičenie 10.5

Zobrazte graf funkcie $f(x) = x^3 - 2x + 3$ a jeho dotyčnicu v bode $[5, ?]$ do jedného obrázku.

Pomôcka: Dotyčnica grafu funkcie $f(x)$ v bode $(x_0, f(x_0))$ má rovnicu $f(x) = f'(x_0)(x - x_0) + f(x_0)$.

Cvičenie 10.6

Vytvorte užívateľskú funkciu (pozri kapitolu 8), ktorá ako vstup dostane predpis funkcie a jej bod a ako výstup vráti dotyčnicu grafu zadanej funkcie v zadanom bode.

10.1.2 Funkcia D

Na derivovanie Leibnizovským štýlom $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ slúži vstavaná funkcia `D[f,x]`. Jej prvým vstupom je výraz, ktorý sa má zderivovať a druhým je premenná, podľa ktorej sa derivuje.

```
In[15]:= f[x_]:= x^2
         D[f[x], x]

Out[15]= 2 x
```

Všimnite si jeden dôležitý rozdiel. Vstupom tu nie je priamo funkcia (ako pri Lagrangeovskom štýle derivovania) ale výraz, ktorý reprezentuje predpis funkcie. Funkcia `D` najskôr vyhodnotí `f[x]` a až potom derivuje. Takže príkazu na derivovanie z predchádzajúceho príkladu zodpovedá príkaz z nasledujúceho príkladu.

```
In[16]:= D[x^2, x]

Out[16]= 2 x
```

Príkaz z nasledujúceho cvičenia teda nefunguje tak, ako by sme mohli zdanlivo očakávať.

```
In[17]:= D[f, x]

Out[17]= 0
```

Namiesto derivovania funkcie `f` dostávame derivovanie výrazu `f` podľa premennej `x`. A keďže sa vo výraze `f` žiadne `x` nenachádza, dostávame ako výstup deriváciu konštanty.

Aj v prípade, že by sme pri derivovaní pomocou vstavanej funkcie `D` použili ako vstupný argument, podľa ktorého sa má derivovať parameter bez názvu `#`, tak ako výstup dostaneme len zápis derivácie všeobecnej funkcie `f` podľa tohto parametra bez ohľadu na to, že premennej `f[x_]` je priradený výraz `x^2` (pozri nasledujúci príklad).

```
In[18]:= f[x_]:= x^2
         D[f, #] &

Out[18]= ∂#1f &
```

Pripomeňme, že pri použití Lagrangeovského štýlu derivovania by sme dostali nasledujúci výstup.

```
In[19]:= f[x_]:= x^2
         f'

Out[19]= 2 #1 &
```

Cvičenie 10.7

Definícia derivácie je daná vzťahom $\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)}{\Delta x}$. Pomocou funkcie **Limit** (pozri sekciu 3.2) overte, že derivácia funkcie $f(x) = x^2$, ktorú vracia funkcia D, je rovnaká ako $\lim_{\Delta x \to 0} \frac{(x+\Delta x)^2-x^2}{\Delta x}$.

Cvičenie 10.8

Zobrazte graf funkcie $f(x) = \ln(x)$ a jej deriváciu do jedného obrázku. Použite funkciu D.

Najjednoduchší spôsob, ako vyčísliť hodnotu derivácie v nejakom bode získanú pomocou funkcie D, je použitie nahradenia (pozri sekciu 7.1).

```
In[20]:= D[f[x], x] /. x→3

Out[20]= 6
```

Príkaz z predchádzajúceho príkladu je verziou zápisu $\frac{\mathrm{d}f(x)}{\mathrm{d}x}\Big|_{x=3}$. Častou chybou je nasledujúce zadanie derivácie v bode.

```
In[21]:= x = 3;
         D[f[x], x]

         General::ivar: 3 is not a valid variable. ≫

Out[21]= ∂39
```

Funkcia D totiž najskôr dosadí za x hodnotu 3 a následne sa snaží zderivovať číslo 9 ($f(x) = 9$) podľa čísla 3. A keďže podľa čísla sa derivovať nedá, zobrazí sa chybové hlásenie.

Poznámka 10.1

Výstupom neúspešného pokusu o derivovanie v predchádzajúcom príklade je výraz $\partial_3 9$, ktorý je alternatívnym zápisom funkcie `D`. Takéto označenie derivácie nájdeme aj v palete, pozri Pozn. 3.2.

Cvičenie 10.9

Vypočítajte hodnotu derivácie funkcie $f(x) = \ln(x)$ v bode $[3, ?]$ pomocou funkcie `D`.

Funkcia závislá od dvoch alebo viacerých zadefinovaných funkcií (napr. súčet funkcií) sa v prípade použitia funkcie `D` derivuje jednoduchšie. Stačí sčítať predpisy funkcií bez vytvárania pomocnej funkcie súčtu, ktorá sa využívala pri Lagrangeovskom štýle derivovania.

```
In[22]:= D[h[x] + f[x], x]

Out[22]= 2x - 2Sin[x]
```

Deriváciu `n`-tého rádu môžeme získať `n`-násobným aplikovaním funkcie `D`.

```
In[23]:= D[D[D[D[x^5, x], x], x], x]

Out[23]= 120x
```

Praktickejšie je však použiť funkciu `D` len jedenkrát s tým, že za prvý vstupný argument n-krát zapíšeme premennú, podľa ktorej sa má derivovať. Príkaz z predchádzajúceho príkladu prepíšeme teda na stručnejší tvar nasledujúcim spôsobom.

```
In[24]:= D[x^5, x, x, x, x];
```

V prípade vyššieho rádu derivácie odporúčame nahradiť zápis `n` premenných `x` dvojrozmerným zoznamom `{x,n}`.

```
In[25]:= D[x^5, {x, 4}];
```

Cvičenie 10.10

Zderivujte súčin funkcii `f` a `h`. Použite funkciu `D`.

Aj keď funkcia `D` reprezentuje zápis derivácie v Leibnitzovskom štýle, v prípade, že pomocou nej derivujeme všeobecnú funkciu, jej výstup bude v tvare Lagrangeovho zápisu.

```
In[26]:= D[l[x], x]

Out[26]= l'[x]
```

Funkcia `D` umožňuje aplikovať základné pravidlá derivovania aj na všeobecné funkcie.

```
In[27]:= D[c l[x], x]

Out[27]= c l'[x]

In[28]:= D[l[c * x], x]

Out[28]= c l'[x]

In[29]:= D[k[x] * l[x], x]

Out[29]= l[x] k'[x] + k[x] l'[x]

In[30]:= D[k[x] / l[x], x]

Out[30]= k'[x]/l[x] - k[x] l'[x]/l[x]^2
```

10.1.3 Aplikácie derivácií

Využitie derivácií v praxi je široké. Sú súčasťou diferenciálnych rovníc, ktoré reprezentujú mnohé fyzikálne javy v technických vedách, v biomedicíne, v ekonómii ako aj v ďalších odvetviach. Dôležité miesto zaujímajú aj pri výpočtoch lokálnych extrémov funkcií v praxi napr. v optimalizačných úlohách.

V tejto podsekcii si ukážeme, ako sa v softvéri Mathematica riešia základné typy úloh na aplikácie derivácií.

Dotyčnica grafu funkcie

Pripomeňme si, že derivácia funkcie v danom bode sa rovná smernici dotyčnice grafu funkcie v tomto bode. Táto hodnota vyjadruje veľkosť zmeny funkcie v danom bode. Dotyčnica grafu funkcie $f(x)$ v bode $[x_0, f(x_0)]$ je daná nasledovnou rovnicou $t:\ y = f(x_0) + f'(x_0)(x - x_0)$. Na rozdiel od riešených príkladov z predchádzajúcich kapitol, odteraz už budeme používať v dotyčnici vstavanú funkciu na derivovanie.

Zadanie: *Nájdite rovnicu dotyčnice ku grafu funkcie* $f(x) = \frac{x^2}{6}$ *v bode* $[1.5, ?]$. *Zobrazte graf funkcie spolu s touto dotyčnicou.*

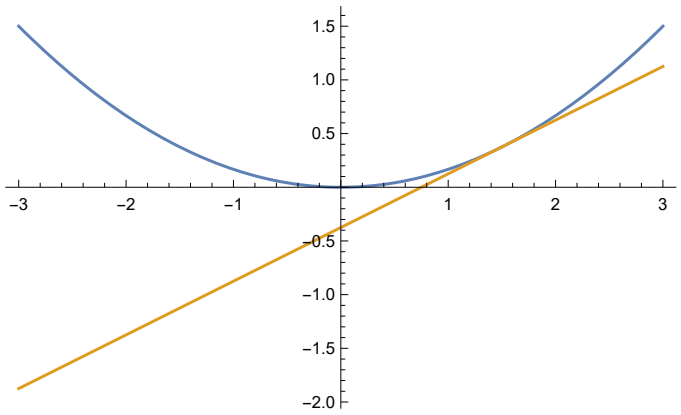
```
In[31]:= f[x_] := x^2/6
         x0 = 1.5;
         t[x_] := f[x0] + f'[x0] (x - x0)
         t[x]

Out[31]= 0.375 + 0.5(-1.5 + x)
```

```
In[32]:= Simplify[%]

Out[32]= -0.375 + 0.5x
```

Rovnica dotyčnice má predpis t: $y = 0.5x - 0.375$.

```
In[33]:= Plot[{f[x], t[x]}, {x, -3, 3}, AspectRatio → Automatic]
```

Out[33]=



Cvičenie 10.11

Pomocou funkcie `Manipulate` (pozri sekciu 5.3) vytvorte výstup, v ktorom sa bude interaktívne meniť bod funkcie $f(x) = \frac{x^2}{6}$ na intervale $(-2; 2)$ s krokom 0.1. Výstupom bude dvojprvkový zoznam. Prvým prvkom bude derivácia v danom bode (t. j. smernica dotyčnice) a druhým graf funkcie a jeho dotyčnica v danom bode.

Pomôcka: Použite voľbu `PlotRange`, aby sa pri animácii nemenil obor hodnôt.

Taylorov rozvoj

Taylorov rozvoj (polynóm) n-tého stupňa v bode $[x_0, f(x_0)]$ je najlepšou aproximáciou funkcie v okolí bodu x_0 zo všetkých polynómov n-tého stupňa. Pripomeňme si, že jeho predpis je nasledovný: $T_n(f(x), x_0, x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$. Všimnite si, že v prípade prvého stupňa je Taylorov rozvoj funkcie v bode $[x_0, f(x_0)]$ lineárnou funkciou, ktorej grafom je dotyčnica grafu funkcie v danom bode $[x0, f(x0)]$. Predpis Taylorovho rozvoja sa dá stručnejšie napísať v nasledovnom tvare:
$T_n(f(x), x_0, x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i$, kde $f^{(i)}$ je i-tá derivácia funkcie. Pripomeňme, že $0! = 1$.

Zadanie: *Vytvorte užívateľskú funkciu* `taylor` *(pozri kapitolu 8), ktorá ako vstup dostane matematickú funkciu, bod x_0 a stupeň; a vráti Taylorov rozvoj vstupnej funkcie v bode $[x_0, f(x_0)]$ daného stupňa.*

Pomocou funkcie `taylor` *nájdite Taylorov rozvoj 5-teho stupňa funkcie:*
$f(x) = (x^7 - 14x^5 + 49x^3 - 36x)/25$ *v bode* $[0, ?]$.

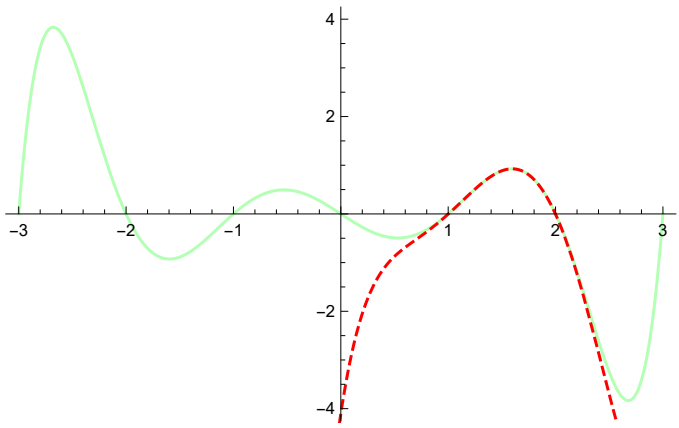
Dajte vykresliť do jedného grafu túto funkciu a jej Taylorov rozvoj.

Funkciu `taylor` zadefinujeme so vstupmi `f, x0, n` a `x`. Prvé tri vstupy určujú pre akú funkciu, okolo ktorého bodu a koľkého stupňa má byť nájdený Taylorov rozvoj. Štvrtý oddelený vstup je premenná vstupnej funkcie a zároveň aj jej Taylorovho rozvoja. Vo vnútri definície si pomocou `Module` zadefinujeme lokálnu premennú `y`. Túto premennú použijeme pri derivovaní a potom ju nahradíme hodnotou `x0`.

In[34]:=
```
taylor[f_, x0_, n_][x_] :=
 Module[{y}, Sum[D[f[y], {y, i}] /. y → x0/i! (x - x0)^i, {i, 0, n}]]
f[x_] := (x^7 - 14x^5 + 49x^3 - 36x)/25
x0 = 1.5;
n = 5;
taylor[f, x0, n][x]
graffunkcia = Plot[f[x], {x, -3, 3}, PlotStyle → {Green, Opacity[0.3]}];
graftaylor = Plot[taylor[f, x0, n][x], {x, -3, 3},
                  PlotStyle → {Red, Dashed}];
Show[graffunkcia, graftaylor]
```

Out[34]=
```
0.885938 + 0.804375(-1.5 + x) - 3.70125(-1.5 + x)^2 -
3.5525(-1.5 + x)^3 + 0.525(-1.5 + x)^4 + 1.33(-1.5 + x)^5
```

Out[35]=



Funkciu **`taylor`** sme vytvorili len z dôvodu, aby sme si precvičili programátorské zručnosti. V softvéri Mathematica je zabudovaná funkcia **`Series[f,{x,x0,n}]`**, ktorá ako výstup vráti Taylorov rozvoj vstupnej funkcie **`f`** okolo bodu **`x0`** stupňa **`n`**. Príkaz **`Series[f,{x,x0,n}]`** však na rozdiel od príkazu **`taylor[f, x0, n][x]`** vráti Taylorov polynóm aj so zvyškom (zvyšok zapísaný v tvare $O(x^{n+1})$ vyjadruje, že v Taylorovom rozvoji sa nachádzajú ešte ďalšie členy, ktoré neboli vyčíslené a ich "najväčší" člen má rád x^{n+1}).

```
In[36]:= Series[f,{x,x0,n}]

Out[36]= 0.885938 + 0.804375(-1.5 + x) - 3.70125(-1.5 + x)^2 -
         3.5525(-1.5 + x)^3 + 0.525(-1.5 + x)^4 + 1.33(-1.5 + x)^5 +
         O[-1.5 + x]^6
```

Ak by sme chceli zobraziť Taylorov polynóm so zvyškom pomocou funkcie **Plot**, zobrazilo by sa chybové hlásenie. Taylorov polynóm bez zvyšku dostaneme, ak funkciu **Series** použijeme ako vstupný argument vstavanej funkcie **Normal**.

```
In[37]:= Normal[Series[f,{x,x0,n}]]

Out[37]= 0.885938 + 0.804375(-1.5 + x) - 3.70125(-1.5 + x)^2 -
         3.5525(-1.5 + x)^3 + 0.525(-1.5 + x)^4 + 1.33(-1.5 + x)^5
```

Cvičenie 10.12

Pomocou funkcie **Manipulate** (pozri sekciu 5.3) vyskúšajte, ako sa mení aproximácia funkcie pomocou Taylorovho rozvoja so zvyšujúcim sa stupňom.

