FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Ukázkový interaktivní grafický výstup z výpočtu
Student:	Bc. Dominika Králiková
Vedoucí:	doc. Ing. Štěpán Starosta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

1) Seznamte se s volně dostupnými vykreslovacími systémy umožňující publikovat na webu interaktivní grafické ukázky výpočtů (např. tikz, PyPlot, Bokeh, D3.js, Raphael atd.).
2) Sestavte seznam modelových příkladů z předmětů KAM vhodných pro interaktivní ukázky při výuce. Zahrňte: zobrazení tečny a tečné nadroviny (pro jedno- a dvourozměrné funkce), vykreslení Taylorova polynomu (pro jedno- a případně i pro dvourozměrné funkce), vizualizace definice Darbouxova integrálu, vizualizace Lagrangeovy metody pro vázané extrémy. Vizualizace se vyžaduje 2D, vizualizace ve 3D je vítaná.
3) Ve vybraném vykreslovacím systému implementujte modelové příklady. Implementaci proveďte tak, aby bylo možné ji nasadit na systému MARAST.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 20. října 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Ukázkový interaktivní grafický výstup z výpočtu

Bc. Dominika Králiková

Katedra softwarového inženýrství
Vedoucí práce: doc. Ing. Štěpán Starosta, Ph.D.

2. května 2021

Poděkování

Ráda bych poděkovala vedoucímu své práce doc. Ing. Štěpánu Starostovi, Ph.D., za odborné konzultace, podporu a spolupráci při její tvorbě. Také bych chtěla poděkovat své rodině, svému příteli a kamarádům za to, že mi byli během psaní této práce a během celého studia oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. května 2021 ……

Odkaz na tuto práci

Králiková, Dominika. *Ukázkový interaktivní grafický výstup z výpočtu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce se zabývá výběrem vhodných technologií pro vytváření interaktivních grafických ukázek výstupů z výpočtů a využitím navrženého řešení při implementaci modelových příkladů určených pro podporu výuky matematických předmětů. První část práce je věnována analýze dostupných vykreslovacích systémů. Následně je sestaven seznam modelových příkladů vhodných pro interaktivní ukázky při výuce. Poté se práce věnuje výběru vhodného vykreslovacího systému, návrhu a implementaci webové aplikace s jednotlivými ukázkami, jejímu testování a zkušebnímu nasazení. Výstupem práce je hotová webová aplikace implementovaná s využitím knihovny Plotly, která zahrnuje modelové příklady jako vykreslení aproximace funkce pomocí Taylorova polynomu nebo vizualizaci Lagrangeovy metody. Aplikace podporuje jednoduché přidávání nových vizualizací a jejich propojení se systémem pro podporu výuky MARAST.

Klíčová slova interaktivní výstup výpočtu, grafický výstup výpočtu, interaktivní aplikace, vykreslovací systém, webová aplikace, podpora výuky, Python, Plotly, Dash, MARAST

Abstract

This diploma thesis focuses on the selection of appropriate technologies for the creation of interactive graphical visualizations of calculation output and on implementation of model examples designated to support the teaching of mathematical subjects with the use of proposed solution. First part of the thesis is dedicated to the analysis of available rendering systems. Subsequently, a list of model examples suitable for interactive demonstrations during a lecture is assembled. Afterwards the thesis focuses on the selection of a suitable rendering system, on the design and the implementation of a web application containing individual model examples, on the testing of this application and on its trial deployment. The output of this thesis is a finished web application implemented with the use of the Plotly library, which includes defined model examples such as function approximation with the help of Taylor polynomial or visualization of the method of Lagrange multipliers. The application supports easy creation of new visualizations and their integration with the MARAST teaching support system.

Keywords interactive calculation output, graphical calculation output, interactive application, rendering system, web application, teaching support, Python, Plotly, Dash, MARAST

Obsah

Seznam obrázků

Seznam výpisů kódu

Seznam tabulek

Úvod

Při studiu na vysoké škole se matematické předměty považují za jedny z nejnáročnějších. Studenti mívají problém pochopit význam definic a vět, uvést si daná sdělení do kontextu a propojit je mezi sebou. Jednou z výukových pomůcek, která může studentům pomoci látku lépe uchopit, vizuálně si ji představit a nakonec si ji i lépe zapamatovat, je její grafické znázornění pomocí interaktivních ukázek. Tyto ukázky mohou doprovázet a doplňovat výuku například znázorněním geometrického významu definice, nebo ilustrací významu matematické věty.

Tato práce se zabývá tvorbou interaktivních webových ukázek pro vysokoškolské matematické předměty. Jejím cílem je sestavit soubor technologií a postupů pro vytváření takových ukázek a implementovat prototypové příklady s využitím navrženého řešení. Výstup práce bude přínosný zejména pro studenty Fakulty informačních technologií ČVUT v Praze, a to především pro studenty předmětů zaštítěných Katedrou aplikované matematiky. Výsledné řešení totiž bude propojené s webovým portálem MARAST, který slouží pro podporu výuky (nejen) matematických předmětů. Portál umožňuje opakování látky v lekcích, její procvičování v cvičebnicích a testování nabytých znalostí v pravidelných kvízech [1].

Toto téma jsem si vybrala, protože věřím, že pro studenty bude výsledné řešení při snaze porozumět učivu výrazným přínosem. Fakulta navíc dosud nemá jednotný systém pro vytváření a publikování zmíněných interaktivních vizualizací látky probírané v matematických předmětech.

Cíle práce

Hlavním cílem této práce je zvolit vhodné technologie, které umožní vytváření interaktivních vizualizací a jejich publikování ve webovém prostředí, a využít je při implementaci modelových příkladů. Hlavním požadavkem na navržené řešení přitom je jeho jednoduchá rozšiřitelnost – měl by být navržen takový

postup tvorby vizualizace, který umožní přímočaré a jednoduše pochopitelné vytváření a publikování nových ukázek.

K dosažení tohoto záměru budou sloužit tři dílčí cíle. Prvním je analýza dostupných vykreslovacích systémů, které umožňují publikovat interaktivní ukázky na webu. Tento cíl zahrnuje seznámení se s dostupnými technologiemi a také jejich vyzkoušení na jednoduchých příkladech.

Dalším cílem je sestavení seznamu modelových ukázek z předmětů Katedry aplikované matematiky vhodných pro interaktivní ukázky při výuce a seznámení se s matematickou teorií, která je základem pro vytvoření těchto ukázek.

Posledním cílem je ve vybraném vykreslovacím systému implementovat modelové ukázky tak, aby hotovou implementaci bylo možné nasadit v systému MARAST.

Členění práce

Tato práce je členěna do několika kapitol. V kapitole 1 se budu věnovat analýze dostupných vykreslovacích systémů. Stanovím kritéria pro posouzení těchto systémů a na základě dostupných dokumentací pak zhodnotím jejich splnění. Pohodlnost práce se systémy navíc otestuji na implementaci jednoduchých příkladů.

V kapitole 2 sestavím seznam modelových ukázek z předmětů Katedry aplikované matematiky vhodných pro interaktivní ukázky při výuce. Ke každé ukázce uvedu matematickou teorii, která bude sloužit jako podklad pro její implementaci. Zároveň také navrhnu, jak by výsledné interaktivní ukázky měly vypadat a co by mělo být jejich vstupy, výstupy a parametry.

Kapitola 3 se bude věnovat výběru vykreslovacího systému pro další implementaci. Stanovím v ní požadavky na zvolený systém vzhledem k vybraným modelovým ukázkám a k jejich zamýšlenému webovému zobrazení. Poté zhodnotím jejich splnění na základě poznatků z kapitoly 1 a provedu výběr vykreslovacího systému.

Poslední kapitola 4 bude zaměřena na podrobnější popis zvolených technologií, návrh řešení, implementaci modelových ukázek a na testování a nasazení výsledného řešení.

Kapitola 1

Vykreslovací systémy

V této kapitole stanovím v souladu s cíli práce klíčové vlastnosti vykreslovacích systémů. S důrazem na tyto nezbytné vlastnosti představím jak důležité programovací jazyky, systémy a technologie, tak především dostupné vykreslovací systémy, například TikZ, Pyplot, Bokeh, D3.js a Raphaël. Tyto systémy zhodnotím a porovnám mezi sebou tak, abych dále v práci mohla zvolit ten nejvhodnější pro účely této práce.

1.1 Zkoumaná kritéria

Pro seznámení s vykreslovacími systémy je nutné stanovit, jaké jejich vlastnosti a schopnosti jsou pro účel této práce zajímavé – co je z pohledu práce klíčové a nezbytné, případně jaké vlastnosti jsou přidanou hodnotou. Díky tomu bude možné při seznamování porovnávat systémy mezi sebou a také později vyhodnotit, který z nich je pro účel této práce nejvhodnější.

Tyto vlastnosti je nutné vybírat podle očekávaného použití systému – bude sloužit k tvorbě matematických vizualizací (nutně ve 2D a ve 3D zploštěném do vrstevnic, nejlépe i ve 3D) a k interaktivní manipulaci s výstupem. Výsledek bude zobrazen na webových stránkách. Je nutné, aby byl systém jednoduše použitelný a uživatelsky příjemný. S tím souvisí i dobrá dokumentace a existující tutoriály pro pochopení základů.

Na základě tohoto očekávaného použití jsem vybrala následující vlastnosti, na které se dále zaměřím:

možnosti jazyka – možnosti pro matematické operace od základních matematických funkcí, jako jsou například mocniny, až po pokročilé operace, například derivace,

možnosti systému – možnosti pro vykreslování, především pro vykreslení matematických objektů a pro interaktivitu,

vykreslení matematických objektů – jakými způsoby se vykreslují grafy funkcí, jaké typy grafů je možné vykreslit a zda je jich možno vykreslit v jednom souřadnicovém systému více, jaké jsou možnosti pro úpravu vzhledu grafu, vykreslení doprovodných popisků či geometrických tvarů,

vykreslení 3D objektů – možnosti pro vykreslování 3D matematických objektů a funkcí,

interaktivita – podpora pro různé interaktivní widgety, které umožní na základě zpracování uživatelského vstupu přepočítat a zobrazit výsledek, podpora přibližování nebo zobrazení informačních boxů (tooltipů) po přejetí myší,

umožnění exportu na webové stránky – podpora pro zobrazení výsledné ukázky na webu, možnosti zobrazení na webových stránkách a jejich vlastnosti,

dokumentace – kvalita dokumentace systému, existence tutoriálů pro rychlé pochopení základních principů práce se systémem, jak snadno je možné se naučit používat systém z dokumentace,

náročnost práce se systémem – množství kódu, intuitivita vykreslování, test systému na jednoduchém statickém příkladu grafu polynomiální funkce a její tečny v určitém bodě, test přidání interaktivity do příkladu u slibných systémů.

1.2 Důležité technologie

Než se budu věnovat popisu jednotlivých vykreslovacích systémů, představím programovací jazyky, knihovny, balíčky a jiné technologie, které jsou pro jejich fungování nezbytné. Zaměřím se přitom zejména na podporu matematických operací, případně zobrazování na webových stránkách.

1.2.1 LaTeX

Podle `latex-project.org` [2] je LaTeX „*systém přípravy dokumentů pro vysoce kvalitní typografickou sazbu.*“ Je používán především pro technické a vědecké dokumenty. Zjednodušuje proces tvorby dokumentu tím, že se autor nemusí zabývat vzhledem, ale pouze obsahem tvořeného dokumentu. Mimo jiné umožňuje sazbu časopisových článků, technických zpráv nebo knih. Kontroluje reference, odkazy a zdroje. Podporuje také sázení komplexních matematických formulí.

Balíček PGF PGF je balíček, na němž dle jeho oficiální dokumentace [3] staví níže zmiňovaný vykreslovací systém TikZ. Podle této dokumentace je jeho součástí také matematický engine, který umožňuje aritmetické operace včetně některých složitějších matematických operací, jako jsou logaritmus nebo goniometrické funkce.

V dokumentaci [3] je uvedeno, že není možné očekávat vysokou rychlost a přesnost těchto operací, jelikož je balíček implementován v jazyce TEX, který je určen spíše pro sazbu dokumentů, než matematické výpočty. Žádná hodnota navíc v průběhu výpočtů nesmí překročit $\pm 16\,383,999\,99$.

1.2.2 Python

Dle dokumentace jazyka [4] je Python interpretovaný, dynamicky typovaný programovací jazyk. Je objektově orientovaný, ale podporuje i další programovací paradigmata jako procedurální a funkcionální programování.

Modul math dle [4] přímo zahrnuje základní matematické operace – mocniny, odmocniny, exponenciální funkci, logaritmus i goniometrické funkce, stejně jako základní konstanty jako π a e. Pro komplikovanější matematické a vědecké výpočty pak existují například balíčky z ekosystému SciPy – NumPy, SciPy a SymPy.

Pro tvorbu a sdílení dokumentů obsahujících funkční kód a vizualizace pak je možné využít Jupyter Notebook [5], který je dle [6, 7] možné sdílet na webu.

NumPy NumPy dle je dle svých stránek [8] knihovna pro numerické výpočty. poskytuje mimo jiné rozsáhlou infrastrukturu pro práci s jednorozměrnými i vícerozměrnými poli. NumPy pole se od pole v Pythonu liší – má pevnou velikost a obsahuje pouze objekty jednoho typu. NumPy se také soustředí na rychlé a efektivní zpracování operací nad těmito poli.

SciPy SciPy dle své dokumentace [9] poskytuje mnoho uživatelsky přívětivých a efektivních numerických postupů, mimo jiné také integraci, derivaci, lineární algebru a statistiku.

SymPy SymPy pak je dle dokumentace [10] knihovna pro symbolické výpočty. Jejím cílem je stát se plnohodnotným počítačovým algebraickým systémem. Mimo jiné zahrnuje důležité operace matematické analýzy, jako jsou limity, derivace, integrály a Taylorovy polynomy.

Jupyter Notebook Jupyter Notebook je podle své dokumentace [5] webová aplikace, umožňující tvorbu a sdílení dokumentů obsahujících funkční kód, rovnice, vizualizace a doprovodný text. Podporuje mnoho jazyků, mezi jinými také Python.

Existuje více variant pro sdílení Jupytek Notebooků na webu. Mimo jiné je možné je exportovat do částečně statických HTML a LaTeX dokumentů [5]. Pro jejich sdílení ale existují i různé další nástroje.

- **nbviewer** – Tento nástroj dle [6] umožní zobrazení notebooku jako statické webové stránky, zachová ale vložený JavaScript (a tedy i interaktivitu knihoven jako například Bokeh). Nástroji je třeba poskytnout URL notebooku (například odkaz na GitHub), poté vykreslí z notebooku HTML a vrátí stabilní odkaz, který je možné sdílet.
- **Binder** – Tento nástroj dle [7] umožňuje sdílení notebooků publikovaných v GitHub repozitářích. Z repozitáře je vytvořen Docker image. Pomocí poskytnutého odkazu je pak možno zobrazit notebook, se kterým lze standardně interagovat. Dle [11] nicméně trvá zobrazení Jupyter Notebooku sdíleného pomocí této služby i několik desítek sekund.

1.2.3 JavaScript

Dle webových stránek MDN [12] je JavaScript *„skriptovací programovací jazyk, který umožňuje implementaci komplexních vlastností webových stránek.“* Jedná se o třetí ze standardních webových technologií po HTML a CSS. Je zodpovědný za dynamické chování webové stránky a interakci s uživatelem. JavaScript může být použitý jak na straně klienta v prohlížeči, tak na straně serveru.

O základní matematiku se dle [12] v JavaScriptu stará objekt Math. Ten zahrnuje matematické konstanty e, π ale například i hodnotu odmocniny ze dvou. Dále pak zahrnuje základní matematické operace – absolutní hodnotu, mocniny a odmocniny, exponenciální funkci i logaritmy, goniometrické funkce a zaokrouhlování.

math.js Pro pokročilou matematiku v JavaScriptu lze použít například rozsáhlou knihovnu math.js. Tato knihovna dle své dokumentace [13] zahrnuje mimo jiné flexibilní parser matematických výrazů, podporu pro symbolické operace včetně derivace, operace pro počítání s komplexními čísly a maticemi, kombinatorické operace, pravděpodobnostní a statistické operace, množinové operace a další.

Technologie pro vykreslení pokročilé grafiky pomocí JavaScriptu Interaktivní vykreslení pokročilé grafiky, jakou jsou například grafy matematických funkcí, je na webu možné například pomocí následujících technologií.

- **Canvas API** – Rozhraní Canvas poskytuje dle [14] prostředky *„pro kreslení grafiky s využitím JavaScriptu a HTML elementu* `<canvas>`*.“* Zaměřuje se zejména na 2D grafiku, používá se pro animace, herní grafiku, ale například i vizualizaci dat. Při použití tohoto přístupu obsahuje

HTML dokument element `<canvas>`, který je potom pomocí JavaScriptu „naplněn obsahem“ s využitím definovaných funkcí.

