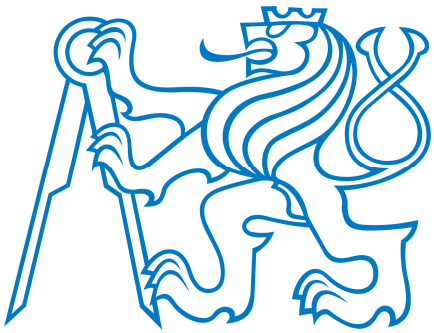
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav přístrojové a řídicí techniky



BAKALÁŘSKÁ PRÁCE

PYTHON MODUL PRO GENEROVÁNÍ TECHNICKÝCH DOKUMENTŮ V LATEX FORMÁTU

PYTHON MODULE FOR GENERATING TECHNICAL DOCUMENTS IN LATEX FORMAT

Richard Kokštein

2023

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím literárních zdrojů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů.

Datum: . . . .

Podpis

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kokštein** Jméno: **Richard** Osobní číslo: **501412**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Python modul pro generování technických dokumentů v LaTeX formátu

Název bakalářské práce anglicky:

Python module for generating technical documents in LaTeX format

Pokyny pro vypracování:

Navrhněte a vytvořte prototyp Python modulu pro dynamické generování LaTeX dokumentů zaměřených na technické a vědecké výpočty. Cílem je co nejpřímější integrace generování obsahu dokumentu přímo ve zdrojovém Python kódu generujícím data pro cílový dokument. Pro řešení využijte v plné šíři již existující moduly, zejména moduly pro formátování LaTeX dokumentů.

- Proveďte rešerši aktuálních možností generování LaTeX dokumentů v jazyce Python.
- Seznamte se se základními principy a typy prezentací dat v technických a vědeckých pracích.
- Navrhněte architekturu modulu.
- Vyberte několik typů prezentací dat a ty implementujte ve formě modulu založeného na navržené architektuře.

Seznam doporučené literatury:

1. S. Few, Show Me the Numbers: Designing Tables and Graphs to Enlighten. California, Berkeley: Analytics Press, 2004.
2. P. Olšák, První setkání s TeXem. 2022. [Online]. Dostupné z: https://petr.olsak.net/tat/aprvni.pdf
3. B. Esslinger, PythonTeX and LATEX - Familiarize us with PythonTeX in order to build the CrypTool Book: Learning via Samples and Counter Examples. 2023. [Online]. Dostupné z: https://www.cryptool.org/download/ctb/PythonTex-by-Examples.pdf
4. J. Fennema, PyLaTeX 1.3.2 documentation, PyLaTeX, c2015. https://jeltef.github.io/PyLaTeX

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Cyril Oswald, Ph.D. U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **28.04.2023** Termín odevzdání bakalářské práce: **08.06.2023**

Platnost zadání bakalářské práce: _____

Ing. Cyril Oswald, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Anotace

Bakalářská práce se zaměřuje na metody generování technických dokumentů obsahujících data v podobě tabulek a grafů. V práci jsou popsané způsoby prezentace dat a metody jejich generace pomocí programovacího jazyka Python a typografického systému TEX, konkrétně jeho nadstavby LATEX. Výsledkem práce je prototyp Python modulu, který zjednodušuje prezentaci dat a generaci technických dokumentů.

Klíčová slova: Python, LaTeX, generace, tabulka, graf

Annotation

This thesis focuses on methods of generating technical documents containing data in the form of tables and charts. The thesis describes the methods of data presentation and its generation using the Python programming language and the TeX typographic system, specifically the LaTeX format. The result of my work is a prototype of a Python module that simplifies data presentation and generation of technical documents.

Keywords: Python, LaTeX, generation, table, graph, chart, plot

Obsah

1 Úvod

Tabulky a grafy jsou důležitou součástí každého technického dokumentu. Můžeme je vytvářet pomocí různých nástrojů. Většina programů, ve kterých pracujeme s daty, nabízí alespoň základní funkce pro tvorbu zprávy obsahující jednoduché formy prezentace dat. Pokud data získáváme nebo zpracováváme námi napsaným skriptem v libovolném programovacím jazyce, naskytuje se nám možnost rozšířit skript i na generaci výsledné zprávy, ve které budeme mít pod kontrolou každý její aspekt.

Pro zpracovávání dat se ve velké míře využívá programovací jazyk Python. Pro tvorbu technických zpráv a dokumentů se s oblibou, především v akademické sféře, používá typografický systém TEX. Pro obě technologie existují rozšíření, které dovolují tvorbu různých forem prezentace dat. Tato práce se zaměřuje na spojení zmíněných technologií pro snadnou a rychlou generaci tabulek a grafů pro technické dokumenty.

První část práce bude zaměřena na seznámení se se způsoby prezentace technických dat a rešerši již existujících řešení na propojení Pythonu a TEXu. V druhé praktické části bude vytvořen prototyp Python modulu, který se pokusí zjednodušit stávající řešení pro generaci tabulek a grafů.

2 Prezentace dat

Data v technických dokumentech jsou obvykle ve formě čísel, která chceme vhodným způsobem prezentovat. Jako autoři bychom měli volit vhodnou formu prezentace, abychom čtenáři ulehčili pochopení informací vycházejících z dat. Všechna data v dokumentu by měla být v textu představena. Autor zde má možnost vysvětlit, proč jsou data v dokumentu prezentována, za jakých podmínek vznikla a na jaké hodnoty by se měl čtenář zaměřit. Následující výpis uvádí formy prezentace dat společně s případy jejich použití. [1]

- Text – popis jedné až dvou číselných hodnot.
- Tabulka
  - ⋆ prezentace přesných hodnot,
  - ⋆ porovnávání a vyhledávání konkrétních hodnot a jejich zvýrazňování,
  - ⋆ prezentace číselných hodnot s větším počtem fyzikálních jednotek.
- Graf
  - ⋆ prezentace vývoje a tvaru dat,
  - ⋆ pozorování vzorů v datech,
  - ⋆ prezentace dlouhých sérií dat. [1]

Formátování textu, tabulek a grafů by mělo být v celém dokumentu konzistentní. Grafické prvky tabulek a grafů by měly být v dokumentu uloženy ve vektorové formě. Měli bychom se vyhnout používání rastrové grafiky nízké kvality. V následujících dvou podkapitolách je popsána struktura, jednotlivé konstrukční prvky a hlavní typy tabulek a grafů.

2.1 Tabulky

Tabulky slouží pro zobrazení hodnot zarovnaných do řádků a sloupců. Hodnoty jsou čtenáři prezentovány v textové podobě – pomocí písmen a číslic. Jejich použití je vhodné v případě, kdy chceme dát čtenáři možnost zkoumat jednotlivé hodnoty v tabulce, porovnávat hodnoty mezi sebou nebo čtenáři prezentovat souhrnné informace, například součet nebo aritmetický průměr hodnot v konkrétním sloupci. Tabulky umožňují prezentovat přesná data složená z různých fyzikálních veličin s rozdílnými jednotkami. Hlavní prvky tabulek jsou:

- nadpis, podnadpis, popisek,
- záhlaví sloupců a řádků,
- okraj, svislé a vodorovné ohraničení,
- tělo tabulky, jednotlivé buňky,
- zápatí, souhrnné informace. [1, 2]

Pro zvýraznění konkrétních hodnot v tabulce lze změnit barvu textu v buňce, barvu jejího pozadí nebo barvu a tloušťku ohraničení. Hodnoty v buňkách lze slučovat do skupin pomocí mezer mezi buňkami, stejné barvy pozadí nebo stylem ohraničení.

Kombinace svislého a vodorovného ohraničení tvoří mřížku, která zvýrazňuje hranice mezi jednotlivými sloupci a řádky. Příliš výrazná mřížka může data fragmentovat a zhoršit naši orientaci při prohlížení hodnot. Styl ohraničení se proto volí méně výrazný. Mřížka může být nahrazena nebo doplněna většími mezerami mezi hodnotami nebo barevným pruhováním řádků nebo sloupců. [1–3]

Při prezentování číselných hodnot bychom měli volit vhodnou přesnost a tu použít pro všechny hodnoty stejného původu, které poté zarovnáme podle desetinné čárky. Pro dlouhá čísla je vhodné použít oddělovač tisíců.

2.2 Grafy

Grafy prezentují data v grafické podobě. Pozornost čtenáře se přesouvá od vnímání jednotlivých hodnot k pozorování tvaru a průběhu dat. Jejich použití je vhodné v případě, kdy chceme čtenáři prezentovat průběh, trend, vývoj a tvar dat. Oproti tabulkám je v grafech jednodušší porovnávat velké série dat, pozorovat vzory, odhalit oblasti růstu, poklesu a body globálních a lokálních extrémů. [1]

Velikosti číselných hodnot nebo kategorické informace jsou v grafech uloženy pomocí různých objektů a jejich atributů. K převodu z jedné formy reprezentace hodnot do další slouží osy, měřítka a legendy, pomocí kterých dokáže čtenář odečítat hodnoty uložené v objektech a jejich atributech. Využívané objekty ve grafech jsou:

- body,
- čáry, křivky,
- svislé a vodorovné sloupce,
- obdélníky, krabice,
- obecné tvary. [1]

Jejich atributy, které mohou reprezentovat naše hodnoty, jsou:

- forma – délka, šířka, orientace, tvar, velikost, ohraničení,
- barva – barevný odstín, barevná sytost,
- poloha. [1]

Pro zakódování číselných hodnot je vhodné použít délku a pozici. Dále můžeme využít šířku, velikost a barevnou sytost. Tyto atributy dokáže čtenář snadno převést zpět na jejich číselnou reprezentaci a porovnávat je, poměrově i absolutně. Ostatní atributy jsou vhodné

pro kategorické informace, které odlišují jednotlivé datové série, například typy měřících přístrojů nebo oddělení ve firmě. Hlavní prvky grafů jsou:

- nadpis, podnadpis, popisek,
- osy, popisy os, značky a hodnoty intervalů,
- svislá a vodorovná mřížka,
- tělo grafu, objekty s atributy,
- popisky datových objektů,
- čáry a křivky trendu,
- referenční čáry,
- měřítka, legenda,
- okraj. [1, 2]

Grafy prezentující funkční závislosti mezi číselnými daty mají ve většině případů dvě osy – vodorovnou osu x a svislou osu y. Osy jsou označeny popiskem, který čtenáři říká, jaká veličina v jakých jednotkách se na osu vynáší. Na osách se nacházejí značky s uvedenými hodnotami v pravidelných intervalech, které pomáhají čtenáři odečítat hodnoty a orientovat se v datech. Závislost mezi vizuální délkou intervalu a hodnotami u značek vychází z použitého měřítka pro konkrétní osu. Mezi nejpoužívanější měřítka patří:

- lineární,
- logaritmické,
- reciproční. [1]

Značkami na ose by se měla vyznačovat „pěkná“ zaokrouhlená čísla. Měli bychom se vyhnout nepravidelným intervalům a náhodným číslům se zbytečným desetinným rozvojem. Intervaly mezi hlavními značkami mohou být rozděleny vedlejšími značkami, které už nejsou popsány odpovídajícími hodnotami. Vedlejší značky/intervaly mohou čtenáři urychlit odečítání hodnot a rozpoznání použitého měřítka. [1]

Osy by měly začínat nulovou hodnotou. Pokud je potřeba zobrazit kladné i záporné hodnoty, může se nulová hodnota zvýraznit. Zabrání se tím zavádějící prezentaci dat, při které můžeme ovlivnit čtenářovo pochopení dat. Podobné zkreslující účinky může mít špatný poměr stran grafu nebo špatné využití barev s rozdílnou sytostí. [1–3]

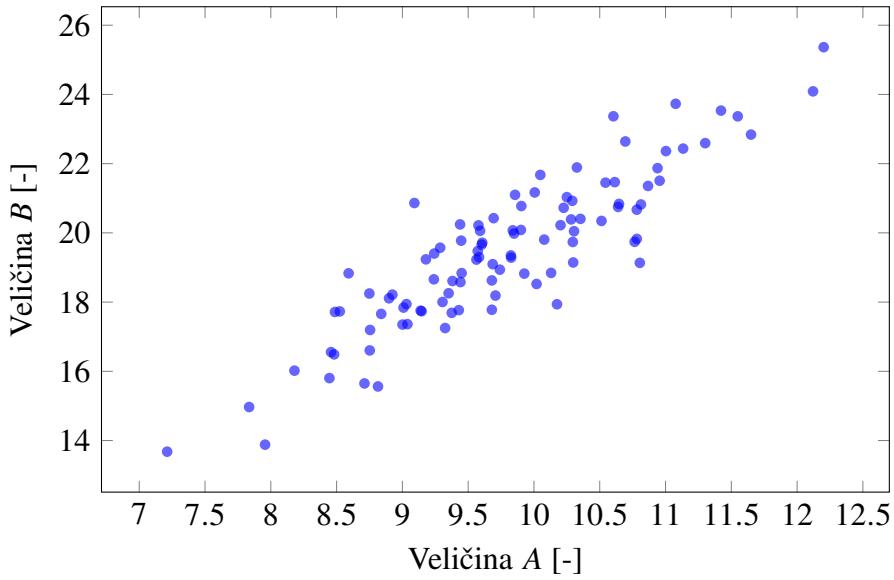
Následující seznam představí české i anglické názvy základních typů grafů, se kterými se můžeme setkat v technických dokumentech. Některé grafy spolu sdílí charakteristické prvky a může na ně být pohlíženo jako na podkategorie jiných grafů. V podkapitolách budou popsány rovinné (dvourozměrné) varianty některých zmíněných grafů.

- XY, bodový – scatter plot,
- spojnicový, čárový, liniový – line chart,
- sloupcový, řádkový, pruhový – vertical/horizonal bar chart,
- plošný – area chart,
- výsečový – pie chart,
- prstencový – donut chart,
- bublinový – bubble chart,
- krabicový – vertical/horizonal box chart, box-and-whisker chart,
- histogram – histogram,
- teplotní mapa – heat map,
- povrchový – surface chart,
- polární, paprskový – radar chart, polar chart,
- vrstevnicový, obrysový – contour plot,
- graf vektorového pole – vector field plot, quiver plot, stream plot,
- vodopádový – waterfall chart,
- Ganttův – Gantt chart,
- geografický, mapový – map chart,
- kartogram, choropletová mapa – choropleth map. [2, 4–6]

2.2.1 Bodový graf

Bodový graf má dvě spojité číselné osy obecně označované x a y. Osa má vždy svou veličinu s jednotkou a měřítkem. Data jsou v grafu prezentována jako jednotlivé body, které se mohou překrývat a nemusejí mezi sebou mít pravidelnou vzdálenost podle jedné z os. [1, 2]

Graf je vhodný pro prezentaci dat, ve kterých chceme pozorovat funkční závislost mezi dvěma veličinami nebo vznik shluků. Graf může být doplněn o křivku trendu. Ukázkový graf můžeme vidět na obrázku 1. [1, 2]

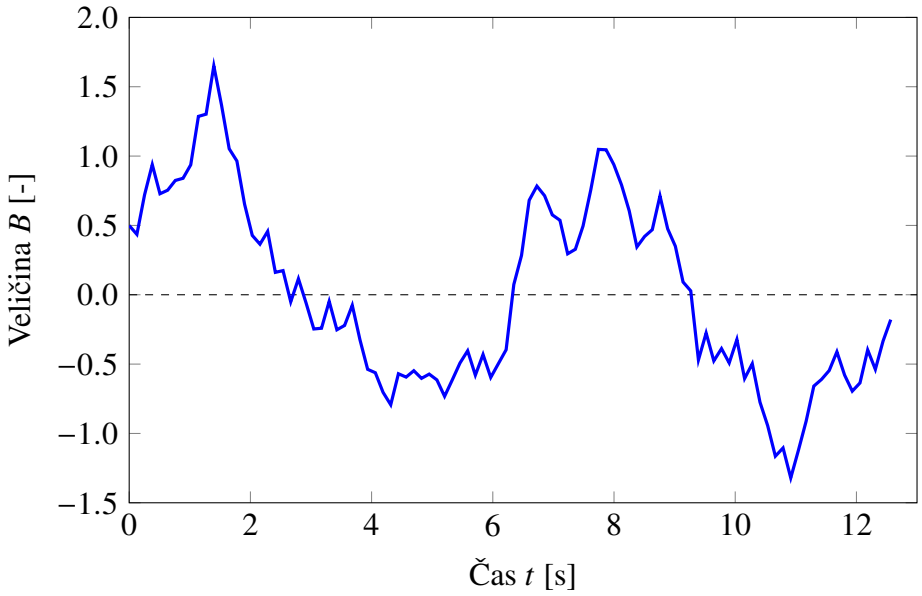


Obr. 1: Příklad bodového grafu

2.2.2 Čárový graf

Čárový graf je možné vnímat jako podkategorii bodového grafu. Osy jsou číselné, spojité a mají přidělenou veličinu, jednotku a měřítko. Vodorovná osa x často představuje časovou osu, u které je možné použít spojité číselné hodnoty i nespojité kategorické hodnoty. Popisky značek nespojité časové osy odpovídají například názvům jednotlivých měsíců v roce nebo dnům v týdnu. Data jsou prezentována jako jednotlivé body spojené spojnicí, která zdůrazňuje spojitost a návaznost dat. Body by měly být rozmístěny v pravidelných intervalech na ose x. [1, 2]

Graf je vhodný pro prezentaci spojité série hodnot získaných v pravidelných časových intervalech. Z grafu lze vypozorovat extrémy i časovou změnu hodnot – trend, fluktuaci, oblasti růstu a poklesu. Ukázkový graf můžeme vidět na obrázku 2. [1, 2]

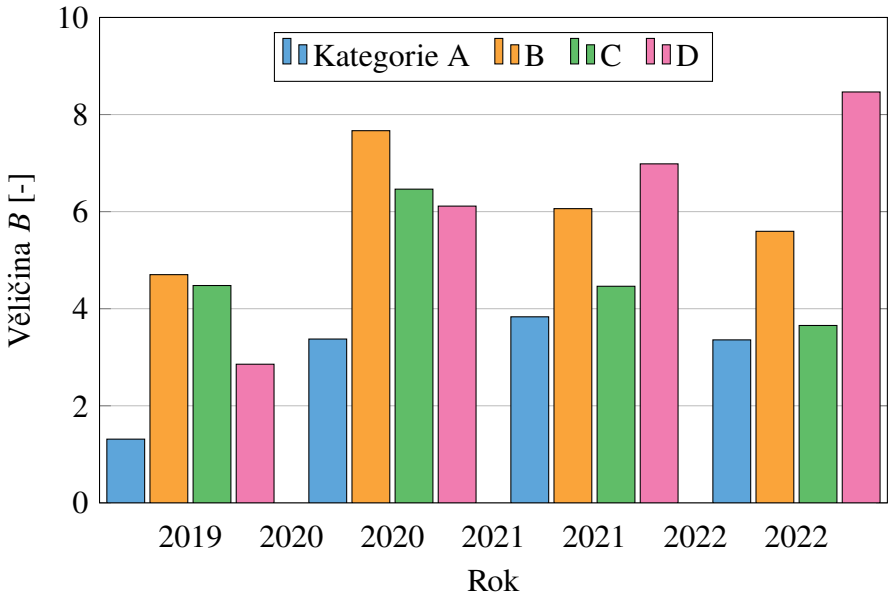


Obr. 2: Příklad čárového grafu

2.2.3 Sloupcový a řádkový graf

Sloupcový graf má dvě osy – vodorovnou kategorickou osu x a svislou číselnou osu y. Řádkový graf má osy prohozené. Data jsou v grafu prezentována jako svislé nebo vodorovné sloupce. Číselné hodnoty jsou zakódované ve výšce jednotlivých sloupců, zatímco šířka sloupců závisí pouze na formátování grafu. Kategorie se často seřazují podle jejich číselné hodnoty. [1, 2]