Čo sa stane, ak stupeň Taylorovho rozvoja bude rovnaký alebo vyšší ako stupeň vstupnej polynomickej funkcie?

10.2 Integrovanie

Základnou vstavanou funkciou na integrovanie je funkcia **Integrate** (z angl. zintegruj). Umožňuje výpočty neurčitých aj určitých integrálov.

10.2.1 Neurčitý integrál pomocou funkcie Integrate

Predpis funkcie **Integrate** na neurčitý integrál je nasledovný: **Integrate[f,x]**, kde **f** je výraz, ktorý sa má zintegrovať a **x** je premenná, podľa ktorej sa integruje.

```
In[38]:= Integrate[Sin[x],x]

Out[38]= -Cos[x]
```

Neurčitý integrál sa dá zadať aj pomocou matematického symbolu na integrovanie z palety (pozri Pozn. 3.2) a taktiež aj pomocou klávesovej skratky **[Esc]intt[Esc]**.

```
In[39]:= ∫ Sin[x] dx

Out[39]= -Cos[x]
```

Výstupom je primitívna funkcia ku integrovanej funkcii. Vzhľadom na fakt, že neurčitý integrál je inverznou operáciou k derivovaniu, je z predchádzajúceho príkladu zrejmé, že funkcia $f(x) = \sin(x)$ je deriváciou funkcie $F(x) = -\cos(x)$. Avšak v skutočnosti $F(x) = -\cos(x)$ nie je jedinou funkciou, ktorá má deriváciu $f(x) = \sin(x)$ (keďže deriváciou konštanty je nula). Všeobecne môžeme všetky funkcie, ktorých deriváciou je funkcia $f(x) = \sin(x)$ zapísať ako $F(x) = -\cos(x) + c$, kde c je ľubovoľná reálna konštanta. Softvér Mathematica však túto konštantu vo výstupe neurčitého integrálu nepíše. Množinu všetkých primitívnych funkcií ku funkcii $f(x)$ označujeme pojmom "neurčitý integrál funkcie $f(x)$".

V nasledujúcom príklade ukážeme, že integrovanie a derivovanie sú inverzné operácie.

```
In[40]:= int = ∫ x^2Sin[x] dx

Out[40]= -(-2 + x^2)Cos[x] + 2xSin[x]

In[41]:= der = D[int, x]

Out[41]= 2Sin[x] - (2 - x^2)Sin[x]
```

Hoci sme ako výstup derivovania očakávali výraz `x^2Sin[x]`, dostali sme iný výsledok. Z kapitoly 6 vieme, že softvér Mathematica automaticky neupravuje všetky výrazy. Výraz môžeme zjednodušiť pomocou funkcie **`Simplify`**.

```
In[42]:= der // Simplify

Out[42]= x^2Sin[x]
```

Cvičenie 10.13

Ukázali sme, že ak zintegrujeme funkciu a potom ju derivujeme, dostaneme pôvodnú funkciu. Ukážte, že to platí aj opačne, t. j. ak zderivujeme funkciu a výsledok zintegrujeme, dostaneme pôvodnú funkciu.

Poznámka 10.2

Neurčitý integrál sa označuje aj pojmom "antiderivácia". Tento názov vychádza z faktu, že neurčitý integrál je opačná operácia k derivovaniu. Táto vlastnosť je zabudovaná aj vo funkcii **`Derivative`**. Ak chceme pomocou nej integrovať, stačí namiesto kladného rádu derivácie zadať rád záporný reprezentujúci antideriváciu (t. j. neurčitý integrál).

```
In[43]:= Derivative[-1][Sin]

Out[43]= -Cos[#1]&
```

Takýto zápis dáva zmysel, keďže n-tá derivácia m-tej derivácie funkcie je $(n+m)$-tá derivácia. A definovaním derivácií záporného rádu ako integrálu môžeme túto vlastnosť rozšíriť aj na záporné čísla.

Vo funkcii **Integrate** tak ako aj vo funkcii **D** sa najskôr nahrádzajú symboly a až potom sa integruje. Ak je teda v premennej, podľa ktorej sa integruje uložená nejaká číselná hodnota, zobrazí sa chybové hlásenie.

```
In[44]:= y = 3;
         Integrate[y^2, y]

         Integrate::ilim : Invalid integration variable or limit(s) in 3. >>

Out[44]= ∫ 9 d3
```

Rovnako ako vstavané funkcie na derivovanie aj funkcia **Integrate** umožňuje pracovať aj so symbolmi.

```
In[45]:= ∫ x^n dx

Out[45]= x^(1+n)/(1+n)
```

V tomto prípade sme dostali všeobecný výsledok, ktorý však neplatí, ak **n** sa rovná -1. Pri integrovaní výrazov, ktoré obsahujú symboly, je potrebné skontrolovať, či je výsledok vo výstupe platný pre všetky prípustné hodnoty symbolov.

Cvičenie 10.14

Predchádzajúci príklad platí len za podmienky, že $n \neq -1$. Ak $n = -1$, tak ide o integrál funkcie $f(x) = \frac{1}{x}$. Z hodín matematiky vieme, že primitívnou funkciou k tejto funkcii je funkcia $F(x) = \ln(x)$. Tento výsledok získame, keď vypočítame limitu $\lim_{n\to -1} \frac{x^{1+n}}{1+n}$. Pomocou vstavanej funkcie **Limit** (pozri sekciu 3.2) overte toto tvrdenie.

Niektoré integrály vo výstupe vracajú výsledky v netypickom tvare.

```
In[46]:= ∫ E^(-x^2) dx

Out[46]= 1/2 Sqrt[π] Erf[x]
```

Vo výstupe je funkcia **Erf[x]**. Podľa dokumentového centra softvéru Mathematica (pozri Pozn. 2.8) je táto funkcia definovaná nasledovne $erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z \mathrm{e}^{-t^2} \mathrm{d}t$. S výsledkami podobného typu sa stretneme aj pri hľadaní koreňov polynómov v sekcii 11.1. Výsledok tohto integrálu nevedia síce matematici vypočítať, v praxi je však často používaný. Z tohoto dôvodu je funkcia **Integrate**

naprogramovaná tak, že vo výstupe vracia funkciu, ktorá tento výsledok reprezentuje. Približnú hodnotu tejto funkcie získame, keď zadáme ako vstup nejakú číselnú hodnotu. Ak táto hodnota nie je reálne (t. j. približné) číslo (pozri sekciu 3.3), musíme naviac aplikovať aj funkciu `N`.

```
In[47]:= Erf[0.1]
         Erf[1 / 10]
         Erf[1 / 10] // N

Out[47]= 0.112463
         Erf[1/10]
         0.112463
```

10.2.2 Určitý integrál pomocou funkcie Integrate

Predpis funkcie **`Integrate[f,{x,x0,x1}]`** na určitý integrál sa od predpisu na neurčitý integrál **`Integrate[f,x]`** líši v tom, že druhým vstupným argumentom nie je premenná **`x`**, podľa ktorej sa integruje, ale zoznam troch prvkov. Prvým z nich je premenná podľa, ktorej sa integruje a druhý a tretí sú dolná a horná hranica oblasti – intervalu integrácie.

```
In[48]:= Integrate[Sin[x], {x, -2, 3}]

Out[48]= Cos[2] - Cos[3]
```

Určitý integrál sa dá tiež zadať pomocou matematického symbolu integrálu z palety alebo pomocou klávesovej skratky **[Esc]dintt[Esc]**. **d** v klávesovej skratke reprezentuje prvé písmeno z pojmu "definite integral" (z angl. určitý integrál).

```
In[49]:= ∫_-2^3 Sin[x] dx

Out[49]= Cos[2] - Cos[3]
```

Softvér Mathematica umožňuje výpočet určitého integrálu aj v prípade, že integrovaný výraz obsahuje symboly.

```
In[50]:= ∫_-2^3 Sin[a x] dx

Out[50]= (Cos[2a] - Cos[3a]) / a
```

Platí to aj v prípadoch, keď sú symboly súčasťou hraníc integrovania.

```
In[51]:= ∫_a^b 1/x dx
```

```
Out[51]= ConditionalExpression[-Log[a] + Log[b], ((Im[a] ≤ Im[b] &&
    (Im[a] ≥ 0 || Im[b] ≤ 0 || Im[b] Re[a] ≥ Im[a] Re[b])) ||
    (Im[a] ≥ Im[b] && (Im[a] ≤ 0 || Im[b] ≥ 0 ||
    Im[b] Re[a] ≤ Im[a] Re[b]))) && ((a/(a - b) ≠ 0 &&
    Re[a/(-a + b)] ≥ 0) || a/(a - b) ∈ Reals ||
    Re[a/(a - b)] > 1) && (a/(a - b) ∈ Reals ||
    Re[a/(a - b)] ≥ 1 || Re[a/(-a + b)] ≥ 0) && ((b ∈ Reals &&
    ((Re[a] ≠ 0 && (a ∈ Reals || (Im[a] == Im[b] && Re[b] == 0))) ||
    (Im[a] == Im[b] && Re[b] ≠ 0 &&
    2 b Conjugate[b] ≠ b Conjugate[a] + a Conjugate[b]))) ||
    (b ∈ Reals && ((a ∈ Reals && b ≠ 0) || (Re[a] < b &&
    (b < 0 || (Re[a] > 0 && b > 0))) || (Re[a] > b &&
    (b > 0 || (b < 0 && Re[a] < 0))))) || (Re[a] > 0 &&
    Re[b] == 0 && ((Im[a] < 0 && Im[b] > 0) || (Im[a] > 0 &&
    Im[b] < 0))) || (Im[a] < 0 && ((Im[a] > Im[b] && Im[b] < 0) ||
    (Im[b] > 0 && Re[b] ≠ 0 && Im[b] Re[a] > Im[a] Re[b]))) ||
    (Im[a] < Im[b] && (Im[b] < 0 || (Im[a] > 0 && Im[b] > 0))) ||
    (Im[a] > 0 && Im[b] < 0 && Re[b] ≠ 0 &&
    Im[b] Re[a] < Im[a] Re[b]) || (Im[a] > Im[b] && Im[b] > 0))]
```

V poslednom výstupe sme nielen dostali výsledok vo veľmi komplikovanom tvare, ale naviac aj výpočet trval dlho. Z výsledku je zrejmé, čo to zapríčinilo. Softvér Mathematica implicitne považuje všetky symboly (teda aj symboly `a` a `b`) za komplexné čísla. Ak chceme poznať výsledok za predpodkladu, že `a` a `b` sú reálne čísla, môžeme použiť funkciu **`Assuming`** (pozri str. 95).

```
In[52]:= Assuming[a∈Reals && b∈Reals, ∫_a^b 1/x dx]

Out[52]= ConditionalExpression[Log[b/a], 0 < a < b]
```

Aj v tomto prípade je výstupom podmienený výraz, avšak tentokrát je podmienka na hranice integrovania podstatne stručnejšia. Vyžaduje sa len platnosť nerovnosti $0 < a < b$.

Cvičenie 10.15

Z predchádzajúceho výsledku je zrejmé, že výsledkom integrálu $\int_a^b \frac{1}{x}dx$ je $\ln(b/a)$, ak $a > 0$ a $b > a$. Zvoľte si ľubovoľnú dvojicu reálnych čísel spĺňajúcich nerovnosť $0 < a < b$ a overte, či tento integrál dáva takýto výsledok.

Softvér Mathematica dokáže počítať aj nevlastné integrály.

```
In[53]:= ∫_-∞^0 x^2 E^(x^3) dx
```

Out[53]= $\frac{1}{3}$

Cvičenie 10.16

Matematická definícia predchádzajúceho integrálu je $\lim_{a\to-\infty}\int_a^0 x^2\mathbb{e}^{x^3}\mathbb{d}x$. Pomocou funkcie **Limit** overte, či takto vypočítaný integrál dáva rovnaký výsledok.

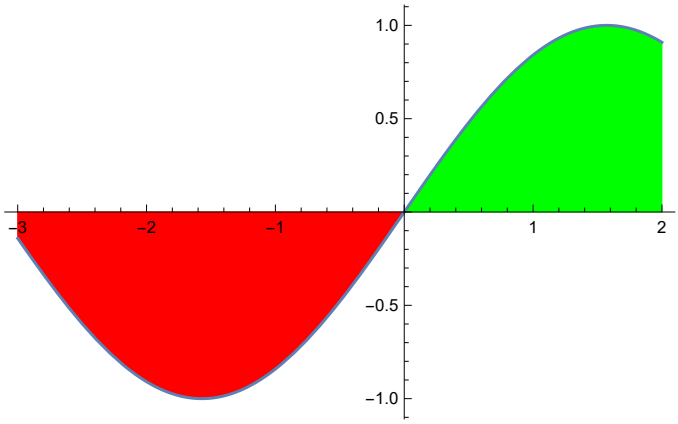
In[54]:=
```
Integrate[E^-x^2, {x, -∞, ∞}]
```

Out[54]= $\sqrt{\pi}$

Určitý integrál funkcie $f(x)$ na intervale (a, b) sa dá voľne interpretovať ako rozdiel obsahu plochy medzi kladnou časťou funkcie $f(x)$ a osou x a obsahu plochy medzi zápornou časťou funkcie $f(x)$ a osou x. Toto vizuálne ukážeme pomocou funkcie **Plot**; a volieb **Filling→Axis** a **FillingStyle→{Red, Green}**.

In[55]:=
```
Plot[Sin[x], {x, -3, 2}, Filling → Axis, FillingStyle → {Red, Green}]
```

Out[55]=



Voľbu **FillingStyle→{Red, Green}** sme použili, aby sme odlíšili plochu pod x-ovou osou a nad x-ovou osou. Plocha pod osou prispieva k výsledku integrálu zápornou hodnotou a plocha nad osou kladnou hodnotou.

Cvičenie 10.17

Aj keď sú definície určitého a neurčitého integrálu odlišné, používame na popis obidvoch ten istý názov *integrál* a naviac aj rovnaké značenie. Súvislosť medzi nimi je daná vzťahom $\int_a^b g(x)\mathbb{d}x = \int g(x)\mathbb{d}x|_{x=b} - \int g(x)\mathbb{d}x|_{x=a}$.

Vypočítajte určitý integrál $\int_0^\pi \sin(x)\mathbb{d}x$ len pomocou neurčitého integrálu a nahradenia (pozri sekciu 7.1). Výsledok overte pomocou funkcie na určitý integrál.

V prípade, že integrál nekonverguje (t. j. jeho výsledkom nie je konečná hodnota), upozorní nás na to chybové hlásenie.

In[56]:=
$$\int_{-\infty}^{\infty} x^2 E^{x^3}\, dx$$

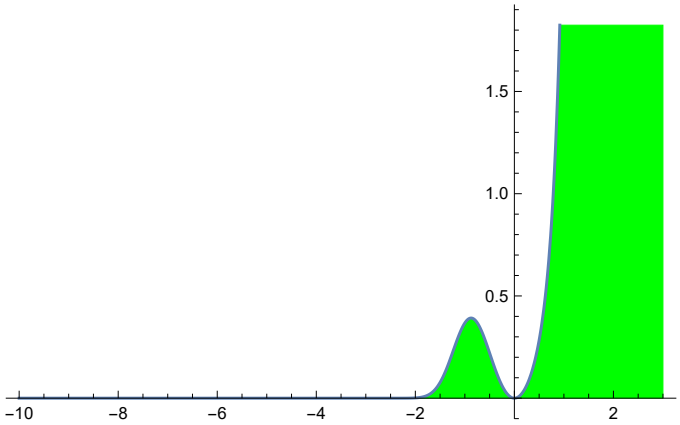
Integrate::idiv : Integral of $x^2 E^{x^3}$ does not converge on $\{-\infty, \infty\}$ ≫

Out[56]=
$$\int_{-\infty}^{\infty} x^2 E^{x^3}\, dx$$

Z grafického zobrazenia tejto funkcie je zrejmé, prečo tento integrál diverguje, keď sú jeho hranice $-\infty$ a ∞, ale konverguje, ak sú jeho hranice $-\infty$ a 0.