- **SVG** – Značkovací jazyk SVG – Scalable Vector Graphics – dle MDN [15] slouží k popisu dvourozměrné vektorové grafiky. Základní elementy SVG zahrnují mimo jiné kruhy, elipsy, obdélníky, obrázky, cesty a text. Pro účel této práce jsou nejdůležitějším elementem cesty – `paths`, používané pro tvoření linií, oblouků, křivek a dalšího. Tvar cesty je popsán pomocí několika příkazů – přesun do bodu, nakreslení rovné čáry do bodu, nakreslení Bézierovy křivky do bodu, nakreslení oblouku do bodu a uzavření cesty. K modifikaci SVG elementů slouží atributy, mimo jiné barva výplně, styl, tloušťka a barva čáry a transformace (rotace, posunutí, škálování, sklopení).

1.3 Dostupné vykreslovací systémy

V této sekci představím dostupné vykreslovací nástroje se zaměřením na TikZ, Pyplot, Bokeh, D3.js, Raphaël a Plotly. Soustředím se při tom na důležité vlastnosti zmíněné v sekci 1.1.

1.3.1 TikZ

Online LaTeX editor Overleaf [16] uvádí, že balíček TikZ je „*pravděpodobně nejkomplexnější a nejmocnější nástroj pro tvoření grafických elementů v LaTeX.*“ Podle [3] je TikZ postaven na balíčku PGF.

Možnosti systému LaTeX a balíčku PGF jsou popsány výše v sekci 1.2.1

Vykreslení matematických objektů Portál Overleaf [16] a stručný úvod k TikZ [17] popisují následující vlastnosti balíčku.

- **Způsob vykreslení matematické funkce** – Balíček dovede vykreslit matematické funkce přímo podle jejich vzorce. Navíc je při jejich vykreslování možné využít celou řadu integrovaných základních funkcí, jako jsou například druhá odmocnina, mocnina, exponenciála, logaritmus, goniometrické funkce a další.

- **Podporované typy grafů** – Vzhledem k nízkoúrovňovému rozhraní balíčku jsou jeho možnosti velmi flexibilní – je možné v něm vytvořit jak spojnicové, tak bodové, či sloupcové grafy. Spojnicové jsou však díky podpoře vykreslování funkcí podle vzorce nejjednodušší. Je také možné zobrazit vícero typů grafů v jednom systému souřadnic.

- **Možnosti úpravy vzhledu grafu funkce** – Balíček umožňuje nastavit vzhled všem elementům – okraje, výplň, styl čáry apod.

- **Vykreslení geometrických tvarů** – Balíček dovede vykreslit základní elementy jako body, čáry a křivky, jednoduché geometrické tvary. Navíc je možné kreslení lomených čar, dekorování čar šipkami, zaoblení rohů.
- **Podporované úpravy souřadnicových os** – Souřadnicové osy je nutné vykreslit manuálně, stejně jako důležité body na nich. Uživatel tedy má velkou volnost co do měřítek a vzhledu os, ale zároveň se o jejich vykreslení musí starat manuálně, systém mu to neusnadní.
- **Vkládání textových popisků** – TikZ umožňuje přidávat popisky a nastavit jejich pozici vzhledem ke konkrétnímu bodu (jestli budou nad ním, pod ním apod.). Je možné nastavit jejich barvu nebo zarovnání textu. Také je možné v rámci popisku využít matematický mód a sázet díky tomu matematické formule, které se vykreslí interpretované pomocí LaTeX.

Vykreslení 3D objektů Na balíčku TikZ podle Overleaf [18] staví balíček PGFplots. Ten je zaměřen přímo na vědeckou grafiku, ještě dále zjednodušuje vykreslování matematických objektů a navíc umožňuje 3D projekci objektů a matematických funkcí.

Interaktivita Podle odpovědí na fóru TeXStack Exchange [19] by interaktivní změna hodnot parametrů byla při použití balíčku TikZ významně omezená. Jednou z mála možností by bylo připravení grafů pro předem určenou sadu hodnot a jejich zobrazování/skrývání.

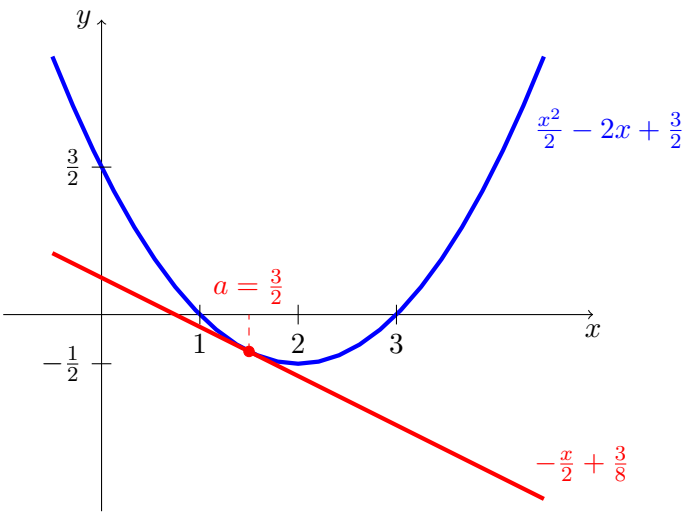
Zobrazení na webu Z dokumentace PGF [3] vyplývá, že výstup je možné exportovat do formátů PostScript, PDF a SVG. Soubory ve formátech PDF a SVG je možné zobrazit na webu.

Dokumentace Sama oficiální dokumentace k balíčku PGF je velmi rozsáhlá, manuál má přes 1 200 stran [3] a orientace v něm je náročnější. Další krátké tutoriály [17, 16] popisují především základní funkce.

Náročnost práce se systémem V rámci otestování náročnosti práce se systémem jsem vytvořila graf funkce a její tečny v daném bodě. Hotová ukázka je na obrázku 1.1, kód potřebný k jejímu vytvoření je na výpisu 1.1.

Z kódu je patrné, že bylo potřeba zvlášť vytvořit souřadnicové osy, stejně jako důležité body na nich. Rovněž bylo nutné manuálně vypočítat derivaci zobrazené funkce a manuálně získat rovnici popisující její tečnu ve zvoleném bodě – kvůli omezeným možnostem matematického engine systému PGF.

Na druhou stranu velmi jednoduché bylo vykreslení funkcí pomocí jejich vzorce, do modelu se dobře přidávaly textové popisky i jiné popisné struktury,



Obrázek 1.1: Výsledek testování vykreslovacího systému TikZ vytvořený pomocí kódu ve výpisu 1.1.

jako zvýraznění bodu a na grafu funkce nebo vodící linie od tohoto bodu k ose x.

Interaktivitu jsem vzhledem k omezeným možnostem balíčku v tomto směru netestovala.

Celkově bylo vytvoření ukázky poměrně pracné, a to jak na množství manuálně tvořených prvků, tak na manuální výpočty. Vytvořené modely jsou ale díky nízkoúrovňovosti balíčku velmi flexibilní a použitý kód se v nich často opakuje – takže se stačí naučit syntaxi několika málo příkazů. Celkově si myslím, že je balíček velmi dobrý ve vytváření jednoduchých statických ukázek, složitější ukázky by byly spojené s rychle rostoucím množstvím kódu a tím pádem i s vysokou náročností na vytváření.

Shrnutí Balíček TikZ, eventuálně jeho nadstavba PGFplots, by byl použitelný pro statické vykreslení matematických objektů jak ve 2D, tak ve 3D. Při jeho použití je ale nutné počítat s větší náročností vytváření vizualizací, jelikož je nutné např. manuálně vykreslovat osy a popisky na nich.

Problematické by mohly být matematické výpočty, na jejichž základě by se měly objekty vykreslovat – překážkou zde bude omezená přesnost a rychlost matematického engine balíčku PGF. Jediným řešením by bylo vypočítat hodnoty externě a do kódu je poté vložit.

Dalším problémem bude požadovaná interaktivita. Balíček ji nedokáže poskytnout na dostatečné úrovni.

```
\begin{tikzpicture}[xscale=1.5,yscale=1.5]

% axes
\draw [->] (-1, 0) -- (5, 0);
\node [below] at (5, 0) {$x$};
\draw [->] (0, -2) -- (0, 3)
\node [left] at (0, 3) {$y$};

% main function
\draw [blue, ultra thick, domain=-0.5:4.5] plot (\x, {\x*\x/2-2*\x+1.5});
\node [below right, blue] at (4.3, 2.2) {$\frac{x^2}{2}-2x+\frac{3}{2}$};
\draw (1, -0.1) -- (1, 0.1);
\node [below] at (1, -0.1) {$1$};
\draw (2, -0.1) -- (2, 0.1);
\node [below] at (2, -0.1) {$2$};
\draw (3, -0.1) -- (3, 0.1);
\node [below] at (3, -0.1) {$3$};
\draw (-0.1, -0.5) -- (0.1, -0.5);
\node [left] at (-0.1, -0.5) {$-\frac{1}{2}$};
\draw (-0.1, 1.5) -- (0.1, 1.5);
\node [left] at (-0.1, 1.5) {$\frac{3}{2}$};

% tangent
\draw [red, ultra thick, domain=-0.5:4.5] plot (\x, {-0.5*\x+0.375});
\node [above right, red] at (4.3, -1.8) {$-\frac{x}{2}+\frac{3}{8}$};
\draw [fill, red] (1.5, -0.375) circle  [radius=1.5pt];
\draw [red, dashed] (1.5, -0.375) -- (1.5, 0);
\node [above, red] at (1.5, 0) {$a=\frac{3}{2}$};

\end{tikzpicture}
```

Výpis kódu 1.1: Vytvoření ukázky 1.1 v systému TikZ.

1.3.2 Pyplot

Podle oficiální dokumentace knihovny Matplotlib [20] je její část Pyplot „*kolekcí funkcí, které umožňují Matplotlibu fungovat jako MATLAB*“ (MATLAB je dle [21] programovací platforma, která dovede analyzovat data, vyvíjet algoritmy a tvořit modely a aplikace). Každá funkce nějak ovlivní výsledný model – od jeho vytvoření, přes přidání grafové oblasti a vlastní vykreslení grafu, až po přidání různých dekorativních elementů, například popisků.

Matplotlib samotný je dle dokumentace [20] „*obsáhlá knihovna pro tvoření statických, animovaných a interaktivních vizualizací v Pythonu.*“ Umožňuje

vytvořit kvalitní grafy pomocí pár řádků kódu, tvořit interaktivní modely, exportovat modely do mnoha prostředí a mnoho dalšího.

Matplotlib může být dle své dokumentace [20] v zásadě používán dvěma způsoby.

- Objektově orientovaný styl umožňuje explicitně vytvářet modely a grafy a volat nad nimi metody.
- S využitím Pyplotu a jeho automatického vytváření a spravování modelů a grafů skrze funkce – dochází zde mimo jiné k implicitnímu určení konkrétního grafu v modelu, jehož se úpravy týkají.

Možnosti jazyka Python a jeho knihoven byly popsány výše v sekci 1.2.2

Vykreslení matematických objektů Pyplot má podle dokumentace [20] následující možnosti pro vykreslení matematických objektů.

- **Způsob vykreslení matematické funkce** – Je možné vykreslit graf matematické funkce pomocí seznamů nebo NumPy polí popisujících souřadnice bodů grafu – například tak, že se vytvoří pole hodnot na ose x, pomocí funkce se spočítají odpovídající hodnoty na ose y a výsledky se předají Pyplotu k vykreslení.
- **Podporované typy grafů** – Knihovna Matplotlib jako celek podporuje dle příkladů v dokumentaci sloupcové, spojnicové i bodové grafy. Důležitá je také podpora vrstevnicových grafů. Je možné vykreslení vícero grafů funkcí (i různých druhů) v jednom souřadnicovém systému.
- **Možnosti úpravy vzhledu grafu funkce** – Jsou dostupné velmi bohaté možnosti úprav vzhledu grafu pomocí argumentů, samostatné funkce, nebo pomocí speciálních formátovacích textových řetězců. Mimo jiné je možné nastavit barvu a tloušťku linie grafu, styl čáry (plná, čárkovaná aj.), styl bodu (kruh, čtverec, trojúhelník).
- **Vykreslení geometrických tvarů** – Dle dokumentace je možné vykreslit geometrické tvary jako obdélníky, elipsy a šipky pomocí modulu `patches`. Všechny tyto tvary, stejně jako různé čáry a křivky, lze vytvořit pomocí cest (tzv. `paths`).
- **Podporované úpravy souřadnicových os** – Osám grafu je možné nastavit rozsah, je možné použít logaritmické a jiné nelineární osy. Rovněž je možné osám nastavit popisky.
- **Vkládání textových popisků** – Každý graf může mít název. Je také možné vložit text na libovolné místo modelu. Text je možné použít jako anotaci – popis konkrétního místa na grafu. Takovýto text pak může

být doplněn šipkou, která ukazuje na bod, jehož se anotace týká. Jakýkoli text může obsahovat zápis matematických vzorců pomocí TEX– ve výsledném modelu se takový text zobrazí interpretovaný.

Vykreslení 3D objektů Podle dokumentace [20] existuje sada nástrojů `mplot3d`, která podporuje vykreslování jednoduchých 3D grafů včetně povrchových a sloupcových. Tato sada je navíc součástí každé standardní instalace Matplotlibu.

Interaktivita Aby bylo možné využívat interaktivní funkce Matplotlibu, je podle dokumentace [20] nutné používat nějaký interaktivní backend – těch je celá řada, umožňují vykreslování například v TkInteru, Qt nebo Jupyter Notebooku. Pak lze využívat interaktivní funkce jako přibližování, posouvání, reakce na polohu a kliknutí myší, ale také vkládat do modelu widgety, které umožní interaktivní změnu hodnot. Mezi nabízené widgety patří například `slider`, `text box`, `button`, `radio button` a další.

Zobrazení na webu Podle dokumentace [20] je možný export do několika statických formátů, včetně PDF a SVG. Takové soubory je sice na webu možné zobrazit bez problémů, ale neumožňují požadovanou interaktivitu.

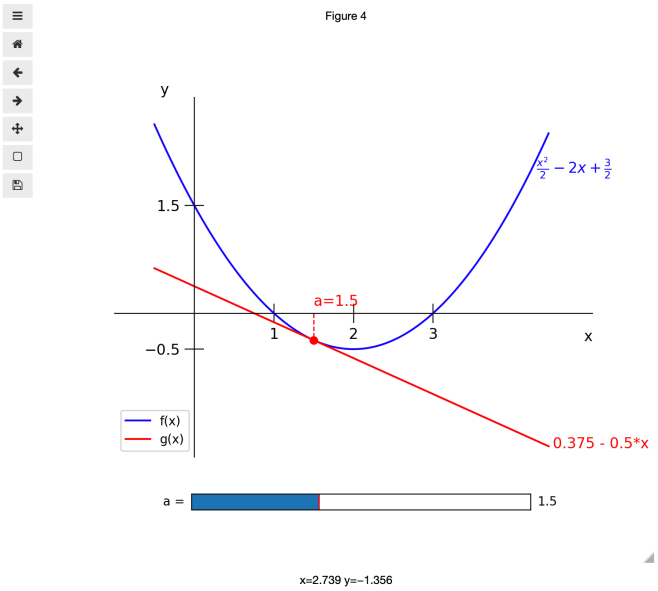
Dále je dle [20] možné používat Matplotlib na webovém aplikačním serveru, takové použití ale vyžaduje vynechání Pyplotu (kvůli práci s pamětí) a z příkladu uvedeného v dokumentaci se zdá, že je takto možné přenášet pouze statické obrázky.

Další možností je vytvoření Jupyter Notebooku a jeho sdílení na webu – tato varianta je dle dokumentace [20] plně interaktivní a je při ní možné využít vestavěné widgety.

Dokumentace Dokumentace knihovny Matplotlib [20] je obsáhlá a občas pro mě osobně byla poněkud nepřehledná. Při troše hledání se v ní nicméně dají najít všechny potřebné informace. Obsahuje navíc mnoho příkladů, ze kterých je možné čerpat inspiraci.

Náročnost práce se systémem Práci se systémem Pyplot jsem otestovala včetně interaktivních funkcí na vizualizaci grafu funkce a její tečny v daném bodě. Ukázku jsem tvořila v Jupyter Notebooku a její hotová verze je na obrázku 1.2. Zdrojový Notebook je dostupný na přiloženém médiu.

Tvorba statické ukázky byla přímočará a narazila jsem jen na velmi málo problematických částí – nejnáročnější tak bylo nastavit souřadnicové osy, aby procházely počátkem systému souřadnic a zobrazovaly se na nich správné hodnoty. Jediným požadavkem, který se mi nepodařilo naplnit, pak byly šipky na koncích souřadnicových os.



Obrázek 1.2: Výsledek testování vykreslovacího systému Pyplot.

Vykreslení funkce bylo snadné – pomocí seznamu hodnot na ose x a funkce pro výpočet hodnot na ose y jsem získala souřadnice pro vykreslovací funkci. Stejně tak bylo velmi pohodlné nastavení vzhledu i tvorba ostatních elementů v grafu. Pro ilustraci použití knihovny uvádím na výpisu 1.2 celý kód potřebný pro vykreslení základní funkce (na obrázku v modré barvě).

```
# plot main function
f, f_expr = getFunctionFormula()
yvalues = np.array([f(x) for x in XVALUES])
plt.plot(XVALUES, yvalues, "b", label="f(x)")
ax.text(4.3, f(4.3)-0.2, r"$\frac{x^2}{2} - 2x + \frac{3}{2}$",
    color="blue", fontsize=14)
```

Výpis kódu 1.2: Vykreslení průběhu funkce při vytváření ukázky 1.2 v systému Pyplot.