Graf je vhodný pro zobrazení číselných hodnot, které jsou přidělené k určité kategorii. Může nahrazovat jednoduché tabulky. Je možné prezentovat i procentuální hodnoty, které vznikly rozložením celku na jednotlivé kategorie, a nahradit tak výsečové grafy. Z grafu lze snadno vypozorovat kategorie s nejnižší a nejvyšší hodnotou. Ukázkový graf můžeme vidět na obrázku 3. [1, 2]



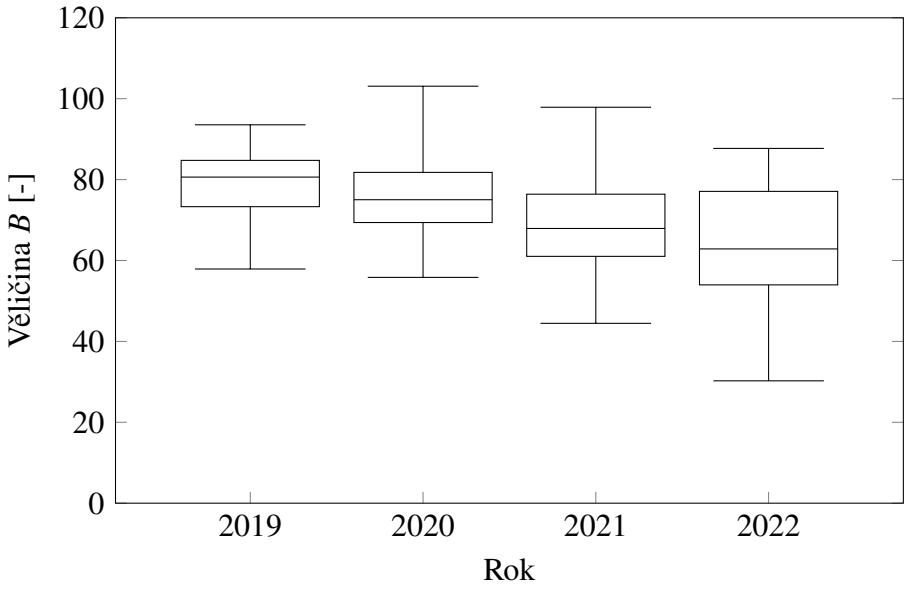
Obr. 3: Příklad sloupcového grafu

2.2.4 Krabicový graf

Krabicový graf má stejné rozložení os jako sloupcový graf – jednu kategorickou osu a jednu číselnou osu. Existují svislé i vodorovné varianty. Tento graf slouží pro prezentaci rozložení[1] hodnot v datových sériích pomocí krabic (plovoucích obdélníků). [1, 2]

Krabice je zezdola ohraničena hodnotou 1. kvartilu[2] a seshora hodnotou 3. kvartilu[3]. Krabice může být rozdělena na dvě části pomocí čáry odpovídající mediánu (2. kvartilu[4]). Z krabice mohou vystupovat svislé čáry („vousy“), které svým koncem označují minimální a maximální hodnotu v sérii dat. Konce čar mohou označovat i jiné, menší parametry než extrémy, a umožnit tak zobrazení hodnot, které leží mimo definované pásmo. Takové hodnoty označujeme za odlehlé[5] a v grafu se zobrazují jako body. Dalším bodem může být označen aritmetický průměr série dat. Některé varianty krabicového grafu využívají proměnnou šířku krabic, popřípadě šikmé zářezy, pro zakódování dalších parametrů. Při použití krabicového grafu je vhodné v textu vysvětlit, které hodnoty jsou kterým grafickým objektem, především svislými čárami, reprezentovány. [1–3]

Graf je vhodné použít pro zobrazení parametrů, které se používají při statistické analýze dat. Jde především o minimální a maximální hodnotu, medián a hodnotu 1. a 3. kvartilu. Tyto parametry můžeme v grafu porovnávat mezi jednotlivými kategoriemi nebo pozorovat jejich časový průběh. Ukázkový graf můžeme vidět na obrázku 4. [1, 2]



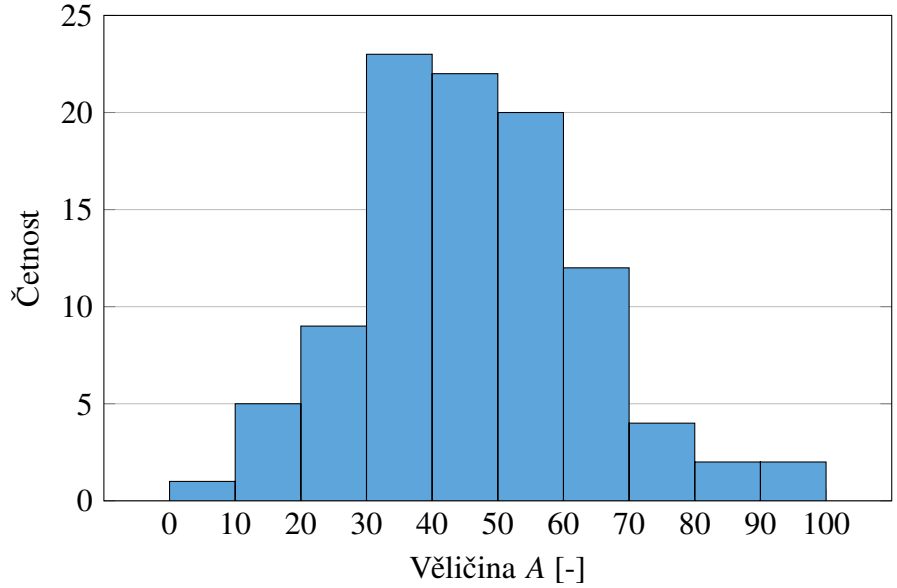
Obr. 4: Příklad krabicového grafu

[1] Ve statistice se používá odborný termín *rozdělení* nebo *distribuce*.
[2] 25% prvků má hodnotu menší než 1. kvartil.
[3] 75% prvků má hodnotu menší než 3. kvartil.
[4] 50% prvků má hodnotu menší než 2. kvartil.
[5] Označováno také anglickým výrazem *outliers*.

2.2.5 Histogram

Histogram je podkategorie sloupcového grafu, který slouží pro zobrazení rozložení hodnot v jedné, popřípadě více datových sériích. Graf je možné sestavit ze svislých i vodorovných sloupců. Graf má dvě číselné osy. V případě svislých sloupců má spojitou svislou osu y a nespojitou vodorovnou osu x, která vznikla rozdělením spojité číselné osy na stejně široké intervaly. Výška jednotlivých sloupců odpovídá počtu hodnot, které spadají do daného intervalu. Sloupce, na rozdíl od normálního sloupcového grafu, mezi sebou nemají mezery, což zdůrazňuje spojitosti dat a návaznost intervalů. Spojením středů vrcholů sloupců vzniká spojnicový graf nazývaný polygon četností. [1, 2]

Graf je vhodné použít pro zobrazení tvaru rozložení dat v datové sérii. Z grafu lze vypozorovat četnost hodnot ve sledovaných intervalech a druh rozložení, například rovnoměrné nebo normální (Gaussovo). Ukázkový graf můžeme vidět na obrázku 5. [1, 2]

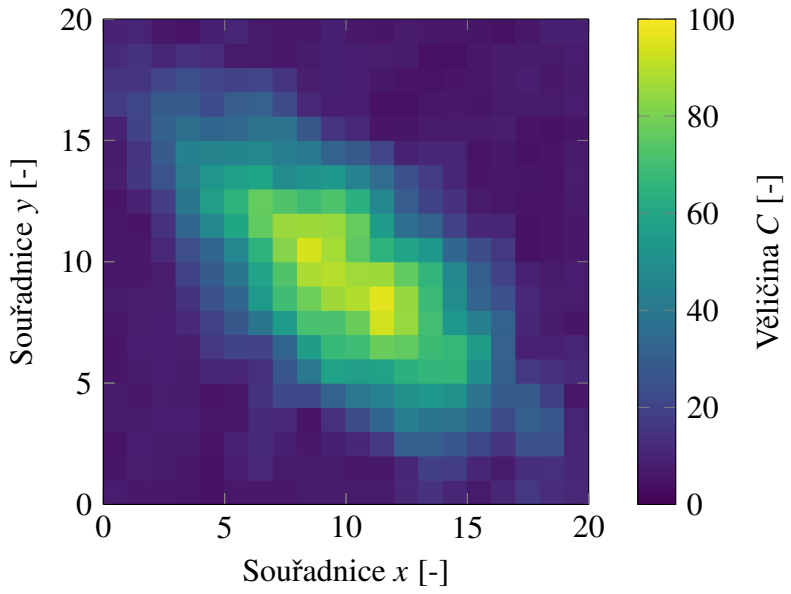


Obr. 5: Příklad histogramu

2.2.6 Teplotní mapa

Teplotní mapa zobrazuje číselné hodnoty nebo četnost dat pomocí dvourozměrné barevné mřížky. Při prezentaci četnosti dat lze na teplotní mapu pohlížet jako na histogram s přidaným rozměrem. Graf má většinou tři číselné osy – dvě nespojité a jednu spojitou. Dvě nespojité osy rozdělují graf na jednotlivá políčka. Třetí osa je umístěna vedle grafu a slouží jako legenda nebo měřítko pro převod barev na číselné hodnoty. Existují i jiné konfigurace os. Prezentovaná data mají dvě souřadnice, které definují jejich umístění na mapě, a velikost, která určuje barvu bodu nebo políčka. Číselnou velikost dat je možné chápat jako třetí souřadnici a data prezentovat i pomocí prostorového povrchového grafu. [2]

Graf je vhodné použít pro data ve formě tabulky nebo matice, u kterých chceme graficky zobrazit oblasti vyšších i nižších hodnot, tvar a trend dat. Graf umožňuje v datech vypozorovat obrazce, které by byly v tabulce náročné rozpoznat. Je také vhodný pro zobrazení matematických funkcí skalárních polí. Ukázkový graf můžeme vidět na obrázku 6. [2, 7]

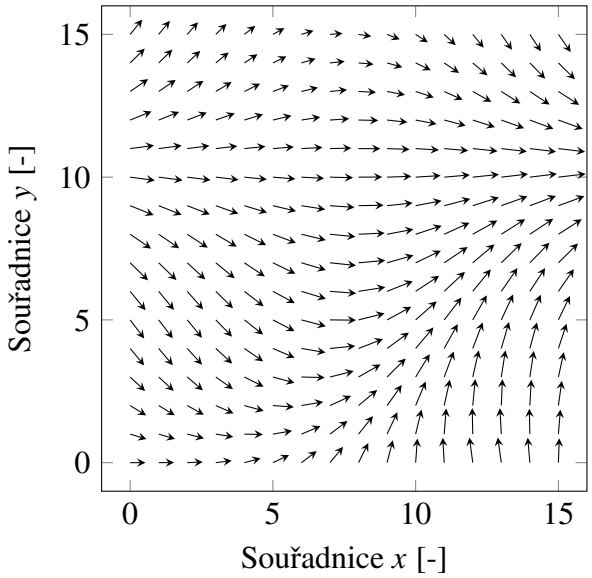


Obr. 6: Příklad teplotní mapy

2.2.7 Graf vektorového pole

Graf vektorového pole je obdoba teplotní mapy, ve které jednotlivé datové body zobrazujeme pomocí šipek. Jednotlivé šipky lze sestrojit ze znalosti souřadnic polohy, velikosti a směru. Velikost a směr můžeme zadat i relativními souřadnicemi polohy konce šipky, čehož se například může využít při zobrazování matic komplexních čísel. [2, 8]

Graf je vhodný pro zobrazení matematických funkcí vektorových polí a výsledků numerických simulací. V grafu lze pozorovat chování vektorového pole a odhalit místa zřídel, propadů a vírů. Ukázkový graf můžeme vidět na obrázku 7. [2, 3, 7, 8]



Obr. 7: Příklad grafu vektorového pole

3 Python: programovací jazyk

Python je rozšířený programovací jazyk, jehož první verze 0.9.0 byla světu představena v roce 1991 nizozemským programátorem Guido van Rossumem. Dnes je Python spravován americkou neziskovou organizací Python Software Foundation. Jeho hlavní charakteristiky jsou:

- vysokoúrovňový – poskytuje vysokou úroveň abstrakce od fungování výpočetní techniky, programátor nemusí přemýšlet nad jednotlivými instrukcemi pro procesor, upřednostňuje se srozumitelnost kódu a rychlost vývoje programu před jeho rychlostí a využitím výpočetního výkonu,
- hybridní/multiparadigmatický – programátor může kombinovat různé způsoby, kterými manipuluje s daty a během programu, hlavní dva modely jsou objektově orientované a funkcionální programování,
- interpretovaný – program nemusí být kompilován, pro spuštění je potřeba pouze zdrojový kód a program interpretující daný kód (interpret), výhodou je snadné ladění programu, nevýhodou pak pomalejší běh programu,
- multiplatformní – díky interpretaci je možné spustit zdrojový kód na široké škále výpočetních platforem, pro které existuje interpret, bez větších zásahů do kódu,
- dynamicky typovaný – programátor nemusí definovat typy proměnných, jedna proměnná může během svého života nabýt hodnot rozdílných typů, před během programu se neprovádí statická kontrola typů, typy jsou důležité až při běhu programu,
- automatická správa paměti – programátor nemusí žádat výpočetní platformu o volnou paměť a poté ji ručně uvolňovat, přidělování a uvolňování paměti probíhá automaticky na pozadí programu,
- rozsáhlá standardní knihovna – možnost využít různorodé funkce a datové struktury bez nutnosti přidávat externí rozšíření,
- bohatý ekosystém přídavných modulů – aktivní komunita tvořící volně šiřitelné i komerční moduly, které přidávají nové funkce nebo rozšiřují ty stávající,
- snadná čitelnost kódu,
- jednoduchá syntaxe,
- rychlé intuitivní programování,
- otevřený zdrojový kód. [9–13]

Python je kromě obecného programování používán i jako skriptovací jazyk v různých programech a aplikacích, například pro tvorbu rozšíření nebo jednoduchých maker. Python umožňuje využívat kompilované knihovny nízkoúrovňových programovacích jazyků,

především C a C++. Tohoto propojení, kdy z Pythonu ovládáme procedury optimalizované pro naši výpočetní platformu, se hojně využívá při výpočetně náročných operacích. Jedná se o jeden z hlavních důvodů, proč se Python v průběhu let stal velmi populárním v oblasti zpracování dat a vědeckého výzkumu. S Pythonem se můžeme setkat v následujících oblastech:

- automatizace,
- vědecké a numerické simulace,
- strojové učení a vidění,
- zpracování a analýza dat,
- vývoj desktopových nebo webových aplikací a her,
- testování. [11, 12]

3.1 Externí moduly

Externí Python moduly[6] jsou uloženy ve veřejném repositáři PyPI (Python Package Index). Pro jejich stahování, instalaci a správu slouží univerzální nástroj `pip` (Package Installer for Python). Pro stažení modulu stačí zadat jeho název do příkazu v příkazové řádce a `pip` zajistí stažení a instalaci modulu včetně všech jeho závislostí. Alternativou pro `pip` je nástroj `conda`, který cílí převážně na vědeckou komunitu používající operační systém Windows. `conda` nabízí svůj vlastní repositář modulů pro programovací jazyky Python a R. Volně dostupné externí moduly usnadňují vývojářům práci, poskytují přístup k mnoha užitečným funkcím, snižují náročnost vývoje aplikací a zkracují dobu vývoje. Pro technické účely se nejčastěji využívají následující moduly a nástroje:

- `numpy` – rychlé a efektivní datové struktury (matice, případně víceprostorové pole čísel nebo objektů) a funkce pro náročné numerické výpočty, kritické části funkcí jsou psané v programovacích jazycích C, C++ a Fortran,
- `scipy` – funkce a algoritmy pro lineární algebru, optimalizační úlohy, numerickou integraci a řešení soustav lineárních, nelineárních i diferenciálních rovnic, využívá datové struktury z `numpy`,
- `pandas` – manipulace a analýza tabulkových dat včetně načítání a ukládání dat v různých formátech,
- `sympy` – operace se symbolickými matematickými výrazy, umožňuje symbolické řešení soustav rovnic, derivaci, integraci a následné převedení do Python funkcí pro numerické výpočty,

[6] Označováno také anglickým výrazem *packages*.