In[57]:=
```
Plot[x^2 E^x^3, {x, -10, 3}, Filling → Axis, FillingStyle → {Red, Green}]
```

Out[57]=



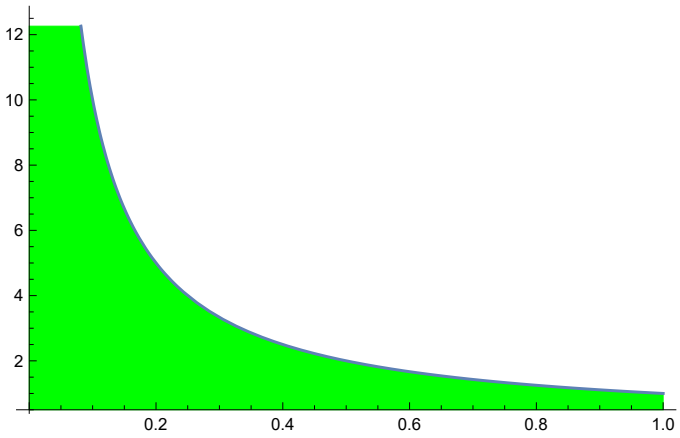
To, že hranica integrovania je nekonečno, nemusí byť jediný dôvod, prečo integrál nekonverguje. Divergencia integrálu nastáva aj v prípadoch, keď funkcia na danom intervale integrovania rýchlo diverguje do nekonečna.

In[58]:=
$$\int_0^1 \frac{1}{x}\, dx$$
```
Plot[1/x, {x, 0, 1}, Filling → Axis, FillingStyle → {Red, Green}]
```

Integrate::idiv : Integral of $\frac{1}{x}$ does not converge on $\{0, 1\}$ ≫

Out[58]=
$$\int_0^1 \frac{1}{x}\, dx$$

Out[59]=

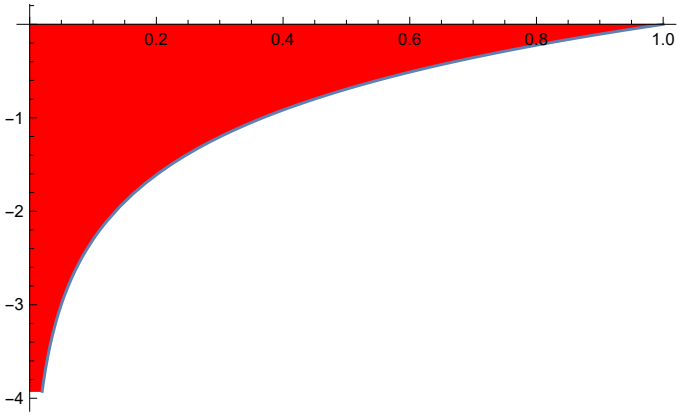


Výraz "rýchlo diverguje do nekonečna" sme nepoužili náhodou, keďže existujú aj funkcie, ktoré na intervale integrovania divergujú do nekonečna, ale napriek tomu majú konečný integrál.

In[60]:=
$$\int_0^1 \text{Log}[x]\ dx$$
```
Plot[Log[x], {x, 0, 1}, Filling → Axis, FillingStyle → {Red, Green}]
```

Out[60]= -1

Out[61]=



V nasledujúcom príklade vypočítame približne obsah plochy pod grafom funkcie $f(x) = x^2$ v intervale $(2, 3)$. Najskôr zadefinujeme vstupné údaje.

In[62]:=
```
f[x_]:= x^2
a = 2;
b = 3;
```

Potom oblasť (interval) integrovania rozdelíme na `n` rovnakých úsekov (za `n` volíme veľké číslo).

In[63]:=
```
n = 100;
```

Na každom úseku budeme funkciu $f(x) = x^2$ aproximovať pomocou konštatnej funkcie. Hodnota tejto konštanty sa bude rovnať hodnote funkcie v začiatočnom bode úseku. Keďže sme interval rozdelili na `n` rovnakých úsekov, tak dĺžku jedného úseku vypočítame nasledovne.

In[64]:=
```
dx:= (b - a)/n
```

`dx` definujeme pomocou oneskoreného priradenia (pozri podsekciu 8.1.1), aby sa v prípade zmeny hodnoty `n` automaticky zmenila aj hodnota `dx`. Ďalej pre každý úsek vypočítame obsah oblasti ohraničenej touto konštantnou funkciou a osou x. Hodnota funkcie na i-tom úseku bude daná nasledovne.

In[65]:=
```
F:= f[a + (i-1) * dx]
```

Nakoniec sčítame obsahy plôch všetkých úsekov a výsledok dáme pomocou funkcie `N` zapísať ako reálne číslo.

In[66]:=
```
∑_{i=1}^{n} F dx //N
```

Out[66]= 6.30835

Tento výsledok, ktorý sme získali približnou metódou porovnáme s presným výsledkom vypočítaným pomocou určitého integrálu.

In[67]:=
```
∫_2^3 x^2 𝐝x
```

Out[67]= 6.33333

Ak v približnej metóde zvýšime presnosť (t. j. zvýšime hodnotu **n**), dostaneme presnejší výsledok.

In[68]:=
```
n = 100 000;
∑_{i=1}^{n} F dx //N
```

Out[68]= 6.33331

V cvičení 10.7 sme na funkcii $f(x) = x^2$ overili pomocou funkcie **Limit** platnosť vzorca na derivovanie. Niečo podobné si ukážeme aj pre určitý integrál.

In[69]:=
```
∫_a^b x^2 𝐝x
```

Out[69]= $-\frac{a^3}{3} + \frac{b^3}{3}$

Obsah plochy pod grafom funkcie v intervale (a,b) vypočítaný pomocou delenia na konštantné úseky, za predpokladu, že počet týchto úsekov ide do nekonečna, sa bude rovnať výsledku získanému pomocou určitého integrálu.

In[70]:=
```
Clear[a, b, n]
Limit[∑_{i=1}^{n} F dx, n → ∞]
```

Out[70]= $\frac{1}{3}(-a^3 + b^3)$

Cvičenie 10.18

Vypočítajte rovnakým postupom určitý integrál funkcie $f(x) = \sin(x)$.

Poznámka 10.3

Z predchádzajúceho príkladu je zrejmé, odkiaľ pochádza značenie integrálu $\int_a^b f(x)\mathrm{d}x$. Znak $\int$ reprezentuje sumu a znak $\mathrm{d}x$ reprezentuje nekonečne malú dĺžku dielika.

10.2.3 Aplikácie integrálov

Integrály majú uplatnenie v mnohých odvetviach. Používajú sa napr. pri riešení diferenciálnych rovníc, z ktorých mnohé reprezentujú fyzikálne javy využívané v praxi. Ich aplikácie nájdeme aj pri výpočtoch dĺžky krivky, obsahu plochy, objemu a povrchu rotačných telies.

V tejto učebnici sme však limitovaní rozsahom učiva prvého semestra, a tak sa obmedzíme len na ukážku riešenia základného učebnicového príkladu pomocou softvéru Mathematica.

Výpočet obsahu ohraničenej plochy

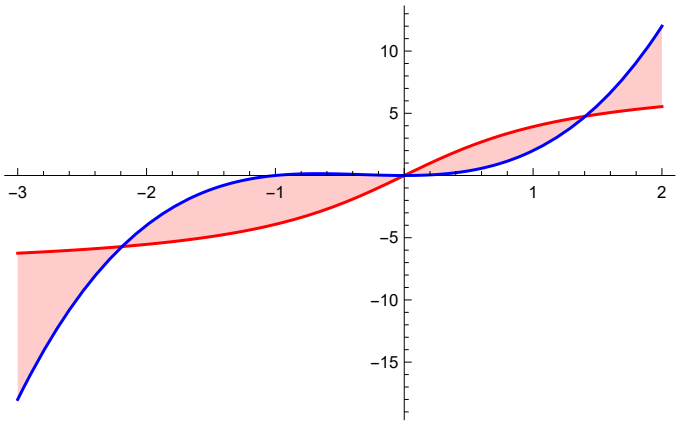
Zadanie: *Vypočítajte obsah plochy ohraničenej grafmi funkcií* $f(x) = 5artg(x)$ *a* $g(x) = x^3 + x^2$ *na intervale* $(-3; 2)$.

```
In[71]:= f[x_] := 5ArcTan[x]
         g[x_] := x^3 + x^2
         a = -3;
         b = 2;
```

Najskôr dáme vykresliť plochu, ktorej obsah počítame.

```
In[72]:= Plot[{5ArcTan[x], x^3 + x^2}, {x, a, b}, PlotRange → All,
           Filling → True, PlotStyle → {Red, Blue}]
```

Out[72]=



Potom pomocou funkcie **Solve** (pozri sekciu 11.1) vypočítame priesečníky grafov funkcií $f(x)$ a $g(x)$.

```
In[73]:= korene = Solve[f[x] == g[x], x, Reals];
         {b1, b2, b3} = Sort[N[x /. korene]]
```

```
Out[73]= {-2.19053, 0., 1.40693}
```

Dosadzovací operátor `/.` (pozri sekciu 7.1) sme použili, pretože funkcia **Solve** vracia korene v tvare `x → hodnota`. Keďže poradie riešení vo výstupe nie je vzostupné, usporiadali sme ich pomocou

funkcie **Sort** (pozri str. 48). V premenných b1, b2 a b3 sú uložené hodnoty, ktoré sú x-ovými súradnicami priesečníkov grafov funkcií $f(x)$ a $g(x)$ a zároveň aj koncovými bodmi podintervalov, na ktorých budeme integrovať. Keď integrujeme oblasť ohraničenú grafmi dvoch funkcií, tak od funkcie s vyššími funkčnými hodnotami odčitujeme funkciu s nižšími funkčnými hodnotami. Poradie funkcií $f(x)$ a $g(x)$ vo funkcii **Integrate** na jednotlivých podintervaloch môžeme určiť vizuálne z grafu. Rovnako dobre však funguje aplikovanie absolútnej hodnoty pri použití jednotného rozdielu $f(x) - g(x)$ pre všetky podintervaly.

```
In[74]:= S1 = Integrate[Abs[f[x] - g[x]], {x, a, b1}]
         S2 = Integrate[Abs[f[x] - g[x]], {x, b1, b2}]
         S3 = Integrate[Abs[f[x] - g[x]], {x, b2, b3}]
         S4 = Integrate[Abs[f[x] - g[x]], {x, b3, b}]

Out[74]= 4.13815
         5.86738
         2.06596
         1.68473
```

Nakoniec sčítame obsahy všetkých podoblastí.

```
In[75]:= S1 + S2 + S3 + S4

Out[75]= 13.7562
```

Obsah plochy ohraničenej grafmi funkcií $f(x)$ a $g(x)$ v intervale $(-3, 2)$ je 13.7562.

Poznámka 10.4

V skutočnosti pri výpočte obsahu plochy ohraničenej grafmi funkcií $f(x)$ a $g(x)$ nebolo potrebné počítať priesečníky grafov týchto funkcií. Stačilo zadať integrál z absolútnej hodnoty rozdielu funkcií na celom intervale.

```
In[76]:= Integrate[Abs[f[x] - g[x]], {x, a, b}] // N

Out[76]= 13.7562
```

Všimnite si, že v tomto prípade bol výpočtový čas o niečo dlhší v porovnaní s výpočtom na podintervaloch. Ako sa presne rieši úloha druhým spôsobom vo výpočtovom jadre sa môžeme len domnievať, ale pravdepodobne to nebude veľmi odlišné od postupu, akým sme to riešili my. Dlhší výpočtový čas funkcie **Integrate** v jednopríkazovom postupe v sebe zahŕňa aj čas výpočtov priesečníkov.

Nami prezentovaný postup slúžil na precvičenie použitia rôznych vstavaných funkcií.

10.3 Zhrnutie

V tejto kapitole sme si predstavili funkcie na derivovanie a integrovanie. V softvéri Mathematica používame na derivovanie dve základné vstavané funkcie **`Derivative`** a `D`. Funkcia **`Derivative`** derivuje funkciu v programátorskom zmysle slova podľa parametra. Funkcia `D` derivuje výraz (predpis funkcie) podľa nami zvolenej premennej (symbolu). Pri derivovaní funkcie v konkrétnom bode funkcia `D` najskôr nahradí symboly a až potom derivuje, čo pri nesprávnom zadaní príkazu môže zapríčiniť neočakávané chyby.

Na integrovanie je v softvéri Mathematica zabudovaná funkcia **`Integrate`**. Slúži na výpočet neurčitých aj určitých integrálov. Táto funkcia má podobný spôsob zadávania aj fungovania ako funkcia `D`. Pri jej používaní musíme mať tiež na pamäti, že nahrádzanie symbolov má prednosť pred integrovaním.

Softvér Mathematica umožňuje derivovať a integrovať aj funkcie a výrazy obsahujúce symboly, dokonca aj v prípadoch, keď sú tieto symboly súčasťou hraníc integrovania. Vstavané symbolické derivovanie a integrovanie je oproti väčšine programovacích jazykov veľkou prednosťou softvéru Mathematica.

Kapitola 11

Riešenie rovníc

V tejto kapitole si bližšie predstavíme funkcie softvéru Mathematica, ktoré slúžia na riešenie rovníc a sústav rovníc. Niektoré z nich riešia rovnice priamo, iné hľadajú riešenia približnými metódami.

Špeciálnym prípadom sústavy rovníc je lineárna sústava rovníc. Takéto sústavy sú spolu s vektormi a maticami predmetom štúdie lineárnej algebry. Funkciu **LinearSolve** si preto predstavíme až v kapitole 12.

11.1 Funkcie Solve a NSolve

Základnou vstavanou funkciou na riešenie rovníc a sústav rovníc je funkcia **Solve**. Zadáva sa dvoma spôsobmi: **Solve[sustava, nezname]** alebo **Solve[sustava, nezname, obor]**, kde **sustava** je rovnica alebo sústava rovníc (prípadne aj nerovníc) zapísaných ako zoznam, **nezname** je neznáma alebo zoznam neznámych (napr. **{x,y,...}**) a **obor** určuje množinu, na ktorej hľadáme riešenie.

Poznámka 11.1

Za **obor** môžeme dosadiť hodnoty: **Integers** (množina celých čísel), **Rationals** (množina racionálnych čísel), **Reals** (množina reálnych čísel) a **Complexes** (množina komplexných čísel).

Výstupom je zoznam riešení v tvare `{{x→riesenieX1, y→riesenieY1,...}, {x→riesenieX2, y→riesenieY2,...},...,{x→riesenieXn, y→riesenieYn,...}}`

```
In[1]:= sol=Solve[2x^2 + 4x - 10 == 0, x]

Out[1]= {{x → -1 - √6}, {x → -1 + √6}}
```

Rovnica z predchádzajúceho príkladu má dve riešenia, takže výstupom je zoznam s dvoma prvkami. A keďže táto rovnica je rovnicou len s jednou neznámou, každý prvok je zoznam s jedným prvkom v tvare `x→riesenie`.