Pro matematické výpočty jsem použila knihovnu SymPy pro symbolické výpočty. Ta mi umožnila z textového řetězce se vzorcem funkce vytvořit zpracovaný výraz, který bylo možné zderivovat. Z reprezentace funkce a její derivace bylo možné odvodit rovnici tečny v konkrétním bodě. Z takto získaných výrazů pak knihovna vytvořila pro další výpočty běžné funkce v Pythonu.

Méně pohodlné pak bylo přidání interaktivity do již hotové ukázky. Nejprve bylo nutné pomocí příkazu `%matplotlib widget` vloženého na první řá-

dek zapnout interaktivní backend. Ten zpřístupnil interaktivní funkce jako přibližování a posouvání grafu, ale zároveň změnil poměry velikostí jednotlivých prvků, takže bylo nutné kód upravit. Dále bylo nutné manuálně upravit velikost modelu tak, aby se pod něj vešel widget `slider`, jehož pozice a velikost byla následně ne zcela přímočaře určena pomocí souřadnic. Toto podrobné nastavování sice přináší flexibilitu ohledně rozložení, ale je to velké množství práce navíc. Při samotné aktualizaci hodnot je třeba všem dotčeným datovým řadám a textovým popiskům změnit jejich parametry (data, případně pozici), což je sice přímočará, ale pracná operace. Kód potřebný pro nastavení widgetu `slider` je na výpisu 1.3.

```
# setup slider
axslider = plt.axes([0.25, 0.1, 0.55, 0.03])
slider = Slider(axslider, "a =", 0, 4, valinit=POINT, valstep=0.25)

def update(val):
    new_point = slider.val
    new_t, new_texpr = getTangentFormula(new_point)
    new_yvalues = np.array([new_t(x) for x in XVALUES])

    tan_function.set_ydata(new_yvalues)
    tan_point_marker.set_data(new_point, new_t(new_point))
    tan_line_marker.set_data([new_point, new_point], [0, new_t(new_point)])
    tan_formula_text.set_text(f"{new_texpr}")
    tan_formula_text.set_position((4.5, new_t(4.5)))
    tan_point_text.set_text(f"a={new_point}")
    tan_point_text.set_position((new_point, 0.1))
    fig.canvas.draw_idle()

slider.on_changed(update)
```

Výpis kódu 1.3: Aktualizace modelu při interaktivní změně hodnot v ukázce 1.2 v systému Pyplot.

Práce s touto knihovnou byla při vytváření statické ukázky poměrně příjemná – kromě problematického nastavování souřadnicových os byly všechny funkce přímočaré a jednoduché. Přidání interaktivity už tolik uživatelsky přívětivé nebylo, dle mého názoru se jednalo o zbytečně pracnou operaci. Výhodou naopak je široká komunita okolo Pyplotu, která poskytuje kvalitní podporu při hledání správných řešení – a supluje tak místy ne úplně přehlednou dokumentaci.

Shrnutí Pyplot umožňuje jednoduchou tvorbu 2D a také 3D matematických modelů. Výhodou jsou pokročilé možnosti Pythonu jako jazyka s ohledem na

matematické výpočty.

Nevýhodou může být ne zcela přímočará interaktivita a mírně zastaralý vzhled nabízených widgetů. Požadavek na interaktivitu také značně omezí možnosti pro zobrazení na webu.

1.3.3 Bokeh

Bokeh je dle svých stránek [22] *„interaktivní vizualizační knihovna pro moderní prohlížeče.“* Umožňuje jednoduchou tvorbu běžných grafů i komplikovanějších modelů v Pythonu. Má celou řadu nástrojů a widgetů, které podporují interaktivitu. V neposlední řadě pak umožňuje zobrazení výsledků na webu nebo v Jupyter Notebooku.

Podle dokumentace [22] sestává Bokeh ze dvou knihovních komponent – první je JavaScriptová knihovna BokehJS, která běží v prohlížeči a je odpovědná za všechno vykreslování. Druhou je knihovna Bokeh v Pythonu, která je schopná generovat JSON objekty – ty potom slouží jako vstupy pro JavaScriptovou knihovnu v prohlížeči.

Možnosti jazyka Python a jeho knihoven jsem zhodnotila v sekci 1.2.2.

Vykreslení matematických objektů Bokeh podle dokumentace [22] nabízí dvě úrovně rozhraní. První, nízkoúrovňové `bokeh.models`, pracuje na úrovni základních prvků modelu. Druhé rozhraní, `bokeh.plotting`, je již podobné např. Matplotlibu a umožňuje tvořit modely rychleji a pohodlněji.

Dokumentace [22] dále popisuje schopnosti knihovny.

- **Způsob vykreslení matematické funkce** – Data pro vykreslení grafu matematické funkce lze zadat pomocí polí souřadnic (jedno pole popisuje souřadnice na ose x, druhé na ose y) nebo pomocí slovníku, ve kterém jsou klíče jména datasetů a hodnoty pole dat.

- **Podporované typy grafů** – Knihovna podporuje mimo jiné spojnicové, bodové i sloupcové grafy. Podle příkladů v dokumentaci umožňuje vykreslení vícero grafů funkcí (i různých druhů) v rámci jednoho systému souřadnic. Podpora pro klasické vrstevnicové grafy dle dokumentace chybí – je však možné vykreslit v podstatě bitmapové obrázky barevně indikující, do jakého rozsahu hodnot spadá hodnota funkce v daném bodě.

- **Možnosti úpravy vzhledu grafu funkce** – Co se úpravy vzhledu týče, umožňuje knihovna nastavit mimo jiné barvu, tloušťku a styl čáry, barvu výplně a styl okraje, font, velikost a styl textu, styl šipek. Je možné nastavit odlišný styl objektů při najetí myší. Grafu samotnému pak je možné nastavit barvu a styl pozadí a okrajů, barvu os a jejich popisky.

- **Vykreslení geometrických tvarů a dalších prvků** – Do modelu je možné přidat různé mnohoúhelníky, elipsy aj. Bokeh dále umožňuje přidávat do grafů a modelů různé popisné prvky – od šipek až po různé linie a barevně odlišené oblasti.
- **Podporované úpravy souřadnicových os** – Je možné nastavit rozsah os nebo použít logaritmické osy místo lineárních.
- **Vkládání textových popisků** – Grafu je možné nastavit jeho titulek. Dále je možné na libovolná místa v grafu vkládat popisné štítky a nastavovat jim vzhled.

Vykreslení 3D objektů Bokeh je dle dokumentace [22] primárně určen pro 2D modelování. Nicméně dokumentace nabízí i jeden příklad 3D grafu, který byl vytvořen pomocí přizpůsobitelných rozšíření Bokehu a propojení s knihovnou VisJS Graph3d.

Interaktivita Bokeh poskytuje řadu nástrojů pro interaktivní manipulaci s grafem. Patří sem přibližování, posouvání, zobrazení tooltipů při přejetí myší nad částmi grafu, výběr oblastí a bodů. Je možné využít interaktivní legendy například pro zobrazování a skrývání částí grafu. Lze také reagovat na kliknutí a pohyb myši. Důležitými interaktivními prvky pak jsou widgety – interaktivní ovladače pro manipulaci s grafy a datasety. Widgety je možné použít v kombinaci s Bokeh serverem, ale také v samostatných HTML dokumentech, pokud jim jsou poskytnuty JavaScriptové callbacky (viz další odstavec). Mezi dostupné widgety patří například `button`, `checkbox`, `text box`, `select`, `slider` a `range slider`.

Zobrazení na webu Podle dokumentace [22] je možností zobrazení na webu vícero.

- První je použití Bokeh serveru, který umožňuje reagovat na uživatelský vstup a události z uživatelského rozhraní pomocí Pythonu – zejména je tak možné použití widgetů spolu s callbacky psanými v Pythonu. Server je možné použít pro spuštění vícero aplikací naráz, každá z nich pak bude mít vlastní URL adresu. Dokumentace obsahuje varování před použitím jednoho serveru pro webové sdílení více aplikací od různých tvůrců, a to kvůli bezpečnostním hrozbám. V takovém případě se doporučuje infrastruktura s jedním serverem pro každou aplikaci nebo alespoň pro každého tvůrce.
- Druhou možností je použití Bokehu v rámci Jupyter Notebooku a publikování daného notebooku. Jupyter Notebook umožňuje jak použití Bokeh serveru a callbacků psaných v Pythonu, tak také tvorbu vizualizace bez serveru s použitím callbacků psaných v JavaScriptu.

- Dále je možné exportovat grafy do obrázků formátu PNG nebo SVG, tím se ale ztratí interaktivní možnosti.

- Poslední variantou je využití samostatných dokumentů – Bokeh dokumentů, za kterými nestojí server, ale jedná se o vygenerované HTML, CSS a JavaScript, které je možné použít samostatně, nebo jako součást jiné webové stránky. Do takovýchto dokumentů je možné připojit i interaktivní widgety, je ale nutné těmto widgetům poskytnout callbacky psané v JavaScriptu. Použití JavaScriptových knihoven typu math.js v callbacích je možné.

Dokumentace Dokumentace knihovny [22] je i přes svou obsáhlost přehledná, uživatelská příručka je rozdělená na srozumitelné kapitoly, kde je možné najít všechny potřebné informace. Obsahuje mnoho příkladů hotových aplikací postavených nad serverem i samostatných dokumentů. Dále obsahuje kvalitní tutoriál ve formě Jupyter Notebooků.

Náročnost práce se systémem Také náročnost práce s knihovnou Bokeh (včetně přidání interaktivity) jsem testovala na příkladu funkce a její tečny v bodě. Vizualizaci jsem vytvářela v Jupyter Notebooku, který je dostupný na přiloženém médiu. Hotová vizualizace je na obrázku 1.3.

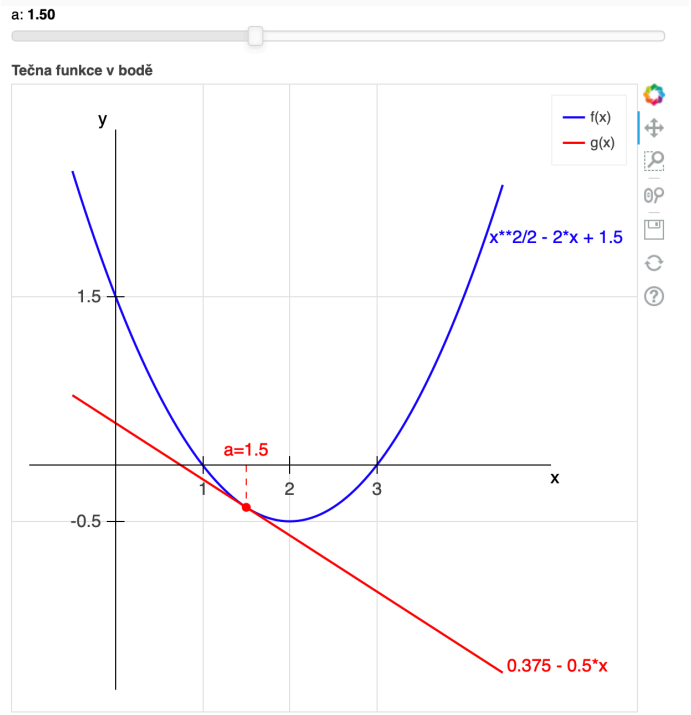
Podobně jako s Pyplotem se mi i s Bokehem při tvorbě statické ukázky pracovalo dobře a opět nejproblematičtější bylo nastavení souřadnicových os (v porovnání se mi ale zdálo více přímočaré). Stejně jako u Pyplotu se mi nepodařilo vytvořit šipky na koncích souřadnicových os.

Výsledný kód pro vykreslování statických prvků je Pyplotu také velmi podobný – toto lze dobře srovnat například na kódu pro vykreslení průběhu hlavní (modré) funkce – viz výpis 1.4. Obecně bylo tvoření všech elementů grafu jednoduché a přímočaré. Tak jako u testování Pyplotu jsem i v tomto případě pro matematické výpočty použila knihovnu SymPy.

```
1 # plot function
2 p.line(x="x", y="f", legend_label="f(x)", line_width=2, color="blue",
3     source=SOURCE)
4 p.text(x=4.3, y=f(4.3)-0.2, text=bm.value(f"{f_expr}"), text_color="blue")
```

Výpis kódu 1.4: Vykreslení průběhu funkce při vytváření ukázky 1.3 v systému Bokeh.

Přidanou hodnotou oproti Pyplotu jsou automatické interaktivní funkcionality, jako je posouvání grafu nebo jeho přibližování. Samotný kód pro aktualizaci vizualizace byl ale poměrně zdlouhavý – dle příkladů v dokumentaci [22] jsem použila speciální třídu `ColumnDataSource`, která obsahuje přiřazení polí



Obrázek 1.3: Výsledek testování vykreslovacího systému Bokeh.

dat ke klíčům a která spustí automatickou aktualizaci vizualizace při změně těchto dat. V rámci jednoho `ColumnDataSource` nicméně musí být všechna pole stejně dlouhá – není tedy možné v jednom zdroji uchovávat například informace o grafu funkce, význačném bodě na něm a spojovací linii mezi tímto bodem a osou x – v ukázce jsem proto potřebovala zdroje celkem tři. Aktualizace všech zdrojů pak představuje poměrně významné množství kódu navíc, jak je vidět na výpisu 1.5. Výhodou tohoto přístupu je, že není třeba přepočítávat hodnoty, které se nemění, a aktualizace je tak potenciálně efektivnější.

Práce s touto knihovnou byla při tvoření statické ukázky příjemná, potřebných efektů bylo obvykle možné dosáhnout rychle a ne příliš pracně. Přidání interaktivity bylo poměrně náročné, problematická byla zejména nutnost mít vícero datových zdrojů. Dobrou oporou byla kvalitní dokumentace, na druhou stranu komunita oproti Pyplotu není tolik aktivní a řešení méně obvyklých problémů se tak dohledává o něco hůře.

Shrnutí Knihovna Bokeh nabízí kvalitní interaktivní zpracování 2D modelů a navíc poskytuje celou řadu možností pro jejich zobrazení na webu. Je tedy dobrou volbou pro vykreslování parametrizovaných příkladů matematických funkcí. Navíc je možné se opřít o rozsáhlé matematické možnosti Pythonu.

Hlavním problematickým bodem jsou 3D grafy a jejich zobrazení. Ačkoli

```
def updateData(data, point):
    t, t_expr = getTangentFormula(point)
    data = {
        "x": data["x"],
        "f": data["f"],
        "t": np.array([t(x) for x in data["x"]])
    }
    point_data = {
        "x": [point],
        "y": [t(point)],
        "x_point_text": [point],
        "y_point_text": [0.05],
        "point_text": [f"a={point}"],
        "x_expr_text": [4.5],
        "y_expr_text": [t(4.5)],
        "expr_text": [f"{t_expr}"],
    }
    line_data = {
        "x": [point, point],
        "y": [0, t(point)],
    }
    return data, point_data, line_data

#slider
def callback(attr, old, new):
    data, point_data, line_data = updateData(DATA, new)
    SOURCE.update(data=data)
    POINT_SOURCE.update(data=point_data)
    LINE_SOURCE.update(data=line_data)

slider = bm.Slider(start=0, end=4, value=1.5, step=0.25, title="a")
slider.on_change("value", callback)
```

Výpis kódu 1.5: Aktualizace modelu při interaktivní změně hodnot v ukázce 1.3 v systému Bokeh.

dle poskytnutých příkladů je možné Bokeh tímto směrem rozšířit, není to vestavěná funkcionalita a tudíž se jedná o potenciální zdroj problémů.

Další, tentokrát již menší, problém by mohl nastat, pokud by se Bokeh měl zobrazovat na webu ve formě samostatných HTML dokumentů – JavaScriptové callbacky, které by v takovém případě musely být napsány ve formě textových řetězců, by mohly být nepřehledné a vyžadují znalost dalšího programovacího jazyka.

1.3.4 D3.js

Podle dokumentace knihovny D3.js [23] je D3 zkratka pro *„Data-Driven Documents.“* Jedná se o *„JavaScriptovou knihovnu pro manipulaci s dokumenty založenou na datech.“* D3.js používá HTML, SVG a CSS a klade velký důraz na webové standardy. Díky tomu dovede využít všechny možnosti moderních prohlížečů.

Knihovna D3.js je dle své dokumentace [23] flexibilní díky svému nízkoúrovňovému přístupu – soustředí se spíše na skládání základních prvků (tvarů a měřítek) místo konfigurace schémat. D3.js nabízí také možnosti pro animaci a interakci. Knihovna zahrnuje přes 30 modulů a více než 1 000 metod.

Možnosti JavaScriptu, jeho knihoven a příbuzných technologií jsem popsala výše v sekci 1.2.3.

Vykreslení matematických objektů Dle dokumentace [23] je možné při vykreslení matematických objektů využít následující schopnosti systému.