- `matplotlib` – tvorba statických, animovaných i interaktivních grafů, vhodné pro prezentaci dat v technických dokumentech i uživatelských rozhraních,
- `jupyterlab`, `notebook` a `jupyter-book` – balíček nástrojů pro tvorbu interaktivních dokumentů s výpočty, tabulkami a grafy v editoru běžícím ve webovém prohlížeči, je možné využít i jiné programovací jazyky než Python, například R nebo Julia,
- `scikit-learn` - funkce pro datovou analýzu (klasifikace, regrese, shluková analýza), tvorbu prediktivních modelů a strojové učení,
- `tensorflow` a `pytorch` – funkce pro tensorové výpočty, strojové učení a vidění, zpracování přirozeného jazyka a tvorbu a trénování neuronových sítí a umělé inteligence,
- `opencv-python` – algoritmy pro strojové vidění a manipulaci s obrázky. [14–28]

4 TEX: program, jazyk, formát

TEX[7] *(vyslovujeme tech) je volně dostupný systém k vytváření elektronické sazby vysoké kvality. Je vybaven sofistikovaným makrojazykem. TEX vytvořil Donald Knuth v 70. letech minulého století. Přesto se používá i v dnešní době a v mnoha vlastnostech dosud nemá konkurenci.* [29, s. 3]

Vstupem pro TEXový program[8] je textový soubor. Obsahuje text dokumentu a řídicí sekvence (příkazy, značky, makra), kterými ovlivňujeme sazbu obsahu. TEX ve svém základu obsahuje zhruba 300 primitivních řídicích sekvencí, které můžeme skládat do složitějších maker. Aby uživatel před psaním dokumentu nemusel definovat makra pro běžné operace, na které je zvyklý z klasických textových procesorů, existují předdefinované formáty[9], soubory obsahující definice maker, které můžeme do TEXu načítat. Základní formáty jsou:

- plainTEX,
- LATEX,
- ConTEXt,
- CSplain,
- OPmac,
- EncTEX,
- OpTEX. [31]

Původní program vytvářející zobrazitelnou sazbu se nazývá TEX. Převádí textový vstup v podobě `.tex` souboru do binárního formátu `.dvi`. Pro zobrazení obsahu `.dvi` souboru se používají speciální prohlížeče nebo programy, který obsah převedou do jiného, uživatelsky přívětivějšího formátu, například `.pdf` souboru. [30]

Existují i nástavby původního programu, které rozšiřují možnosti makrojazyka nebo umožňují výstup v nových formátech. Základní programy jsou:

- TEX,
- pdfTEX – umožňuje výstup ve formátu `.pdf`,
- XETEX – umožňuje použití Unicode znaků ve vstupním souboru a tvorbu výstupu ve formátu `.pdf`,
- LuaTEX – rozšiřuje makrojazyk o skriptovací jazyk Lua, umožňuje použití Unicode znaků ve vstupním souboru a tvorbu výstupu ve formátu `.pdf`. [29]

[7] Slovem TEX můžeme, kromě celého sázecího systému, označovat i jeho komponenty, zmíněný makrojazyk nebo program, který převádí textovou reprezentaci dokumentu do zobrazitelné sazby.

[8] Označováno také anglickým výrazem *engine*.

[9] Označení formát vyplývá z původního využití souborů pro nastavení stylu dokumentu. Dnes dává smysl používat i označení „makrobalík“ [29, 30].

Programy podporující sazbu Unicode znaků jsou výhodné především pro dokumenty v jazycích vyžadujících speciální znaky nebo diakritiku. Při použití programu pdfTEX, případně pdfLATEX, je nutné využít balíčky maker umožňující správnou sazbu Unicode znaků. Společně s podporou Unicode znaků programy většinou nabízejí i možnost změnit font sázeného textu.

Aby se běžný uživatel nemusel učit vytvářet makra pro konkrétní TEXový formát a program, nabízí TEX, převážně LATEX, bohatý ekosystém volně šiřitelných balíčků maker[10]. Uživatel si makra z balíčku přidá pomocí speciálních řídicích sekvencí, které se vkládají na začátek zdrojového souboru. Některé TEX distribuce obsahují možnost automatického stažení chybějících balíčku při kompilaci[11] dokumentu.

Většina zmíněných formátů a programů je obsažena v běžných TEX distribucích TEX Live a MikTEX, které obsahují vše potřebné pro tvorbu dokumentů pomocí systému TEX. Před samotnou tvorbou dokumentu si uživatel zvolí TEX program, formát a zadefinuje přídavné balíčky, které má program načíst. Nejčastěji uživatelé volí formát LATEX a program podporující znaky jejich jazyku a tvorbu `.pdf` souborů. Uživatelé mohou své dokumenty kompilovat lokálně na svých počítačích nebo pomocí online služeb, například Overleaf. [29]

4.1 Balíčky maker

Existuje velké množství balíčků maker a pro uživatele je často náročné zvolit, který balíček pro vyřešení dané problematiky použít. Musí si také ohlídat, jestli použité balíčky jsou s sebou kompatibilní. Uživatelé se většinou rozhodují pomocí obsahu dokumentace k danému balíčku, zvyklostem instituce, pro kterou dokument vytvářejí, a radám na internetových fórech.

TEXové programy, balíčky maker, dokumentace a další pomocné nástroje jsou uloženy především v internetovém archivu CTAN (Comprehensive TeX Archive Network), který vznikl kolem let 1992 a 1993. V následujících letech podobný formát archivů převzaly i jiné projekty, například CPAN pro programovací jazyk Perl nebo CRAN pro jazyk R. [32]

Velká část balíčků maker využívá externí programy, které se už nacházejí v základní instalaci TEXové distribuce nebo se automaticky přidají při instalaci daného balíčku. Některé velké programy si uživatel musí instalovat sám, například Python nebo Inkscape. Pokud je při kompilaci povoleno volání externích příkazů a programů, může je prostý uživatel nebo tvůrce balíčku využít pomocí maker `\write18{...}` a `\input{|...}`. Použití externích programů dovoluje snadné rozšíření funkčnosti LATEXu bez nutnosti psát složité makra nebo zasahovat do zdrojového kódu kompilačního programu.

[10] Označováno také anglickým výrazem *packages*.

[11] Spuštění TEX programu a následná sazba dokumentu.

4.2 Tvorba tabulek

Pro tvorbu jednoduchých tabulek lze využít LaTeXové prostředí `tabular`. Lze v něm specifikovat zarovnání textu v jednotlivých sloupcích a nastavit svislé i vodorovné okraje. Obsah buněk se vkládá po řádcích, kde se jednotlivé sloupce oddělují znakem `&`, a konec řádku se značí řídicí sekvencí `\\`. Pro vytvoření jednoho nebo více okrajů se používá znak `|` pro svislé okraje a makro `\hline` pro vodorovné okraje. Následující ukázka kódu 1 vytvoří příkladnou tabulku se třemi sloupci s vycentrovaným textem a třemi řádkami, viz obrázek 8.

```
\begin{tabular}{| c || c c }
\hline
A & B & C \\
\hline \hline
D & E & F \\
G & H & I \\
\end{tabular}
```

Kód 1: Příklad jednoduché tabulky

A	B	C
D	E	F
G	H	I

Obr. 8: Příklad jednoduché tabulky

LaTeX nabízí možnosti pro přizpůsobení svého tabulkového prostředí. Můžeme například měnit umístění tabulky na stránce, tloušťku okrajů nebo volný prostor kolem textu v buňce. Do tabulek je možné vkládat popisek a označení, pomocí kterého lze v textu na tabulku odkazovat. Pro tabulky, kde chceme ovládat přesnou šířku sloupců, mít přesnější kontrolu nad jednotlivými buňkami, spojovat více buněk do jedné nebo načítat data z externích souborů, můžeme využít následující balíčky maker:

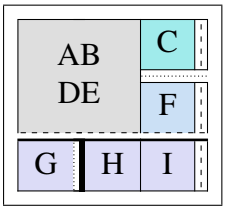
- `array` nebo `booktabs` – vylepšení základního tabulkového prostředí,
- `tabularx` nebo `tabulary` – nástavba `array`, přidává vlastní tabulkové prostředí,
- `longtable` – automatické rozdělování dlouhých tabulek na více stránek, přidává vlastní tabulkové prostředí,
- `multirow` – spojování buněk ve sloupcích,
- `xcolor` nebo `colortbl` – nástavba `color`, umožňuje změnu barvy okrajů a pozadí buněk,
- `tabularray` – nahrazuje většinu předchozích balíčků svou funkčností a ucelenostní, využívá nových funkcí přidaných v LaTeX3 a přidává vlastní tabulkové prostředí,

- `nicematrix` – nástavba `array`, složité tabulky a matice pomocí systému PGF/TikZ pro tvorbu grafiky,
- `csvsimple` – tvorba tabulek z `.csv` souborů, využívá PGF/TikZ,
- `datatool` – obsáhlý balíček pro tvorbu tabulek a grafiky z `.csv` souborů, využívá PGF/TikZ. [33–42]

Následující ukázka kódu 2 obsahuje příklad vizuálně složité tabulky vytvořené prostředím `tblr` z balíčku maker `tabularray`. Barvy, styl okrajů a sloučení buněk jsou jednoduše nastaveny předem na začátku prostředí a tělo tabulky poté obsahuje pouze text buněk bez nepřehledných řídicích sekvencí. Výsledná tabulka lze vidět na obrázku 9.

```
\begin{tblr}{
    colspec={|c|[dotted]|[2pt]c|c|[solid]|[dashed]|},
    rowspec={|Q[cyan9]||[dotted]|Q[azure9]|[dashed]|[1pt]Q[blue9]|},
    cell{1}{1}={c=2,r=2}{gray9}
}
{AB \\ DE} &   & C \\
           &   & F \\
G          & H & I \\
\end{tblr}
```

Kód 2: Příklad složité tabulky pomocí balíčku `tabularray`



Obr. 9: Příklad složité tabulky pomocí balíčku `tabularray`

4.3 Tvorba grafů

Existují dva způsoby, kterými můžeme do LaTeXového dokumentu přidávat grafy, včetně rastrové i vektorové grafiky.

1. Využijeme balíčky maker pro tvorbu grafiky přímo ve zdrojovém souboru,
    - výhody – autor nemusí opouštět textový editor, možnost tvorby grafiky pomocí matematických vztahů, existence nástavbových modulů pro specializované funkce,
    - nevýhody – pomalejší kompilace dokumentu, vyšší spotřeba operační paměti při kompilaci, náročné pro začínající uživatele,

2. grafy zpracujeme v externím programu, například Microsoft Excel, LibreOffice Calc, Matlab nebo gnuplot, a využijeme balíčky maker pro vložení grafického materiálu do vygenerovaného dokumentu,
    - výhody – možnost využít interaktivní grafické editory, uživatel může použít své každodenní programy, vhodné pro vygenerovaný obsah,
    - nevýhody – rozdílný styl dokumentu a vložené grafiky, opakované exportování a přenášení souborů při častých změnách.

4.3.1 Tvorba grafů uvnitř LaTeXu

První metoda je vhodná pro jednodušší grafiku, diagramy a grafy. Uživatel se před samotnou tvorbou grafiky musí nejdříve seznámit s vybranými balíčky, protože narozdíl od vizuálních editorů, kde uživatel vidí, co zrovna tvoří, je zde tvorba grafických objektů ovládaná pouze pomocí textového popisu a maker. Existují tři hlavní balíčky maker, které tuto funkčnost nabízejí. Zároveň je možné ke každému balíčku načíst další doplňující balíčky pro rozšíření jeho možností.

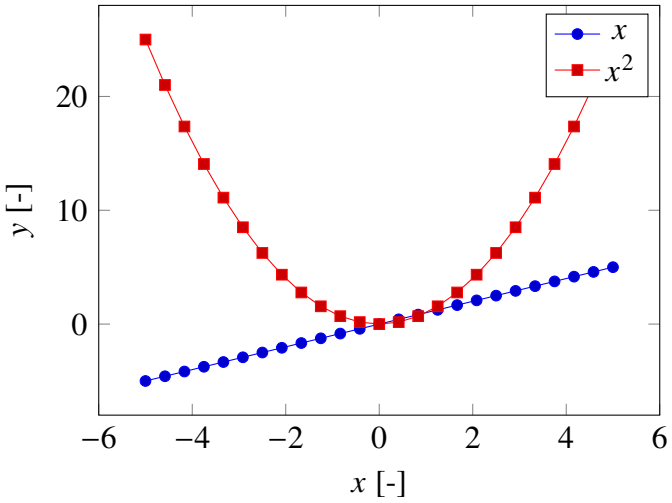
- `pgf`, `tikz` – populární balíček pro tvorbu vektorové grafiky pomocí vlastního jazyka TikZ, který se dále převádí do specifikace PGF, balíček je často označován spojením zkratek PGF/TikZ, dokumenty je možné kompilovat programem pdfLaTeX,
    - ⋆ `pgfplots` – nástavba dovolující vytvářet grafy z matematických definic nebo tabulkových dat, nabízí různé předpřipravené druhy grafů a dovoluje využívat externí programy, například gnuplot pro náročné grafy,
- `pstricks` – méně populární, znemožňuje přímou generaci `.pdf` dokumentů kvůli nekompatibilitě s programem pdfLaTeX, řešením je generace `.dvi` souborů nebo využití dalších balíčků,
    - ⋆ `pst-plot` – nástavba pro tvorbu grafů, nabízí podobné funkce jako balíček `pgfplots`, méně uživatelsky přívětivé,
- `asymptote` – balíček umožnující tvorbu grafiky ze zdrojového LaTeXového souboru pomocí externího programu Asymptote, program samostatně nabízí generaci obsahu do různých statických i interaktivních formátů, využívá vlastní programovací jazyk pro tvorbu grafiky. [43–48]

Následující ukázka kódu 3 slouží pro vytvoření jednoduchého grafu pomocí balíčku `pgfplots`. Graf obsahuje dva průběhy matematických funkcí $y = x$ a $y = x^2$, viz obrázek 10.

```
\begin{tikzpicture}
\begin{axis}[
    xlabel={$x$ [-]},
    ylabel={$y$ [-]},
```

```
    legend entries={$x$, $x^2$},
    width=9cm, height=7cm
]
\addplot{x};
\addplot{x^2};
\end{axis}
\end{tikzpicture}
```

Kód 3: Příklad jednoduchého grafu pomocí balíčku `pgfplots`

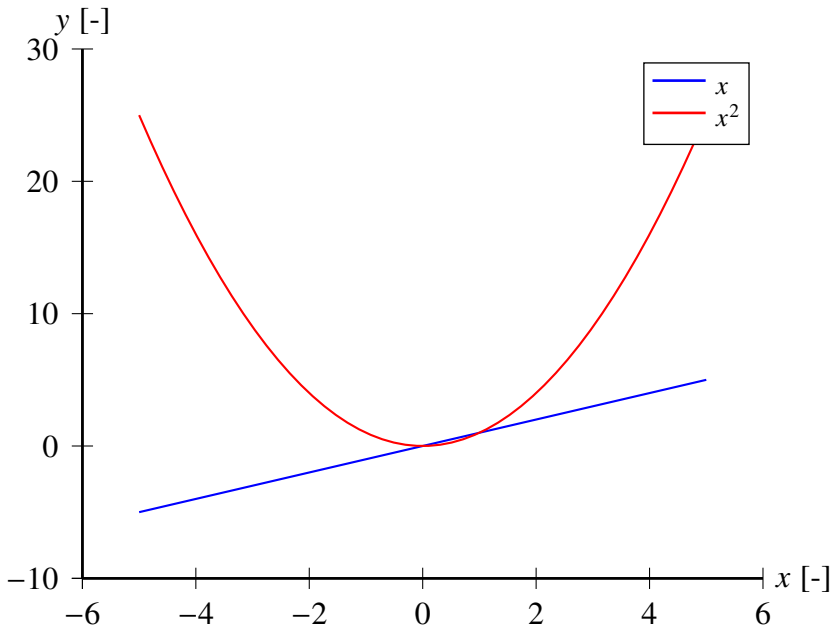


Obr. 10: Příklad jednoduchého grafu pomocí balíčku `pgfplots`

Další ukázka kódu 4 využívá maker z balíčku `pst-plot` a vytváří podobný graf jako v předchozí ukázce. Můžeme zde vidět, že tento balíček od uživatele vyžaduje nastavení více parametrů pro získání obstojného výsledku. Dále můžeme vidět, že se výchozí nastavení stylu a formátování grafu liší mezi balíčky, například ve stylu okraje nebo značek intervalů, viz obrázek 11.

```
\begin{psgraph}[
    xAxisLabel={$x$ [-]},
    yAxisLabel={$y$ [-]},
    Dx=2, Dy=10,
    Ox=-6, Oy=-10,
](-6,-10)(6,30){9cm}{7cm}
\pslegend[rt]{
    \blue\rule[1ex]{2em}{1pt} & $x$ \\
    \red\rule[1ex]{2em}{1pt} & $x^2$
}
\psplot[linecolor=blue]{-5}{5}{x}
\psplot[linecolor=red]{-5}{5}{x dup mul}
\end{psgraph}
```

Kód 4: Příklad jednoduchého grafu pomocí balíčku `pst-plot`



Obr. 11: Příklad jednoduchého grafu pomocí balíčku `pst-plot`

4.3.2 Vkládání grafů z externích programů

Druhá metoda je vhodná pro všechny druhy grafiky. Autoři většinou zpracovávají svá data v externích programech, které často nabízejí funkce pro tvorbu grafů, například Matlab. Vytvořené grafy se následně ručně nebo automaticky uloží do některého z používaných formátů pro grafický materiál, například `.png`, `.tiff`, `.jpeg`, `.svg`, `.eps` nebo `.pdf`. V tomto případě je pro autora snazší vložit vygenerované grafy přímo do dokumentu než exportovat data do meziformátu a načítat a zpracovávat je pomocí LaTeXových maker. Pro vkládání hotové grafiky do dokumentů existují následující balíčky maker:

- `graphics` – původní balíček umožňující vkládat do dokumentů externí grafiku a manipulovat s ní, přidává základní makra `\includegraphics`, `\rotatebox`, `\scalebox` a `\resizebox`,
- `graphicx` – nástavba `graphics`, přidává další možnosti pro definovaná makra,
- `svg` – podpora grafiky v `.svg` formátu, umožňuje využít externí program Inkscape pro rozdělení textu od grafiky, a tím zajistit jednotný font a velikost písma v dokumentu,
- `pdfpages` – přidávání celých i modifikovaných stránek z jiných `.pdf` dokumentů,
- `media9` – vkládání interaktivního obsahu pro čtečky `.pdf` dokumentů podporující funkce Adobe Reader-9/X,
- `embedfile`, `attachfile(2)`, `navigator` – přidávání příloh (libovolných souborů) do výsledného `.pdf` dokumentu. [49–57]

5 Způsoby generace obsahu

Generovat dokumenty z externích dat a výpočtů můžeme pomocí Pythonu a LaTeXu dvěma hlavními způsoby, které budou popsány v následujících dvou kapitolách.

5.1 Generace z Python prostředí

První způsob generace spočívá v použití Python skriptu, který za nás vytvoří textový soubor a naplní ho řídicími sekvencemi a textem reprezentujícím naše data. Skript může používat lokálně stažená data, například `.csv`, `.xlsx`, `.ods`, `.json` a `.xml` soubory, nebo může komunikovat s webovými API[12] a databázemi.