Správnosť získaného výsledku overíme veľmi jednoducho. Do premennej **x** priradíme prvé riešenie a po zavolaní výrazu reprezentujúceho ľavú stranu rovnice dostaneme ako výsledok pravú stranu rovnice. Všimnite si, že sme na tento výraz aplikovali funkciu **Simplify**. Jej použitie bolo vynútené faktom, že hodnota premennej **x** je v príliš zložitom tvare a výsledný výraz by nebol softvérom automaticky upravený na jednoduchší tvar.

```
In[2]:= x = -1 - √6;
        2x^2 + 4x - 10 // Simplify

Out[2]= 0
```

Rovnakým spôsobom by sme overili aj správnosť druhého riešenia. Keďže sme si výstup funkcie **Solve** uložili do premennej **sol**, tak k jednotlivým riešeniam môžeme pristupovať pomocou **sol[[i]]**, kde **i** je poradové číslo riešenia (prvku v zozname **sol**). Tieto riešenia môžeme potom pomocou dosadzovacieho operátora **/.** dosádzať za premennú **x**, pozri sekciu 7.1.

```
In[3]:= sol[[1]]

Out[3]= {x → -1 - √6}

In[4]:= Clear[x]
        2x^2 + 4x - 10 == 0 /. sol

Out[4]= {True, True}
```

Keďže ľavá strana rovnice sa rovná pravej, tak sme dostali ako výstup zoznam odpovedí **{True, True}**, t. j. obidve riešenia vyhovujú rovnici.

Pri riešení sústavy rovníc zadávame ako vstupné argumenty zoznam rovníc tejto sústavy a zoznam neznámych týchto rovníc.

```
In[5]:= Solve[{x + y == 0, x - y == 1}, {x, y}]

Out[5]= {{x → 1/2, y → -1/2}}
```

Výstupom je zoznam riešení (v predchádzajúcom príklade len jedno) a každé riešenie je zapísané ako zoznam nahradení pre jednotlivé neznáme.

Cvičenie 11.1

Táto sústava rovníc sa dá riešiť aj graficky. Stačí zobraziť množinu bodov, pre ktorú platí $x + y = 0$ a množinu bodov, pre ktorú platí $x - y = 1$. Obidve množiny sú v tomto prípade priamky. V sekcii 5.2.2 sme predstavili funkciu **ContourPlot**, ktorá ako vstup môže dostať rovnicu. Využite túto funkciu na zobrazenie obidvoch rovníc. Výstup funkcie **Solve** použite na vykreslenie riešenia ako bodu. Zobrazte rovnice a riešenie v jednom grafe.

V prípade, že sústava rovníc nemá žiadne riešenie, výstupom bude prázdny zoznam.

```
In[6]:= Solve[{x + y == 0, x + y == 1}, {x, y}]

Out[6]= { }
```

Cvičenie 11.2

Rovnice z predchádzajúceho príkladu zobrazte graficky rovnakým spôsobom ako v cvičení 11.1.

Ak má sústava rovníc nekonečne veľa riešení, vo výstupe bude zápis reprezentujúci tieto riešenia.

```
In[7]:= Solve[{x + y == 2, 2x + 2y == 4}, {x, y}]

        Solve: Equations may not give solutions for all "solve" variables. ≫

Out[7]= {{y → 2 - x}}
```

V predchádzajúcom príklade je riešenie pre neznámu y vyjadrené pomocou neznámej x. Tento zápis reprezentuje priamku $y = 2 - x$ a každý z jej nekonečne veľa bodov je riešením zadanej sústavy rovníc. Naviac dostaneme správu o tom, že sme nemuseli dostať riešenie pre všetky neznáme.

Cvičenie 11.3

Rovnice z predchádzajúceho príkladu zobrazte graficky rovnakým spôsobom ako v cvičení 11.1.

V rovniciach môžeme používať aj symboly, ktoré nie sú v zozname neznámych. Ak tieto symboly nemajú priradenú hodnotu, sú považované za konštanty.

```
In[8]:= Solve[{x + d y == c, 2x + 2y == 4}, {x, y}]

Out[8]= {{x → -(c - 2d)/(-1 + d), y → -(2 - c)/(-1 + d)}}
```

Výstupom je všeobecné riešenie. Upozorňujeme, že aj v prípade, ak toto riešenie platí len za určitých podmienok, softvér Mathematica na to neupozorní. Napr. všeobecné riešenie z predchádzajúceho príkladu by malo byť doplnené podmienkou $d \neq 1$.

Namiesto zoznamu rovníc môžeme ako vstupný argument zadať rovnice sústavy tak, že každé dve za sebou nasledujúce rovnice sú oddelené od seba nie čiarkou, ale logickým operátorom `&&` ("a zároveň").

```
In[9]:= Solve[x^2 + y == 0 && x - y == 1, {x, y}]

Out[9]= {{x → 1/2 (-1 - √5), y → 1/2 (-3 - √5)},
         {x → 1/2 (-1 + √5), y → 1/2 (-3 + √5)}}
```

Keďže sústava z predchádzajúceho príkladu obsahuje kvadratickú a lineárnu rovnicu, výstupom je zoznam dvoch riešení.

Cvičenie 11.4

Lineárna funkcia má všeobecný predpis $f(x) = a\ x + b$, kde a, b sú nejaké reálne čísla.

Pomocou **Solve** vypočítajte hodnoty a, b funkcie, ktorej graf prechádza bodmi $[1, 3]$ a $[2, 8]$. Zobrazte graf tejto funkcie (pozri kapitolu 5) spolu s bodmi $[1, 3]$ a $[2, 8]$ (pozri kapitolu 9).

Takáto aproximácia matematickej funkcie medzi jej dvoma bodmi sa nazýva lineárna interpolácia. Funkcia **ListLinePlot** (pozri kapitolu 5) využíva podobný postup na vykreslenie spojitých dát.

Pomôcka: Pre funkciu, ktorej graf prechádza bodom $[x, y]$ platí $y = f(x)$.

Cvičenie 11.5

Kvadratická funkcia má všeobecný predpis $f(x) = a\ x^2 + b\ x + c$, kde a, b, c sú nejaké reálne čísla.

Pomocou **Solve** vypočítajte hodnoty a, b, c funkcie, ktorej graf prechádza bodmi $[-1, 4]$, $[0, 0]$ a $[1, 4]$. Zobrazte graf tejto funkcie spolu s bodmi $[-1, 4]$, $[0, 0]$ a $[1, 4]$.

Takáto aproximácia matematickej funkcie medzi jej troma bodmi sa nazýva kvadratická interpolácia. Funkcia **ListLinePlot** s voľbou **InterpolationOrder→2** využíva podobný postup na vykreslenie spojitých dát.

Pri riešení zložitejších úloh vracia funkcia **Solve** niekedy ako výstup riešenie zapísané v komplikovanom tvare.

```
In[10]:= sol = Solve[Sin[x] == 0, x]

Out[10]= {{x → ConditionalExpression[2π c₁, c₁ ∈ ℤ]},
          {x → ConditionalExpression[π + 2π c₁, c₁ ∈ ℤ]}}
```

V predchádzajúcom príklade sa hodnota funkcie $f(x) = sin(x)$ rovná nule v nekonečne veľa bodoch. A keďže výstupom nemôže byť nekonečný zoznam, tak je riešenie zapísané s použitím funkcie `ConditionalExpression`, kde prvý argument je všeobecné riešenie a druhý udáva podmienky, za ktorých je toto riešenie platné. V tomto prípade teda funkcia vrátila ako výstup riešenia $2\pi c$ a $\pi + 2\pi c$, za podmienky, že c je celé číslo. Pomocou dosadzovacieho operátora `/.` vieme získať hodnotu pre ľubovoľné c_1.

```
In[11]:= sol /. C[1] → 1

Out[11]= {{x → 2 π}, {x → 3 π}}
```

Namiesto znaku c_1 sme použili **C[1]**. Je to len iný zápis toho istého znaku, tak ako je **Pi** iný zápis π.

Výsledok v netypickom tvare dostaneme aj pri riešení niektorých polynomických rovníc.

```
In[12]:= sol = Solve[x^3 + 2x^2 + 4x + 10 == 0, x]

Out[12]= {{x → Root[10 + 4# + 2#^2 + #^3& , 1]},
          {x → Root[10 + 4# + 2#^2 + #^3& , 2]},
          {x → Root[10 + 4# + 2#^2 + #^3& , 3]}}
```

V predchádzajúcom príklade bola vo výstupe na zápis riešenia použitá funkcia **Root** (z angl. koreň). Jej prvým parametrom je anonymná funkcia (pozri sekciu 8.2) vytvorená z ľavej strany zadanej rovnice a druhý parameter je poradie koreňa (riešenia), ktorý hľadáme. Keďže funkcia **Solve** nedokáže priamo vypočítať konkrétne čísla, tak riešenia vracia v takomto tvare. Na prvý pohľad sa môže zdať, že vo výstupe je len zopakované naše zadanie (tri korene zadanej kubickej rovnice). Avšak s týmito zdanlivo abstraktnými tvarmi riešenia sa dá ďalej pracovať. Môžeme ich napríklad približne vyčísliť.

```
In[13]:= sol // N

Out[13]= {{x → -2.2236}, {x → 0.111802 - 2.11771 i}, {x → 0.111802 + 2.11771 i}}
```

Z numerického zápisu riešenia zistíme, napr., obor jednotlivých riešení. Zadaná kubická rovnica má jeden koreň reálny a dva komplexné. V prípade, že chceme pracovať len na vybranom obore, napr., na množine reálnych čísel, môžeme pomocou tretieho parametra ohraničiť obor riešenia.

```
In[14]:= sol = Solve[x^3 + 2x^2 + 4x + 10 == 0, x, Reals]

Out[14]= {{x → Root[10 + 4# + 2#^2 + #^3& , 1]}}
```

Pomocou parametra **Reals** bola množina riešení zúžená na obor reálnych čísel. Pripomeňme si, že parameter **Reals** sa rovnakým spôsobom využíval aj vo funkcii **Simplify** (pozri sekciu 6.1).

Funkcia **NSolve** rieši sústavu rovníc numericky a vracia výstup v tvare približných čísel. Rovnaký výstup by sme dostali, keby sme použili funkciu **Solve** a potom na jej výstup aplikovali funkciu **N**.

```
In[15]:= NSolve[x^3 + 2x^2 + 4x + 10 == 0, x, Reals]

Out[15]= {{x → -2.2236}}
```

Cvičenie 11.6

Pomocou funkcie **Plot** zobrazte ľavú stranu rovnice a vizuálne overte, či graf funkcie pretína x-ovú os v bode $x = hodnota$, ktorú sme dostali vo výstupe posledného príkladu.

V prípade, že funkcie **Solve** alebo **NSolve** nedokážu nájsť riešenie, vrátia ako výstup zadaný príkaz so správou oznamujúcou, že zadaný systém (rovnica, sústava rovníc...) nedokáže byť vyriešený metódami zabudovanými v tejto vstavanej funkcii.

```
In[16]:= NSolve[2Sin[x] == x, x]

         NSolve: This system cannot be solved with the methods available to NSolve. ≫

Out[16]= NSolve[2Sin[x] == x, x]
```

V prvom vstupnom argumente funkcií **Solve** a **NSolve** môžu byť zahrnuté aj nerovnice. Napríklad, ak nás zaujímajú len záporné riešenia, môžeme pridať nerovnicu $x < 0$.

```
In[17]:= Solve[x^3 + x^2 == 0 && x < 0, x]

Out[17]= x → -1
```

Upozorňujeme, že nerovnicu $x < 0$ nemôžeme pridať ako tretí vstupný argument reprezentujúci obor riešení. Ak by sme sa o to pokúsili, výstupom by bol zadaný príkaz a zobrazila by sa informácia oznamujúca, že $x < 0$ nie je platný vstupný argument na určenie oblasti.

```
In[18]:= Solve[x^3 + x^2 == 0, x, x < 0]

         Solve: Warning: x < 0 is not a valid domain specification. Assuming it is
         a variable to eliminate.

         Solve: x < 0 is not a valid variable.

Out[18]= Solve[x^3 + x^2 == 0, x, x < 0]
```

Funkcie **Solve** a **NSolve** nemôžu dostať ako prvý vstupný argument takú sústavu `n` neznámych, ktorej riešenie je `n`-rozmerná oblasť. V prípade, že takú sústavu zadáme s výstupom obdržíme správu informujúcu, že množina riešení obsahuje `n`-rozmerný prvok a odporučenie použiť funkciu **Reduce**, pozri sekciu 11.2.

```
In[19]:= Solve[x^2 + y^2 < 1 && x > 0, {x, y}, Reals]

         Solve: A solution set contains a full−dimensional component; use Reduce
         for complete solution information.

Out[19]= {{}}
```

Pri riešení niektorých úloh sa môže stať, že výpočet trvá príliš dlho a nezdá sa, že skončí. Vtedy môžeme tento výpočet prerušiť (predčasne ukončiť) príkazom z horného menu `Evaluation → Abort Evaluation` (z angl. preruš vyhodnocovanie). Preruší sa len konkrétny výpočet, všetky predtým zadané definície a priradenia zostanú platné.

```
In[20]:= Solve[Cos[x] Log[x] + Sin[x]/x == 0, x]

Out[20]= $Aborted
```

11.2 Funkcia Reduce

Funkcia **Reduce[tvrdenie, premenne]** sa používa na riešenie podobných úloh ako funkcia **Solve**, ale výstup zapisuje iným spôsobom. Rovnako ako pri funkcii **Solve**, aj pri funkcii **Reduce** je možné zúžiť množinu riešení pomocou tretieho (nepovinného) parametra **oblast**. Prvým vstupným parametrom môže byť tak isto rovnica alebo sústava rovníc (prípadne aj nerovníc). Avšak funkcia **Reduce** umožňuje zadať aj samostatnú nerovnicu alebo sústavu nerovníc, ktorá neobsahuje žiadnu rovnicu.

```
In[21]:= Reduce[t > 3, t, Integers]
         Reduce[x^2 + y^2 < 1 && x > 0, {x, y}, Reals]
```

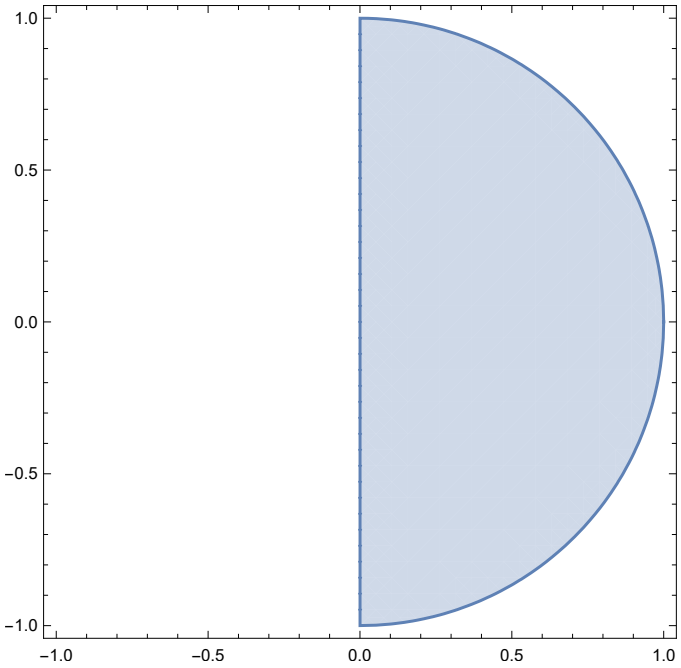
$$\text{Out[21]=}\quad t \in \mathbb{Z} \;\&\&\; t \geq 4$$

$$0 < x < 1 \;\&\&\; -\sqrt{1 - x^2} < y < \sqrt{1 - x^2}$$

Riešením predchádzajúceho príkladu je množina bodov $[x, y]$, ktorá reprezentuje polkruh. Na zobrazenie oblasti roviny vyjadrenej pomocou logických tvrdení slúži funkcia **RegionPlot**, pozri podsekciu 5.2.5.

```
In[22]:= RegionPlot[x^2 + y^2 < 1 && x > 0, {x, -1, 1}, {y, -1, 1}]
```

Out[22]=



Funkcia **Reduce** vracia ako výstup zjednodušenú verziu zadaného tvrdenia. Ak je tvrdením rovnica, tak výstupom (zjednodušeným tvrdením) bude riešenie tejto rovnice.

```
In[23]:= Reduce[2x^2 + 4x - 10 == 0, x]
```

$$\text{Out[23]=}\quad x == -1 - \sqrt{6} \;||\; x == -1 + \sqrt{6}$$

Výstupom v predchádzajúcom príklade nie je zoznam dvoch riešení, ale jedno tvrdenie, že premenná **x** sa rovná $-1-\sqrt{6}$ alebo $-1+\sqrt{6}$. Slovo "alebo" je vyjadrené logickým operátorom ||.