- **Způsob vykreslení matematické funkce** – Lze vykreslit graf funkce z dat ve formě pole souřadnic – může se jednat například o pole JavaScriptových objektů s atributy pro jednotlivé souřadnice. Nejprve je nutné vytvořit měřítka pro jednotlivé osy, poté pomocí těchto měřítek vykreslit linii funkce a na závěr vše propsat do HTML.
- **Podporované typy grafů** – Vedle spojnicových je možné vykreslit i další druhy grafů – například bodové, sloupcové nebo vrstevnicové. Je možné v rámci jednoho souřadnicového systému zobrazit vícero grafů (i různých druhů).
- **Možnosti úpravy vzhledu grafu funkce** – Je možné nastavit barvu a tloušťku linie, velikost a barvu bodu. Je možné použít barevný gradient pro barvené odlišení hodnot v grafu.
- **Vykreslení geometrických tvarů** – Lze vykreslovat cesty, linie, oblouky, ale i geometrické obrazce jako kruhy a obdélníky.
- **Podporované úpravy souřadnicových os** – Je možné použít logaritmické i lineární osy a nastavit jim vzhled a popisky.
- **Vkládání textových popisků** – Knihovna umožňuje vkládat popisky a upravovat jejich vzhled.

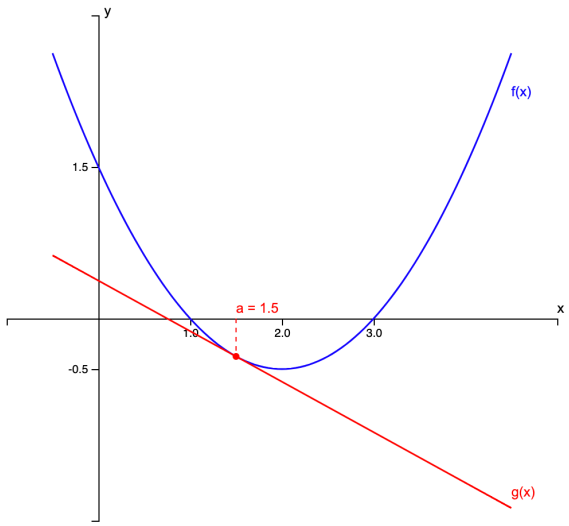
Vykreslení 3D objektů V dokumentaci [23] se mi nepodařilo najít zmínku o nativní podpoře 3D grafiky. Existuje však plugin `d3-3d` [24], který se zaměřuje na 3D vizualizace v prohlížeči. Umí vykreslit například 3D povrchové grafy, 3D bodové grafy nebo 3D spojnicové grafy.

Interaktivita Z interaktivních prvků umožňuje D3.js dle [23] například zobrazit tooltipy po najetí myší nad linii funkce, přibližovat nebo posouvat graf, reagovat na kliknutí a pohyb myši. Zpracování uživatelských vstupů je možné například pomocí HTML formulářových elementů.

Zobrazení na webu Knihovna D3.js je přímo určená pro zobrazení v prohlížeči. Dle dokumentace [23] od verze 5 podporuje moderní prohlížeče, jako Chrome, Edge, Firefox a Safari. Verze 4 a starší podporují Internet Explorer 9+. Rozsah použitelných funkcionalit tedy závisí především na možnostech prohlížeče, u moderních prohlížečů nicméně je dostupný celý.

Dokumentace Knihovna je velmi obsáhlá, a tak je náročné postihnout všechny její možnosti. Dokumentace [23] z velké části spoléhá na dostupné příklady a jejich úpravu do požadované podoby. Obsahuje řadu tutoriálů, které usnadňují pochopení základních principů knihovny. Je ale těžké se v nich zorientovat, když uživatel hledá konkrétní vlastnost (např. jak nastavit barvu linie grafu). Chybí mi také nějaký přehled možností knihovny jiný než pouze příklady hotových vizualizací v galerii.

Náročnost práce se systémem V rámci testování obtížnosti práce s JavaScriptovou knihovnou D3.js jsem vytvořila graf funkce a její tečny v bodě. Hotová ukázka je na obrázku 1.4 a kompletní kód je dostupný na přiloženém médiu.



Obrázek 1.4: Výsledek testování vykreslovacího systému D3.js.

Knihovna je poměrně nízkoúrovňová. Na rozdíl od knihovny TikZ nabízí už částečně automatizované tvoření souřadnicových os a hodnot na nich, nicméně tento postup není příliš přímočarý. Při vytváření grafu je také nutné nejprve vytvořit měřítka – jakýsi přepočet mezi souřadnicovými systémy grafu a kořenového SVG elementu.

Vykreslení průběhu funkce je pracné – je třeba poskytnout data, funkci pro jejich přepočet do měřítka a mnoho dalších atributů pro nastavení vzhledu. Příklad vykreslení průběhu funkce je na výpisu 1.6.

```
const drawPath = (svg, data, xScale, yScale) => svg.append('path')
  .datum(data)
  .attr('fill', 'none')
  .attr('stroke', 'black')
  .attr('stroke-width', 1.5)
  .attr('d', d3.line()
    .x(function(d) { return xScale(d.x) })
    .y(function(d) { return yScale(d.y) })
  );

drawPath(
    svg,
    generateFunctionValues(functionFormula, {from: -0.5, to: 4.5}), x, y
).attr('stroke', 'blue');
```

Výpis kódu 1.6: Vykreslení průběhu funkce při vytváření ukázky 1.4 v systému D3.js.

Jako pozitivum bych vyzdvihla matematické možnosti knihovny math.js, která v této ukázce umožnila parsovat vzorce funkcí zadané pomocí textových řetězců, vypočítat derivace takových funkcí a získat hodnoty těchto derivací v bodech – což umožnilo dynamicky vykreslit tečnu funkce v bodě.

Interaktivitu jsem vzhledem k pracnosti vytváření statické ukázky netestovala.

Obecně pro mě práce s touto knihovnou nebyla pohodlná. Nabízí mnoho možností a je velmi flexibilní, ale právě díky tomu je obtížné naučit se s ní pracovat v krátkém čase – i pro zvládnutí základního vykreslení grafu je potřeba mnoho znalostí. Při práci jsem také často narážela na pro mě nepřehlednou dokumentaci a oficiální tutoriály, kde bylo náročné dohledat, jak dosáhnout potřebné funkcionality.

Shrnutí Knihovna D3.js nabízí velké množství funkcí a velkou flexibilitu zobrazení. Dala by se použít pro vykreslení 2D objektů a s použitím pluginu také pro vykreslení 3D objektů. Podporuje interaktivní změny hodnot

a samozřejmě také zobrazení na webu. Výhodou jsou matematické možnosti JavaScriptu jako jazyka a existence knihoven pro pokročilé matematické operace.

Jedná se nicméně o knihovnu zaměřenou spíše na datovou analýzu a použití pro matematické vizualizace tak není tak přímočaré, jako u ostatních nástrojů. Nízkoúrovňová orientace knihovny D3.js umožňuje velkou flexibilitu, ale zároveň znamená, že tvorba vizualizací je pracnější a je náročnější se ji naučit. Tomu nepomáhá ani dokumentace, která dle mého názoru není tak přehledná, jak by měla být, a nedostatečně představuje možnosti knihovny.

1.3.5 Raphaël

Podle [25] je Raphaël JavaScriptová knihovna, jejíž jméno je odvozeno od umělce Raffaella Sanzia da Urbina. Je zaměřena na umělce a grafické designéry.

Podle aktuální dokumentace [26] by knihovna *„měla usnadnit práci s vektorovou grafikou na webu.“* Základem pro tvoření grafiky v této knihovně je doporučení W3C pro SVG a VML. Cílem knihovny je snazší tvoření vektorové grafiky a poskytnutí adaptéru, který umožní kompatibilitu napříč prohlížeči.

Možnosti JavaScriptu jako jazyka, jeho knihoven a příbuzných technologií jsem hodnotila již dříve v sekci 1.2.3.

Vykreslení matematických objektů Podle tutoriálů [27, 28] má knihovna Raphaël následující možnosti.

- **Způsob vykreslení matematické funkce** – Pokročilé tvary, jako například právě grafy funkcí, je možné tvořit pomocí cest. Těm je možné určit výchozí bod, body na cestě a jestli má konkrétní dva body spojovat křivka nebo rovná čára (k tomu se používá specifikace totožná se specifikací SVG elementu `path`).

- **Podporované typy grafů** – Vzhledem k nízkoúrovňové povaze této knihovny jsou její možnosti velmi flexibilní – lze vytvořit libovolný typ grafu, lze zobrazovat více grafů různých typů v jednom souřadnicovém systému. Teto flexibilita však přináší značnou pracnost tvoření vizualizace.

- **Možnosti úpravy vzhledu grafu funkce** – Všem objektům je možné nastavit styl pomocí JavaScriptového objektu s atributy, jako je například barva výplně, barva linie nebo tloušťka linie.

- **Vykreslení geometrických tvarů** – Je možné kreslit jednoduché tvary jako obdélníky, kruhy a elipsy.

- **Podporované úpravy souřadnicových os** – Vzhledem k povaze této knihovny je nutné, aby uživatel vytvářel souřadnicové osy manuálně.

Možnosti pro jejich vzhled a měřítka jsou tedy velmi flexibilní za cenu obtížnějšího vytváření vizualizací.

- **Vkládání textových popisků** – Raphaël umožňuje vykreslit text, a to jak jednoduchý bez nastavitelného stylu, tak také pokročilejší, který umožňuje nastavit font, velikost písma, tloušťku písma a barvu.

Vykreslení 3D objektů Podle [29] existuje rozšiřující knihovna Raphaël3d, která umožňuje tvorbu 3D objektů. V dokumentaci se mi však nepodařilo dohledat podporu například pro povrchové grafy – musely by se tvořit manuálně z dílčích komponent.

Interaktivita V dokumentaci [26] se mi nepodařilo dohledat žádné zabudované interaktivní prvky ve formě widgetů zpracovávajících uživatelský vstup. Nicméně díky propojení s JavaScriptem a HTML by bylo pro interakci s uživatelem možné využít běžné HTML elementy.

Raphaël také dle [27] umožňuje pracovat s JavaScriptovými událostmi, jako jsou kliknutí myší nebo najetí myší nad objekt – ke každému elementu je možné připojit callback zavolaný po této události.

Zobrazení na webu Raphaël je dle [26] knihovna určená přímo pro zobrazení na webu. Navíc poskytuje širokou kompatibilitu napříč prohlížeči. Podpora zahrnuje prohlížeče Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera 9.5+ a Internet Explorer 6.0+.

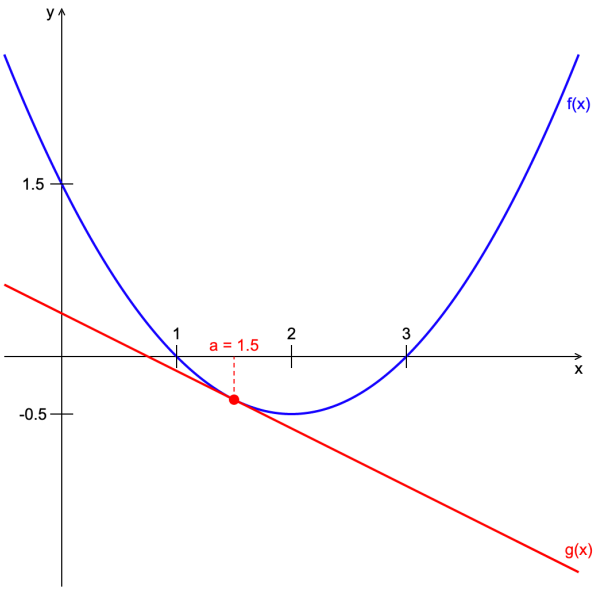
Dokumentace Oficiální dokumentace [26] obsahuje přehled všech metod často včetně příkladů použití. Chybí nicméně nějaký detailnější úvod, tutoriál nebo přehled základních schopností, takže je nutné pro pochopení základních principů využívat externí, často zastaralé zdroje.

Náročnost práce se systémem V rámci testování náročnosti práce s touto knihovnou Raphaël jsem vytvořila graf funkce a její tečny v bodě. Hotová ukázka je na obrázku 1.5, kompletní kód je k nahlédnutí na přiloženém médiu.

Raphaël je podobně jako TikZ velmi nízkoúrovňová knihovna, a tak je tvoření vizualizací poměrně pracné. Je nutné vytvořit manuálně osy a vyznačit na nich důležité body, stejně tak se manuálně přidávají potřebné popisky. Velkou nevýhodou oproti TikZ je, že systém souřadnic Raphaëlu má počátek v levém horním rohu a je tak třeba provádět přepočet, například pomocí funkce na výpisu 1.7, aby se dalo se systémem pohodlně pracovat.

Další velkou nevýhodou knihovny je způsob vykreslování linií, které v grafu reprezentují průběhy funkcí – je třeba používat složitou syntaxi SVG elementu `path`, jak je vidět ve výpisu 1.8.

Nízkoúrovňový přístup na druhou stranu přináší velkou flexibilitu v možnostech vykreslování. Tato flexibilita je navíc podpořena knihovnou math.js,



Obrázek 1.5: Výsledek testování vykreslovacího systému Raphaël.

kterou jsem zde stejně jako u knihovny D3.js použila pro pokročilé matematické operace.

Interaktivitu jsem vzhledem k pracnosti vytváření statické ukázky netestovala.

Celkově si myslím, že je knihovna pro použití snadno pochopitelná, i když je práce s ní časově náročná. Dle mého názoru by mohla více abstrahovat od místy složitější syntaxe SVG a být tak pro použití jednodušší.

Shrnutí Knihovna Raphaël je velmi jednoduchá a malá knihovna, která usnadňuje práci s vektorovou grafikou na webu. Je nízkoúrovňová, což znamená podporu pouze pro základní tvary a vykreslování cest. Tyto cesty by bylo možné použít pro vykreslení matematických funkcí, nicméně musely by se k nim manuálně tvořit osy i popisky. Díky propojení s JavaScriptem a HTML nemá knihovna problém s interaktivitou ani vykreslením na webu.

Podpora pro 3D zobrazení funkcí není v knihovně zahrnutá a rozšiřující knihovna ji také neposkytuje v dostatečné míře. Chybějící oficiální tutoriál nebo bližší úvod pak ztěžují pochopení základních principů knihovny.

```
// where the center point is in the model
const CENTER = {
  x: 100,
  y: 350,
};

// how many pixels represent one step on the graph
const SCALE = {
  x: 100,
  y: 100,
}

const getCanvasPoint = (point) => ({
  x: CENTER.x + point.x * SCALE.x,
  y: CENTER.y - point.y * SCALE.y,
});
```

Výpis kódu 1.7: Převod mezi různými systémy souřadnic při vytváření ukázky 1.5 v systému Raphaël.

```
const drawFunction = (paper, formula, range, style = {}) => {
  let path = ''
  for (let i = range.from; i <= range.to; i += 0.01) {
    const command = i === range.from ? 'M' : ' L';
    const point = getCanvasPoint({x: i, y: formula(i)});

    path += `${command} ${getTextPoint(point)}`;
  }
  const element = paper.path(path);
  element.attr(style);
}
```

Výpis kódu 1.8: Vykreslení průběhu funkce při vytváření ukázky 1.5 ve vykreslovacím systému Raphaël.

1.3.6 Plotly

Knihovna Plotly je dle [30] open source grafová knihovna, která existuje ve třech variantách – pro jazyky Python, R a JavaScript – a umožňuje tvorbu interaktivních grafů a map. V rámci této práce jsem se rozhodla zaměřit na dvě z těchto variant, konkrétně pro Python a pro JavaScript.

Plotly pro Python je dle své dokumentace [31] knihovna, která dovede

vytvořit interaktivní grafy v publikační kvalitě. Podporuje přes 40 různých typů grafů, od statistických, přes vědecké až po 3D grafy. Tato knihovna je postavená nad JavaScriptovou Plotly knihovnou a vytvořené vizualizace tak mohou být zobrazeny nejen v rámci Jupyter Notebooků, ale také uloženy do samostatných HTML dokumentů, případně mohou být součástí v Pythonu vytvořených webových aplikací s použitím frameworku Dash.

Dle své dokumentace [32] je Dash open source framework, který slouží k vytváření webových analytických aplikací čistě v jazyce Python, je ideální pro aplikace sloužící k vizualizaci dat. Staví na frameworcích a knihovnách Flask, Plotly pro JavaScript a React.js. Aplikace vytvořené ve frameworku Dash mohou být nasazeny na server a zobrazovány v prohlížečích, případně sdíleny pomocí URL adresy.

Plotly pro JavaScript je dle své dokumentace [33] *„vysokoúrovňová deklarativní grafová knihovna,“* která staví na knihovnách D3.js a stack.gl. Stejně jako sesterská knihovna pro Python podporuje širokou škálu různých typů grafů.

Možnosti Pythonu a JavaScriptu, jejich knihoven, balíčků a příbuzných technologií jsem hodnotila již dříve v sekcích 1.2.2 a 1.2.3.

Vykreslení matematických objektů Plotly pro Python dle své dokumentace [31] umožňuje reprezentovat grafy jako objekty `Figure`. Před předáním vykreslovací knihovně Plotly pro JavaScript dojde k serializaci do formátu JSON. Tuto serializaci je možné manuálně zobrazit a pracovat s ní.