LaTeXové příkazy musíme psát uvnitř Python skriptu, většinou ve formě textových řetězců[13]. Následně pomocí funkcí operujících s textovými objekty (`f-string`, `str.format`, `str.replace`, `re.sub`) můžeme nahradit dočasné klíče[14] načtenými daty. V ukázce kódu 5 můžeme vidět názorný příklad, ve kterém pomocí Pythonu vytváříme soubor a vkládáme do něj řídicí sekvence pro vytvoření jednoduchého LaTeXového dokumentu. Výsledný obsah vygenerovaného souboru můžeme vidět v ukázce kódu 6. [58]

```
# Proměnná obsahující textový řetězec.
# [1] představuje dočasný klíč, který bude nahrazen.
template = "Vložené číslo: [1]\n"
# Proměnná s uloženým číslem, které chceme zobrazit.
number = 1989
# Operátor 'with' a funkce 'open' otevře/vytvoří
# soubor a následně ho i zavře.
with open("dokument.tex", "w", encoding="utf8") as f:
    # Funkce 'write' vloží textový řetězec do souboru.
    f.write("\\documentclass{article}\n")
    f.write("\\begin{document}\n")
    # Funkce 'replace' nahradí dočasný klíč za jiný textový řetězec.
    f.write(template.replace("[1]", str(number)))
    f.write("\\end{document}")
```

Kód 5: Vytvoření zdrojového souboru jednoduchého LaTeXového dokumentu

```
\documentclass{article}
\begin{document}
Vložené číslo: 1989
\end{document}
```

Kód 6: Zdrojový soubor jednoduchého LaTeXového dokumentu

[12] Application Programming Interface – programátorské rozhraní mezi dvěma programy.

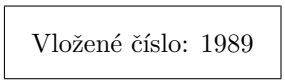
[13] Označováno také anglickým výrazem *strings*.

[14] Sekvence znaků, které programu/skriptu/funkci říkají, kam v textovém řetězci umístit data. Používá se anglický výraz *placeholder*.

Pro převedení zdrojového kódu do čitelné podoby musíme spustit některý z LaTeXových programů a nasměrovat ho na náš nově vytvořený `.tex` soubor. Můžeme pro to využít Python funkci `run` z modulu `subprocess` a program pdfLaTeX, který spustíme pomocí konzolového příkazu `pdflatex`, jak můžeme vidět v ukázce kódu 7. Po spuštění skriptu se nám v adresáři vytvoří `.pdf` soubor obsahující text „Vložené číslo: 1989“, viz obrázek 12.

```
import subprocess
# Funkce 'run' spustí daný konzolový příkaz.
# Vrácený objekt 'result' obsahuje informace o průběhu programu.
result = subprocess.run(
    # Příkaz je rozdělený na název programu a jednotlivé argumenty.
    # Argument '-interaction=batchmode' zamezí pozastavení programu
    # při nalezení chyby ve zdrojovém kódu LaTeXového dokumentu.
    ["pdflatex", "-interaction=batchmode", "dokument.tex"],
    capture_output=True,  # Zaznamenáme výstup z volaného programu.
    encoding="cp852",     # Nastavíme správné kódování znaků výstupu.
    errors="replace",     # Neznámé znaky budou nahrazeny otazníkem.
)
```

Kód 7: Spuštění pdfLaTeXu pomocí Python skriptu



Obr. 12: Výsledný dokument vygenerovaný z jednoduchého skriptu

Pokud pro psaní Python skriptů využíváme IDE[15], přicházíme o zvýraznění a kontrolu syntaxe, případně i o návrhy a doplňování kódu, pro LaTeXové části skriptu. Snižujeme tím svoji rychlost vytváření dokumentu a zvyšujeme šanci na chyby a překlepy.

5.1.1 Šablonovací systémy

Pro vytváření složitějších dokumentů je výše popsaný způsob generace nedostačující. LaTeXové značky a textové části dokumentu jsou zapletené do funkčních částí Python skriptu. To zhoršuje udržitelnost[16] projektu, především má-li být Python skript využíván pro různé typy dokumentů. Nabízí se řešení, ve kterém rozdělíme text a kód do separovaných souborů a využijeme interní nebo externí Python moduly pro získání šablonovacích funkcí.

Šablonovací systémy se nejčastěji používají u serverového renderování[17] webových stránek, a jsou proto přizpůsobené k práci s `.xml/.html` semi-strukturovanými soubory. Je ovšem možné je využít i pro generování jiných souborů, v našem případě LaTeXového zdrojového kódu. Systémy se dělí podle možností, které poskytují, na dvě základní skupiny.

[15] Integrated Development Environment – textový editor s rozšířenými funkcemi pro psaní kódu.
[16] ve smyslu rychlosti opravovat chyby a možnosti vytvářet rozšíření
[17] Vypořádání se s podmínkami a cykly uvnitř šablony a nahrazení dočasných klíčů za poskytnutá data.

Do první skupiny spadají jednodušší systémy pracující pouze s nahrazováním dočasných klíčů, například interní Python modul `string.Template`. Do druhé skupiny patří složitější systémy dovolující zakomponovat do šablony klasické programovací prvky, například podmínky, cykly, filtry a komentáře. Často nabízejí i možnost tvorby vlastních maker nebo kombinovat šablony z více souborů. Některé systémy dovolují spouštět Python příkazy i celé bloky kódu zevnitř šablony. Následující výčet obsahuje výběr externích modulů, které poskytují výše zmíněné funkce. [58]

- `jinja2`,
- `django.template`,
- `mako`,
- `cheetah3`,
- `pyratemp`. [58]

V následujících ukázkách kódu 8, 10 a 12 lze vidět Python skripty využívající základní funkce šablonovacích systémů `jinja2`, `mako` a `pyratemp`. Skripty v ukázkách se ve svém obsahu téměř neliší. Větší rozdíl můžeme vidět v použitých kontrolních symbolech v šablonách, které jsou ukázány ve výpisech kódu 9, 11 a 13.

```
import jinja2
# Otevření souboru se šablonou.
with open("./_jinja2.tex", "r", encoding="utf8") as f1:
    # Vytvoření jinja2 šablony z načteného souboru.
    template = jinja2.Template(f1.read())
    # Vyrenderování šablony pomocí vložených dat.
    result = template.render(
        text="Rozdělení ocelí podle použití:",
        data=["konstrukční", "nástrojové"]
    )
    # Uložení zdrojového kódu LaTeX dokumentu do souboru.
    with open("./jinja2.tex", "w", encoding="utf8") as f2:
        f2.write(result)
```

Kód 8: Generace zdrojového kódu pomocí modulu `jinja2`

Systém `jinja2` využívá ve svých šablonách značky `{{...}}` pro proměnné a `{%...%}` pro podmínky a cykly. [59]

```
\documentclass{article}
\begin{document}
{{text}}
\begin{itemize}
{%- for x in data %}
    \item {{x}}
```

```
{%- endfor %}
\end{itemize}
\end{document}
```

Kód 9: Šablona pro modul `jinja2`

```
import mako.template
# Vytvoření mako šablony ze souboru.
template = mako.template.Template(
    filename="./_mako.tex",
    input_encoding="utf8"
)
# Vyrenderování šablony pomocí vložených dat.
result = template.render(
    text="Rozdělení ocelí podle použití:",
    data=["konstrukční", "nástrojové"]
)
# Odstranění problému s novými řádky.
result = result.replace("\r\n", "\n")
# Uložení zdrojového kódu LaTeX dokumentu do souboru.
with open("./mako.tex", "w", encoding="utf8") as f:
    f.write(result)
```

Kód 10: Generace zdrojového kódu pomocí modulu `mako`

Systém `mako` využívá ve svých šablonách značky `${...}` pro proměnné a `\% ...` pro podmínky a cykly. [60]

```
\documentclass{article}
\begin{document}
${text}
\begin{itemize}
% for x in data:
    \item ${x}
% endfor
\end{itemize}
\end{document}
```

Kód 11: Šablona pro modul `mako`

```
import pyratemp
# Vytvoření pyratemp šablony ze souboru.
template = pyratemp.Template(
    filename="./_pyratemp.tex",
    encoding="utf8"
)
# Vyrenderování šablony pomocí vložených dat.
```

```
result = template(
    text="Rozdělení ocelí podle použití:",
    data=["konstrukční", "nástrojové"]
)
# Uložení zdrojového kódu LaTeX dokumentu do souboru.
with open("./pyratemp.tex", "w", encoding="utf8") as f:
    f.write(result)
```

Kód 12: Generace zdrojového kódu pomocí modulu `pyratemp`

Systém `pyratemp` využívá ve svých šablonách značky `@!...!@` pro proměnné a `<!--(...)-->` pro podmínky a cykly. Navíc zde ještě definujeme typ souboru pomocí `<!--(set_escape)--> LaTeX <!--(end)-->`. [61]

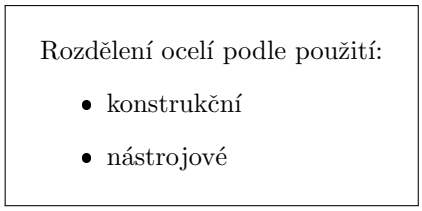
```
<!--(set_escape)--> LaTeX <!--(end)-->
\documentclass{article}
\begin{document}
@!text!@
\begin{itemize}
<!--(for x in data)-->
    \item @!x!@
<!--(end)-->
\end{itemize}
\end{document}
```

Kód 13: Šablona pro modul `pyratemp`

Spuštění výše uvedených skriptů nám vytvoří totožné zdrojové soubory s obsahem viditelným v ukázce kódu 14. Po spuštění programu pdfLaTeX dostaneme totožné `.pdf` dokumenty, viz obrázek 13.

```
\documentclass{article}
\begin{document}
Rozdělení ocelí podle použití:
\begin{itemize}
    \item konstrukční
    \item nástrojové
\end{itemize}
\end{document}
```

Kód 14: Zdrojový LaTeXový soubor vygenerovaný ze šablony



Obr. 13: Výsledný dokument vygenerovaný ze šablony

Při používání šablonovacích systémů narážíme na problém nekompatibility kontrolních symbolů a značek s LaTeXovým prostředím. Pokud bychom chtěli šablonu dokumentu, například tu v ukázce kódu 9, kompilovat pomocí pdfLaTeXu bez nahrazení zmíněných značek, program bude hlásit chybu, protože šablonovací značky nejsou platnými LaTeXovými makry. Některé šablonovací systémy nabízejí možnost změnit své značky. Toho využijeme pro vytvoření šablony s platnou LaTeXovou syntaxí.

V ukázce kódu 16 vidíme vytvoření konfigurace pro modul `jinja2` a následné použití nové šablony. Na následující ukázce kódu 15 vidíme šablonu pro tento příklad. Na začátek zdrojového LaTeXového souboru bylo nutné přidat definice prázdných maker `\VAR` a `\BLOCK`, které při kompilaci samotné šablony zamezí vzniku chyb. Tyto značky/makra budou přepsány našimi daty při použití šablony pro generaci zdrojového kódu pomocí skriptu v ukázce kódu 16. [62]

```
\documentclass{article}
% Definice prázdných maker, abychom mohli
% samotnou šablonu kompilovat bez chyb.
\newcommand{\VAR}[1]{}
\newcommand{\BLOCK}[1]{}
\begin{document}
\VAR{text}
\#{Ukázka komentáře}
\begin{itemize}
\BLOCK{for x in data}
    \item \VAR{x}
\#{Sem se vloží jednotlivé položky}
\BLOCK{endfor}
\end{itemize}
\end{document}
```

Kód 15: Rozšířená šablona pro modul `jinja2` [62]

Výsledný dokument při kompilaci šablony lze vidět na obrázku 14. Vygenerovaný zdrojový kód nám vytvoří stejný dokument jako v předchozích příkladech, viz obrázek 13. Komentáře v šabloně jsou systémem `jinja2` vymazány.

```
import os
import jinja2
# Vytvoření konfigurace pro modul jinja2 (nastavení značek).
LaTeX_env = jinja2.Environment(
    # Značky pro cykly a podmínky.
    block_start_string=r"\BLOCK{",
    block_end_string=r"}",
    line_statement_prefix=r"%-",
    # Značky pro proměnné.
    variable_start_string=r"\VAR{",
    variable_end_string=r"}",
    # Značky pro komentáře.
    comment_start_string=r"\#{",
    comment_end_string=r"}",
    line_comment_prefix=r"%#",
    # Odstraňování prázdných řádků.
    trim_blocks=True,
    # Ošetření pro XML/HTML speciální znaky.
    autoescape=False,
    # Nastavení výchozího umístění šablon.
    loader=jinja2.FileSystemLoader(os.path.abspath(".")),
)
# Vytvoření jinja2 šablony ze souboru.
template = LaTeX_env.get_template("_jinja2_env.tex")
# Vyrenderování šablony pomocí vložených dat.
result = template.render(
    text="Rozdělení ocelí podle použití:",
    data=["konstrukční", "nástrojové"]
)
# Uložení zdrojového kódu LaTeXové dokumentu do souboru.
with open("jinja2_env.tex", "w", encoding="utf8") as f:
    f.write(result)
```

Kód 16: Rozšířený příklad použití modulu `jinja2` [62]

#Ukázka komentáře

- #Sem se vloží jednotlivé položky

Obr. 14: Výsledný dokument vygenerovaný ze samotné rozšířené šablony

`jinja2` a další moduly nabízejí možnost spouštět Python funkce, někdy dokonce i celé bloky kódu. V takových případech už můžeme mluvit o generaci dokumentů z kombinovaného prostředí.

5.1.2 Specializované Python moduly

Kromě šablonovacích systémů můžeme využít specializované Python moduly, které řeší problematiku vytváření dokumentů pomocí Pythonu. Nejvýznamnějším je `pylatex` modul. Nabízí funkce, kterými můžeme volat libovolná makra, vkládat nadpisy i text a přidávat obrázky a grafy. Dokáže spolupracovat s populárními Python moduly `numpy` a `matplotlib`, které slouží pro matematické výpočty a vytváření grafů. `pylatex` hlídá potřebné balíčky maker a při generaci zdrojového kódu je automaticky umístí do souboru, aby kompilace výsledného dokumentu proběhla bez chyb. Po vytvoření zdrojového kódu dokumentu vyhledá `pylatex` dostupné programy pro kompilaci LaTeXových dokumentů a vytvoří výsledný `.pdf` soubor. Díky otevřenosti a přístupu ke zdrojovému kódu modulu je možné jej rozšiřovat o vlastní funkce. [63]

Na ukázce kódu 17 vidíme, jak vypadá Python skript pro vytvoření našeho vzorového dokumentu, viz obrázek 13. Vygenerový LaTeXový zdrojový kód je rozdílný od našich ručně vytvářených šablon. `pylatex` sám vložil potřebné balíky maker, v tomto případě hlavně `fontenc`, `inputenc` a `lmodern`, které zaručí správné vytváření znaků s diakritikou a zlepší vzhled použitého fontu. Obsah výsledného zdrojového LaTeXového souboru můžeme vidět v ukázce kódu 18.

```
import pylatex
# Definice našich dat.
text = "Rozdělení ocelí podle použití:"
data = ["konstrukční", "nástrojové"]
# Vytvoření pylatex dokumentu.
doc = pylatex.Document("pylatex")
# Vložení prostého textu.
doc.append(text)
# Vytvoření bodového výčtu.
with doc.create(pylatex.Itemize()) as itemize:
    # Vložení jednotlivých položek do výčtu.
    for x in data: itemize.add_item(x)
# Vygenerování .tex a .pdf souborů.
doc.generate_pdf(clean_tex=False)
```

Kód 17: Jednoduchý příklad využití modulu `pylatex`

```
\documentclass{article}%
\usepackage[T1]{fontenc}%
\usepackage[utf8]{inputenc}%
\usepackage{lmodern}%
\usepackage{textcomp}%
\usepackage{lastpage}%
\begin{document}%
\normalsize%
```

```
Rozdělení ocelí podle použití:%
\begin{itemize}%
\item%
konstrukční%
\item%
nástrojové%
\end{itemize}%
\end{document}
```

Kód 18: Zdrojový soubor vygenerovaný modulem `pylatex`

V ukázce kódu 19 se nachází příklad využití pokročilejších funkcí modulu `pylatex`. Vytváříme v něm tři kapitoly, jednu nečíslovanou a dvě číslované, matici a vektor pomocí Python modulu `numpy` a následně jednoduchý graf pomocí modulu `matplotlib.pyplot`. Následující výpis představí zajímavé funkce z modulu `pylatex`, které v ukázce používáme. Každá funkce nabízí řadu volitelných argumentů, kterými můžeme ovlivnit tvorbu LaTeXových prvků a celého dokumentu.

- `pylatex.Document(...)` – vytvoří Python objekt, který po čas běhu skriptu ponese informace o našem dokumentu,
- `pylatex.Section(...)` – vytvoří nadpis,
- `pylatex.utils.NoEscape(...)` – vloží do dokumentu text obsahující řídicí sekvence, například makro pro sazbu monofontem[18] `\texttt{...}`,
- `pylatex.Matrix(...)` – vytvoří LaTeXovou reprezentaci vektoru nebo matice,
- `pylatex.VectorName(...)` – vytvoří matematický symbol pro vektor nebo matici,
- `pylatex.Math(...)` – vytvoří LaTeXové prostředí pro sazbu matematických výrazů,
- `pylatex.Figure(...)` – vytvoří plovoucí blok pro vložení grafického materiálu. [63]

Pokud bychom udělali inspekci vygenerovaného LaTeXového kódu, všimli bychom si způsobu, kterým `pylatex` vložil graf do dokumentu. Při generaci uložil graf v `.pdf` formátu do dočasné složky[19] a do zdrojového kódu vložil makro `\includegraphics[...]{...}`, které na daný soubor odkazuje. Podle výchozího nastavení se daný soubor po kompilaci vymaže, takže není možné kompilovat zdrojový soubor LaTeXového dokumentu samostatně bez použití Python skriptu. Výsledný zkompilovaný dokument můžeme vidět na obrázku 15.

[18] Font, který má stejně široké znaky. Typický pro psací stroje nebo sazbu kódu.
[19] Například `C:/Users/.../AppData/Local/Temp/...` v operačním systému Windows 10.

```
import pylatex
import numpy
import matplotlib.pyplot
doc = pylatex.Document("pylatex2")
# Kapitola 0: Úvod
with doc.create(pylatex.Section("Úvod", numbering=False)):
    doc.append(pylatex.utils.NoEscape((
        r"Následující dokument ukáže možnosti modulů \texttt{pylatex},\
        \texttt{numpy} a \texttt{matplotlib.pyplot}.")))
# Kapitola 1: Matice a vektor
with doc.create(pylatex.Section("Matice a vektor")):
    mat = pylatex.Matrix(numpy.matrix([
        [2, 3, 4], [0, 0, 1], [0, 0, 2]]), mtype="b")
    mat_name = pylatex.VectorName("A")
    vec = pylatex.Matrix(numpy.array([[100, 10, 20]]).T, mtype="b")
    vec_name = pylatex.VectorName("v")
    result = pylatex.Matrix(mat.matrix * vec.matrix, mtype="b")
    doc.append(pylatex.utils.NoEscape(
        "Mějme matici {} a vektor {}.".format(
            pylatex.Math(inline=True, data=[mat_name]).dumps(),
            pylatex.Math(inline=True, data=[vec_name]).dumps())))
    doc.append(pylatex.Math(data=[mat_name, "=", mat]))
    doc.append(pylatex.Math(data=[vec_name, "=", vec]))
    doc.append("Jejich vynásobením dostaneme vektor:")
    doc.append(pylatex.Math(data=[mat_name, vec_name, "=", result]))
# Kapitola 2: Graf
with doc.create(pylatex.Section("Graf")):
    matplotlib.pyplot.plot([0,1,2,3,4,5,6], [15,2,7,1,5,6,9])
    matplotlib.pyplot.xlabel("Veličina $x$ [-]")
    matplotlib.pyplot.ylabel("Veličina $y$ [-]")
    with doc.create(pylatex.Figure(position="htbp")) as plot:
        plot.add_plot(
            width=pylatex.NoEscape(r"0.75\textwidth"),
            bbox_inches="tight",
            pad_inches=0)
        plot.add_caption(pylatex.utils.NoEscape("Průběh $y=f(x)$"))
doc.generate_pdf(clean_tex=False)
```

Kód 19: Pokročilá ukázka modulu `pylatex` [63]

Úvod

Následující dokument ukáže možnosti modulů `pylatex`, `numpy` a `matplotlib.pyplot`.