V prípade sústavy rovníc s jedným riešením je výstupom tvrdenie obsahujúce dve tvrdenia o hodnotách neznámych spojené logickým operátorom && ("a zároveň").

```
In[24]:= Reduce[x + y == 0 && x - y == 1, {x, y}]

Out[24]= x == 1/2 && y == -1/2
```

Cvičenie 11.7

Pomocou funkcie **Reduce** zjednodušte tvrdenie $x^2+y=0$ a $x-y=1$.

Cvičenie 11.8

Pomocou funkcie **Reduce** zjednodušte tvrdenie $\sin x = 0$.

Ďalší rozdiel medzi funkciami **Solve** a **Reduce** spočíva v tom, že ak sa v prvom vstupnom argumente nachádzajú aj iné symboly ako v druhom vstupnom argumente, tak vo funkcii **Solve** sú považované za konštanty, zatiaľ čo funkcia **Reduce** zahrnie do výstupného tvrdenia aj špeciálne prípady súvisiace s týmito symbolmi.

```
In[25]:= Reduce[{x + d y == c, 2x + 2y == 4}, {x, y}]

Out[25]= (d == 1 && c == 2 && y == 2 - x) || (-1 + d ≠ 0 &&
         x == (-c + 2d)/(-1 + d) && y == 2 - x)
```

Ak použijeme funkciu **Reduce** na riešenie polynomickej rovnice, ktorá nemá analytické riešenie, tak vo výstupe bude riešenie zapísané pomocou funkcie **Root**, ktorú sme si predstavili v sekcii 11.1.

```
In[26]:= Reduce[x^3 + 2x^2 + 4x + 10 == 0, x]

Out[26]= x == Root[10 + 4#1 + 2#1^2 + #1^3 &, 1] ||
         x == Root[10 + 4#1 + 2#1^2 + #1^3 &, 2] ||
         x == Root[10 + 4#1 + 2#1^2 + #1^3 &, 3]
```

Rovnako ako funkcie **Solve** a **NSolve**, ani funkcia **Reduce** nedokáže nájsť riešenia rovnice $2\sin x = x$.

```
In[27]:= Reduce[2Sin[x] == x, x]

         Reduce: This system cannot be solved with the methods available to Reduce. ≫

Out[27]= Reduce[2Sin[x] == x, x]
```

Cvičenie 11.9

Pomocou **Reduce** nájdite definičný obor funkcie $f(x) = \arcsin(\frac{3x-9}{4})$.
Pomôcka: Pre argument x funkcie $f_1(x) = \arcsin(x)$ platí: $-1 \leq x \leq 1$.

Cvičenie 11.10

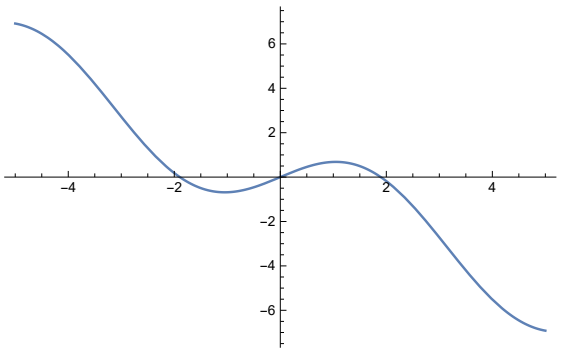
Pomocou **Reduce** nájdite definičný obor funkcie $g(x) = \sqrt{-2 + \log_3(2x - 7)}$.
Pomôcka: Pre argument x funkcie $g_1(x) = \sqrt{x}$ platí: $x \geq 0$.
Pomôcka: Pre argument x funkcie $g_2(x) = \log_a(x)$ platí: $x > 0$.

11.3 Funkcia FindRoot

Obidve predchádzajúce funkcie riešia zadané úlohy priamo, a preto zlyhávajú pri niektorých komplikovanejších úlohách. Takéto úlohy môžeme riešiť iteračne. Základnou vstavanou funkciou na riešenie rovníc a sústav rovníc je funkcia **FindRoot[vyraz, {x, x$_0$}]**, ktorá pomocou Newtonovej iteračnej metódy nájde hodnotu premennej **x**, pre ktorú sa **vyraz** rovná nule, t. j. nájde koreň rovnice **vyraz=0**. Newtonova iteračná metóda vyžaduje zadanie štartovacieho bodu **x$_0$**, t. j. počiatočný odhad hodnoty, ktorú hľadáme. **x$_0$** najjednoduchšie určíme z grafu. Pomocou funkcie **Plot** dáme vykresliť graf pre **vyraz** a približnú hodnotu, v ktorej funkcia $f(x) =$ **vyraz** pretína x-ovú os dosadíme za **x$_0$**. Ak sa na prvýkrát nepodarí nájsť tento priesečník (pre nevhodne zvolený interval zobrazenia), rozšírime dostatočne tento interval.

```
In[28]:= Plot[2Sin[x] -  x, {x, -5, 5}]
         FindRoot[2Sin[x] -  x, {x, 1}]
```

Out[28]=




```
Out[29]= x → 0.
```

Keďže výsledkom iteračnej metódy je len približné riešenie, tak výstupom je reálne číslo (pozri sekciu 3.3). Aj keď je riešení viac, je nájdené len jedno a zvyčajne to, ktoré je najbližšie k počiatočnému odhadu.

Funkcia **FindRoot** môže dostať prvý vstupný argument aj v tvare **lava == prava**, kde **lava** a **prava** sú ľavá a pravá strana rovnice.

```
In[30]:= FindRoot[2Sin[x] ==  x, {x, 1}]

Out[30]= x → 0.
```

Ak chceme nájsť súčasne viacero riešení, tak ich počiatočné odhady zadávame ako zoznam.

```
In[31]:= FindRoot[2Sin[x] == x, {x, {-2, 0, 2}}]

Out[31]= {x → {-1.89549, 0., 1.89549}}
```

Cvičenie 11.11

Rovnica $9\sin(x) = x$ má niekoľko riešení. Zobrazte ľavú aj pravú stranu rovnice ako dve funkcie v jednom grafe. Z grafu je zrejmé, že nula nie je jediné riešenie. Vypočítajte ďalšie riešenia.

11.4 Aplikácia riešenia rovníc

Nájdenie predpisu dotyčnice

Dotyčnica grafu funkcie $f(x)$ v bode $[x_0, f(x_0)]$ je priamka, ktorej graf má s grafom funkcie $f(x)$ jediný spoločný bod (tzv. dotykový) $[x_0, f(x_0)]$ a jej smernica sa rovná hodnote derivácie funkcie $f(x)$ v bode $[x_0, f(x_0)]$.

Zadanie: *Nájdite rovnicu dotyčnice grafu funkcie $f(x)$ v bode $[x_0, f(x_0)]$.*

Keďže dotyčnica je priamka (t. j. je grafom lineárnej funkcie), vieme ju vyjadriť predpisom $d(x) = ax + b$.

```
In[32]:= d[x_]:= a x + b
```

Informáciu, že bod $[x_0, f(x_0)]$ je spoločným bodom grafu funkcie a dotyčnice zapíšeme matematicky nasledovne: $f(x_0) = d(x_0)$. Podobne vieme vyjadriť aj vzťah, že smernica dotyčnice (t. j. derivácia lineárnej funkcie $d(x)$) sa rovná derivácii funkcie $f(x)$ v bode $[x_0, f(x_0)]$; $d'(x) = f'(x_0)$. Obidva tieto vzťahy sú rovnosti, ktoré môžeme riešiť ako sústavu rovníc.

```
In[33]:= sol = Solve[{f[x0] == d[x0], d'[x0] == f'[x0]}, {a, b}]

Out[33]= {{a → f'[x0], b → f[x0] - x0 f'[x0]}}
```

Výsledkom je jedno riešenie pre neznáme a a b. Nahraďme ich do pôvodného predpisu.

```
In[34]:=  a x + b /. sol[[1]] // FullSimplify

Out[34]= f[x0] + (x - x0) f'[x0]
```

Zameňme pôvodný predpis všeobecnej priamky predpisom všeobecnej dotyčnice.

```
In[35]:= d[x_] := Evaluate[d[x] /. sol[[1]]]
         d[x]

Out[35]= f[x0] + (x - x0) f'[x0]
```

Výsledkom je predpis dotyčnice, ktorý poznáme z hodín matematickej analýzy a používali sme ho v predchádzajúcich príkladoch.

Lokálne extrémy

Pomocou derivácií vieme získať užitočné informácie o grafoch funkcií. Ak je prvá derivácia funkcie na nejakom intervale kladná (resp. záporná), tak je funkcia na tomto intervale rastúca (resp. klesajúca). Ak je prvá derivácia v nejakom bode funkcie nulová, tak je tento bod kandidátom na lokálny extrém funkcie, t. j. funkcia v ňom môže, ale nemusí mať lokálny extrém. Takýto bod označujeme pojmom "stacionárny bod". Ak je druhá derivácia v stacionárnom bode kladná (resp. záporná), tak funkcia má v tomto bode lokálne minimum (resp. maximum).

Ak je druhá derivácia na nejakom intervale kladná (resp. záporná), tak je funkcia na tomto intervale konvexná (resp. konkávna). Ak je druhá derivácia v nejakom bode funkcie nulová, tak je tento bod kandidátom na inflexný bod (zmena konvexnosti funkcie na konkávnosť alebo opačne).

Zadanie: *Nájdite lokálne extrémy a inflexné body funkcie* $f(x) = \frac{x^3}{3} - x^2 - 3x$.

Zobrazte graf tejto funkcie a vyznačte na ňom body lokálnych extrémov funkcie a inflexné body grafu.

Najskôr pomocou funkcie `Solve` (pozri sekciu 11.1) nájdeme stacionárne body funkcie.

```
In[36]:= f[x_] := x^3/3 - x^2 - 3x
         stacbody = Solve[D[f[x], x] == 0, x]

Out[36]= {{x → -1}, {x → 3}}
```

Tieto dva body sú kandidátmi na lokálne extrémy. Pomocou druhej derivácie overíme, či funkcia má v týchto bodoch skutočne lokálne extrémy a ak áno, tak zistíme, o aký typ extrému ide. Využijeme dosadzovací operátor `/.` (pozri sekciu 7.1) a fakt, že funkcia `Solve` vracia ako výstup zoznam výsledkov v tvare `x → hodnota`.

```
In[37]:= D[f[x], {x, 2}] /. stacbody[[1]]
         D[f[x], {x, 2}] /. stacbody[[2]]

Out[37]= -4
         4
```

Obdobným spôsobom zistíme funkčnú hodnotu v týchto bodoch.

```
In[38]:= f[x] /. stacbody[[1]]
         f[x] /. stacbody[[2]]

Out[38]= 5/3
         -9
```

Funkcia má v bode $x = -1$ lokálne maximum a nadobúda v ňom hodnotu $\frac{5}{3}$ a v bode $x = 3$ lokálne minimum a nadobúda v ňom hodnotu -9.

Pomocou druhej derivácie nájdeme kandidátov na inflexné body a pomocou tretej derivácie potvrdíme, že skutočne ide o inflexné body.

```
In[39]:= inflex = Solve[D[f[x], {x, 2}] == 0, x]

Out[39]= {{x → 1}}

In[40]:= D[f[x], {x, 3}] /. inflex[[1]]

Out[40]= 2

In[41]:= f[x] /. inflex[[1]]

Out[41]= -11/3
```

Funkcia má v bode $x = 1$ inflexný bod a nadobúda v ňom hodnotu $-\frac{11}{3}$.

Graf funkcie dáme vykresliť pomocou **Plot**, body pomocou **Graphics** a **Point** (pozri kapitolu 9) a spolu do jedného grafu ich zobrazíme pomocou funkcie **Show** (pozri str. 62).

Keďže x-ové súradnice maxima, minima a inflexného bodu máme k dispozícii len v tvare `x → hodnota` pomocou dosadzovacieho operátora ich uložíme do pomocných premenných.

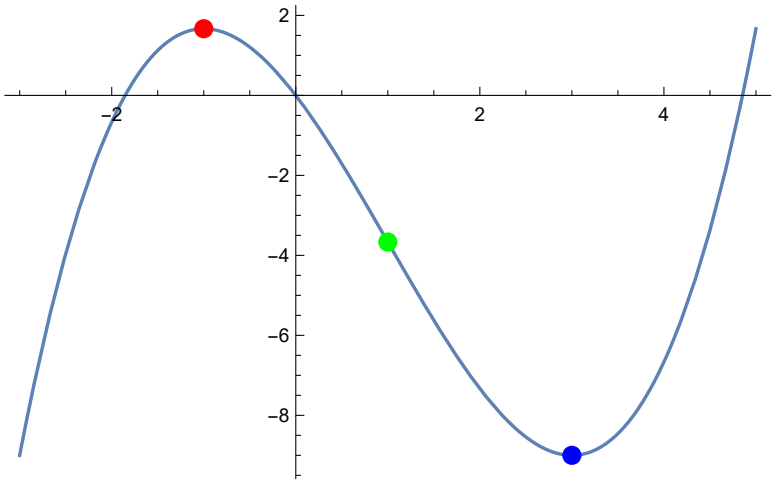
```
In[42]:= xmax = x /. stacbody[[1]]
         xmin = x /. stacbody[[2]]
         xinfl = x /. inflex[[1]]

Out[42]= -1
         3
         1
```

Tieto premenné potom využijeme pri zobrazovaní týchto bodov.

```
In[43]:= graff = Plot[f[x], {x, -3, 5}];
         max = Graphics[{Red, PointSize[0.025], Point[{xmax, f[xmax]}]}];
         min = Graphics[{Blue, PointSize[0.025], Point[{xmin, f[xmin]}]}];
```

```
infl = Graphics[{Green, PointSize[0.025], Point[{xinfl, f[xinfl]}]}];
Show[graff, max, min, infl]
```

Out[43]=



Cvičenie 11.12

Pomocou derivácií zistite, či je funkcia v bode $x = 2.5$

a) rastúca alebo klesajúca,

b) konvexná alebo konkávna.

11.5 Zhrnutie

V tejto kapitole sme si predstavili niekoľko vstavaných funkcií, ktoré slúžia na riešenie rovníc, nerovníc a ich sústav. Funkcie `Solve` a `NSolve` dávajú výsledky v tvare zoznamu riešení. `Solve` vracia presné čísla, ktoré môžu byť zapísané v komplikovanom tvare a `NSolve` približné čísla. Dokážu vyriešiť len systémy (s `n` neznámymi), ktorých riešenia sú maximálne `n-1`-rozmerné množiny bodov, napr. pri 2 neznámych, riešením môžu byť body, alebo priamka riešení, ale nie rovina ani žiadna 2-rozmerná podoblasť.