Podle dokumentací variant knihovny pro Python [31] a JavaScript [33] se obě varianty ve svých možnostech v podstatě shodují. Jejich hlavní vlastnosti jsou následující.

- **Způsob vykreslení matematické funkce** – Ve variantě pro Python je možné vykreslit graf funkce pomocí seznamu bodů na ose x a seznamu hodnot, jichž v nich funkce nabývá. Ve variantě pro JavaScript je celý graf včetně všech vlastností popsán pomocí JavaScriptového objektu, v němž jsou mimo jiné také data pro vykreslení grafu zadaná pomocí polí – seznamů hodnot na jednotlivých osách.

- **Podporované typy grafů** – Obě varianty knihovny podporují bodové grafy, spojnicové grafy i sloupcové grafy. Umožňují zobrazení vícero grafů funkcí v rámci jednoho souřadnicového systému, přičemž tyto grafy nemusí být stejného typu. Zahrnuta je také podpora pro vrstevnicové grafy.

- **Možnosti úpravy vzhledu grafu funkce** – Je možné měnit barvu grafů, velikost bodů a případně tloušťku či styl linie grafu (plná čára, tečkovaná, čárkovaná atd.). Existují rovněž barevné šablony, které nastavují barvu a styl pozadí, stejně jako výchozí barvy grafů.

- **Vykreslení geometrických tvarů** – Do grafu je možné vkládat geometrické tvary jako například obdélníky a kruhy. Je možné zobrazovat úsečky v různých směrech, případně vertikální nebo horizontální přímky. Složitější tvary a křivky je možné vykreslit s použitím rozhraní pro vkládání cest podle specifikace SVG elementu `path`.
- **Podporované úpravy souřadnicových os** – Je možné použít lineární i logaritmické osy.
- **Vkládání textových popisků** – Obě varianty umožňují nastavit titulek grafu, popisky os a zobrazení legendy. Lze rovněž vkládat samostatné textové anotace, které mohou mít šipku pro určení přesného bodu, ke kterému se vztahují. Je možné formátovat text nastavením fontu, barvy a velikosti. Obě varianty umožňují navíc interpretaci textu sázeného v matematickém módu v LaTeX.

Vykreslení 3D objektů Obě varianty knihovny Plotly dle svých dokumentací [31, 33] obsahují vestavěnou podporu pro 3D grafy, a to od zobrazení bodových a spojnicových grafů ve vícedimenzionálním prostoru, až po povrchové grafy.

Interaktivita Plotly obecně [31, 33] umožňuje celou řadu základních interaktivních operací – přibližování a oddalování, posouvání grafu, přiblížení vybrané sekce. Je také možné schovávat a opětovně zobrazovat jednotlivé grafy v modelu klikáním na jejich popisky v legendě. Grafy v knihovně Plotly mají automatizované zobrazování tzv. tooltipů s výchozím obsahem po najetí myší nad konkrétní místo grafu. Jejich obsah lze následně upravit. V kombinaci s Dash je dle [32] také možné implementovat reakci na klikání a pohyby myši v grafu.

V základním stavu má knihovna dle dokumentace [31, 33] možnosti pro jednoduchou interaktivní reakci na uživatelský vstup – tyto možnosti spočívají zejména ve změně stylu grafu, případně schování/zobrazení předem předpočítaných datových sad.

Pro skutečně interaktivní dopočítávání hodnot v reakci na uživatelský vstup je pro verzi v Pythonu dle [31] třeba spojit knihovnu s frameworkem Dash, případně s Jupyter Notebookem, které oba nabízejí moderní interaktivní widgety pro zpracování uživatelského vstupu (jako například `text box`, `select`, `slider`, `range slider`, `radio button`), a které umožňují v reakci na uživatelský vstup spustit funkci v Pythonu [32, 34].

Tuto míru interaktivity by mělo být možné dosáhnout i pro JavaScriptovou variantu Plotly, a to propojením s běžnými HTML formulářovými prvky a JavaScriptovými událostmi. Dle dokumentace [33] může knihovna reagovat také na specifické události, jako například kliknutí na určité místo v grafu nebo najetí myší nad určité místo v grafu.

Zobrazení na webu Pro webové zobrazení vizualizací vytvořených v Plotly pro Python existuje dle dokumentace [31] několik variant.

- V dokumentaci často doporučovanou variantou je použití frameworku Dash pro postavení celé webové aplikace, kterou je poté možné nasadit na server. Takovéto použití umožní zapojit do vizualizace interaktivní widgety a ostatní interaktivní prvky a umožní reagovat na uživatelský vstup pomocí callbacků v Pythonu. Webová aplikace ve frameworku Dash může mít dle jeho dokumentace [32] vícero stránek (a tedy obsahovat vícero vizualizací) s různými URL adresami.

- Dále je možné využít Jupyter Notebooků, které podporují základní interaktivitu a poskytují widgety – tzv. `ipywidgets` – pro interaktivní zpracování uživatelského vstupu [34]. Plotly potom poskytuje infrastrukturu pro promítnutí aktualizovaných dat do existující vizualizace.

- Je rovněž možné staticky exportovat obrázky ve formátech PNG, JPG, SVG nebo PDF.

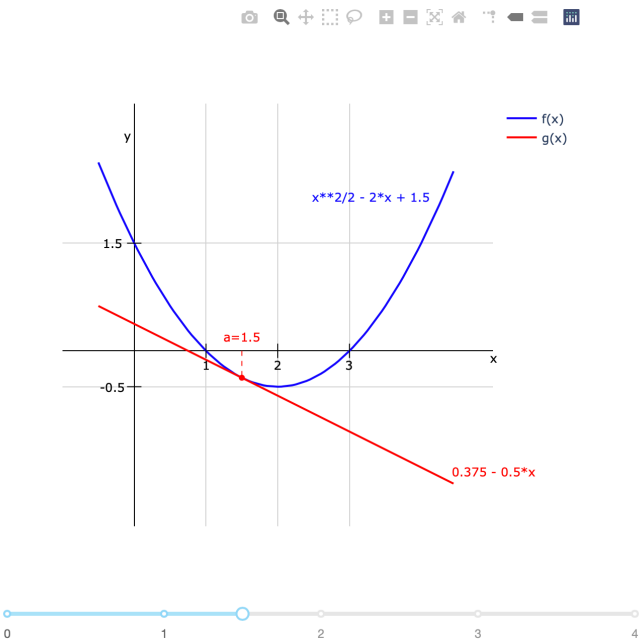
- Poslední možností je export do HTML – vytvoření samostatné HTML stránky, která si zachová základní interaktivní prvky, jako přibližování nebo zobrazování tooltipů. Je možné i zachování interakce v podobě změn vzhledu grafu a schování/zobrazení některé datové řady. V dokumentaci jsem nenalezla zmínku o podpoře dynamického přepočítávání hodnot při tomto způsobu exportu.

Webové zobrazení vizualizací vytvořených v Plotly pro JavaScript je pak přímočaré, jelikož je tato varianta pro zobrazení na webu přímo určená.

Dokumentace Dokumentace pro variantu Plotly pro Python dostupná na stránkách [31] pro mě byla na první pohled nepřehledná. Nicméně po chvilce jsem pochopila její strukturování a práce s ní se zjednodušila. Dokumentace obsahuje popis základních vlastností knihovny včetně ilustrace na mnoha příkladech. Problémy s nalezením informací jsem měla zejména při dohledávání pokročilých vlastností například v oblasti interaktivity.

Dokumentace varianty pro JavaScript [33] je méně propracovaná, obsahuje sice řadu příkladů, ale už méně vysvětlivek. Dokumentace zahrnuje užitečný „cheat sheet“ se základními nastaveními a ovládacími prvky.

Náročnost práce se systémem Při testování náročnosti práce s knihovnou jsem vyzkoušela obě její varianty na stejném příkladě jako předchozí systémy – na grafu funkce a její tečny. Začala jsem verzí pro Python, ve které jsem vyzkoušela i přidání interaktivity, hotová ukázka je na obrázku 1.6. Pracovala jsem ve frameworku Dash, kód je dostupný na přiloženém médiu.



Obrázek 1.6: Výsledek testování vykreslovacího systému Plotly pro Python.

Nejproblematičtější bylo stejně jako u předchozích knihoven pro Python nastavení souřadnicových os. U této knihovny se mi nepodařilo najít nastavení, které by upravovalo jejich polohu tak, aby procházely počátkem soustavy souřadnic. Situaci jsem vyřešila pomocí zvýraznění automaticky vytvořených linií procházejících počátkem, čímž jsem vytvořila falešné osy. Poté jsem skutečné osy schovala a na falešných jsem vyznačila sledované hodnoty.

Kromě toho jsem už na žádné další obtíže nenarazila a práce s knihovnou byla pohodlná a rychlá. Kód použitý k vykreslení všech součástí grafu (mimo souřadnicových os) je v podstatě ekvivalentní s knihovnami Bokeh a Plotly, jak je ilustrováno na ukázce kódu pro vytvoření hlavní funkce na výpisu 1.9. Stejně jako u obou jmenovaných knihoven jsem pro matematické výpočty použila knihovnu SymPy.

Přidání interaktivity bylo ve frameworku Dash velmi jednoduché – celou vizualizaci jsem vytvořila znovu zavoláním stejné funkce jako při jejím vytváření, jen s jinými vstupními parametry. Aktualizace je tak sice méně efektivní při provádění, ale její naprogramování je velmi jednoduché. Při práci s frameworkem Dash je dále nutné definovat podobu webové stránky s vizualizací, k čemuž je potřeba dodatečný kód. Stránka je ale díky tomu flexibilnější – může obsahovat doprovodný text atp. Celý kód definující podobu stránky a aktualizaci vizualizace je na výpisu 1.10.

Výsledek, který jsem vytvořila ve verzi Plotly pro JavaScript, byl z podstaty věci v podstatě totožný s vizualizací tvořenou v Pythonu (kromě inter-

```
# plot main function
f, f_expr = getFunctionFormula()
fvalues = np.array([f(x) for x in XVALUES])
fig.add_trace(go.Scatter(x=XVALUES, y=fvalues, name="f(x)", mode="lines",
    line={"color": "blue"}))
fig.add_annotation(x=5, y=f(4.3), text=r"$\frac{x^2}{2}-2x+\frac{3}{2}$",
    showarrow=False, font={"color": "blue"})
```

Výpis kódu 1.9: Vykreslení průběhu funkce při vytváření ukázky 1.6 v systému Plotly pro Python.

aktivních funkcí, které jsem zde netestovala). Hlavní rozdíl byl ve způsobu jeho tvoření – nepoužívají se funkce, místo toho se knihovně předá několik Javascriptových objektů obsahujících všechny potřebné informace. Tyto informace jsou v podstatě shodné s informacemi předávanými funkcím při tvorbě vizualizace v Pythonu, jsou jen jinak uspořádané. Atributy se často i stejně jmenují a navíc je možné jako podklad použít JSON vygenerovaný vizualizací v Pythonu. Hotový kód je k nahlédnutí na připojeném médiu.

Při tvorbě jsem narazila na stejný problém se souřadnicovými osami a také jsem ho stejným způsobem vyřešila. Ostatní elementy se tvořily poměrně přímočaře. Jako ilustraci připojuji kód definující hlavní funkci a její popisek – viz výpis 1.11. Pro matematické výpočty jsem využila knihovnu math.js.

Jako výhodu knihovny vnímám připravenost na interaktivitu – od zabudované sady interaktivních operací pro automatické zobrazení tooltipů po najetí myší nad místo v grafu.

Příjemněji se mi pracovalo s variantou v Pythonu, nebýt problémů se souřadnicovými osami, tak byla práce s ní bezproblémová. Práce s JavaScriptovou verzí byla o něco náročnější, ale po adaptaci na jiný způsob předávání informací bylo nakonec tvoření vizualizace velmi jednoduché. Jako nevýhodu u obou variant knihovny vnímám kvalitu dokumentace, některé informace se dohledávaly špatně, navíc komunita okolo knihovny zřejmě není příliš aktivní, a tak ani dohledávání řešení problémů v jiných zdrojích příliš nefungovalo.

Shrnutí Knihovna jako celek nabízí velmi zajímavé možnosti pro vykreslování jak 2D, tak také 3D grafů a modelů. Má vestavěné základní interaktivní prvky, mnoho možností pro úpravu a vylepšení vzhledu, případně pro přidání popisných prvků.

Varianta pro Python nabízí řadu možností zobrazení na webu. Při výběru konkrétní technologie pro účely této práce však je nutné brát do úvahy omezení pro úroveň interaktivity – pouze použití frameworku Dash nebo Jupyter Notebooku umožní plné interaktivní zpracování uživatelského vstupu.

Variantu pro JavaScript lze použít pro zobrazení na webu a není nutné

```
# ========== DISPLAY ==========

app.layout = html.Div(children=[
    dcc.Graph(
        id="tangent-graph",
        figure=createFigure(1.5),
    ),
    dcc.Slider(
        id="a-slider",
        min=0, max=4, value=1.5, step=0.25,
        marks={i: str(i) for i in range(0, 5)},
    ),
])

# ========== UPDATE ==========

@app.callback(
    Output("tangent-graph", "figure"),
    Input("a-slider", "value")
)
def update_figure(new_point):
    return createFigure(new_point)
```

Výpis kódu 1.10: Definice webového zobrazení a aktualizace modelu při interaktivní změně hodnot v ukázce 1.6 v systému Plotly pro Python.

omezovat interaktivní zpracování uživatelského vstupu. Problematická však je nedostačující dokumentace této varianty a její použití skrz méně přehledné JavaScriptové objekty s mnoha atributy.

1.3.7 Další vykreslovací systémy

Během rešerší dostupných vykreslovacích systémů jsem narazila také na několik knihoven, které sice neposkytují veškerou funkcionalitu potřebnou pro účely této práce, ale mají zajímavé vlastnosti.

Manim Manim je dle [35] knihovna pro Python a slouží k vytváření matematických animací. Jak je uvedeno v dokumentaci, projekt není zcela dokončený a stále je ve fázi vývoje. Přesto nabízí zajímavé možnosti.

Dle dokumentace [35] je tato knihovna schopná vygenerovat animované video ve formátech MP4 nebo GIF na základě poskytnutého kódu. Tento kód je navíc velmi jednoduchý – i komplikovanější animace je možné popsat pomocí několika málo řádek. Animace jsou tvořeny na základě interpolace mezi

```
const f = getFunctionFormula(func);
const line = {
    type: 'scatter',
    name: 'f(x)',
    x: xvalues,
    y: xvalues.map(x => f(x)),
    mode: 'lines',
    line: {
      color: 'blue',
    },
};
const annotation = {
  x: 5,
  y: f(4.3),
  text: '$\\frac{x^2}{2} - 2x + \\frac{3}{2}$',
  showarrow: false,
  font: { color: 'blue' },
};
```

Výpis kódu 1.11: Vykreslení průběhu funkce při vytváření ukázky v systému Plotly pro JavaScript (kód byl pro lepší popisnost mírně upraven).

dvěma matematickými objekty a jejich možnosti zahrnují postupné vykreslení, transformace na jiné objekty, posuny, změny průhlednosti, rotace a podobně.

Objekty, které je možné vytvořit, zahrnují dle dokumentace [35] jednoduché geometrické tvary jako čtverec, trojúhelník a kruh, ale i komplikovanější struktury jako jsou souřadnicové osy nebo grafy matematických funkcí – přičemž tyto grafy je možné zadávat přímo pomocí funkce, nejsou třeba pole hodnot jako u většiny dříve zmiňovaných nástrojů. Výhodou je také možnost vykreslit popisky pomocí matematického módu v LaTeX.

Celkově knihovna nabízí mnoho funkcí, které by pro účel této práce byly velmi vhodné. Nicméně je vytvořena tak, aby jejím výstupem byla hotová videa a chybí zde tak interaktivita, která je klíčovou součástí zadání.

Altair Altair je dle [36] další knihovna pro Python, která umožňuje deklarativní statistickou vizualizaci a je založená na gramatikách Vega a Vega-Lite. Podle svých stránek [37, 38] jsou Vega a Vega-Lite vizualizační gramatiky – jazyky pro popis interaktivních vizualizačních designů, které umožní například podle popisu ve formátu JSON vygenerovat webová zobrazení s použitím technologií Canvas nebo SVG, přičemž Vega-Lite je vysokoúrovňová a tedy pohodlnější na používání.

Dle příkladů v dokumentaci [36] je možné vytvořit sloupcové, bodové

i spojnicové grafy. Hodnoty funkce jsou popsány pomocí seznamů popisujících souřadnice bodů na grafu. Grafům je možné jednoduše upravovat vzhled a mají různé interaktivní prvky – dokáží mimo jiné reagovat na výběr určité sekce grafu (například dynamicky počítat průměr hodnot) nebo na uživatelský vstup z widgetů typu `radio button`, `select` nebo `slider`.

Zobrazení vizualizace je dle [36] možné pomocí JavaScriptového balíčku Vega-Embed, případně v rámci Jupyter Notebooků.