1 Matice a vektor

Mějme matici $\mathbf{A}$ a vektor $\mathbf{v}$.

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 4 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 100 \\ 10 \\ 20 \end{bmatrix}$$

Jejich vynásobením dostaneme vektor:

$$\mathbf{Av} = \begin{bmatrix} 310 \\ 20 \\ 40 \end{bmatrix}$$

2 Graf

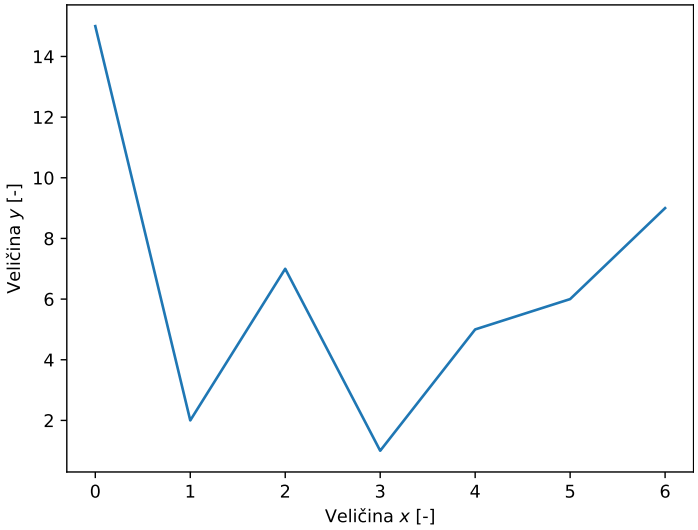


Figure 1: Průběh $y = f(x)$

Obr. 15: Výsledný dokument z pokročilé ukázky modulu `pylatex`

Jak lze z ukázky kódu 19 vidět, některé Python funkce operující s LaTeXem jsou zbytečně dlouhé a opakují se. Máme zde šanci a prostor pro sloučení často používaných funkcí do našich procedur, například pro operace s maticemi, které poté můžeme využívat i v ostatních projektech.

5.2 Generace z LaTeXového prostředí

Druhý způsob generace využívá balíčky maker, které nám dovolují psát bloky cizího kódu uvnitř LaTeXu a následně tento kód při kompilaci dokumentu spustit a uložit jeho výstup. Kompilaci dokumentu je většinou nutné provést několikrát za sebou. Při první kompilaci se posbírají příkazy a bloky kódu, které se uloží do externího souboru. Následně se zavolá program, který spustí posbíraný kód a zachytí jeho výstup. Při další kompilaci se využije uložený výstup a jeho jednotlivé části se vloží na správná místa do dokumentu tak, jak autor ve zdrojovém LaTeXovém souboru specifikoval. Pro vícefázové kompilace existuje nástroj `latexmk`, který podle konfiguračního souboru dokáže správně sřetězit více kompilací a spouštění programů. Do dokumentu je možné takto přidávat obecné řídicí sekvence, obyčejný text, čísla, rovnice, tabulky a grafy. Pokud blok kódu vygeneruje grafiku, musí se uložit do externího souboru a do zdrojového LaTeXové souboru vložit příkaz na její zahrnutí ve výsledném dokumentu.

Balíčky maker nabízející popsané funkce můžeme dělit do dvou základních skupin. Do první skupiny patří balíčky, které nabízejí spouštění externích Python skriptů. Ve druhé skupině jsou balíčky, které dovolují psát Python kód přímo ve zdrojovém LaTeXovém souboru. Následující výpis představí balíčky, které můžeme využít pro generaci obsahu pomocí Pythonu. Jejich vznik mohl být inspirován již existujícími nástroji `sweave` a `knitr`, které umožňují použití jazyka R z LaTeXových souborů pro výpočty a tvorbu grafů. [64, 65]

- `runcode` – nabízí makra pro spouštění libovolných externích programů a skriptů v různých programovacích jazycích (Python, Julia, R, Matlab), umožňuje zobrazovat barevný výpis kódu ze skriptu v dokumentu pomocí balíčku maker `minted`, komunikace s externími programy může být řešena pomocí Python modulu `talk2stat`,
- `sympytex` – starý projekt pro Python2 pro symbolické výpočty uvnitř LaTeXu pomocí Python modulu `sympy`,
- `python` – jednoduchý balíček pro spouštění bloků Python kódu s možností načítat obecný kód z externího pomocného souboru,
- `pythontex` – balíček obsahuje široký výběr maker a prostředí pro spouštění, substituci a zobrazování Python kódu, při kompilaci je kód spuštěn pouze v případě, že došlo k jeho změně od poslední kompilace, je možné použít i jiné jazyky, například Octave, Rust nebo JavaScript,
  - ⋆ `depythontex` – nástroj pro nahrazení Python kódu ve zdrojovém LaTeXovém souboru pro snadnější sdílení s ostatními lidmi, kteří dokument chtějí kompilovat bez nutnosti mít na počítači nainstalovaný Python interpret,
- `hybrid-latex`, `pylatex` – podobné jako `pythontex`, umožňuje v Python kódu vícekrát dosazovat do jedné proměnné a poté využít její historické hodnoty při

substitucích v LaTeXu, nabízí kromě Pythonu i spouštění kódu v programech Maple a Mathematica,

- `sagetex` – balíček pro propojení LaTeXu a balíčku matematických nástrojů SageMath, který je odnoží Pythonu a jeho výpočetních a vědeckých modulů,
- `pythonimmediate` – balíček na pozadí vytváří obousměrnou komunikaci mezi Pythonem a LaTeXem, je proto možné spustit Python kód a zahrnout jeho výstup do dokumentu během jednoho kompilačního cyklu, obousměrná komunikace také umožňuje vytvářet LaTeXová makra pomocí Python kódu,
- `pyluatex` – balíček, který během kompilace posílá Python kód do interaktivního Python interpretu, a umožňuje tak dokončení dokumentu během jednoho kompilačního cyklu, funguje pouze v programu LuaLaTeX. [66–74]

Některé z vypsaných balíčku maker lze použít při tvorbě dokumentu v online nástroji Overleaf, který má předinstalovaný Python interpret, spouští LaTeXový program s povolením volat externí příkazy a dovoluje využít vlastní konfigurační soubor pro `latexmk`. Overleaf omezuje délku trvání kompilace a je obtížné do něj nahrávat externí Python moduly, proto je vhodné kompilaci přesunout na vlastní počítač při výpočetně náročných operacích nebo potřebě využívat široké spektrum externích modulů. [75, 76]

Pokud bychom chtěli pro generaci využít nerozšířené programy nebo programovací jazyky, pro které nebyly vytvořeny konkrétní balíčky maker, můžeme využít již zmíněné příkazy `\write18{...}`, `\input{|...}` a nástroj `latexmk` a ručně propojit kompilaci dokumentu s naším externím programem.

Při využívání balíčků maker, které spouští kód pomocí externích programů, je nutné mít na paměti bezpečnost našeho systému a nekompilovat dokumenty obsahující bloky kódu z neověřených zdrojů. Útočník může do zdrojového LaTeXového souboru ukrýt nežádoucí kód, který se při kompilaci spustí.

5.2.1 Příklady použití

Na ukázce kódu 20 můžeme vidět příklad jednoduché generace dat pomocí Pythonu s využitím balíčku `python`. Mezi příkazy `\begin{python}...\end{python}` máme vložený Python kód, který chceme při kompilaci spustit a nahradit ho jeho výstupem. V tomto případě pomocí kódu vytváříme pole mocnin čísla 2 a pomocí funkce `print(...)` vkládáme výsledek do dokumentu. Výsledný dokument lze vidět na obrázku 16. [68]

```
\documentclass[12pt,border=5pt,varwidth]{standalone}
\usepackage{python} % Zde načítáme balíček maker.
\begin{document}
Data:
% Využití prostředí 'python' pro spuštění celého bloku kódu.
```

```
\begin{python} # Zde začíná Python kód.
result = []
for i in range(10):
    result.append(2**i)
print(result)
\end{python} % Zde končí Python kód.
\end{document}
```

Kód 20: Jednoduchá generace dat v LaTeXu pomocí balíčku `python`

Data: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

Obr. 16: Výsledek jednoduché generace dat v LaTeXu pomocí balíčku `python`

V další ukázce kódu 21 vytváříme podobný příklad pomocí balíčku `pythonimmediate`. Pro spuštění kódu můžeme využít makro `\py{...}` nebo prostředí ohraničené příkazy `\begin{pycode}...\end{pycode}`. V této ukázce zobrazujeme pomocí Pythonu i datum a čas při kompilaci. Výsledný dokument lze vidět na obrázku 17. [73]

```
\documentclass[12pt,border=5pt,varwidth]{standalone}
% Speciální nastavení balíčku pro spouštění v Overleafu.
\usepackage[abspath]{currfile}
\usepackage[
    python-executable={
        PYTHONPATH=pythonimmediate-tex-0.4.0.zip/%
        pythonimmediate-tex-0.4.0/
        python
    },
    args={--mode=unnamed-pipe}
]{pythonimmediate}
\begin{document}
% Využití makra '\py{..}' pro spuštění jednoho řádku kódu přímo v textu.
Data: \py{[2**i for i in range(10)]}\\
Datum:
% Využití prostředí 'pycode' pro spuštění celého bloku kódu.
\begin{pycode}
from datetime import datetime
from pythonimmediate import print_TeX
print_TeX(datetime.now())
\end{pycode}
\end{document}
```

Kód 21: Generace dat v LaTeXu pomocí balíčku `pythonimmediate`

Data: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
Datum: 2023-05-04 16:51:32.386149

Obr. 17: Výsledek generace dat v LaTeXu pomocí balíčku `pythonimmediate`

Balíček `pythonimmediate` nabízí také možnost definovat LaTeXová makra pomocí Python kódu. Příklad využití lze vidět na zkrácené ukázce kódu 22, kde v bloku Python kódu vytváříme funkci `sumManyArgs`, která se převede na odpovídající makro `\sumManyArgs...`. Funkcí makra je sečíst jednotlivá čísla vložená do složených závorek `{...}` za makrem. Výsledný dokument lze vidět na obrázku 18. [73]

```
...
\begin{pycode}
from pythonimmediate import (
    newcommand, peek_next_char, get_arg_str, print_TeX
)
@newcommand
# Definice nového makra pomocí speciálních funkcí
# 'peek_next_char' a 'get_arg_str', které nám
# dovolují pracovat s argumenty vloženými do makra.
def sumManyArgs(result = 0):
    # Smyčka přes všechny argumenty začínající složenou závorkou.
    while peek_next_char() == "{":
        # Převod argumentu na číslo a přičtení k celkovému součtu.
        result += int(get_arg_str())
    print_TeX(str(result))
\end{pycode}
...
% Použití makra '\sumManyArgs...' definovaného pomocí Pythonu.
$10 + 35 = \sumManyArgs{10}{35}$\\
$1 + 2 + 3 = \sumManyArgs{1}{2}{3}$
...
```

Kód 22: Definice LaTeXového makra v Pythonu pomocí balíčku `pythonimmediate`

$$10 + 35 = 45$$
$$1 + 2 + 3 = 6$$

Obr. 18: Výsledek použití makra z Pythonu pomocí balíčku `pythonimmediate`

V poslední ukázce kódu 23 lze vidět použití balíčku maker `pythontex` s Python modulem `sympy`. Kód je v ukázce vložen mezi příkazy `\begin{sympycode}...\end{sympycode}`. Toto prostředí vychází ze základního prostředí `pycode` a automaticky se v něm načítá Python modul `sympy`, pomocí kterého vytváříme symbolické proměnné a, b a roznásobujeme výraz $(a + b)^3$. V textu výsledek

vyvoláme pomocí makra `\sympy{...}`, které automaticky převede Python objekt uložený ve vložené proměnné na LaTeXovou reprezentaci matematického výrazu. Výsledný dokument lze vidět na obrázku 19. [69, 77]

```
\documentclass[12pt,border=5pt,varwidth]{standalone}
% Načtení potřebných balíčků.
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{pythontex}
\begin{document}
\begin{sympycode}
# Vytvoření symbolických proměnných.
a, b = symbols("a, b")
# Roznásobení výrazu '(a+b)^3'.
result = expand((a + b)**3)
\end{sympycode}
% Vypsání výsledku do dokumentu.
Výsledek: $\sympy{result}$.
\end{document}
```

Kód 23: Symbolické operace v LaTeXu pomocí balíčku `pythontex` a modulu `sympy` [77]

Výsledek: $a^3 + 3a^2b + 3ab^2 + b^3$.

Obr. 19: Výsledek symbolické operace pomocí balíčku `pythontex` a modulu `sympy`

6 Tvorba prototypu modulu

Pro modul jsem zvolil název Data2LaTeX, který vyjadřuje možnost modulu přenášet data uživatelů do LaTeXového formátu. Název modulu nemá v průběhu vývoje kolizi s žádným jiným názvem existujícího Python modulu nebo LaTeXového balíčku. Logo na obrázku 20 obsahuje název modulu a vytvoří se pomocí následujícího LaTeXového kódu: `$\Delta$\texttt{ata2}\LaTeX{}`.

Obr. 20: Logo modulu

6.1 Výběr technologií

Pro vytvoření modulu, který bude pomocí Python skriptu generovat a kompilovat LaTeXový zdrojový kód, jsem vybral následující technologie:

- Python 3.10,
- TeX – formát LaTeX, program `pdflatex` nebo `latexmk`,
- PyLaTeX – externí modul, který ulehčuje tvorbu a kompilaci LaTeXových dokumentů pomocí Pythonu,
- `tabularray` – balíček maker pro tvorbu tabulek pomocí LaTeXu,
- `pgfplots` – balíček maker pro tvorbu grafů pomocí LaTeXu.

Externí Python modul PyLaTeX byl zvolen kvůli otevřenosti kódu a svým datovým strukturám, pomocí kterých je možné reprezentovat základní LaTeXová makra a řídicí sekvence. Základní struktury a objekty lze skládat do větších celků pro vytvoření libovolného dokumentu. Umožňuje také automatickou správu potřebných balíčků maker pro jednotlivé objekty. PyLaTeX lze snadno rozšířit o nová uživatelem definovaná LaTeXová prostředí a makra, převážně díky autorovu využití objektově orientovaného programování při tvorbě modulu.

Tvorba tabulek většinou probíhá na straně LaTeXu pomocí balíčku maker, proto se výběr technologie omezil pouze na obsáhlé balíčky, ze kterých svými možnostmi nejvíce vystupoval právě vybraný `tabularray`. U grafů byl výběr technologie složitější, protože existuje bohatá nabídka projektů, které umožňují generovat grafiku z LaTeXu i z Pythonu. Pro svůj prototyp jsem zvolil generaci grafů pomocí LaTeXových prostředí nabízených v balíčku `pgfplots` z následujících důvodů:

- balíček maker `pgfplots` nabízí velké množství funkcí a předpřipravených grafů,

- text ve vygenerovaných grafech má stejný font a velikost jako ostatní text v dokumentu,
- legendy, anotace a popisy os a grafů mohou obsahovat LaTeXová makra a využívat načtené balíčky maker,
- LaTeXovou konstrukci grafu je možné umístit přímo do hlavního zdrojového `.tex` souboru, do externího souboru nebo graf zkompilovat separovaně do samostatného `.pdf` souboru a do hlavního dokumentu ho vložit jako hotový grafický objekt,
- v kombinaci s tabulkami vytvořenými pomocí `tabularray` je možné vygenerovat pouze jeden `.tex` soubor, který bude obsahovat všechna prezentovaná data.

6.2 Rozsah prototypu

Do prototypu Python modulu jsem se rozhodl implementovat následující funkce, které uživateli umožní tvorbu různorodých prezentací dat, a dovolí tak otestovat modul z funkční stránky i uživatelské přívětivosti.

- Základní prvky – tvorba nadpisů a textu, který může obsahovat speciální znaky pro využití LaTeXových maker, v této kategorii nebudou implementovány další funkce pro usnadnění formátování textu,
- tabulky – generace tabulek z dat uložených v dvourozměrných Python polích, případně v rozšířených datových strukturách `numpy.ndarray` a `pandas.DataFrame` z externích modulů, funkce budou nabízet možnosti pro hrubé formátování, například zarovnání textu a čísel v těle tabulky nebo zvýraznění textu v hlavičce,
- grafy – generace grafů obsahujících body a čáry pro prezentaci numerických dat uložených ve dvou separovaných polích pro souřadnice x a y, důležitá je možnost zobrazit více datových sérií s různým formátováním v jednom grafu, funkce budou nabízet větší počet možností pro formátování, například výběr barvy bodů a čar, použití lineárního a logaritmického měřítka nebo nastavení mřížky v těle grafu.

6.3 Založení projektu

Pro snadný vývoj externího Python modulu je nutné správně vytvořit strukturu projektu. Následující výčet obsahuje jednotlivé kroky, které byly podstoupeny pro založení projektu, nastavení důležitých parametrů a instalaci potřebných nástrojů.