Funkcia `Reduce` vracia ako výstup zjednodušené logické tvrdenia reprezentujúce riešenia sústav rovníc a nerovníc. Dokáže vyriešiť aj systémy (s `n` neznámymi), ktorých riešenia sú `n`-rozmerné množiny bodov. Vyššie spomenuté funkcie riešia systémy len symbolicky, takže niektoré obtiažnejšie úlohy sú pre ne neriešiteľné.

Takéto typy úloh (len rovnice alebo sústavy rovníc, ktorých riešenia sú len body alebo zoznamy bodov s konečným počtom prvkov) sa dajú riešiť numerickými metódami s použitím funkcie `FindRoot`. Táto funkcia hľadá riešenie iteračne a vyžaduje zadanie počiatočného odhadu riešenia.

Kapitola 12

Vektory a matice

Vektory a matice sú základným nástrojom v počítačovej grafike, pri numerickom riešení diferenciálnych rovníc a v mnohých ďalších oblastiach.

V tejto kapitole sa oboznámime s tým, ako sa v softvéri Mathematica zadávajú a používajú vektory, matice a ich aplikácie. Predstavíme tiež funkciu **LinearSolve**, ktorá slúži na riešenie lineárnych sústav rovníc.

12.1 Vektory

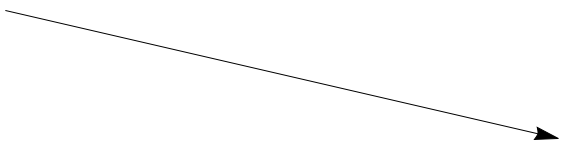
Vektory sú v softvéri Mathematica reprezentované zoznamom čísel. Zoznam n čísel reprezentuje n rozmerný vektor.

```
In[1]:= vektor = {5.4, -1.25};
        vektor2 = {3, -2.832, 0, 11/3, 7};
```

Na vektor sa môžeme pozerať ako na geometrický objekt so začiatkom v bode 0. Graficky vieme zobraziť dvojrozmerné vektory pomocou funkcie **Graphics** ako grafický objekt **Arrow**.

```
In[2]:= Graphics[Arrow[{{0, 0}, vektor}]]
```

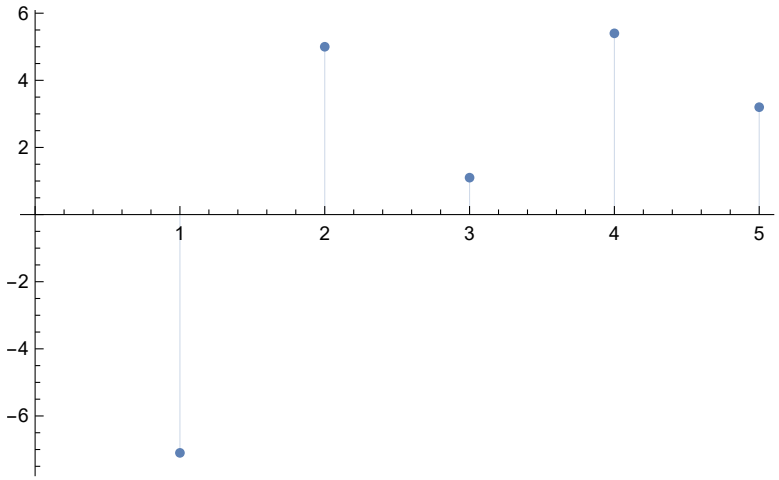
In[3]:=



Trojrozmerné vektory zobrazujeme rovnakým spôsobom pomocou funkcie **Graphics3D**. Viacrozmerné vektory sa dajú zobraziť pomocou funkcie **ListPlot**.

```
In[4]:= ListPlot[{-7.1, 5., 1.1, 5.4, 3.2}, Filling → Axis]
```

In[5]:=



Poznámka 12.1

Takto zobrazený vektor ponúka iný pohľad na vektory. n-rozmerný vektor je funkcia, ktorej definičný obor sú celé kladné čísla od 1 po n. Tento pohľad na funkcie sa často využíva v počítačových vedách. Funkcia sa aproximuje po častiach konštantnou funkciou, ktorá sa dá jednoduchšie reprezentovať v počítači ako konečnorozmerný vektor.

V softvéri Mathematica je zabudovaných viacero operácií s vektormi. Aritmetické operácie (pozri sekciu 3.1) fungujú po zložkách vektora. Napríklad **+** sčíta hodnoty na rovnakých pozíciách sčitovaných vektorov.

In[6]:=
```
{1/4, -1.2\} + \{1/3, 2.6\}
```

Out[6]= $\{\frac{7}{12}, 1.4\}$

Funkcia **Norm** vracia dĺžku vektora (v matematike je označovaná aj pojmom "Euklidova norma").

In[7]:=
```
Norm[vektor]
```

Out[7]= 5.67362

Vzdialenosť dvoch bodov sa dá vypočítať ako norma vektora, ktorého začiatočným a koncovým bodom sú dané body alebo v softvéri Mathematica priamo pomocou vstavanej funkcie **EuclideanDistance**.

In[8]:=
```
A = {1, 1}; B = {4, 5};
Norm[B - A]
EuclideanDistance[A, B];
```

Out[8]= 5

Cvičenie 12.1

Zadefinujte užívateľskú funkciu (pozri kapitolu 8), ktorá ako vstup dostane zoznam troch bodov reprezentujúcich vrcholy trojuholníka; a vráti zoznam dĺžok jeho strán.

Funkcia **Norm** dokáže pracovať aj s vektormi, ktorých súradnice sú symboly.

```
In[9]:= Norm[{x, y, z, w}]
```

$$\text{Out[9]=}\ \sqrt{\text{Abs[w]}^2 + \text{Abs[x]}^2 + \text{Abs[y]}^2 + \text{Abs[z]}^2}$$

Výstupom v predchádzajúcom príklade je vzorec na výpočet dĺžky vektora. Na každú súradnicu vektora je pred umocnením aplikovaná absolútna hodnota. V prípade výpočtu Euklidovskej dĺžky (pomocou Euklidovej 2-normy) sa môže táto operácia zdať zbytočná, keďže následne číslo umocníme na druhú. Avšak, ak by hodnoty vstupného vektora boli komplexné čísla, vynechaním tejto operácie by sme dostali nesprávny výsledok (keďže $|a + bi| = \sqrt{a^2 + b^2}$). Okrem toho funkcia **Norm** umožňuje po zadaní nepovinného vstupného parametra `p` aj výpočet všeobecnejšej p-normy, v ktorej je prítomnosť absolútnej hodnoty zrejmá. Jej definíciu dostaneme vo výstupe nasledujúceho príkazu.

```
In[10]:= Norm[{x, y, z}, p]
```

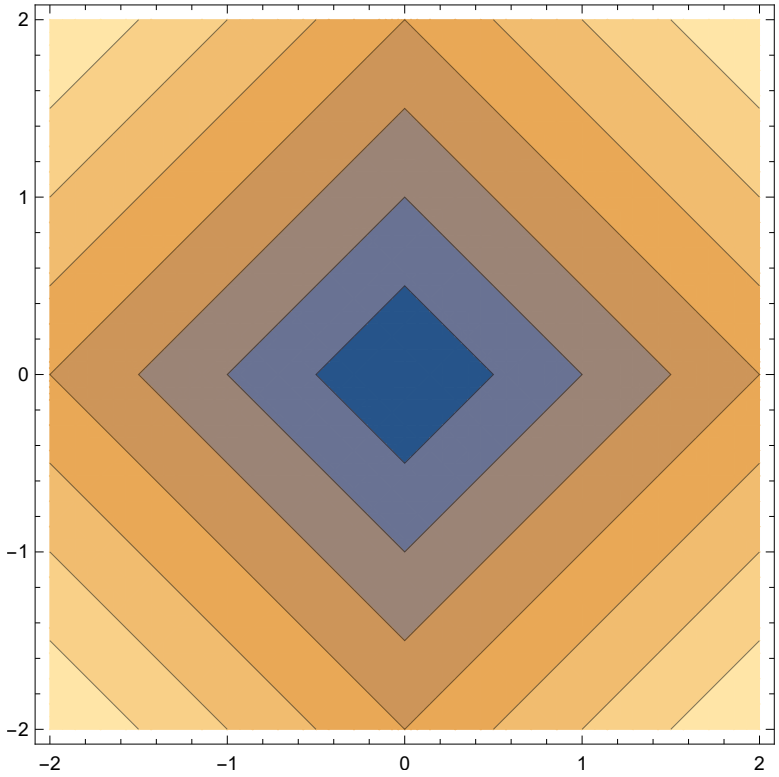
$$\text{Out[10]=}\ \left(\text{Abs[x]}^p + \text{Abs[y]}^p + \text{Abs[z]}^p\right)^{1/p}$$

Z množiny p-noriem sú okrem základných 2-noriem najčastejšie používané 1-normy a ∞-normy.

1-norma (označovaná aj pojmami "Manhattan norma" alebo "norma taxikára") reprezentuje vzdialenosť dvoch bodov ako ich najkratšiu spojnicu po pravouhlej mriežke. Graficky sa táto norma dá zobraziť pomocou funkcie **ContourPlot**. Zobrazíme hodnotu 1-normy pre body(vektory) na štvorci okolo bodu 0.

```
In[11]:= ContourPlot[Norm[{x, y}, 1], {x, -2, 2}, {y, -2, 2}]
```

In[12]:=



∞-norma (označovaná aj ako "maximová norma") je definovaná ako maximálna absolútna hodnota zo všetkých súradníc.

In[13]:=
```
Norm[{x, y, z}, ∞]
Norm[vektor, ∞]
```

Out[13]=
```
Max[Abs[x], Abs[y], Abs[z]]
5.4
```

Umocňovanie čísel na nekonečno a nekonečná odmocnina v klasickom ponímaní mocnín a odmocnín nedávajú zmysel. Z tohoto dôvodu sa pod pojmom "∞-norma" rozumie limita p-normy pre $p \to \infty$ a výsledná hodnota tejto limity je maximálna absolútna hodnota zo všetkých súradníc vstupného vektora.

Cvičenie 12.2

Predchádzajúce tvrdenie nebudeme dokazovať. Ukážeme si však, že to platí pre nejaký konkrétny vektor pre dostatočne veľké p. Vytvorte zoznam p-noriem vektora $(1.1, 2.2, 3.3)$ pre $p = 1, 2, 3, ..., 100$. Približujú sa tieto hodnoty postupne k hodnote najväčšej súradnice?

Cvičenie 12.3

1-normu sme zobrazili pomocou funkcie `ContourPlot`. Zobrazte pomocou nej aj Euklidovu a maximovú normu.

Normovaný vektor získame z pôvodného vektora, keď každú jeho súradnicu vydelíme jeho normou alebo použijeme vstavanú funkcia **Normalize**. Jej vstupom je pôvodný vektor a výstupom normovaný vektor.

```
In[14]:= Normalize[vektor]

Out[14]= {0.974239, -0.225518}
```

Normovaný vektor má rovnaký smer ako pôvodný vektor, ale jeho dĺžka sa rovná jednej, keďže bol vytvorený vydelením Euklidovou normou (t. j. dĺžkou pôvodného vektora).

```
In[15]:= Normalize[{x, y, z}]

Out[15]= {x/Sqrt[Abs[x]^2 + Abs[y]^2 + Abs[z]^2],
          y/Sqrt[Abs[x]^2 + Abs[y]^2 + Abs[z]^2],
          z/Sqrt[Abs[x]^2 + Abs[y]^2 + Abs[z]^2]}
```

Cvičenie 12.4

Na výstup z predchádzajúceho príkladu aplikujte funkciu **Norm**. Výsledok by podľa definície normovaného vektora mal byť 1, ale keďže softvér Mathematica neupravuje všetky výrazy automaticky, bude zapísaný v komplikovanejšom tvare. Aplikujte na tento výsledok funkciu **Simplify** (pozri sekciu 6.1).

Cvičenie 12.5

Vytvorte ľubovoľný nenulový 2D vektor a znormujte ho. Zobrazte pôvodný a normovaný vektor do jedného obrázku.

Na výpočet skalárneho súčinu (po angl. dot product) slúži funkcia **Dot**. Skalárny súčin sa dá zadať aj pomocou bodky z klávesnice alebo z palety (pozri Pozn. 3.2).

```
In[16]:= Dot[{1, 2, 7}, {-3, 3/4, 7}];
         {1, 2, 7}.{-3, 3/4, 7}

Out[16]= 95/2
```

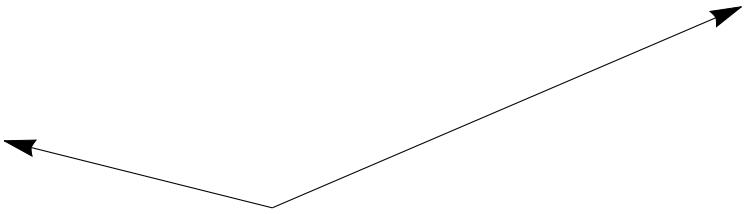
Cvičenie 12.6

Norma vektora sa dá vypočítať aj ako odmocnina zo skalárneho súčinu vektora samého so sebou. Zadefinujte užívateľskú funkciu **norma** pomocou tohto vzťahu.

Pomocou skalárneho súčinu sa dá vypočítať aj uhol medzi vektormi.

```
In[17]:= u = {7, 3}; v = {-4, 1};
         Graphics[{Arrow[{{0, 0}, u}], Arrow[{{0, 0}, v}]}]
```

In[18]:=



Vzorec na jeho výpočet je daný vzťahom $\arccos\left(\frac{\mathbf{u}}{||\mathbf{u}||} \cdot \frac{\mathbf{v}}{||\mathbf{v}||}\right)$ (na overenie, či sú dva nenulové vektory kolmé, stačí, aby platilo $\mathbf{u}.\mathbf{v} = 0$).

```
In[19]:= ArcCos[Normalize[u].Normalize[v]] // N

Out[19]= 2.49172
```

Kedže sme ako súradnice vektorov zadali celé (t. j. presné) čisla, výsledok bez použitia funkcie **N** by bol tiež presné číslo (pravdepodobne v komplikovanom tvare).

Ak chceme previesť radiány na stupne, môžeme tak urobiť pomocou predelenia uhla v radiánoch konštantou **Degree**.

```
In[20]:= 2.49172 / Degree

Out[20]= 142.765
```

Poznámka 12.2

Konštanta **Degree** je hodnota $\pi/180$; t. j. zlomok radiánu prislúchajúci jednému stupňu.
Z tohoto dôvodu nasledujúci zápis vracia hodnotu sínusu pre $\pi/2$.

```
In[21]:= Sin[90 Degree]

Out[21]= 1
```

Ak chceme opačný prevod, z radiánov na stupne, musíme použiť delenie.

Cvičenie 12.7

V softvéri Mathematica je zabudovaná funkcia **VectorAngle** slúžiaca na výpočet uhlov dvoch nenulových vektorov. Overte, či táto funkcia dá rovnaký výsledok ako ten, ktorý sme dostali vo výstupe predchádzajúceho príkladu.

Cvičenie 12.8

Zadefinujte užívateľskú funkciu (pozri kapitola 8), ktorá ako vstup dostane zoznam troch bodov reprezentujúcich vrcholy trojuholníka; a ako výstup vráti zoznam uhlov tohto trojuholníka.

Skalárny súčin sa využíva aj pri pravouhlom premietaní vektorov (označovanom aj pojmom "ortogonálne premietanie"). Vektor **u** sa dá premietať na vektor **v** pomocou skalárneho súčinu $u \cdot \frac{\mathbf{v}}{||\mathbf{v}||}$. Výsledok je dĺžka priemetu na vektor **v**.