Rozhraní pro definici vizualizací je dle mého názoru o něco méně uživatelsky přívětivé než v jiných knihovnách pro Python – používá řetězení metod pro popis specifických vlastností výstupu a na první pohled komplikovanější přiřazení seznamů hodnot k jednotlivým osám. Nepodařilo se mi také nalézt možnosti pro vykreslení vícerozměrných grafů. Výhodou naopak jsou přímočaré možnosti pro zobrazení na webu.

Chart.js Chart.js je dle dokumentace [39] jednoduchá a flexibilní JavaScriptová knihovna pro tvoření grafů. Tato knihovna pracuje s elementy typu `canvas` a se skriptem vloženým do HTML kódu.

Graf je dle [39] popsán JavaScriptovým objektem se stanovenými atributy. Popis tak může být poněkud zdlouhavější, než například u knihoven typu Pyplot nebo Bokeh, ale je velmi přehledný. Graf funkce je popsán pomocí pole bodů na ose x a pole hodnot, kterých funkce v těchto bodech nabývá, případně pomocí pole souřadnic jednotlivých bodů na grafu. Je možné nastavit vzhled grafu. Vstupní objekt obsahuje i identifikaci typu grafu – knihovna podporuje 7 základních typů včetně spojnicového, sloupcového a bodového. Zajímavou vlastností spojnicového grafu je možnost vyhlazování křivky pomocí různých typů interpolace.

Dále knihovna umožňuje dle [39] používat různé typy souřadnicových os – lineární, nebo například logaritmické.

Graf může podle dokumentace [39] reagovat na běžné události typu kliknutí myší nebo najetí myší nad objekt. Knihovna neobsahuje vestavěné widgety pro interaktivní zpracování uživatelského vstupu, ale vzhledem k jejímu charakteru je to pochopitelné – je možné použít běžné HTML elementy a propojit je s grafem.

Velkou výhodou této knihovny je jednoduchost jejího použití ve webovém prostředí a přehledná definice grafů a jejich vzhledu. Je však nutné dodat, že knihovna je primárně určena spíše pro datovou analýzu než pro grafické zobrazení matematických výpočtů a na jejích možnostech je to velmi znatelné.

JSXGraph JSXGraph je dle své dokumentace [40] *„JavaScriptová knihovna pro interaktivní geometrii, vykreslování grafů funkcí, vykreslování schémat a vizualizaci dat ve webovém prohlížeči.“* Tato knihovna je open source a má široké interaktivní možnosti.

Dle [40] je možné například vykreslovat body, spojovat je přímkami nebo úsečkami, vykreslovat kružnice pomocí středu a bodu na jejich obvodu. Lze zobrazit souřadnicové osy a do nich vykreslovat grafy matematických funkcí pomocí JavaScriptové funkce pro jejich výpočet. Všem elementům je možné určit vzhled – barvu, velikost, tloušťku aj.

Možnosti pro interakci jsou dle dokumentace [40] také velmi široké. Body vykreslené v modelu je možné posouvat a sledovat tak, jaký má tento posuv význam na vytvořený model (např. jak ovlivňuje velikost kruhu či těžiště trojúhelníku). Je možné reagovat na uživatelský vstup ve formě např. textových polí. Jednotlivé modely je mezi sebou možné propojit a manipulace v jednom z nich tak může ovlivňovat zobrazení ve druhém.

V dokumentaci [40] se mi bohužel nepodařilo dohledat podporu pro 3D nebo alespoň vrstevnicové grafy.

Knihovna JSXGraph je velmi šikovná pro rychlé interaktivní vizualizace geometrie a geometrických vztahů, stejně tak pro vykreslování grafů matematických funkcí. Bohužel však chybí podpora pro 3D grafy nebo alespoň pro vrstevnicové grafy.

1.4 Zhodnocení nalezených řešení

Na závěr této kapitoly shrnu důležité vlastnosti vykreslovacích systémů a možnosti, které nabízejí. Výběr vykreslovacího systému založený na zjištěních z této kapitoly následně provedu v kapitole 3.

Možnosti jazyka Základem pro zmíněné vykreslovací systémy jsou tři různé jazyky – LaTeX, Python a JavaScript.

- LaTeX je pro potřeby této práce nejméně vhodný, jelikož jeho možnosti nezahrnují přesné matematické výpočty ani pokročilé matematické operace, navíc příliš nepodporuje interaktivitu.
- Python má širokou nabídku knihoven pro podporu vědeckých výpočtů a pokročilé matematiky. Knihovny, které jsou na něm postavené, prakticky vždy umožňují zobrazení na internetu ve formě Jupyter Notebooků a podporují interaktivitu.
- JavaScript má také knihovny pro vědecké výpočty, navíc je určený přímo pro webová zobrazení a je interaktivní – knihovny na něm postavené mohou využívat běžné HTML formulářové elementy.

Vykreslení matematických objektů Všechny zmíněné knihovny mají nějaký způsob pro vykreslování matematických funkcí. Zdaleka nejlepší, ale také nejméně častý způsob, je přímo poskytnutí matematické funkce jako vstupu

pro vykreslovací funkci – toto podporuje například TikZ. Častější je explicitní zadání seznamu souřadnic, přes které linie grafu funkce prochází (v různých formátech). Zde je nutné zabezpečit výpočet nad dostatečným množstvím bodů na ose x, aby byla výsledkem plynulá křivka a nikoli lomená čára. Možnosti nastavení vzhledu i vykreslení různých typů grafů (spojnicový, sloupcový, bodový) jsou většinou napříč všemi systémy velmi podobné. Stejně tak možnosti vykreslení popisků a dalších objektů, jako jsou různé geometrické tvary a horizontální či vertikální linie. U popisků je zajímavé, když umožňují zpracování pomocí syntaxe matematického módu systému LaTeX, jako to umožňuje například Pyplot nebo Plotly.

Vykreslení 3D objektů 3D grafy jsou méně často podporovaná funkcionalita. Většinou je nutné použít nějaký balíček třetí strany. Dobrou podporu má například Pyplot nebo Plotly.

Interaktivita Úroveň podpory interaktivity se mezi jednotlivými knihovnami velmi liší. Častá je podpora přibližování nebo posouvání grafu, případně zobrazení tooltipů při přejetí myší. Již vestavěné interaktivní widgety pro zpracování uživatelského vstupu nabízejí například Bokeh a Pyplot. JavaScriptové knihovny pak mohou využívat HTML elementy pro uživatelský vstup.

Zobrazení na webu Nejjednodušší pro zobrazení na webu jsou samozřejmě JavaScriptové knihovny. Knihovny pro Python mohou většinou využít možnosti pro webové sdílení Jupyter Notebooků. Některé knihovny (například Bokeh a Plotly) navíc mají i své vlastní způsoby sdílení ve formě samostatných HTML dokumentů, případně interaktivního serveru.

Dokumentace Dostupná oficiální dokumentace knihoven se často orientuje spíše na příklady než obsáhlejší strukturované tutoriály. Je potom mnohem náročnější zhodnotit celkově možnosti knihovny, případně nalézt řešení hledaného problému. Relativně kvalitní dokumentaci má například knihovna Bokeh.

Náročnost práce se systémem Knihovny lze rozdělit na nízkoúrovňové (TikZ, Raphaël a svým způsobem i D3.js) a vysokoúrovňové (Pyplot, Bokeh a Plotly v obou variantách). První skupina se vyznačuje flexibilitou, která je ale „vykoupena“ větší pracností tvoření vizualizací. Druhá skupina je co do pracnosti přívětivější, může se ale s větší pravděpodobností stát, že bude potřeba nějaká vlastnost, kterou knihovna z této skupiny kvůli své specializaci nemá (např. posunout souřadnicové osy tak, aby procházely počátkem soustavy souřadnic).

Kapitola 2

Modelové příklady

V této kapitole se zaměřím na vybrané modelové příklady vhodné pro interaktivní ukázky při výuce – vykreslení tečny a tečné nadroviny, vykreslení Taylorova polynomu, vizualizaci Darbouxovy konstrukce Riemannova integrálu, vizualizaci Langrangeovy metody pro vázané extrémy. Shrnu matematickou teorii potřebnou k vytvoření a pochopení ukázek. Popíši, jak budou tyto ukázky vypadat – jejich vstupy, výstupy, interaktivně nastavitelné parametry – a tento popis doplním o náčrtky ukázek pro názornější ilustraci.

2.1 Vykreslení tečny a tečné nadroviny

Podle [41, s. 84] tečna ke grafu funkce v daném bodě vystihuje okamžité chování této funkce v daném bodě. Konkrétně její „sklon“, tedy koeficient přírůstku. Obdobně podle [42] je tečná nadrovina objekt sídlící ve stejném prostoru jako graf funkce, který v nějakém smyslu nejlépe postihuje chování přírůstku funkce v daném bodě definičního oboru.

2.1.1 Tečna

Pro zadefinování pojmu tečna jsou nutné následující definice převzaté z přednášek [43, 44] a skript [41, s. 9] předmětu ZMA.

Rozšířená množina reálných čísel Množinu $\overline{\mathbb{R}} := \mathbb{R} \cup \{+\infty, -\infty\}$ nazýváme rozšířenou množinou reálných čísel (případně též rozšířenou reálnou osou).

Limita funkce Buďte f reálná funkce reálné proměnné a $a \in \overline{\mathbb{R}}$. Nechť f je definovaná na okolí bodu a, s možnou výjimkou bodu a samotného. Řekneme, že $c \in \overline{\mathbb{R}}$ je limitou funkce f v bodě a, právě když pro každé okolí H_c bodu c

existuje okolí H_a bodu a takové, že z podmínky

$$x \in H_a \setminus \{a\}$$

plyne

$$f(x) \in H_c.$$

Zapisujeme

$$\lim_{x \to a} f(x) = c, \quad \text{případně} \lim_{a} f = c.$$

Derivace funkce v bodě Nechť f je funkce definovaná na okolí bodu $a \in \mathbb{R}$. Pokud existuje limita

$$\lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

a její hodnota je konečná, řekneme, že funkce f je diferencovatelná v bodě a. Hodnotu limity nazveme derivací funkce f v bodě a a označíme $f'(a)$.

Tečna funkce v bodě Nechť existuje $f'(a)$ a je konečná. Tečnou funkce f v bodě a pak nazýváme přímku s rovnicí $y = f(a) + f'(a)(x - a)$.

2.1.2 Tečná nadrovina

Obdobou tečny ve vícerozměrném prostoru je tečná nadrovina. Pro její definování jsou potřeba následující definice z předmětu MPI [42].

Limita funkce více proměnných Řekneme, že funkce $f : D_f \to \mathbb{R}, D_f \subset \mathbb{R}^n$, má limitu $L \in \mathbb{R}$ v hromadném bodě $\boldsymbol{b}$ množiny D_f pokud

$$\forall H(L)\ \exists H(\boldsymbol{b})\ \boldsymbol{x} \in (D_f \cap H(\boldsymbol{b})) \setminus \boldsymbol{b} \implies f(\boldsymbol{x}) \in H(L).$$

Značení:

$$\lim_{\boldsymbol{x} \to \boldsymbol{b}} f(\boldsymbol{x}) = L.$$

Parciální derivace funkce více proměnných Označme jednotlivé proměnné jako $x_1, x_2, x_3, \ldots, x_n$. Parciální derivace funkce f ve směru x_i (nebo podle x_i) v bodě $\boldsymbol{b} = (b_1, b_2, \ldots, b_n) \in D_f$ takovém, že $\exists H(\boldsymbol{b}) \subset D_f$, je

$$\lim_{h \to 0} \frac{(b_1, b_2, \ldots, b_i + h, \ldots, b_n) - f(b_1, b_2, \ldots, b_i, \ldots, b_n)}{h} = L$$

pokud tato limita existuje. Značení:

$$\frac{\partial f}{\partial x_i}(\boldsymbol{b}) = L.$$

Gradient Gradient funkce f v bodě $\boldsymbol{b} \in D_f$ je vektor

$$\nabla f(\boldsymbol{b}) = \left(\frac{\partial f}{\partial x_1}(\boldsymbol{b}), \frac{\partial f}{\partial x_2}(\boldsymbol{b}), \ldots, \frac{\partial f}{\partial x_n}(\boldsymbol{b}) \right).$$

Pro spojitě diferencovatelnou funkci f ukazuje gradient směr jejího nejvyššího růstu.

Parciální derivace ve směru Nechť $\boldsymbol{v} \in \mathbb{R}^{n,1} = \mathbb{R}^n$, $\|\boldsymbol{v}\| = 1$. Derivace funkce f ve směru $\boldsymbol{v}$ v bodě $\boldsymbol{b} \in D_f$ takovém, že $\exists H(\boldsymbol{b}) \subset D_f$, je

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{b}) = \lim_{h \to 0} \frac{f(\boldsymbol{b} + h\boldsymbol{v}) - f(\boldsymbol{b})}{h}.$$

Pokud existuje gradient f v bodě $\boldsymbol{b}$ a pokud jsou všechny parciální derivace funkce f spojité na nějakém okolí bodu $\boldsymbol{b}$, pak platí

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{b}) = \nabla f(\boldsymbol{b}) \cdot \boldsymbol{v}$$

Tečna funkce více proměnných ve směru Z předchozí definice vyplývá, že přírůstek funkce v daném bodě umíme popsat pro jednotlivé směry $v \in \mathbb{R}^n$. Pro každý směr umíme v tomto směru najít tečnu funkce f zúžené na tento směr, tj. zúžené na $D_f \cap \{\boldsymbol{x} : \boldsymbol{x} = \boldsymbol{b} + t\boldsymbol{v}, t \in \mathbb{R}\}$. Toto zúžení lze chápat jako funkci jedné proměnné t, tedy $\mathbb{R} \to \mathbb{R}$, a u té umíme najít tečnu.

Tečná nadrovina Sjednotíme-li tečny ve všech směrech (v bodě $\boldsymbol{b} \in D_f$), dostaneme tečnou nadrovinu funkce f v bodě $\boldsymbol{b}$ (podmínkou pro existenci je spojitá diferencovatelnost f v bodě $\boldsymbol{b}$).

Rovnice této nadroviny je

$$z = \frac{\partial f}{\partial x_1}(\boldsymbol{b})(x_1 - b_1) + \frac{\partial f}{\partial x_2}(\boldsymbol{b})(x_2 - b_2) + \cdots + \frac{\partial f}{\partial x_n}(\boldsymbol{b})(x_n - b_n) + f(\boldsymbol{b}).$$

Její normálový vektor je

$$\left(\frac{\partial f}{\partial x_1}(\boldsymbol{b}), \frac{\partial f}{\partial x_2}(\boldsymbol{b}), \ldots, \frac{\partial f}{\partial x_n}(\boldsymbol{b}), -1 \right).$$

2.1.3 Popis ukázky

K tomuto tématu vzniknou celkem tři ukázky – tečna ke grafu funkce jedné proměnné, tečná nadrovina ke grafu funkce dvou proměnných a tečna ke grafu funkce dvou proměnných ve směru. Měly by sloužit k usnadnění pochopení pojmů „tečna v bodě“, „tečná nadrovina“ a „tečna v bodě ve směru“.

Ukázky budou pro spojitě diferencovatelnou funkci jedné proměnné zobrazovat její graf s tečnou v daném bodě. Pro spojitě diferencovatelnou funkci

dvou proměnných budou zobrazovat buď tečnou nadrovinu v daném bodě, nebo tečnu v daném bodě a směru.

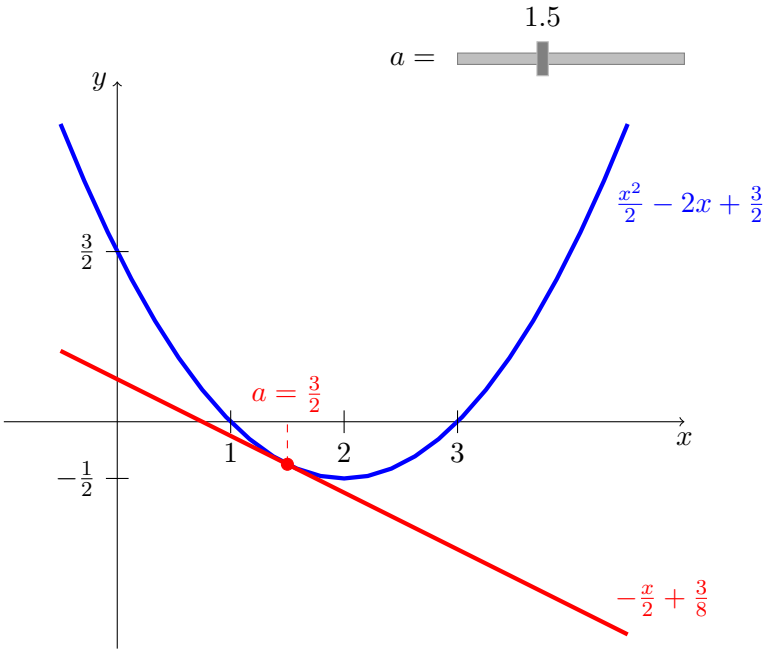
Bude možné interaktivně určovat bod na definičním oboru funkce, případně směr, ve kterém hledáme tečnu vícerozměrné funkce a zobrazovat tak tečny v různých bodech a různých směrech.