1. Vytvoření repositáře, ve kterém bude probíhat vývoj modulu. Použitá struktura složek a souborů vychází z oficiální Python dokumentace.
   - `data2latex/` – vrchní složka projektu,
     - ⋆ `LICENSE` – textový soubor obsahující licenci, která nastavuje pravidla pro použití a šíření modulu – v mém případě svobodná licence MIT, která

neomezuje použití ani šíření za podmínky, že text licence bude distribuován společně s nástrojem využívajícím můj kód [78],

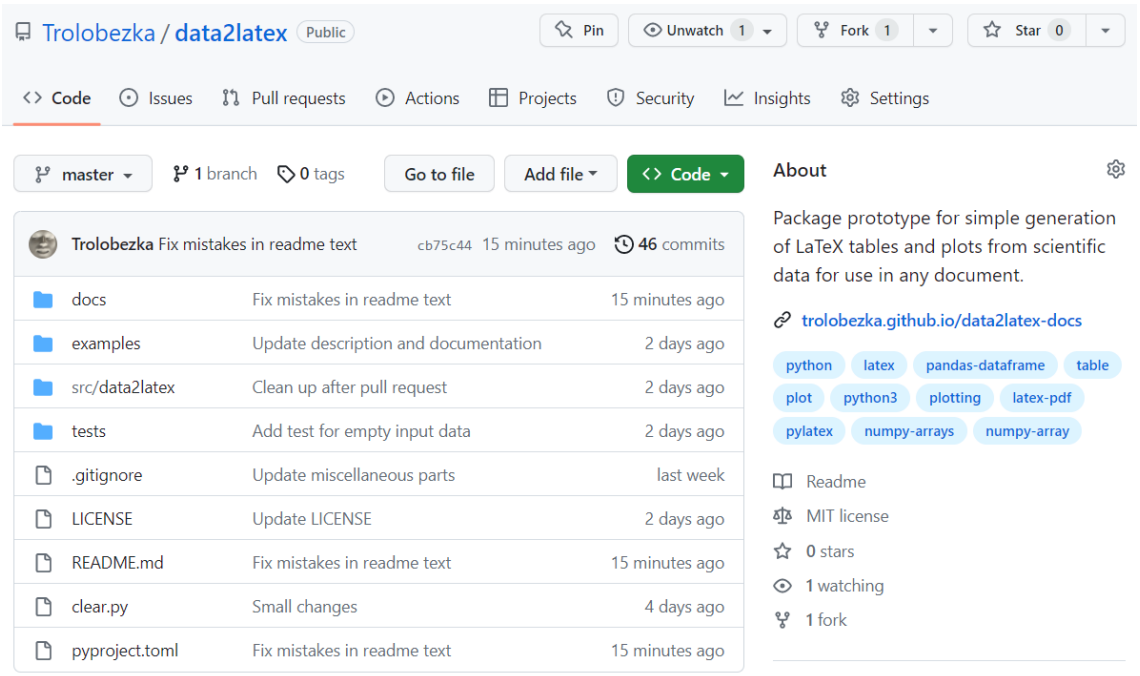
- ⋆ `README.md` – textový soubor ve formátu Markdown obsahující popis modulu, příklady využití, návod k instalaci a další užitečné informace pro uživatele,
- ⋆ `pyproject.toml` – textový soubor ve formátu TOML (Tom's Obvious Minimal Language) obsahující základní údaje o modulu, aby ho bylo možné nahrát do knihovny modulů PyPI, definují se zde například následující parametry: název, autor, verze, kategorie modulu a závislost na jiných modulech – v mém případě PyLaTeX, uvádí se zde také nástroj pro sestavení modulu do šiřitelného balíčku – v mém případě `setuptools`,
- ⋆ `src/data2latex/` – složka `src` s vnořenou složkou `data2latex` obsahující kód vyvíjeného modulu,
  - ∗ `__init__.py` – Python soubor, který ze složky vytváří importovatelný modul,
  - ∗ `...` – zbytek souborů, které tvoří funkčnost modulu, převážně Python skripty,
- ⋆ `tests/` – složka obsahující manuální i automatické skripty pro testování správné funkčnosti modulu.

2. Vytvoření virtuálního Python prostředí pro lokální instalaci potřebných externích modulů pro vývoj a testování. Virtuální prostředí je výhodné pro oddělení jednotlivých projektů, ve kterých potřebujeme využívat rozdílné verze Pythonu a instalovaných modulů. Pro tvorbu prostředí byl využit nástroj `venv`.
3. Instalace potřebných modulů pro následující činnosti:
   - vývoj – `pylatex`, `numpy`, `pandas`, `matplotlib`,
   - formátování kódu – `black`,
   - tvorba dokumentace – `sphinx`, `sphinx-rtd-theme`,
   - sestavení a publikace modulu – `build`, `twine`.
4. Nastavení nástroje `sphinx` pro generaci dokumentace v podobě webové stránky z poznámek a komentářů uložených v kódu. Pro vytvoření správné struktury a konfiguračních souborů slouží přidružený nástroj `sphinx-quickstart`. Obsah a vzhled dokumentace je definován v Python skriptu `./docs/conf.py`, `./docs/*.rst` souborech (ReStructured Text) a případně ve vlastních `./docs/_static/css/*.css` souborech (Cascading Style Sheets). Pro celkový vzhled webové stránky byl zvolen populární styl *Read the Docs*. [79, 80]

5. Vytvoření pomocných skriptů pro automatizaci častých úkonů. V repositáři lze najít dva takové skripty. První slouží pro mazání dočasných souborů po kompilaci LaTeXových dokumentů, generaci dokumentace a sestavení modulu. Druhý usnadňuje převod `.pdf` dokumentů do `.png` obrázků, které se posléze vkládají do dokumentace do kapitoly s příklady použití modulu.

6. Nastavení verzovacího systému `git`, pomocí kterého lze ukládat změny v kódu, vracet se ke starším verzím projektu a sdílet kód na Internetu pomocí neplacené webové služby GitHub. Protože modul vytvářím pod svobodnou licencí MIT, ostatní uživatelé mají možnost procházet kód, navrhovat změny a vytvářet své vlastní verze a nástavby. Náhled do repositáře projektu ve webovém rozhraní je možné vidět na obrázku 21.

Veškeré materiály, které budou v repositáři uloženy, včetně kódu (názvů proměnných a komentářů) a dokumentace, budou vytvářeny v anglickém jazyce. Jedná se o nepsaný standard, který ulehčuje sdílení kódu se zahraničními uživateli. Pro psaní kódu a správu repositáře budou používány aplikace Visual Studio Code a GitHub Desktop.

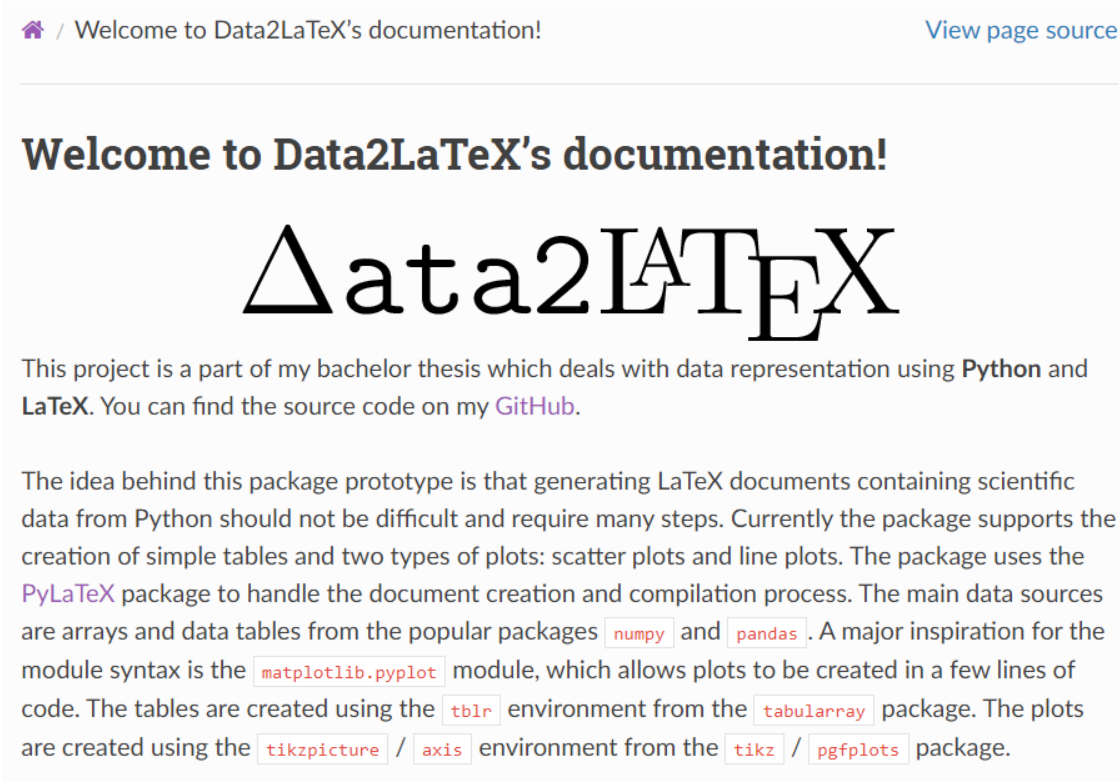


Obr. 21: Repositář ve webové aplikaci GitHub [81]

6.4 Průběh vývoje

Vývoj modulu začal studiem externího modulu PyLaTeX, na kterém je můj prototyp založený. Pro zjištění dostupných funkcí v modulu byla využita online dokumentace. Pro pochopení struktury objektů, které reprezentují LaTeXová prostředí a makra, bylo potřeba procházet zdrojový kód modulu. Společně s PyLaTeX modulem byla procházena i dokumentace k vybraným LaTeXovým balíčkům `tabularray` a `pgfplots`. Následující výčet popisuje postup vývoje a pořadí, ve kterém byly jednotlivé funkce modulu vytvářeny.

1. Vytvoření repositáře a nastavení nástrojů.
2. Vytvoření objektu `DocumentManager`, který spravuje instanci LaTeXového dokumentu a umožňuje k němu přistupovat z ostatních částí kódu. Tento objekt byl inspirován fungováním Python modulu `matplotlib.pyplot`, ve kterém existují podobné objekty pro správu grafů a jejich jednotlivých částí.
3. Vytvoření základních funkcí pro tvorbu dokumentu, vkládání nadpisů a textu, generaci `.tex` souborů a kompilaci `.pdf` dokumentů.
4. Vývoj funkce `table(...)` pro generaci tabulek. Po vzoru PyLaTeX objektů pracujících se základním tabulkovým prostředím `tabular` byl vytvořen Python objekt `tblr`, který reprezentuje stejně pojmenované LaTeXové prostředí z balíčku maker `tabularray`.
5. Vývoj funkce `plot(...)` pro generaci bodových a čárových grafů. Pro implementaci byly využity již existující PyLaTeX objekty pro práci s prostředími `tikzpicture` a `axis` z LaTeXových balíčků `tikz` a `pgfplots`.
6. Rozšíření možností objektu `DocumentManager` pro umožnění generace dokumentů pomocí LaTeXové třídy `standalone`. V dokumentu využívajícím tuto třídu se velikost stránek řídí velikostí jejich obsahu. Výsledkem je dokument s ořezanými stránkami, které je možné vkládat do jiných dokumentů jako grafický materiál.
7. Manuální testování modulu a tvorba ukázkových příkladů do dokumentace. Při tomto procesu došlo k objevení chyb, které byly následně opraveny.
8. Vytvoření textu, příkladů a obrázků pro úvodní stránku dokumentace. Spuštěním nástroje `sphinx` byla poté vygenerována složka obsahující zdrojové soubory webové stránky. Pomocí nástroje `git` a služby GitHub byl založen další repositář, do kterého se uložila vygenerovaná dokumentace. GitHub nabízí možnost publikovat webové stránky uložené ve svých repositářích, čehož zde bylo využito, a dokumentace modulu dostala veřejnou internetovou adresu na doméně `github.io`. Ukázka dokumentace lze vidět na obrázcích 22 a 23.
9. Založení účtu ve službě PyPI a následné sestavení, zabalení a publikace modulu pod názvem `data2latex` s odkazy na repositář s kódem uloženým ve službě GitHub a online vygenerovanou dokumentaci. [82]
10. Stažení a instalace modulu přes nástroj `pip` a otestování funkčnosti publikované verze modulu na jiném počítači. Při testování byly nalezeny další chyby v kódu, které se opravily, a modul byl znovu publikován s vyšším číslem verze.



/ Welcome to Data2LaTeX's documentation! View page source

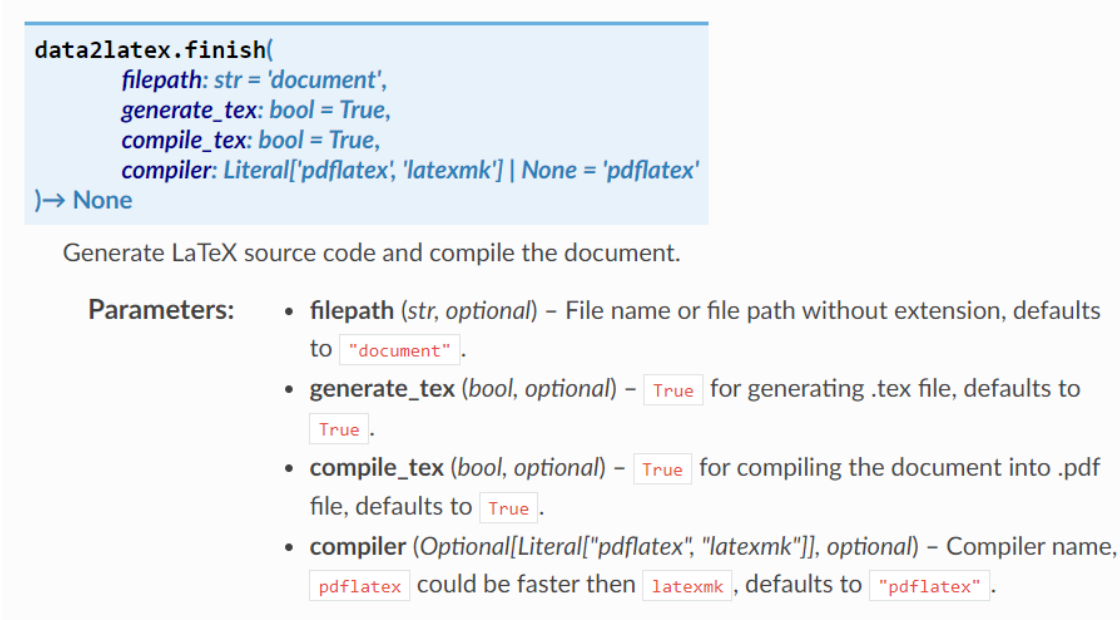
Welcome to Data2LaTeX's documentation!

Data2LaTeX

This project is a part of my bachelor thesis which deals with data representation using **Python** and **LaTeX**. You can find the source code on my GitHub.

The idea behind this package prototype is that generating LaTeX documents containing scientific data from Python should not be difficult and require many steps. Currently the package supports the creation of simple tables and two types of plots: scatter plots and line plots. The package uses the PyLaTeX package to handle the document creation and compilation process. The main data sources are arrays and data tables from the popular packages `numpy` and `pandas`. A major inspiration for the module syntax is the `matplotlib.pyplot` module, which allows plots to be created in a few lines of code. The tables are created using the `tblr` environment from the `tabularray` package. The plots are created using the `tikzpicture` / `axis` environment from the `tikz` / `pgfplots` package.

Obr. 22: Úvod online dokumentace [83]



```
data2latex.finish(
        filepath: str = 'document',
        generate_tex: bool = True,
        compile_tex: bool = True,
        compiler: Literal['pdflatex', 'latexmk'] | None = 'pdflatex'
)→ None
```

Generate LaTeX source code and compile the document.

Parameters:

- **filepath** (*str, optional*) – File name or file path without extension, defaults to `"document"`.
- **generate_tex** (*bool, optional*) – `True` for generating .tex file, defaults to `True`.
- **compile_tex** (*bool, optional*) – `True` for compiling the document into .pdf file, defaults to `True`.
- **compiler** (*Optional[Literal["pdflatex", "latexmk"]], optional*) – Compiler name, `pdflatex` could be faster then `latexmk`, defaults to `"pdflatex"`.

Obr. 23: Příklad dokumentace pro funkci z modulu [83]

7 Návod k použití modulu

V následující kapitole bude vysvětleno základní použití modulu pro vytvoření ukázkové tabulky a grafu. Tvorba složitějšího dokumentu je ukázána v příloze I.

7.1 Instalace

Pro nainstalování modulu je důležité mít na počítači nainstalovaný Python interpret verze 3.10 nebo novější. Modul byl vytvářen ve verzi 3.10. Novější verze 3.11 byla také otestována. Pokud chceme kompilovat dokumenty do `.pdf` formátu, je potřeba mít nainstalovaný TEXový program `pdflatex` a nástroj `latexmk`. Bez TEXového programu bude možné pouze generovat zdrojové `.tex` soubory. Uvedené programy jsou součástí kompletních distribucí systému TEX, například MiKTEX. Pro zjištění dostupnosti potřebných programů můžeme zadat následující příkazy do příkazové řádky, které nám vypíší informace o nainstalovaných verzích daných programů.

```
> python --version
> pdflatex --version
> latexmk --version
> miktex --version
```

Je také vhodné zkontrolovat aktuálnost nainstalovaných balíčků LATEXových maker, abychom předešli kompilačním chybám. U distribuce MiKTEX se pro toto používá aplikace MiKTeX Console. [84]

Modul nainstalujeme pomocí Python nástroje `pip`, který stáhne soubory z online archivu PyPI a uloží je do složky externích Python modulů. Využijeme pro to příkazový řádek, ve kterém nejprve aktualizujeme `pip` a poté nainstalujeme modul `data2latex`. Potřebné příkazy jsou uvedeny na následujících řádcích.

```
> python -m pip install --upgrade pip
> python -m pip install data2latex
```

Pro zobrazení informací o nainstalovaném modulu můžeme využít funkci `show` nástroje `pip`. Vypsané údaje budou obsahovat například verzi modulu a jméno autora.

```
> python -m pip show data2latex
```

7.2 Tvorba tabulky

Pro generaci tabulky vytvoříme soubor pro Python skript `tabulka.py`, který otevřeme v libovolném textovém editoru. Hned na prvním řádku skriptu musíme importovat modul pomocí kódu `import data2latex as dtol`, abychom mohli využívat jeho funkce. Modul `data2latex` je pro tento skript přejmenován na kratší název `dtol`, který se rychleji píše.

Obsah tabulky bude tvořen ukázkovými daty, která budou obsahovat interval celých čísel od 1 do 9 rozdělený na tři řádky. Čísla budou uložena v polích, které reprezentují jednotlivé řádky. Řádky budou dále vloženy do dalšího pole, které reprezentuje celé tělo tabulky. Pomocí kódu `data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]` vytvoříme popsané dvourozměrné číselné pole.

Pro vytvoření tabulky stačí zavolat funkci `table(...)` z modulu `data2latex`, viz kód `dtol.table(data)`, a předat jí naše data. Výchozí nastavení vygeneruje tabulku vycentrovanou na stránce dokumentu, s vycentrovanými čísly v políčkách a bez mřížky.