In[22]:=
```
vN = Normalize[v];
proj = u.vN
```

Out[22]= $-\frac{25}{\sqrt{17}}$

Samotný premietnutý vektor vieme vypočítať ako súčin projekcie a normovaného vektoru: $u \cdot \frac{v}{||v||} \; \frac{v}{||v||}$.

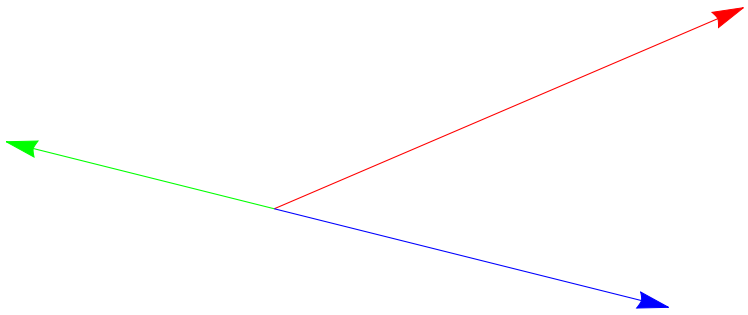
In[23]:=
```
uProj = proj vN
```

Out[23]= $\{\frac{100}{17}, -\frac{25}{17}\}$

Vektory **u** a **v** spolu s výslednou projekciou znázorníme nasledujúcim príkazom.

In[24]:=
```
Graphics[{
 Red, Arrow[{{0, 0}, u}],
 Green, Arrow[{{0, 0}, v}],
 Blue, Arrow[{{0, 0}, uProj}]
}]
```

In[25]:=



Červený vektor **u** bol projektovaný na zelený vektor **v** a výsledná projekcia je modrý vektor. Keby sme spojili koncové body vektora **u** a výslednej projekcie, dostaneme vektor kolmý na vektor **v**.

Cvičenie 12.9

V softvéri Mathematica je zabudovaná funkcia `Projection`, ktorá vracia vo výstupe dĺžku priemetu prvého vstupného vektora na druhý vstupný vektor. Overte, či táto funkcia dá rovnaký výsledok ako ten, ktorý bol vo výstupe premennej `proj`.

Posledná vstavaná funkcia, ktorú si predstavíme v tejto sekcii, je `Cross`. Jej výstupom je vektorový súčin (po angl. cross product) dvoch vstupných trojrozmerných vektorov. Alternatívne sa dá zadať pomocou znaku × zadaného klávesovou skratkou **[Esc]cross[Esc]** alebo z palety (pozri Pozn. 3.2). Upozorňujeme, že napriek tomu, že tento znak pripomína písmeno `x` zadané z klávesnice, nemôže ním byť nahradený.

```
In[26]:= u = {1.5, 0.1, -0.4}; v = {0.2, -0.9, -0.1};
         w = Cross[u, v]
         w = v×u

Out[26]= {-0.37, 0.07, -1.37}
         {0.37, -0.07, 1.37}
```

Všimnite si na predchádzajúcom príklade, že vektorový súčin je antikomutatívny, t. j. platí vzťah $u \times v = -v \times u$. Výsledkom vektorového súčinu dvoch trojrozmerných nenulových a lineárne nezávislých vektorov **u** a **v** je vektor, ktorý je na tieto dva vektory kolmý.

```
In[27]:= u.w
         v.w

Out[27]= 0.
         0.
```

Jeho dĺžka sa rovná obsahu rovnobežníka, ktorého dve susedné strany sú reprezentované dvoma vstupnými vektormi vektorového súčinu.

Cvičenie 12.10

Zadefinujte užívateľskú funkciu (pozri kapitolu 8), ktorá ako vstup dostane zoznam troch bodov reprezentujúcich vrcholy trojuholníka; a ako výstup vráti obsah tohto trojuholníka.

Pomôcka: Posuňte celý rovnobežník do začiatku súradnicovej sústavy a zvoľte dva správne vektory, na ktoré aplikujete vektorový súčin.

Existuje zovšeobecnenie vektorového súčinu na n-rozmerný priestor. n-rozmerný vektorový súčin dostane ako vstup $n-1$ n-rozmerných vektorov. Výsledok tohto vektorového súčinu môžeme intuitívne chápať ako vektor, ktorý je kolmý na všetky vstupné vektory. Pre dvojrozmerný vektor dostaneme kolmý vektor vymenením súradníc a pridaním znamienka $-$ pred prvú z vymenených súradníc.

```
In[28]:= Cross[x, y]

Out[28]= {-y, x}
```

Cvičenie 12.11

Vypočítajte vektorový súčin troch nenulových vektorov štvorrozmerného priestoru. Zistite, či je výsledný vektor kolmý na tieto tri vektory.

Pomôcka: Pre nenulové vektory štvorrozmerného priestoru taktiež platí, že sú na seba kolmé práve vtedy, keď ich skalárny súčin je nulový.

12.2 Matice

Matica $n \times m$ je v softvéri Mathematica reprezentovaná ako n-rozmerný zoznam m-rozmerných zoznamov čísel. Inak povedané, zoznam obsahujúci n m-rozmerných vektorov.

```
In[29]:= M = {{11, 12, 13}, {21, 22, 23}, {31, 32, 33}};
```

Pomocou vstavanej funkcie **MatrixForm** vieme zobraziť maticu v tzv. maticovom tvare. Táto funkcia (rovnako ako funkcia N) sa dá použiť dvoma spôsobmi.

```
In[30]:= M // MatrixForm;
         MatrixForm[M]
```

Out[30]//MatrixForm=

$$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}$$

i-tý riadok matice vieme získať pomocou príkazu M[[i]].

```
In[31]:= M[[1]]

Out[31]= {11, 12, 13}
```

Cvičenie 12.12

Vytvorte maticu, ktorá má v prvku na pozícii i, j skalárny súčin i-tého riadku matice M s jej j-tým riadkom.

`j`-tý stĺpec matice vieme získať pomocou príkazu `M[[;;,j]]` (pozri sekciu 4.1).

```
In[32]:= M[[;;,2]]

Out[32]= {12, 22, 32}
```

Cvičenie 12.13

Vytvorte maticu, ktorá má v prvku na pozícii i, j skalárny súčin i-tého stĺpca matice M s jej j-tým stĺpcom.

V softvéri Mathematica sú zabudované základné matematické operácie s maticami. Aritmetické operácie rovnako ako pri vektoroch fungujú aj pri maticiach po zložkách (t. j. po prvkoch). Podobne ako pri vektoroch, znamienko `*` reprezentuje násobenie matíc po zložkách, zatiaľ čo znamienko `.` maticový súčin matíc.

Funkcia **`Transpose`** transponuje maticu (vymení riadky za stĺpce).

```
In[33]:= Transpose[M] // MatrixForm
```

Out[33]//MatrixForm=

$$\begin{pmatrix} 11 & 21 & 31 \\ 12 & 22 & 32 \\ 13 & 23 & 33 \end{pmatrix}$$

Funkcia **`Inverse`** vracia vo výstupe inverznú maticu ku vstupnej (zadanej) matici.

```
In[34]:= M = {{-2, 1, 0}, {1, -2, 1}, {0, 1, -2}};
         Inverse[M] // MatrixForm
```

Out[34]//MatrixForm=

$$\begin{pmatrix} -\frac{3}{4} & -\frac{1}{2} & -\frac{1}{4} \\ -\frac{1}{2} & -1 & -\frac{1}{2} \\ -\frac{1}{4} & -\frac{1}{2} & -\frac{3}{4} \end{pmatrix}$$

Cvičenie 12.14

Matica `{{11, 12, 13}, {21, 22, 23}, {31, 32, 33}}` je singulárna, t. j. neexistuje k nej inverzná matica. Vyskúšajte, čo sa stane, ak na ňu aplikujeme funkciu **`Inverse`**.

Funkcia **Det** vracia vo výstupe determinant vstupnej (zadanej) štvorcovej matice.

```
In[35]:= Det[M]

Out[35]= -4
```

Cvičenie 12.15

Matica **{{4, 2}, {-2, -1}}** má nulový deteminant. Zobrazte riadky matice ako vektory pomocou funkcie **Graphics**. Ako sa nulový determinant prejaví na tomto obrázku?

Funkcia **Dot** (zadaná aj pomocou bodky) vracia ako výstup výsledok maticového súčinu dvoch matíc, ale len takých, ktoré spĺňajú podmienku, že počet stĺpcov prvej matice sa rovná počtu riadkov druhej matice.

```
In[36]:= H = {{2, 3, 5}, {7, -2, 8}};
         Dot[H, M] // MatrixForm;
         H.M // MatrixForm
```

Out[36]//MatrixForm=

$$\begin{pmatrix} -1 & 1 & -7 \\ -16 & 19 & -18 \end{pmatrix}$$

Ak by sme sa pokúsili vynásobiť matice, ktoré toto pravidlo nespĺňajú, zobrazilo by sa chybové hlásenie upozorňujúce na nekompabilitu matíc.

```
In[37]:= H.H

         Dot: "Tensors {{2,3,5},{7,−2,8}} and {{2,3,5},{7,−2,8}} have incompatible
         shapes"

Out[37]= {{2, 3, 5}, {7, -2, 8}} . {{2, 3, 5}, {7, -2, 8}}
```

Cvičenie 12.16

Len pomocou násobenia matice maticou a transponovania matice; vytvorte maticu, ktorá má v prvku na pozícii i, j skalárny súčin: a) i-tého riadku matice M s jej j-tým riadkom,
b) i-tého stĺpca matice M s jej j-tým stĺpcom.

Špeciálny prípad násobenia matice maticou je násobenie matice vektorom. Ak by sme chceli byť matematicky korektní, napísali by sme, že maticu rozmeru $m \times n$ násobíme stĺpcovým vektorom (t. j. maticou rozmeru $n \times 1$) a výsledok tohto súčinu bude stĺpcový vektor (t. j. matica rozmeru $m \times 1$).

```
In[38]:= vStlpec = {{1}, {0}, {4}};
         M.vStlpec

Out[38]= {{-2}, {5}, {-8}}
```

Keďže v softvéri Mathematica zapisujeme zvyčajne vektory ako zoznamy čísel, tak je v tomto softvéri zabudovaný aj spôsob násobenia matice s vektorom v takomto tvare. Výsledkom takéhoto násobenia je vektor v tvare zoznamu.

```
In[39]:= v = {1, 0, 4};
         M.v

Out[39]= {-2, 5, -8}
```

Matice sa dajú zadávať aj pomocou vstavaných funkcií, ktorých výstupmi sú zoznamy (napr. **Table** (pozri str. 52)). V tejto sekcii si predstavíme niektoré ďalšie funkcie na vytváranie matíc.

Funkcia **IdentityMatrix[n]** vytvorí jednotkovú maticu (po angl. identity matrix) rozmeru **n** × **n**, ktorá má na diagonále jednotky a na ostatných pozíciách nuly.

```
In[40]:= IdentityMatrix[4] // MatrixForm
```

Out[40]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cvičenie 12.17

Pre inverznú maticu M^{-1} platí rovnosť: $M \cdot M^{-1} = M^{-1} \cdot M = I$, kde I je jednotková matica. Overte jej platnosť v softvéri Mathematica. Použite ľubovoľnú (regulárnu) štvorcovú maticu.

Poznámka 12.3

V angličtine sa pre jednotkovú maticu používa pojem "identity matrix", ktorý by sa do slovenčiny dal preložiť ako "matica identity". Tento výraz vychádza z pohľadu na maticu ako na operátor (zovšeobecnenie funkcie). Operátor dostane zadaný vstup (v tomto prípade vektor) a vráti výstup (v tomto prípade iný vektor). Identita je operátor, ktorý vracia ako výstup zadaný vstup. Ukážeme si to na nasledujúcom príklade.

```
In[41]:= IdentityMatrix[4].{x, y, z, w}
```

```
Out[41]= {x, y, z, w}
```

Funkcia **DiagonalMatrix** dostáva ako vstup zoznam prvkov a vytvorí maticu, ktorá má na diagonále prvky zadaného zoznamu a na ostatných pozíciách nuly.

```
In[42]:= DiagonalMatrix[{1, -5, 8, 12}] // MatrixForm
```

Out[42]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -5 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 12 \end{pmatrix}$$

Cvičenie 12.18

Vytvorte maticu, ktorá má na diagonále celé čísla od 1 po 20 a na ostatných pozíciách nuly.

Druhý (voliteľný) parameter funkcie **DiagonalMatrix** určuje, na ktorej diagonále bude zadaný zoznam (z prvého vstupného parametru). Preddefinovaná hodnota 0 reprezentuje hlavnú diagonálu, 1 diagonálu posunutú o jednotku vpravo, -1 diagonálu posunutú o jednotku vľavo a n (kladné celé číslo) diagonálu posunutú o n vpravo. Vo všeobecnosti môžeme zvoliť za tento parameter ľubovoľné celé číslo.

```
In[43]:= DiagonalMatrix[{7, 9, -4}, 2] // MatrixForm
```

Out[43]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Cvičenie 12.19

Vytvorte maticu, ktorá má na hlavnej diagonále všetky prvky s hodnotou -2 a na vedľajších diagonálach (diagonály posunuté o jednotku vľavo a vpravo od hlavnej diagonály) všetky prvky s hodnotou 1.

Pomôcka: Využite sčítanie matíc.

Na výpočet vlastných vektorov a vlastných čísel matice slúžia funkcie **Eigenvectors** a **Eigenvalues**.

```
In[44]:= Eigenvectors[M]
```

```
Out[44]= {{1, -√2, 1}, {-1, 0, 1}, {1, √2, 1}}
```

```
In[45]:= Eigenvalues[M]

Out[45]= {-2 - √2, -2, -2 + √2}
```

Výstupom **Eigenvectors** je zoznam vlastných vektorov matice. Výstupom **Eigenvalues** je zoznam vlastných čísel matice, pričom i-té vlastné číslo prislúcha i-tému vlastnému vektoru.

Poznámka 12.4

Pre vlastné číslo λ matice M prislúchajúce k vlastnému vektoru v platí vzťah $M \cdot v = \lambda v$. Ten môžeme prepísať do nasledujúceho tvaru $(M - \lambda I)v = 0$, kde I je jednotková matica a 0 je nulový vektor.

Cvičenie 12.20

Overte platnosť tvrdenie z Poznámky 12.4 pre všetky prislúchajúce dvojice vlastných vektorov a vlastných čísel ľubovoľnej štvorcovej matice.

12.3 Riešenie lineárnej sústavy rovníc

Veľké množstvo fyzikálnych modelov sa dá vyjadriť pomocou parciálnych diferenciálnych rovníc. A väčšinu diferenciálnych rovníc (okrem tých najjednoduchších) nevieme riešiť analytickými metódami. Z tohoto dôvodu využívame rôzne aproximačné metódy; a tie vedú k riešeniu systému lineárnych rovníc. Riešenie sústav lineárnych rovníc zastáva v matematike, ale aj v softvéri Mathematica špeciálne postavenie.

Funkcia **LinearSolve[M,b]** rieši systém lineárnych rovníc $M \cdot \mathbf{x} = \mathbf{b}$, kde M je matica, $\mathbf{b}$ je vektor pravej strany a $\mathbf{x}$ je vektor neznámych

```
In[46]:= M = {{-2, 1, 0}, {1, -2, 1}, {0, 1, -2}};
         b = {1, 0, 7};
         x = LinearSolve[M, b]

Out[46]= {-5/2, -4, -11/2}
```

Cvičenie 12.21

Urobte skúšku správnosti pre predchádzajúci príklad.

Pomôcka: Overte, či platí rovnosť $M \cdot \mathbf{x} = \mathbf{b}$.