Vstup funkce (jedné nebo dvou proměnných)

Parametry bod z definičního oboru, eventuálně směr

Výstup graf funkce s tečnou v bodě (a v daném směru), případně tečná nadrovina

Na obrázku 2.1 je náčrtek ukázky pro dvourozměrný případ. Je na něm vidět zvolená funkce (modře) a její tečna (červeně) v bodě zvoleném pomocí posuvníku (vpravo nahoře).



Obrázek 2.1: Náčrt modelové ukázky pro vykreslení tečny ke grafu funkce jedné proměnné.

2.2 Vykreslení Taylorova polynomu

Podle skript k předmětu ZMA [41, s. 123] představuje tečna funkce f tzv. lineární aproximaci funkce f v bodě a – dobře vystihuje chování funkce v blízkosti

bodu a. Pro větší přesnost se místo přímek – polynomů prvního stupně – využívají polynomy vyšších stupňů. Taylorovy polynomy využívají derivace vyšších stupňů a jsou v jistém smyslu nejlepší možnou polynomiální aproximací k dané funkci. Lze odhadnout chybu aproximace Taylorovým polynomem [41, s. 127] a díky tomu je možné dodržet požadovanou přesnost aproximace.

2.2.1 Taylorův polynom

Nejprve je potřeba uvést definici derivace vyššího řádu a pojmu polynom. Po zavedení Taylorových polynomů je nezbytné se zmínit o zbytku v Taylorově vzorci, který nám umožňuje odhadnout chybu aproximace. Následující definice jsou převzaty ze skript k předmětu ZMA [41, s. 96–128].

Derivace vyššího řádu Rekurzivně definujeme derivace vyšších řádů (dokud existují) následovně:

$$f^{(0)}(x) = f(x), \quad f^{(n)}(x) = (f^{(n-1)})'(x), \quad n \geq 1.$$

Polynom Reálnou funkci reálné proměnné $p : \mathbb{R} \to \mathbb{R}$ nazveme polynomem, právě když existuje nezáporné celé číslo $n \in \mathbb{N}_0$ a reálná čísla $a_1, \ldots, a_n \in \mathbb{R}$ taková, že rovnost

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

platí pro všechna reálná $x \in \mathbb{R}$. Je-li $a_n \neq 0$, nazýváme číslo n stupněm polynomu p. Jsou-li všechny koeficienty $a_k, k = 0, \ldots, n$ nulové, nazýváme p nulovým polynomem a jeho stupeň nedefinujeme.

Taylorův polynom Nechť reálná funkce reálné proměnné f má v bodě $a \in \mathbb{R}$ konečnou n-tou derivaci. Potom existuje právě jeden polynom $T_{n,a}$ stupně nejvýše n takový, že

$$T_{n,a}^{(k)}(a) = f^{(k)}(a) \text{ pro každé } k = 0, 1, \ldots\ n.$$

Tento polynom má tvar

$$T_{n,a}(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

a nazýváme ho n-tým Taylorovým polynomem funkce f.

Taylorův vzorec Nechť funkce f má v bodě a konečnou n-tou derivaci. Pro všechna přípustná x položme $R_{n,a}(x) := f(x) - T_{n,a}(x)$. Potom vztah

$$f(x) = T_{n,a}(x) + R_{n,a}(x)$$

nazýváme Taylorovým vzorcem a $R_{n,a}$ nazýváme n-tým zbytkem v Taylorově vzorci.

Lagrangeův tvar zbytku Nechť existuje okolí H_a bodu a takové, že funkce f v něm má konečnou $(n+1)$-ní derivaci. Pak zbytek v Taylorově vzorci $f(x) = T_{n,a}(x) + R_{n,a}(x)$ lze pro každé $x \in H_a$ zapsat ve tvaru

$$R_{n,a}(x) = \frac{f^{(n+1)}(\xi)}{(n+1!)}(x-a)^{n+1},$$

kde číslo ξ závisí na x a n a leží uvnitř intervalu s krajními body x a a. Tento tvar zbytku nazýváme Lagrangeův.

2.2.2 Popis ukázky

Ukázka zde bude znázorňovat aproximaci funkce s konečnými derivacemi vyšších řádů pomocí Taylorova polynomu různých stupňů. Měla by usnadňovat pochopení Taylorova polynomu a zlepšování jeho aproximačních schopností s narůstajícím stupněm. Také by měla znázornit, jak se bude měnit tvar polynomu v závislosti na bodě a.

Bude možné interaktivně určovat stupeň polynomu n a měnit tak přesnost aproximace. Také bude možné měnit bod a Taylorova polynomu a ukázat tak, jak se polynom v závislosti na něm mění.

Vstup funkce jedné proměnné

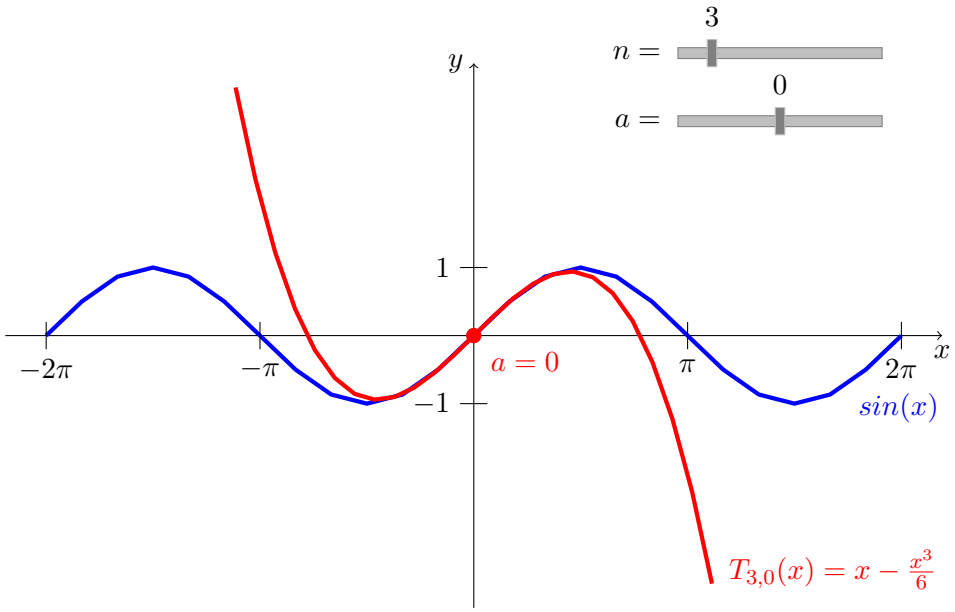
Parametry bod a stupeň Taylorova polynomu

Výstup graf funkce a graf její aproximace Taylorovým polynomem daného stupně v daném bodě

Na obrázku 2.2 je náčrtek ukázky. Modře je na něm ukázána aproximovaná funkce, červeně její aproximace pomocí Taylorova polynomu daného stupně n a s daným bodem a – tyto hodnoty jsou určeny pomocí posuvníků v pravé horní části náčrtku.

2.3 Vizualizace Darbouxovy konstrukce Riemannova integrálu

Skripta k předmětu ZMA [41, s. 146] uvádí, že geometrickou motivací pro výpočet určitého integrálu (také Riemannova, či Darbouxova) je *„výpočet obsahu plochy ohraničené grafem funkce a osou nezávisle proměnné.“* Jak je ale uvedeno dále [41, s. 158], tento obsah se počítá i se znaménkem – obsah plochy „pod“ touto osou bude mít znaménko záporné.



Obrázek 2.2: Náčrt modelové ukázky pro vykreslení aproximace funkce pomocí Taylorova polynomu.

2.3.1 Konstrukce určitého integrálu pro funkci jedné proměnné

Konstrukce Riemannova integrálu vychází v tomto případě ze znalosti obsahu obdélníka, jak uvádí [41, s. 146]. Základní myšlenka spočívá v aproximaci plochy pod grafem pomocí plochy sestavené z mnoha obdélníků. Pro matematický popis této konstrukce je nutné nejprve definovat základní pojmy. Popis a definice níže jsou převzaty z [41, s. 146–148].

Dělení intervalu Buď dán interval $\langle a, b\rangle$. Konečnou množinu

$$\sigma = \{x_0, x_1, \ldots, x_n\}$$

takovou, že

$$a = x_0 < x_1 < \cdots < x_n = b$$

nazýváme dělením intervalu $\langle a, b\rangle$. Bodům $x_k, k = 1, 2, \ldots, n-1$ říkáme dělící body intervalu $\langle a, b\rangle$. Intervalu $\langle x_{k-1}, x_k\rangle$ říkáme částečný interval intervalu $\langle a, b\rangle$ při dělení σ. Číslo

$$\nu := \max\{\Delta_k | k = 1, 2, \ldots, n\}, \quad \text{kde } \Delta_k := x_k - x_{k-1}, k = 1, 2, \ldots, n$$

nazýváme normou dělení σ.

Horní a dolní součet funkce Buďte funkce f definovaná a omezená na intervalu $J = \langle a, b\rangle$ a $\sigma = \{x_0, x_1, \ldots, x_n\}$ dělení intervalu J. Součty

$$S(\sigma, f) = \sum_{i=1}^{n} \Delta_i \sup_{\langle x_{i-1}, x_i\rangle} f \quad \text{a} \quad s(\sigma, f) = \sum_{i=1}^{n} \Delta_i \inf_{\langle x_{i-1}, x_i\rangle} f$$

nazýváme horním součtem funkce a dolním součtem funkce f při dělení σ.

Dolní, resp. horní, součty představují obsah plochy tvořené obdélníky pod, resp. nad, grafem funkce s podstavami tvořenými částečnými dělícími intervaly.

Horní a dolní integrál Pro funkci f definovanou a omezenou na uzavřeném intervalu $J = \langle a, b\rangle$ definujeme čísla

$$\overline{\int_a^b} f(x)\,dx := \inf\{S(\sigma) | \sigma \text{ dělení } J\} \quad \text{a} \quad \underline{\int_a^b} f(x)\,dx := \sup\{s(\sigma) | \sigma \text{ dělení } J\}$$

a nazýváme je horním resp. dolním integrálem funkce f na intervalu J.

Z tohoto způsobu konstrukce lze odvodit, že horní (resp. dolní) integrál představuje jakýsi horní (resp. dolní) odhad obsahu hledané plochy pod grafem. Pokud jsou tyto integrály stejné, pak má dobrý smysl mluvit o obsahu plochy pod grafem.

Určitý integrál Pokud pro funkci f definovanou a omezenou na uzavřeném intervalu J platí

$$\overline{\int_a^b} f(x)\,dx = \underline{\int_a^b} f(x)\,dx \in \mathbb{R},$$

pak jejich společnou hodnotu nazýváme určitým integrálem funkce f na intervalu J a toto číslo značíme symboly

$$\int_a^b f, \quad \text{případně} \quad \int_a^b f(x)\,dx.$$

2.3.2 Konstrukce určitého integrálu pro funkci dvou proměnných

Mějme nějakou funkci $f : D \to \mathbb{R}$ dvou proměnných, kde D je součin dvou intervalů $D = \langle a, b\rangle \times \langle c, d\rangle$. Graf této funkce si lze dle [45] představit jako část povrchu nějakého předmětu a jeho integrálem pak budeme počítat objem pod tímto grafem. Postup konstrukce určitého integrálu pro funkci dvou proměnných je pak z velké části podobný konstrukci pro funkci jedné proměnné. Definice a postup konstrukce v následující části jsou převzaty z přednášky předmětu MPI [45].

Horní a dolní součet funkce Nechť $\sigma_x = \{x_0, x_1, \ldots, x_n\}$ je dělení intervalu $\langle a, b\rangle$ a $\sigma_y = \{y_0, y_1, \ldots, y_m\}$ je dělení intervalu $\langle c, d\rangle$. Potom je součin $\sigma = \sigma_x \times \sigma_y$ dělením obdélníkové oblasti $D = \langle a, b\rangle \times \langle c, d\rangle$.

Označme dále

$$M_{i,j} = \sup\{f(x, y) : (x, y) \in \langle x_{i-1}, x_i\rangle \times \langle y_{j-1}, y_j\rangle\}$$

a

$$m_{i,j} = \inf\{f(x, y) : (x, y) \in \langle x_{i-1}, x_i\rangle \times \langle y_{j-1}, y_j\rangle\}.$$

Horní součet funkce f vzhledem k rozdělení σ pak definujeme jako

$$S_f(\sigma) = \sum_{i=1}^{n}\sum_{j=1}^{m} M_{i,j}(x_i - x_{i-1})(y_j - y_{j-1})$$

a dolní součet funkce f vzhledem k rozdělení σ ekvivalentně jako

$$s_f(\sigma) = \sum_{i=1}^{n}\sum_{j=1}^{m} m_{i,j}(x_i - x_{i-1})(y_j - y_{j-1}).$$

Horní a dolní integrál Horní integrál funkce f na D definujeme jako

$$\overline{\iint_D} f(x, y)\,dx\,dy = \inf\{S_f(\sigma) : \sigma \text{ je (obdélníkové) dělení } D\}$$

a dolní integrál funkce f na D ekvivalentně jako

$$\underline{\iint_D} f(x, y)\,dx\,dy = \sup\{s_f(\sigma) : \sigma \text{ je (obdélníkové) dělení } D\}.$$

Určitý integrál Pokud se horní a dolní integrál rovnají, nazýváme jejich společnou hodnotu (dvojitým) určitým integrálem funkce f na D a značíme ji

$$\iint_D f(x, y)\,dx\,dy, \quad \text{případně zkráceně} \quad \iint_D f.$$

Řekneme, že f je integrabilní na D.

2.3.3 Popis ukázky

Pro toto téma vzniknou dvě ukázky – konstrukce určitého integrálu funkce jedné proměnné a konstrukce určitého integrálu funkce dvou proměnných. Ukázky by měly usnadnit pochopení konstrukce určitých integrálů pro funkce jedné a dvou proměnných. Znázorní, jak tato konstrukce probíhá, a dokáží interaktivně ukázat zpřesňování odhadu se zmenšující se velikostí částečných intervalů.

Pro funkci jedné proměnné bude příslušná ukázka zobrazovat graf dané funkce na určeném intervalu, dělení tohoto intervalu na částečné intervaly

stejné velikosti a vizualizaci obdélníků, z nichž jsou počítány horní a dolní součty této funkce. Pro funkci dvou proměnných obdobně bude ukázka zobrazovat plochu grafu, dělení intervalů na osách x a y a vizualizaci kvádrů tvořících horní a dolní součet.

Bude možné interaktivně měnit krajní body intervalu (eventuálně intervalů v součinu pro funkci dvou proměnných) a měnit tak rozsah oblasti, jejíž obsah (resp. objem) je integrací počítán. Dále bude možné měnit velikost částečných intervalů a ukázat tak postupné zpřesňování odhadu plochy (objemu) pod grafem. Uživatel si také bude moci zvolit, zda chce zobrazit vizualizaci konstrukce horního nebo dolního součtu.

Vstup funkce jedné, případně dvou proměnných

Parametry interval z definičního oboru, případně součin intervalů z definičního oboru pro funkce dvou proměnných, dále velikost částečných intervalů

Výstup graf funkce a vizualizace konstrukce určitého integrálu

Na obrázku 2.3 je náčrtek ukázky pro dvourozměrnou variantu. Modře je na něm vidět graf funkce, červeně je pak vykreslena vizualizace konstrukce určitého integrálu. Je možné interaktivně nastavit velikost částečných intervalů a počáteční a koncový bod intervalu, na kterém je funkce integrována (pomocí posuvníků v pravé horní části). Rovněž lze zvolit druh součtu (horní či dolní), který bude zobrazen (rozbalovací nabídka v levé horní části).