Na závěr vygenerujeme LaTeXový zdrojový kód a zkompilujeme dokument do `.pdf` formátu pomocí kódu `dtol.finish("tabulka")`. Do funkce `finish(...)` vložíme název pro vygenerované soubory. Python skript můžeme spustit pomocí příkazové řádky a příkazu `python <cesta ke skriptu>`. Pro skript uložený na ploše v operačních systému Windows můžeme použít následující tvar příkazu.

```
> python C:/Users/.../Desktop/tabulka.py
```

Pokud skript proběhne bez chyb, vytvoří se v adresáři, ze kterého příkazy spouštíme, dva soubory: `tabulka.tex` a `tabulka.pdf`. V `.pdf` dokumentu bychom měli najít stejnou tabulku jako na obrázku 24. Celý kód z tohoto příkladu je uveden v ukázce kódu 24.

1	2	3
4	5	6
7	8	9

Obr. 24: Tabulka vygenerovaná modulem `data2latex`

```
import data2latex as dtol
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
dtol.table(data)
dtol.finish("tabulka")
```

Kód 24: Skript pro generaci tabulky modulem `data2latex`

7.3 Tvorba grafu

Při tvorbě grafu budeme postupovat stejně jako v předchozím příkladě. Založíme soubor `graf.py`, otevřeme ho a začneme vytvářet ukázkový Python skript.

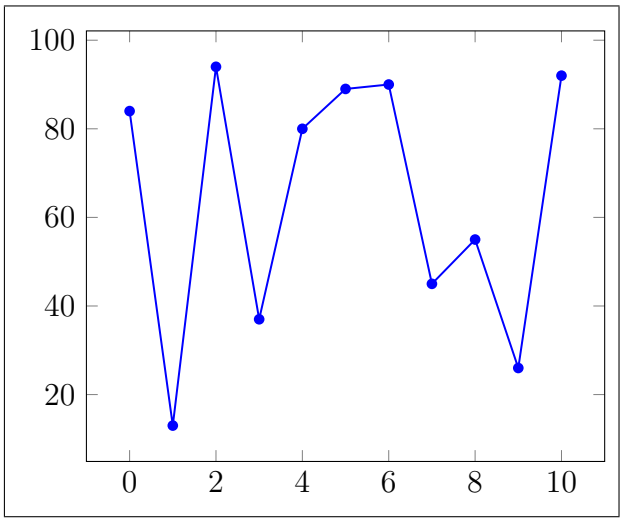
Na začátku skriptu opět importujeme modul pomocí kódu `import data2latex as dtol` a nastavíme LaTeXovou třídu dokumentu na `standalone` třídu zavoláním funkce

`dtol.use_multi_page_standalone()`. Toto nastavení není povinné a je zde pouze pro ukázku tvorby přesně ořezaných grafů na samostatných stránkách pro využití v jiných dokumentech.

Vytvoříme dvě číselná pole pro souřadnice x a y bodů, které chceme v grafu zobrazit. Do prvního pole vložíme interval celých čísel od 0 do 10 pomocí kódu `X = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`. Do druhého pole vložíme kódem `Y = [84, 13, 94, 37, 80, 89, 90, 45, 55, 26, 92]` stejný počet náhodných celých čísel.

Pro tvorbu grafu využijeme funkci `plot(...)`, do které vložíme naše data a nastavíme základní styl bodů a spojnicové čáry. Pomocí kódu `dtol.plot(X, Y, line="-", mark="*")` vytvoříme čárový graf se souvislou spojnicí a zvýrazněnými body. Výchozí barva obou grafických objektů je sytě modrá.

Skript dokončíme kódem `dtol.finish("graf")`. Po úspěšném spuštění skriptu by se měly vytvořit dva soubory: `graf.tex` a `graf.pdf`. Vytvořený graf lze vidět na obrázku 25. Celý kód z tohoto příkladu je uveden v ukázce kódu 25.



Obr. 25: Graf vygenerovaný modulem `data2latex`

```
import data2latex as dtol
dtol.use_multi_page_standalone()
X = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Y = [84, 13, 94, 37, 80, 89, 90, 45, 55, 26, 92]
dtol.plot(X, Y, line="-", mark="*")
dtol.finish("graf")
```

Kód 25: Skript pro generaci grafu modulem `data2latex`

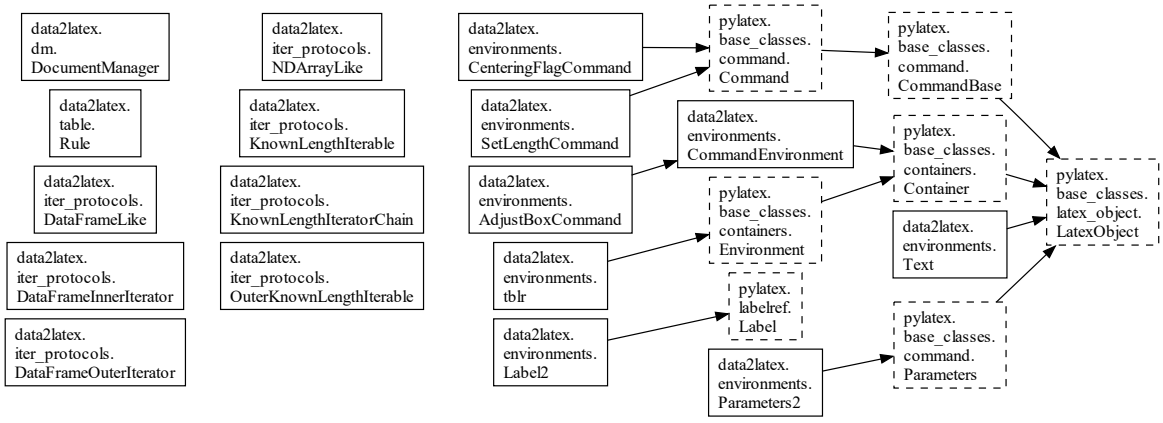
8 Dokumentace hlavních funkcí modulu

Prototyp modulu uživateli nabízí následující funkce:

- `gd` a `gdm` – přístup k objektům, které obsahují data dokumentu,
- `setup`, `use_multi_page_standalone` a `use_one_page_standalone` – nastavení parametrů ovlivňujících generaci dokumentu,
- `section` a `text` – tvorba základních LaTeXových prvků,
- `table` – tvorba tabulek,
- `plot` – tvorba bodových a čárových grafů,
- `finish`, `latex`, `pdf` a `reset` – generace zdrojového souboru a kompilace dokumentu.

Uvedené funkce jsou popsané v online dokumentaci a uživatelé zde mohou zjistit, které vstupní parametry mohou funkcím předat a jaké jsou jejich výchozí hodnoty. V této kapitole budou popsány hlavní a zároveň nejrozsáhlejší funkce `table(...)` a `plot(...)`. V modulu bychom našli i další pomocné funkce, které ovšem nejsou uživateli na první pohled přístupné.

Na obrázku 26 lze vidět hierarchii tříd definovaných v modulu. Nalevo jsou umístěné pomocné třídy a protokoly pro práci s datovými strukturami. Napravo je umístěn diagram ukazující vztahy mezi třídami z modulů `data2latex` a `pylatex`. Většina tříd vychází ze základní třídy `LatexObject` definované modulem `pylatex`, která obsahuje základní proměnné a metody potřebné pro prezentaci všech LaTeXových prvků.



Obr. 26: Hierarchie tříd v modulu

8.1 Funkce `data2latex.table(...)`

Funkce `table(...)` slouží pro generaci tabulek ze vstupních dat, které mohou být ve formě dvourozměrného Python nebo `numpy` pole nebo datové struktury `DataFrame` z modulu `pandas`, která připomíná klasickou tabulku dat. Vygenerovaný výstup z funkce je založený na balíčkách maker `tabularray`, `siunitx`, `adjustbox` a `float`. Při generaci zdrojového `.tex` souboru se automaticky vloží příkazy na načtení zmíněných balíčků pro zajištění bezproblémové kompilace. V následujícím seznamu jsou popsány všechny vstupní parametry funkce `table(...)`.

`data` — Jediný povinný parametr, přes který se do funkce vkládají data pro tvorbu tabulky.

`rules = ""` — Speciální kód pro nastavení svislých a vodorovných čar v tabulce. V kódu se využívají symboly `"|"` a `"_"` pro vybrání směru, ve kterém chceme čáry nastavovat, a za nimi následují symboly modifikující čáry vybraného směru. Pro modifikace lze použít symboly `"1"`, `"2"`, `"3"`, `"A"`, `"a"`, `"B"`, `"b"`, `"#"`, `"O"` a `"o"`. Užitečný je například symbol `"#"`, který do tabulky přidá všechny svislé i vodorovné čáry a vytvoří typickou tabulku s plnou mřížkou.

`caption = None` — Text vložený do popisku tabulky.

`caption_pos = "above"` — Výběr umístění popisku nad (`"above"`) nebo pod (`"below"`) tabulku.

`escape_caption = True` — Nastavení pro nahrazování speciálních LaTeXových symbolů v popisku tabulky, které by mohly způsobovat kompilační chyby. Pokud chceme v popisku využít LaTeXová makra, musíme do tohoto parametru vložit hodnotu `False`.

`label = None` — Označení, pomocí kterého se můžeme v textu na tuto tabulku odkazovat. Může být ve formátu `"typ:označení"` nebo pouze `"označení"` a typ bude automaticky nastaven na hodnotu `"table"`.

`position = "H"` — Nastavení umístění tabulky na stránce v dokumentu. Povolené hodnoty vycházejí z nastavení LaTeXového plovoucího prostředí `table`.

`center = True` — Nastavení vodorovného vycentrování tabulky na stránce dokumentu.

`float_format = "{:0.3f}"`	Formát pro převod desetinných čísel na textové řetězce. Způsob zápisu formátu vychází z použité Python funkce `str.format(...)`. Výchozí hodnota zajistí zobrazení přesně tří desetinných míst.
`str_format = "{{{{{{{:s}}}}}}}"`	Formát pro ošetření textových políček v tabulce. Pokud se pro zarovnání čísel využívá balíček `siunitx`, je potřeba všechen text v políčkách vložit do trojice složených závorek. Formát je využíván funkcí `str.format(...)`.
`str_convertor = str`	Funkce pro převod neznámých objektů na textové řetězce. Výchozí hodnota odkazuje na základní Python funkci `str(...)`.
`str_try_number = True`	Nastavení, které ovlivňuje počítání číslic v jednotlivých políčkách tabulky. Pokud bude vložena hodnota `True`, funkce se bude snažit převést všechny textové řetězce na čísla a určit jejich šířku, která je důležitá pro správnou funkci balíčku `siunitx`. Toto nastavení dovoluje mít ve vstupních datech uložená čísla ve formě textových řetězců.
`escape_cells = True`	Nastavení pro nahrazování speciálních LaTeXových symbolů v jednotlivých políčkách tabulky.
`col_align = "c"`	Nastavení vodorovného zarovnání obsahu v políčkách tabulky. Dostupné jsou obvyklé hodnoty `"l"`, `"c"` a `"r"`, které obsah zarovnají k levému okraji, na střed nebo k pravému okraji.
`row_align = "m"`	Nastavení svislého zarovnání obsahu v políčkách tabulky. Dostupné jsou obvyklé hodnoty `"t"`, `"m"` a `"b"`, které obsah zarovnají k hornímu okraji, na střed nebo k dolnímu okraji.
`left_head_bold = False`	Nastavení pro zvýraznění políček v prvním sloupci tabulky, který může sloužit jako postranní hlavička tabulky.
`left_head_col_align = None`	Nastavení vodorovného zarovnání obsahu políček v prvním sloupci tabulky.
`top_head_bold = False`	Nastavení pro zvýraznění políček v prvním řádku tabulky, který může sloužit jako hlavička tabulky.
`top_head_col_align = None`	Nastavení vodorovného zarovnání obsahu políček v prvním řádku tabulky.

`use_adjustbox = True` Nastavení pro ochranu proti přetečení tabulky přes šířku stránky dokumentu. Tabulka se vkládá do prostředí `adjustbox`, které dovoluje nastavit maximální šířku obsahu. Výsledkem je případné zmenšení tabulky tak, aby nepřekročila šířku stránky.

`use_siunitx = True` Nastavení vodorovného zarovnávání desetinných čísel podle desetinné tečky. Tabulkové prostředí pro tuto funkci využívá balíček `siunitx`, který pro správnou funkci potřebuje údaje o počtu číslic v každém sloupci tabulky. Tyto informace se zjišťují při tvorbě jednotlivých políček tabulky heuristickým postupem, ve kterém se počítají číslice ve výsledných textových řetězcích vkládaných do těla tabulky.

`dataframe_column_names = True` Nastavení umožňující převzetí hlaviček sloupců z dat ve formátu `pandas.DataFrame`, ve kterém lze pojmenovat jednotlivé sloupce.

`dataframe_row_names = True` Nastavení umožňující převzetí hlaviček řádků z dat ve formátu `pandas.DataFrame`, ve kterém lze pojmenovat jednotlivé řádky.

8.2 Funkce `data2latex.plot(...)`

Funkce `plot(...)` slouží pro generaci bodových a čárových grafů ze vstupních dat obsahujících souřadnice bodů. Data musí být rozdělena do dvou polí tak, aby první pole obsahovalo x-ové souřadnice bodů a druhé pole zase ty y-ové. Pokud v grafu chceme zobrazit více datových sérií, můžeme vložit pole souřadnic jednotlivých sérií do nového pole, které poté předáme funkci. Styl zobrazení jednotlivých sérií je nastaven pomocí cyklických polí, která jsou použita například pro barvu, průhlednost a typ spojnic a bodů. Vygenerovaný výstup z funkce je založený na balíčkách maker `tikz`, `pgfplots` (s rozšířením `plotmarks`) a `float`. V následujícím seznamu jsou popsány všechny vstupní parametry funkce `plot(...)`.

`_X` První povinný parametr, do kterého se vkládá pole nebo pole polí x-ových souřadnic bodů.

`_Y` Druhý povinný parametr, do kterého se vkládá pole nebo pole polí y-ových souřadnic bodů.

`caption = None` Text vložený do popisku grafu.

`xlabel = None` Text vložený do popisku x-ové osy. V textu mohou být použita makra a matematické výrazy.

`ylabel = None` — Text vložený do popisku y-ové osy. V textu mohou být použita makra a matematické výrazy.

`grid = None` — Speciální kód pro nastavení svislé a vodorovné mřížky v těle grafu. V kódu se využívají symboly `"|"`, `"_"` a `"#"` pro přidání hlavních svislých a vodorovných čar. Pokud chceme prostor mezi hlavními čarami rozdělit a vytvořit vedlejší mřížku, můžeme za uvedené symboly vložit celé číslo v intervalu 1 až 9, které určí počet přidaných čar.

`mode = ("lin", "lin")` — Nastavení použitého měřítka pro jednotlivé osy. První hodnota určuje měřítko x-ové osy, druhá hodnota zase té y-ové. Můžeme zvolit lineární měřítko hodnotou `"lin"` nebo logaritmické měřítko hodnotou `"log"`.

`legend = None` — Parametr pro vložení pole textových řetězců, které představují jednotlivé položky v legendě.

`legend_pos = "top right"` — Nastavení umístění legendy vzhledem k tělu grafu. Dovolené hodnoty odpovídají hodnotám z balíčku `pgfplots`. Nejdříve se v textovém řetězci uvádí svislá pozice, poté ta vodorovná.

`legend_entry_align = "c"` — Nastavení vodorovného zarovnání textu jednotlivých položek v legendě.

`width = None` — Šířka grafu společně s uvedenou LaTeXovou jednotkou.

`height = None` — Výška grafu společně s uvedenou LaTeXovou jednotkou.

`equal_axis = False` — Nastavení stejného měřítka pro obě osy. Hodnota `True` zaručí, že prezentovaná data nebudou roztažena nebo smrštěna kvůli rozdílným rozpětím na jednotlivých osách.

`xlimits = None` — Krajní limity x-ové osy ve tvaru `(min, max)`. Platná je i hodnota `"exact"`, která zaručí automatické určení limitů podle extrémů vložených dat.

`ylimits = None` — Krajní limity y-ové osy ve tvaru `(min, max)`. Platná je i hodnota `"exact"`, která zaručí automatické určení limitů podle extrémů vložených dat.

`precision = (2, 2)` — Nastavení počtu desetinných míst zobrazených při popisu značek na osách. První číslo určuje přesnost pro popisky x-ové osy, druhé pro popisky y-ové osy.

`zerofill = (False, False)` — Nastavení pro doplnění desetinného rozvoje nulami pro čísla přesnější než nastavená přesnost na dané ose. Parametr očekává dvě hodnoty pro nastavení x-ové a y-ové osy.

`label = None` — Označení, pomocí kterého se můžeme v textu na tento graf odkazovat. Může být ve formátu `"typ:označení"` nebo pouze `"označení"` a typ bude automaticky nastaven na hodnotu `"plot"`.

`caption_pos = "below"` — Výběr umístění popisku nad (`"above"`) nebo pod (`"below"`) graf.

`escape_caption = True` — Nastavení pro nahrazování speciálních LaTeXových symbolů v popisku grafu.

`position = "H"` — Nastavení umístění grafu na stránce v dokumentu. Povolené hodnoty vycházejí z nastavení LaTeXového plovoucího prostředí `figure`.

`center = True` — Nastavení vodorovného vycentrování tabulky na stránce dokumentu.

`line = None` — Nastavení stylu spojnice bodů. Dostupné hodnoty vycházejí z dokumentace balíčku `pgfplots`. Čtyři hlavní hodnoty jsou `"-"` pro plynulou spojnici, `"--"` pro čárkovanou spojnici, `"."` pro tečkovanou spojnici a `"-."` pro čerchovanou spojnici. Hodnota `None` zamezí vykreslování spojnice. Do parametru může být vložen jeden textový řetězec nebo pole, ze kterého bude vytvořeno cyklické pole.

`line_width = "0.75pt"` — Šířka spojnice společně s uvedenou LaTeXovou jednotkou. Do parametru je možné dosadit pole hodnot.

`line_color = ["blue", "red", "black"]` — Barva spojnice ve tvaru trojice celých čísel v intervalu 0 až 255, trojice desetinných čísel v intervalu 0.0 až 1.0, textový řetězec obsahující název definované barvy z balíčku `xcolor`, nebo šestimístný kód webové barvy začínající znakem `"#"`. Do parametru je možné dosadit pole hodnot. Výchozí hodnota je nastavena na cyklické pole, které opakuje modrou, červenou a černou barvu.

`line_opacity = 1.0` — Průhlednost spojnice v rozsahu od 0.0 pro plnou průhlednost do 1.0 pro plnou viditelnost. Do parametru je možné dosadit pole hodnot.

`mark = "*"` — Nastavení stylu bodů. Dostupné hodnoty vycházejí z dokumentace balíčku `pgfplots`. Některé styly vytvářejí pouze obrysy, jejichž vlastnosti musí být nastaveny pomocí parametrů `mark_stroke_*`. Hodnota `None` zamezí vykreslování bodů. Do parametru je možné dosadit pole hodnot.