Tento systém rovníc môžeme vyriešiť aj nasledujúcim spôsobom pomocou funkcie **Solve** (pozri sekciu 11.1).

```
In[47]:= Clear[x]
         X = {x[1], x[2], x[3]};
         rovnice = (M.X == b)
```

$$\text{Out[47]= } \{-2\,x[1] + x[2],\ x[1] - 2\,x[2] + x[3],\ x[2] - 2\,x[3]\} == \{1,\ 0,\ 7\}$$

```
In[48]:= Solve[rovnice, X]
```

$$\text{Out[48]= } \left\{\left\{x[1] \to -\frac{5}{2},\ x[2] \to -4,\ x[3] \to -\frac{11}{2}\right\}\right\}$$

V tomto prípade sme využili fakt, že ľavá a pravá strana rovnice môžu byť zadané ako vektory (zoznamy).

Pozrime sa na špeciálne prípady.

```
In[49]:= B = {{1, 0, 3}, {2, 1, 0}, {6, 2, 6}};
         b = {1, 0 , 7};
         x = LinearSolve[B, b]

         LinearSolve::nosol: Linear equation encountered that has no solution. >>

Out[49]= LinearSolve[{{1, 0, 3}, {2, 1, 0}, {6, 2, 6}}, {1, 0, 7}]
```

Predchádzajúci príklad nemá žiadne riešenie. Softvér Mathematica nás na to upozornil chybovým hlásením a vrátil ako výstup príkaz, ktorý sme mu zadali.

Ak zmeníme pravú stranu rovnosti, dostaneme systém rovníc, ktorý má nekonečne veľa riešení.

```
In[50]:= b = {1, 0 , 2};
         x = LinearSolve[B, b]
```

$$\text{Out[50]= } \left\{-\frac{5}{2},\ -4,\ -\frac{11}{2}\right\}$$

Funkcia **LinearSolve** vráti vždy len jedno z nich. Matica **B** nemá všetky riadky lineárne nezávislé. Tretí riadok je dvojnásobok súčtu prvého a druhého riadku. To znamená, že tretí riadok nám nedáva žiadnu novú informáciu; a ak pravá strana nie je tiež dvojnásobok súčtu prvého a druhého riadku, tak máme dve protichodné tvrdenia, t. j. systém rovníc nemá riešenie. Avšak, ak je pravá strana zladená, tak tretí riadok reprezentuje to isté, čo prvé dva, a v takom prípade má sústava rovníc nekonečne veľa riešení.

Existuje niekoľko spôsobov, ako v softvéri Mathematica zistiť, či je matica regulárna, alebo singulárna. Funkcia **MatrixRank** vracia hodnosť matice, t. j. počet lineárne nezávislých riadkov.

```
In[51]:= MatrixRank[B]

Out[51]= 2
```

Matica `B` má dva lineárne nezávislé riadky z troch, t. j. je singulárna.

V prípade, že je zadaná matica štvorcová, môžeme jej singularitu vyšetriť aj pomocou determinantu. Ak je determinant nulový, matica je singulárna a systém rovníc (v závislosti od pravej strany) nemá riešenie, alebo má nekonečne veľa riešení.

```
In[52]:= Det[B]

Out[52]= 0
```

Ak sme zistili, že zadaná matica je singulárna a funkcia **`LinearSolve`** našla riešenie systému rovníc, vieme, že tento systém má nekonečne veľa riešení. Ďalšie riešenia (t. j. všeobecné riešenie systému rovníc) nájdeme pomocou vstavanej funkcie **`NullSpace`**, ktorá vracia nulový priestor (jadro) matice, t. j. zoznam vektorov $\mathbf{j}$, pre ktoré platí $B \cdot \mathbf{j} = \mathbf{0}$.

```
In[53]:= j = NullSpace[B]

Out[53]= {{-3, 6, 1}}
```

V tomto prípade je to len jeden vektor. Z geometrického hľadiska to znamená, že vektory z jadra sú kolmé na všetky vektory reprezentujúce riadky matice. Naviac, ak máme jedno riešenie $\mathbf{x}$ (nájdené napr. pomocou funkcie **`LinearSolve`**), tak aj $\mathbf{x} + t * \mathbf{j},\ t \in \mathbf{R}$ je tiež riešenie. Toto je zrejmé, keďže platí rovnosť $B \cdot (\mathbf{x} + t * \mathbf{j}) = \mathbf{b}$.

Cvičenie 12.22

Overte platnosť tvrdenia $B \cdot \mathbf{j} = \mathbf{0}$.

Všeobecné riešenie môžeme zapísať nasledovne.

```
In[54]:= vseobecneX = x + Sum[t j[[i]], {i, 1, Length[j]}]

Out[54]= {-5/2 - 3t, -4 + 6t, -11/2 + t}
```

Pomocou determinantu vieme určiť len riešiteľnosť sústav rovníc daných štvorcovou maticou, avšak **`MatrixRank`** umožňuje určiť aj riešiteľnosť sústav daných obdĺžnikovou maticou. Takéto sústavy majú jediné riešenie, ak sa hodnosť obdĺžnikovej matice rovná počtu neznámych sústavy. T. j. ak sústava má n rovníc, ale len m neznámych a hodnosť jej matice je tiež m (nachádza sa v nej $n - m$ lineárne závislých riadkov). Ukážeme si to na nasledujúcom príklade.

Riešme systém 4 lineárnych rovníc o 2 neznámych daný: $F \cdot \mathbf{x} = \mathbf{f}$, kde F je matica, $\mathbf{f}$ je vektor pravej strany a $\mathbf{x}$ je vektor neznámych. Prvý riadok matice je dvojnásobkom druhého a štvrtý riadok je súčtom druhého a tretieho.

```
In[55]:= F = {{4, 6}, {2, 3}, {4, 5}, {6, 8}};
```

Hodnosť takejto matice je 2 a jej nulový priestor je prázdny.

```
In[56]:= MatrixRank[F]
         NullSpace[F]

Out[56]= 2
         { }
```

Keďže nulový priestor matice je prázdny, znamená to, že riešenie systému rovníc nájdené pomocou **LinearSolve** je jediné.

```
In[57]:= f = {82, 41, 73, 114};
         LinearSolve[F, f]

Out[57]= {7, 9}
```

12.3.1 Zle podmienené matice

Doteraz sme ako prvky matice zadávali len celé (t. j. presné) čísla. Vo väčšine aplikácií sa však používajú reálne (t. j. približné) čísla (pozri podkapitolu 3.3). Vyriešme systém rovníc, ktorý v predchádzajúcej podsekcii nemal riešenie tak, že presné čísla matice zmeníme na približné pridaním desatinnej bodky.

```
In[58]:= M = {{1., 0., 3.}, {2., 1., 0.}, {6., 2., 6.}};
         b = {1., 0., 7.};
         x = LinearSolve[M, b]

         LinearSolve::luc:
         Result for LinearSolve of badly conditioned matrix
         {{1.,0.,3.},{2.,1.,0.},{6.,2.,6.}} may contain significant numerical errors. ≫

Out[58]= {1.68885×10^16, -3.3777×10^16, -5.6295×10^15}
```

Teraz sme síce dostali výsledok, ale aj s chybovým hlásením. To tvrdí, že matica je zle podmienená a výsledok môže mať v sebe veľkú numerickú nepresnosť. Môže byť prekvapujúce, že napriek tomu, že riadky matice nie sú lineárne nezávislé, funkcia **LinearSolve** našla nejaké riešenie. Je to spôsobené tým, že prvky zadanej matici sú kvôli pridanej desatinnej bodke považované softvérom za približné čísla (niekde na ďalekom desatinnom mieste sa môže nachádzať nenulová cifra) a v takomto prípade sa o riadkoch matice nedá povedať, že sú lineárne závislé. Avšak podľa chybového hlásenia od toho nemajú ďaleko.

Názorne si to ukážeme na nasledujúcom príklade. K nejakému prvku tretieho riadku (ktorý je lineárnou kombináciou prvých dvoch riadkov) pripočítame veľmi malé číslo. Takto zabezpečíme, že riadky matice budú lineárne nezávislé.

```
In[59]:= M = {{1., 0., 3.}, {2., 1., 0.}, {6., 2., 6.00000001}};
         b = {1., 0., 7.};
         x = LinearSolve[M, b]

         LinearSolve::luc:
         Result for LinearSolve of badly conditioned matrix
         {{1.,0.,3.},{2.,1.,0.},{6.,2.,6.}} may contain significant numerical errors. >>

Out[59]= {1.5×10^9, 3.×10^9, 5.×10^8}
```

V praxi získavame hodnoty dát najčastejšie meraním alebo výpočtom. Tieto hodnoty sú len približné vzhľadom na chybu merania a zaokrúhľovaciu chybu. Teraz si vyskúšame, čo by sa stalo, keby sme hodnotu $6+1.0\times10^{-8}$ z predchádzajúceho príkladu nahradili hodnotou $6+1.5\times10^{-8}$.

```
In[60]:= M = {{1., 0., 3.}, {2., 1., 0.}, {6., 2., 6.000000015}};
         b = {1., 0., 7.};
         x = LinearSolve[M, b]

         LinearSolve::luc:
         Result for LinearSolve of badly conditioned matrix
         {{1.,0.,3.},{2.,1.,0.},{6.,2.,6.}} may contain significant numerical errors. >>

Out[60]= {-1.×10^9, 2.×10^9, 3.33333×10^8}
```

Napriek tomu, že v systéme rovníc sa zmenila iba jediná hodnota a aj to len minimálne (zväčšenie o 5×10^{-9}), riešenie systému je úplne iné. Softvér Mathematica nás na to upozorňuje chybovým hlásením.

Ak sú však všetky prvky matice presné čísla, tak sa chybové hlásenie nezobrazí.

```
In[61]:= M = {{1, 0, 3}, {2, 1, 0}, {6, 2, 6 + 1/100 000 000}};
         b = {1, 0, 7};
         x = LinearSolve[M, b]

Out[61]= {-1 499 999 999, 2 999 999 998, 500 000 000}
```

Poznámka 12.5

Chybové hlásenie o zle podmienenej matici sa zobrazí, ak je aspoň jeden prvok zle podmienenej matice alebo aspoň jeden prvok pravej strany systému približné číslo.

12.4 Riedke matice

V mnohých praktických aplikáciách sa počítajú systémy rovníc s maticami obrovských rozmerov. V prípade, že ide o riedke matice (t. j. matice, ktoré majú väčšinu prvkov nulových), sa používa úsporný spôsob ukladania matice, ktorý šetrí počítačovú pamäť. V softvéri Mathematica sa na efektívne ukladanie riedkych matíc používa funkcia **SparseArray** (z angl. riedke pole). Jej vstupom je zoznam nenulových prvkov, ktoré sa zadávajú pomocou pravidla **{i, j}→hodnota**, kde **i** je číslo riadku, **j** je číslo stĺpca a **hodnota** je hodnota prvku v **i**-tom riadku a **j**-tom stĺpci.

```
In[62]:= M = SparseArray[{{1, 1} → -2, {1, 2} → 1, {2, 1} → 1, {2, 2} → -2,
                          {2, 3} → 1, {3, 2} → 1, {3, 3} → -2}]
```

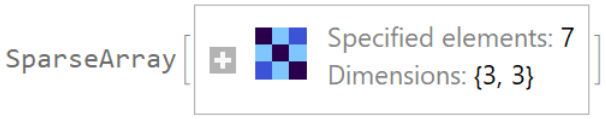
Out[62]=

Výstupom je grafické znázornenie matice a informácie o rozmeroch a nenulových prvkoch.

Väčšina funkcií a operácií, ktoré sme si v tejto kapitole popísali, funguje aj pri práci s riedkymi maticami.

```
In[63]:= 3 M + M
```

Out[63]=




```
In[64]:= Eigenvalues[M]
```

Out[64]= $\{-2 - \sqrt{2}, -2, -2 + \sqrt{2}\}$

```
In[65]:= Eigenvectors[M]
```

Out[65]= $\{\{1, -\sqrt{2}, 1\}, \{-1, 0, 1\}, \{1, \sqrt{2}, 1\}\}$

```
In[66]:= M.{1, 0, 4}
```

Out[66]= {-2, 5, -8}

```
In[67]:= Inverse[M]
```

Out[67]= $\left\{\left\{-\frac{3}{4}, -\frac{1}{2}, -\frac{1}{4}\right\}, \left\{-\frac{1}{2}, -1, -\frac{1}{2}\right\}, \left\{-\frac{1}{4}, -\frac{1}{2}, -\frac{3}{4}\right\}\right\}$

```
In[68]:= Transpose[M]
```

Out[68]=

SparseArray [Specified elements: 7
Dimensions: {3, 3}]

In[69]:=
```
b = {1, 0, 7};
LinearSolve[M, b]
```

Out[69]= $\{-\frac{5}{2}, -4, -\frac{11}{2}\}$

Funkcia **Inverse** aj v prípade, keď dostane ako vstup riedku maticu zadanú pomocou funkcie **SparseArray**, vracia ako výstup plnú maticu, a to aj v situáciách, keď výsledná matica obsahuje nulové prvky.

Riedke matice zadané pomocou funkcie **SparseArray** vieme zobraziť ako plné matice pomocou funkcie **MatrixForm**.

In[70]:=
```
MatrixForm[M]
```

Out[70]//MatrixForm=

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

Riedke matice sa dajú transformovať na plné matice pomocou funkcie **Normal**.

In[71]:=
```
Normal[M]
```

Out[71]=
```
{{-2, 1, 0}, {1, -2, 1}, {0, 1, -2}}
```

12.5 Zhrnutie

V tejto kapitole sme si ukázali, že vektory sa v softvéri Mathematica zadávajú ako zoznamy čísel a matice ako zoznamy zoznamov čísel, pričom všetky vnútorné zoznamy musia mať rovnakú dĺžku (t. j. rovnaký počet prvkov). Predstavili sme základné operácie s vektormi, skalárny súčin pomocou bodky alebo funkcie **Dot**, vektorový súčin **Cross** a rôzne normy pomocou funkcie **Norm**.

Ukázali sme, ako pristupovať k jednotlivým riadkom a stĺpcom matice, základné operácie ako transponovanie matice **Transpose**, výpočet inverznej matice **Inverse** a determinantu matice **Det**, násobenie matíc s maticami a násobenie matíc s vektormi, vytváranie matíc pomocou **IdentityMatrix** a **DiagonalMatrix** a výpočet vlastných čísel **Eigenvalues** a vlastných vektorov **Eigenvectors**.

Ukázali sme tiež, ako sa dajú pomocou funkcie **LinearSolve** riešiť lineárne systémy rovníc definované maticou; a ako sa prejaví, keď systém lineárnych rovníc nemá riešenie, má nekonečne veľa riešení, alebo keď je jeho matica zle podmienená. Na záver sme stručne predstavili používanie riedkych matíc v softvéri Mathematica.

Literatúra

[1] Leonid Shifrin, *Mathematica programming: an advanced introduction*, Version 1.01, ebook, pp. 408, 2008.

[2] Magda Komorniková, Karol Mikula, *Výpočtový systém Mathematica*, Fakulta elektrotechniky a informatiky, ISBN: 80-227-1123-3, Bratislava, 1998.

[3] Stephen Wolfram, *An Elementary Introduction to the Wolfram Language, First Edition*, Wolfram Media, Inc., ISBN: 1944183000 (Paperback), pp. 324, 2015.

Mgr. Oľga Stašová, PhD., Ing. Matej Medľa, PhD.

SOFTVÉR MATHEMATICA: PRVÝ DIEL

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve SPEKTRUM STU, Bratislava, Vazovova 5, v roku 2021.

Edícia vysokoškolských učebníc

Rozsah 206 strán, 150 obrázkov, 10,150 AH, 10,500 VH, 1. vydanie, edičné číslo 6095, vydané v elektronickej forme

85 – 229 – 2021

ISBN 978-80-227-5138-4