`mark_size = "2pt"`	Velikost bodů společně s uvedenou LaTeXovou jednotkou. Do parametru je možné dosadit pole hodnot.
`mark_fill_color = ["blue", "red", "black"]`	Barva výplně bodů. Platí zde stejná pravidla jako pro parametr `line_color`.
`mark_stroke_color = None`	Barva obrysu bodů. Platí zde stejná pravidla jako pro parametr `line_color`.
`mark_fill_opacity = 1.0`	Průhlednost výplně bodů. Platí zde stejná pravidla jako pro parametr `line_opacity`.
`mark_stroke_opacity = 0.0`	Průhlednost obrysu bodů. Platí zde stejná pravidla jako pro parametr `line_opacity`.

9 Závěr

Prvním cílem práce bylo provést rešerši způsobů prezentace dat a možností generace LaTeXových dokumentů pomocí programovacího jazyka Python. Obě rešerše byly provedeny a výsledky, na kterých později zakládám své praktické řešení, byly sepsány do úvodních kapitol této práce.

Druhým cílem práce bylo navrhnout a implementovat prototyp Python modulu, který usnadní generaci vybraných typů prezentace dat. Modul byl navržen po vzoru externích modulů `matplotlib.pyplot` a `pylatex`. Funkčnost modulu jsem omezil na generaci tabulek a bodového a čárového grafu s dostatečným množstvím uživatelského přizpůsobení, aby byl modul použitelný pro skutečné dokumenty. Pro implementaci prototypu byl využit modul `pylatex`, který již nabízel funkce a objekty pro práci s LaTeXem.

Praktickým výsledkem práce je prototyp Python modulu, který byl publikován v archivu modulů PyPI pod názvem `data2latex`. Zdrojový kód modulu je uložený ve veřejném repositáři pomocí služby GitHub pod svobodnou licencí MIT. Pro modul byla vygenerována dokumentace v podobě webové stránky, do které byly ručně vytvořeny příklady použití implementovaných funkcí `table(...)` a `plot(...)`. Další vývoj a práce na modulu se mohou zaměřit na:

- přidání funkcí pro tvorbu dalších základních druhů grafů,
- rozšíření funkcí o další nastavení stylu a formátování generovaných objektů,
- vytvoření automatizovaných testů pro důležité funkce v modulu,
- zamyšlení se nad možností rozšířit existující modul `pylatex` o funkce z tohoto prototypu,
- rozšíření dokumentace o další názorné příklady použití modulu,
- získání názoru na funkčnost a uživatelskou přívětivost modulu od uživatelů využívajících ke své práci kombinaci Pythonu a LaTeXu.

Seznam použité literatury a zdrojů

1. FEW, Stephen. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. 2. vyd. Burlingame (California): Analytics Press, 2012. ISBN 978-0-9706019-7-1.
2. HARRIS, Robert. *Information Graphics: A Comprehensive Illustrated Reference*. Atlanta (Georgia): Management Graphics, 1999. ISBN 0-19-513532-6.
3. TUFTE, Edward. *The Visual Display of Quantitative Information*. 2. vyd. Cheshire (Connecticut): Graphics Press, 2007. ISBN 978-1930824133.
4. *Matplotlib 3.7.1 documentation: Plot types* [online]. c2012–2023. [cit. 2023-05-26]. Dostupné z: https://matplotlib.org/stable/plot_types/index.html.
5. *Podpora Microsoftu: Dostupné typy grafů v Office* [online]. c2023. [cit. 2023-05-26]. Dostupné z: https://support.microsoft.com/cs-cz/office/dostupn%C3%A9-typy-graf%C5%AF-v-office-a6187218-807e-4103-9e0a-27cdb19afb90.
6. *Volba typu grafu* [online]. [cit. 2023-05-26]. Dostupné z: https://help.libreoffice.org/latest/cs/text/schart/01/choose_chart_type.html.
7. DOURMASHKIN, Peter; HUDSON, Eric; LIAO, Sen-Ben. Introduction to TEAL; Fields; Review of gravity; Electric field. In: *Physics II: Electricity And Magnetism* [online]. Massachusetts, 2007 [cit. 2023-05-26]. Dostupné z: https://ocw.mit.edu/courses/8-02-physics-ii-electricity-and-magnetism-spring-2007/resources/chapter1fields.
8. ŠLEGER, Vladimír; VRECION, Pavel. *Mathcad: příručka k matematickému programu Mathcad 7* [online]. 2005. [cit. 2023-05-26]. Dostupné z: http://plarmy.org/cad/mcadprirucka/PriruckaMCAD.pdf.
9. ROSSUM, Guido. *Guido's Personal Home Page* [online]. [cit. 2023-05-18]. Dostupné z: https://gvanrossum.github.io.
10. *Python Software Foundation: About the Python Software Foundation* [online]. c2001-2023. [cit. 2023-05-18]. Dostupné z: https://www.python.org/psf/about.
11. *About Python* [online]. c2012-2022. [cit. 2023-05-18]. Dostupné z: https://pythoninstitute.org/about-python.
12. *Python 3.11.3 documentation: 1. Whetting Your Appetite* [online]. c2001-2023. [cit. 2023-05-18]. Dostupné z: https://docs.python.org/3/tutorial/appetite.html.
13. *Python 3.11.3 documentation: General Python FAQ* [online]. c2001-2023. [cit. 2023-05-18]. Dostupné z: https://docs.python.org/3/faq/general.html.
14. *Python Packaging User Guide: Project Summaries* [online]. c2013–2020. [cit. 2023-05-18]. Dostupné z: https://packaging.python.org/en/latest/key_projects.
15. *Scikit-learn 1.2.2 documentation: scikit-learn: machine learning in Python* [online]. c2007-2023. [cit. 2023-05-18]. Dostupné z: https://scikit-learn.org.

16. *TensorFlow: What Is It and Why Does It Matter?* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://www.nvidia.com/en-us/glossary/data-science/tensorflow.
17. *TensorFlow* [online]. [cit. 2023-05-18]. Dostupné z: https://www.tensorflow.org.
18. *NVIDIA Data Science Glossary: What is PyTorch?* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://www.nvidia.com/en-us/glossary/data-science/pytorch.
19. *PyTorch* [online]. [cit. 2023-05-18]. Dostupné z: https://pytorch.org.
20. *OpenCV: About* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://opencv.org/about.
21. *Anaconda: Free Download* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://www.anaconda.com/download.
22. *NumPy* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://numpy.org.
23. *NumPy v1.24 Manual: Building from source* [online]. Verze 1.24. c2008-2022. [cit. 2023-05-18]. Dostupné z: https://numpy.org/doc/stable/user/building.html.
24. *SciPy* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://scipy.org.
25. *Pandas: Python Data Analysis Library* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://pandas.pydata.org.
26. *SymPy* [online]. c2021. [cit. 2023-05-18]. Dostupné z: www.sympy.org.
27. *Matplotlib: Visualization with Python* [online]. c2012–2023. [cit. 2023-05-18]. Dostupné z: https://matplotlib.org.
28. *Project Jupyter: Home* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://jupyter.org.
29. OLŠÁK, Petr. *První setkání s TeXem* [online]. 2022. [cit. 2022-11-09]. Dostupné z: https://petr.olsak.net/tat/aprvni.pdf.
30. OLŠÁK, Petr. *TeX pro pragmatiky: TeX - plainTeX - CSplain - OPmac* [online]. Brno: CSTUG, 2016 [cit. 2022-11-09]. ISBN 978-80-901950-1-1. Dostupné z: https://petr.olsak.net/ftp/olsak/tpp/tpp.pdf.
31. OLŠÁK, Petr. *TeXbook naruby* [online]. 2. vyd. Brno: Konvoj, 2001 [cit. 2023-05-26]. ISBN 80-730-2007-6. Dostupné z: https://petr.olsak.net/ftp/olsak/tbn/tbn.pdf.
32. *CTAN: What is CTAN?* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://ctan.org/ctan.
33. *Overleaf, Online LaTeX Editor: Tables* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://www.overleaf.com/learn/latex/Tables.
34. TALBOT, Nicola. *CTAN: Package datatool* [online]. 2019. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/datatool.

35. MITTELBACH, Frank. *CTAN: Package array* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/array.
36. CARLISLE, David. *CTAN: Package tabularx* [online]. 2020. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/tabularx.
37. CARLISLE, David. *CTAN: Package longtable* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/longtable.
38. OOSTRUM, Pieter van. *CTAN: Package multirow* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/multirow.
39. KERN, Uwe. *CTAN: Package xcolor* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/xcolor.
40. LYU, Jianrui. *CTAN: Package tabularray* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/tabularray.
41. PANTIGNY, François. *CTAN: Package nicematrix* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/nicematrix.
42. STURM, Thomas F. *CTAN: Package csvsimple* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://ctan.org/pkg/csvsimple.
43. FEUERSÄNGER, Christian. *CTAN: Package pgf* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pgf.
44. FEUERSÄNGER, Christian. *CTAN: Package pgfplots* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pgfplots.
45. VOSS, Herbert. *CTAN: Package pstricks* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pstricks-base.
46. VOSS, Herbert. *CTAN: Package pst-plot* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pst-plot.
47. BOWMAN, John. *CTAN: Package asymptote* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/asymptote.
48. HAMMERLINDL, Andy; BOWMAN, John; PRINCE, Tom. *Asymptote: the Vector Graphics Language: LaTeX usage* [online]. Verze 2.85-27. c2004-2023. [cit. 2023-05-18]. Dostupné z: https://asymptote.sourceforge.io/doc/LaTeX-usage.html.
49. CARLISLE, David. *CTAN: Package graphics* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/graphics.
50. CARLISLE, David. *CTAN: Package graphicx* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/graphicx.
51. HANISCH, Falk. *CTAN: Package svg* [online]. 2020. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/svg.
52. MATTHIAS, Andreas. *CTAN: Package pdfpages* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pdfpages.

53. GRAHN, Alexander. *CTAN: Package media9* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/media9.
54. OBERDIEK, Heiko. *CTAN: Package embedfile* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/embedfile.
55. PAKIN, Scott. *CTAN: Package attachfile* [online]. 2016. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/attachfile.
56. OBERDIEK, Heiko. *CTAN: Package attachfile2* [online]. 2019. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/attachfile2.
57. ISAMBERT, Paul. *CTAN: Package navigator* [online]. 2016. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/navigator.
58. *Python Wiki: Templating* [online]. [cit. 2023-01-31]. Dostupné z: https://wiki.python.org/moin/Templating.
59. *Jinja Documentation (3.1.x): Jinja* [online]. c2007. [cit. 2023-05-18]. Dostupné z: https://jinja.palletsprojects.com.
60. BAYER, Michael. *Welcome to Mako!* [online]. [cit. 2023-05-18]. Dostupné z: https://www.makotemplates.org.
61. KOEBLER, Roland. *Simple is better: pyratemp* [online]. 2013. [cit. 2023-05-18]. Dostupné z: https://www.simple-is-better.org/template/pyratemp.html.
62. ZIEGENHAGEN, Uwe. *Combining LaTeX with Python* [online]. 2019. [cit. 2023-01-31]. Dostupné z: https://tug.org/tug2019/slides/slides-ziegenhagen-python.pdf.
63. FENNEMA, Jelte. *PyLaTeX 1.3.2 documentation: PyLaTeX* [online]. c2015. [cit. 2023-03-14]. Dostupné z: jeltef.github.io/PyLaTeX.
64. LEISCH, Friedrich. *Sweave User Manual* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://stat.ethz.ch/R-manual/R-devel/library/utils/doc/Sweave.pdf.
65. XIE, Yihui. *Knitr: Elegant, flexible, and fast dynamic report generation with R* [online]. c2001-2023. [cit. 2023-05-18]. Dostupné z: https://yihui.org/knitr.
66. BAR, Haim. *CTAN: Package runcode* [online]. 2020. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/runcode.
67. MOLTENO, Tim. *CTAN: Package sympytex* [online]. 2014. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/sympytex.
68. EHMSEN, Martin. *CTAN: Package python* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/python.
69. POORE, Geoffrey. *CTAN: Package pythontex* [online]. 2021. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pythontex.
70. BREWIN, Leo Christopher. *CTAN: Package hybrid-latex* [online]. 2018. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/hybrid-latex.

71. PASECHNIK, Dima. *CTAN: Package sagetex* [online]. 2022. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/sagetex.
72. *SageMath: Open-Source Mathematical Software System* [online]. c2005-2023. [cit. 2023-05-18]. Dostupné z: https://www.sagemath.org.
73. USER202729. *CTAN: Package pythonimmediate* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pythonimmediate.
74. ENDERLE, Tobias. *CTAN: Package pyluatex* [online]. 2023. [cit. 2023-05-18]. Dostupné z: https://www.ctan.org/pkg/pyluatex.
75. LEES-MILLER, John. *Intro to Docker and how we use it at Overleaf*. 2014. Dostupné také z: https://jdlm.info/ds-docker-demo.
76. *Fixing and avoiding compile timeouts* [online]. c2023. [cit. 2023-05-18]. Dostupné z: www.overleaf.com/learn/how-to/Fixing_and_avoiding_compile_timeouts.
77. ESSLINGER, Bernhard. *PythonTeX and LaTeX – Familiarize us with PythonTeX in order to build the CrypTool Book: Learning via Samples and Counter Examples* [online]. Verze 1.01. 2022. [cit. 2023-05-18]. Dostupné z: https://www.cryptool.org/download/ctb/PythonTex-by-Examples.pdf.
78. *Open Source Initiative: The MIT License* [online]. [cit. 2023-05-18]. Dostupné z: https://opensource.org/license/mit.
79. *Sphinx documentation: Welcome* [online]. c2007-2023. [cit. 2023-05-18]. Dostupné z: https://www.sphinx-doc.org.
80. *Read the Docs user documentation 9.13.0 documentation: Getting started with Sphinx* [online]. [cit. 2023-05-18]. Dostupné z: https://docs.readthedocs.io/en/stable/intro/getting-started-with-sphinx.html.
81. KOKŠTEIN, Richard. *Data2latex: Package prototype for simple generation of LaTeX tables and plots from scientific data for use in any document.* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://github.com/Trolobezka/data2latex.
82. KOKŠTEIN, Richard. *Data2latex* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://pypi.org/project/data2latex.
83. KOKŠTEIN, Richard. *Data2latex 1.0.5 documentation: Welcome to Data2LaTeX's documentation!* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://trolobezka.github.io/data2latex-docs.
84. *Manage your TeX installation with MiKTeX Console* [online]. c2023. [cit. 2023-05-18]. Dostupné z: https://miktex.org/howto/miktex-console.

Seznam obrázků

Seznam ukázek kódu

Seznam příloh

Příloha I: Tvorba složitějšího dokumentu

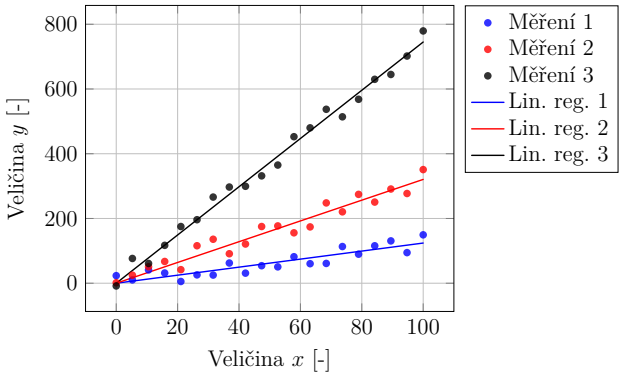
V následující ukázce kódu 26 je umístěn skript pro tvorbu složitějšího dokumentu, který obsahuje nadpis, text, tabulku a graf. Skriptem vytvořený `.pdf` dokument je zobrazen na obrázku 27.

Protokol z měření

Byly provedeny tři měření. Naměřené body z každého měření byly zobrazeny v grafu a proloženy přímkou. Směrnice přímek jsou uvedeny v tabulce.

Tabulka 1: Parametry lineární regrese

Měření	Směrnice	R^2
1	1.241	0.807
2	3.207	0.947
3	7.451	0.991



Obrázek 1: Naměřená data

Obr. 27: Složitější dokument vytvořený ukázkovým skriptem

```
import numpy as np
import data2latex as dtol
import pylatex
# Generace náhodných vstupních dat.
# V praxi bude nahrazeno reálným výpočtem nebo načtením dat ze souboru.
n = 20
X = np.linspace(0, 100, n)
Y1 = np.random.uniform(1, 2) * X + np.random.uniform(-30, 30, n)
Y2 = np.random.uniform(3, 5) * X + np.random.uniform(-35, 35, n)
Y3 = np.random.uniform(6, 9) * X + np.random.uniform(-40, 40, n)
# Lineární regrese.
a1, _, _, _ = np.linalg.lstsq(X[:, np.newaxis], Y1, rcond=None)
a2, _, _, _ = np.linalg.lstsq(X[:, np.newaxis], Y2, rcond=None)
a3, _, _, _ = np.linalg.lstsq(X[:, np.newaxis], Y3, rcond=None)
Xlr = np.array([np.min(X), np.max(X)])
Ylr1 = a1 * Xlr
Ylr2 = a2 * Xlr
Ylr3 = a3 * Xlr
R1 = np.corrcoef(X, Y1)[0, 1] ** 2
R2 = np.corrcoef(X, Y2)[0, 1] ** 2
R3 = np.corrcoef(X, Y3)[0, 1] ** 2
# Tvorba LaTeXového dokumentu.
# Pomocí balíčku babel nastavíme české označení tabulek a grafů.
dtol.gd().packages.add(pylatex.Package("babel", ["english", "czech"]))
dtol.section("Protokol z měření")
dtol.text(("Byly provedeny tři měření. Naměřené body z každého "
           "měření byly zobrazeny v grafu a proloženy přímkou. "
           "Směrnice přímek jsou uvedeny v tabulce."
          ), escape=False)
dtol.table([["Měření", "Směrnice", "$R^2$"],
            ["1", a1[0], R1],
            ["2", a2[0], R2],
            ["3", a3[0], R3]],
           "_2", "Parametry lineární regrese", escape_cells=False)
dtol.plot([X, X, X, Xlr, Xlr, Xlr], [Y1, Y2, Y3, Ylr1, Ylr2, Ylr3],
          "Naměřená data", "Veličina $x$ [-]", "Veličina $y$ [-]", "#",
          mark=["*", "*", "*", None, None, None],
          line=[None, None, None, "-", "-", "-"],
          legend=["Měření 1", "Měření 2", "Měření 3",
                  "Lin. reg. 1", "Lin. reg. 2", "Lin. reg. 3"],
          mark_fill_opacity=0.8,
          legend_pos="outer north east",
          legend_entry_align="l")
dtol.finish("ukazkovy_dokument")
```

Kód 26: Ukázkový skript pro tvorbu složitějšího dokumentu