2.4 Vizualizace Lagrangeovy metody pro vázané extrémy

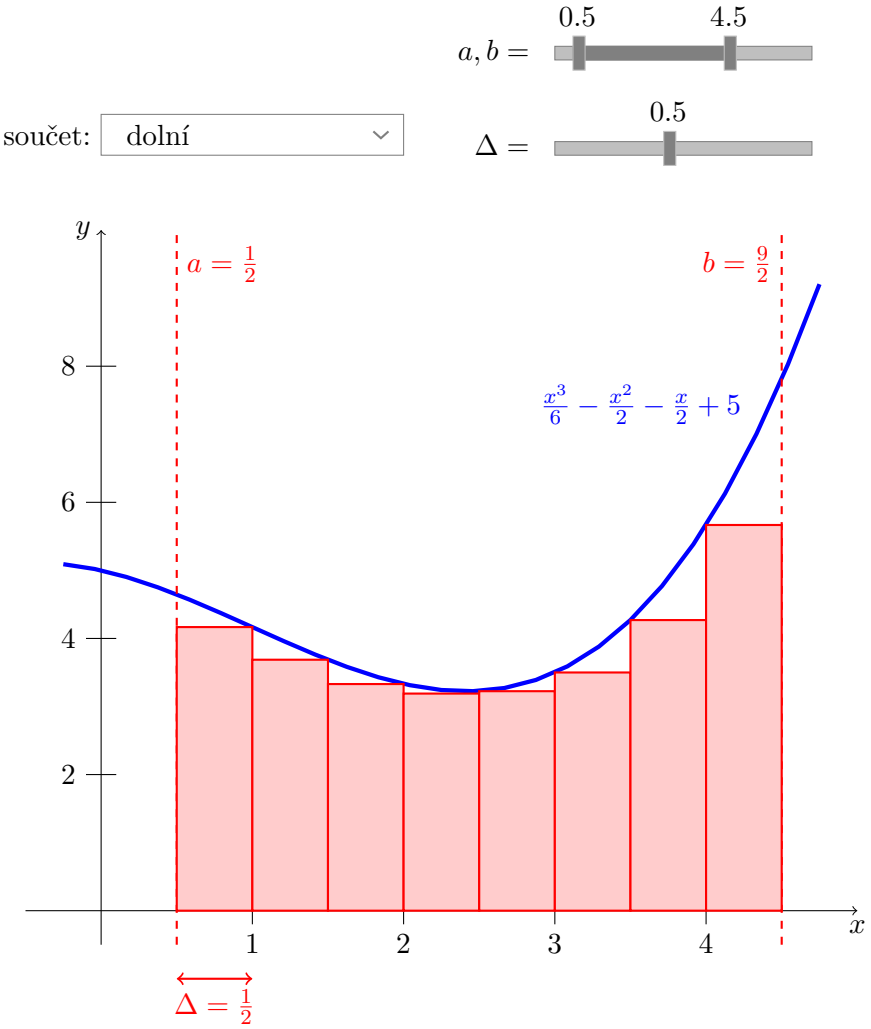
Dle přednášky o vázaných extrémech z předmětu MPI [46] hledá úloha vázaného extrému odpověď na otázku, jaké je minimum a maximum, když se pohybujeme na grafu funkce více proměnných pouze nad nějakou jeho částí. Matematicky přesná formulace toho problému je následující:

Formulace problému Označme $\hat{m} = \{1, \ldots, m\}$ a $\hat{p} = \{1, \ldots, p\}$. Úloha vázaného extrému (minima) je dle [46] obecně následující úloha:

$$\begin{cases} \text{minimalizuj} & f(x), \\ \text{za podmínek} & g_j(x) = 0, j \in \hat{m}, \\ & h_k(x) \leq 0, k \in \hat{p}, \end{cases}$$

kde f, g_j, h_k jsou funkce $D \to \mathbb{R}$, kde $D \subset \mathbb{R}^n$. Funkci f nazýváme objektivní, účelovou, minimalizovanou nebo optimalizovanou funkcí. Funkce g_j se nazývá rovnostní podmínka či vazba a funkce h_k nerovnostní podmínka či vazba.

Úloha vázaného maxima se definuje analogicky.



Obrázek 2.3: Náčrt modelové ukázky pro vizualizaci Darbouxovy konstrukce Riemannova integrálu.

2.4.1 Lagrangeova metoda

Pro nalezení řešení úlohy vázaného extrému je možné použít Lagrangeovu metodu [46]. Než uvedu její plné znění, je třeba zavést několik pojmů a definic převzatých z přednášek předmětu MPI [46].

Množina přípustných řešení Označíme množinu přípustných řešení:

$$\mathcal{M} = \{x \in \mathbb{R}^n : (\forall j \in \hat{m})(g_j(x) = 0) \wedge (\forall k \in \hat{p})(h_k(x) \leq 0)\}.$$

Úlohu lze tedy přeformulovat na hledání lokálního minima vzhledem k množině $\mathcal{M}$:

$$\forall x \in (H(x^*) \cap \mathcal{M}) \quad f(x^*) \leq f(x),$$

pro nějaké okolí $H(x^*)$, případně ostrého lokálního minima vzhledem k množině $\mathcal{M}$:

$$\forall x \in (H(x^*) \cap \mathcal{M} \setminus \{x^*\}) \quad f(x^*) < f(x)$$

pro nějaké okolí $H(x^*)$. Analogicky definujeme lokální maximum a ostré lokální maximum.

Lagrangeova funkce Funkci $L : \mathcal{M} \times \mathbb{R}^m \times \mathbb{R}^p_{\geq 0} \to \mathbb{R}$ definovanou

$$L(x; \lambda; \mu) = f(x) + \sum_{j=1}^{m} \lambda_j g_j(x) + \sum_{k=1}^{p} \mu_k h_k(x)$$

nazýváme Langrangeovou funkcí pro danou úlohu vázaného extrému (v tomto tvaru konkrétně minima). Koeficienty

$$\lambda = \{\lambda_1, \ldots, \lambda_m\} \quad \text{a} \quad \mu = \{\mu_1, \ldots, \mu_p\}$$

nazýváme Lagrangeovy multiplikátory.

Nechť $\widetilde{\mathcal{M}}$ je otevřená nadmnožina množiny $\mathcal{M}$, na níž pro všechny funkce $f, g_j, j \in \hat{m}, h_k, k \in \hat{p}$ existují druhé parciální derivace. Funkci L pak chápeme jako funkci $L : \widetilde{\mathcal{M}} \times \mathbb{R}^m \times \mathbb{R}^p_{\geq 0} \to \mathbb{R}$. Pro jednoduchost lze uvažovat rovnost $\mathcal{M} = \widetilde{\mathcal{M}} = \mathbb{R}^n$.

Množina aktivních omezení Pro bod $x \in \mathcal{M}$ definujeme množinu aktivních omezení jako:

$$\mathcal{B}(x) = \{k \in \hat{p} : h_k(x) = 0\}.$$

Postačující podmínka existence ostrého lokálního minima Nechť mají $f, g_j, j \in \hat{m}, h_k, k \in \hat{p}$ spojité všechny druhé parciální derivace na nějaké otevřené nadmnožině $\widetilde{\mathcal{M}} \supset \mathcal{M}$. Pokud trojice $(x^*; \lambda^*; \mu^*) \in \mathcal{M} \times \mathbb{R}^m \times \mathbb{R}^p_{\geq 0}$ splňuje podmínky:

1. (optimalita) $\forall(i), \frac{\partial L}{\partial x_i}(x^*; \lambda^*; \mu^*) = 0$,
2. (KKT) pro každé $k \in \hat{p}$ platí buď $\mu_k^* = 0$ nebo $h_k(x^*) = 0$,
3. (podmínka 2. řádu) pro každý vektor $0 \neq z \in \mathbb{R}^n$ splňující

$$z^T \nabla g_j(x^*) = 0, \quad \text{pro } \forall j \in \hat{m},$$

$$z^T \nabla h_k(x^*) = 0, \quad \text{pro } \forall k \in \mathcal{B}(x^*),$$

platí

$$z^T \nabla_x^2 L(x^*; \lambda^*; \mu^*) z > 0,$$

kde $\nabla_x^2 L$ je Hessova matice funkce L vzhledem k proměnným $x = (x_1, x_2, \ldots, x_n)$,

potom je x^* bodem ostrého lokálního minima úlohy vázaného lokálního minima.

Analogicky platí pro ostré lokální maximum.

2.4.2 Popis ukázky

Ukázka bude znázorňovat myšlenku věty o postačující podmínce existence ostrého lokálního extrému, zejména její první podmínky – optimality. Konkrétně by měla znázornit, že o lokální extrém při pohybu po vazbě může jít pouze pokud jsou gradienty funkce a vazby v daném bodě kolineární, tzn. pouze pokud se po vazbě pohybujeme ve směru vrstevnice hlavní funkce.

Konkrétně bude ukázka ve 2D grafu zobrazovat vrstevnice zadané spojité funkce, množinu přípustných řešení, bod z této množiny a gradienty funkce a vazby v tomto bodě. 3D graf není potřeba, jelikož je zkoumáno vzájemné chování gradientů, které lze lépe pozorovat právě v 2D pohledu „shora“.

Bude možné interaktivně měnit polohu bodu na vazbě a zobrazovat tak gradienty v různých místech vazby.

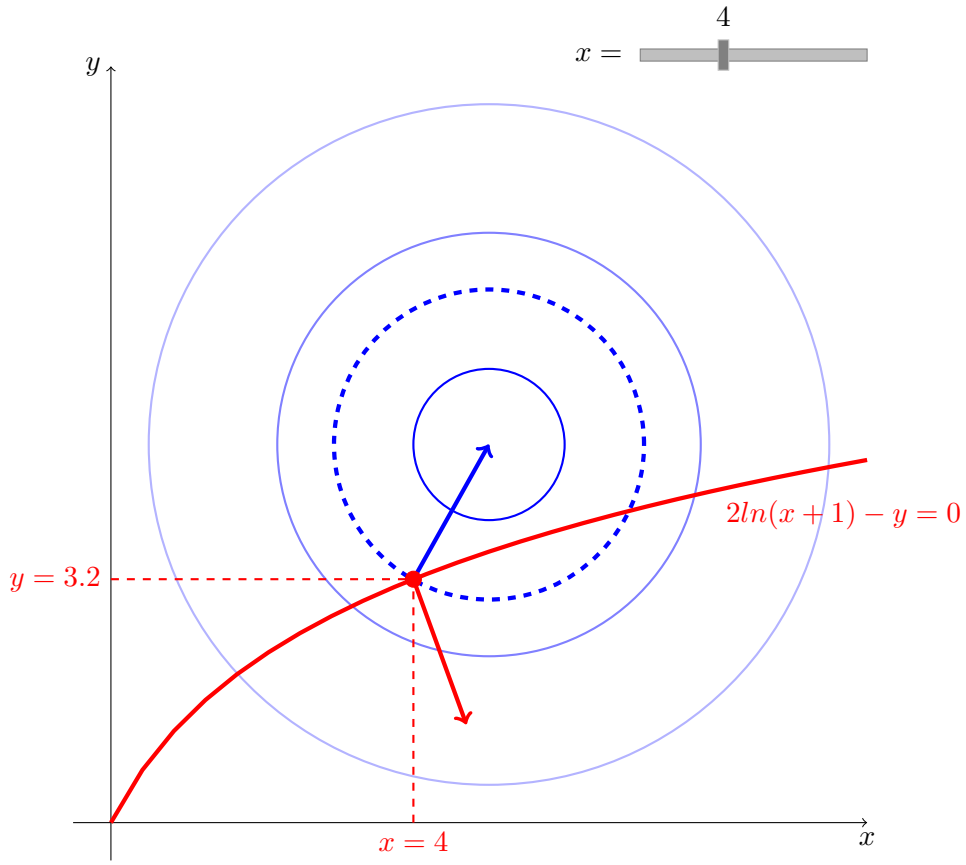
Ukázka bude bude omezena pouze na jednu vazební podmínku, aby byl její obsah snáze srozumitelný. Protože předmětem zájmu je chování gradientů při pohybu po rovnostních vazbách nebo po množině aktivních omezení, bude stačit sloučit tyto případy do jednoho a omezit zobrazované podmínky pouze na rovnostní.

Vstup funkce dvou proměnných, vazba

Parametry bod na vazbě

Výstup vrstevnice funkce, vazba, bod na vazbě, gradienty funkce i vazby v bodě

Na obrázku 2.4 je náčrtek této vizualizace. Modře jsou vyvedeny vrstevnice funkce, červeně příslušná vazba. Podle hodnoty x zvolené pomocí posuvníku v pravé horní části je dopočítána hodnota y. Na základě tohoto zvoleného bodu (x, y) je pak ve vizualizaci modře čárkovaně zobrazena vrstevnice funkce procházející tímto bodem, přesněji $\{(\overline{x}, \overline{y}) \in \mathbb{R}^2 : f(\overline{x}, \overline{y}) = f(x, y)\}$, a gradienty funkce i vazby v tomto bodě.



Obrázek 2.4: Náčrt modelové ukázky pro vizualizaci Lagrangeovy metody pro vázané extrémy.

Kapitola 3

Analýza a výběr vykreslovacího systému

V rámci této kapitoly stanovím funkční a nefunkční požadavky na výsledné řešení. Následně z těchto požadavků postupně podle jejich zaměření odvodím dvě skupiny požadovaných vlastností vykreslovacího systému, zhodnotím jejich splnění jednotlivými systémy a provedu případnou eliminaci systémů z dalšího výběru při zjištění závažných nedostatků. Na závěr kapitoly vyberu vykreslovací systém pro použití v následující implementační fázi.

3.1 Funkční a nefunkční požadavky

V této sekci stanovím funkční a nefunkční požadavky na výsledné řešení. Tyto požadavky budou mít posléze vliv jak na výběr systému pro další implementaci, tak také na návrh, implementaci, testování a nasazení řešení. Každý požadavek bude mít stanovené ID, název a popis.

3.1.1 Funkční požadavky

Funkční požadavky specifikují požadované funkcionality výsledného řešení. Stanovím jim prioritu pomocí systému MoSCoW (must have, should have, could have, won't have), abych odlišila nutné funkcionality a ty, které jsou přidanou hodnotou.

Implementace 2D modelových ukázek (F1) Výsledné řešení musí podporovat implementaci všech navržených 2D ukázek. Musí splňovat veškeré nezbytné nároky kladené na matematické výpočty, na vykreslovací možnosti a na podporu interaktivity.

- **Priorita:** must have

Implementace dalších modelových ukázek (F2) Výsledné řešení by mělo umožňovat také implementaci navržených 3D ukázek. Mělo by proto podporovat potřebné matematické výpočty a mělo by mít možnost tyto ukázky vykreslit. Mělo by mít potenciál pro vytváření různých dalších ukázek – to znamená co nejširší portfólio nabízených funkcionalit.

- **Priorita:** could have

Doplnění ukázek o zajímavé popisné a interaktivní prvky (F3) Výsledné řešení by mělo umožňovat také doplnění základních ukázek o dodatečné popisné prvky (textové anotace, vodící linie…) a zajímavé interaktivní možnosti (zobrazení tooltipů, otáčení a přibližování grafu…), které umožní předat uživateli další informace a usnadní mu pochopení ukázky.

- **Priorita:** should have

Interaktivní webové zobrazení (F4) Výsledné řešení bude možné zobrazit na webu a bude jej možné propojit se systémem MARAST. Webové zobrazení zároveň zachová plnou interaktivitu ve smyslu dynamického přepočítávání ukázky na základě uživatelských vstupů. Statické předpočítání hodnot není kvůli množství kombinací vstupních parametrů vhodné řešení.

- **Priorita:** must have

3.1.2 Nefunkční požadavky

Nefunkční požadavky popisují všechny ostatní nároky na navržené řešení. Zmiňují rozšiřitelnost, uživatelskou přívětivost, testování i výkon řešení.

Rozšiřitelnost (N1) Výsledné řešení musí být v maximální možné míře rozšiřitelné – musí umožňovat přidávání nových vizualizací. Proces přidání vizualizace by vedle jejího vytvoření a otestování neměl zahrnovat téměř žádné další kroky – to znamená, že uživatel mimo jiné nebude muset nastavovat výsledné řešení, aby přidanou vizualizaci zahrnulo jako svoji součást.

Jednoduchá tvorba vizualizací (N2) Výsledné řešení musí umožňovat jednoduchou tvorbu vizualizací. Pro definici vizualizace musí dostačovat přiměřené množství přímočarého a snadno pochopitelného kódu. Vizualizace musí být také jednoduše přizpůsobitelná pro webové interaktivní zobrazení. Musí existovat srozumitelná dokumentace postupu tvorby vizualizací.

Uživatelsky přívětivé zobrazení vizualizace (N3) Řešení musí zajistit, že výsledné vizualizace bude možné propojit se systémem MARAST uživatelsky přívětivým způsobem. Vizualizace by tedy měla být přímo součástí stránky nebo by měla být zobrazena po přesměrování na externí stránku bez dlouhého čekání.

Zabezpečení (N4) Řešení nebude podporovat žádný typ uživatelských účtů nebo zabezpečeného přihlášení. Bude přístupné každému, kdo bude znát jeho URL adresu.

Responzivita (N5) Řešení bude částečně responzivní. Bude uzpůsobeno na obrazovky širší než 1 000 px, ale bude funkční i na obrazovkách do minimální šíře 600 px.

Výkon (N6) Po konečném nasazení na produkční server musí řešení zvládnout obsloužit řádově stovky studentů připojujících se současně.

Pokrytí testy (N7) Automatizovanými testy musí být pokryty pomocné funkce, které jsou používány napříč ukázkami a které nepracují se systémovými funkcemi (například s modulem `os`). Alespoň manuálně je nutné otestovat všechny vytvořené vizualizace.

3.1.3 Výběr systému podle požadavků

Uvedené funkční a nefunkční požadavky ovlivní jak návrh řešení a jeho implementaci, kterým se budu věnovat v kapitole 4, tak také výběr vhodného vykreslovacího systému, který je předmětem této kapitoly. Tuto volbu ovlivní všechny uvedené funkční požadavky, ale také část nefunkčních požadavků (zejména požadavky N2 a N3). Dodržení ostatních nefunkčních požadavků závisí více než na výběru vhodného systému na správném návrhu, implementaci, testování a nasazení.

Požadavky ovlivňující výběr systému lze dále rozdělit do dvou skupin. První skupina obsahuje požadavky F1, F2, F3 a N2 a stanovuje požadavky na systém v závislosti na seznamu modelových ukázek sestaveném v kapitole 2. Druhá skupina obsahuje požadavky F4, N2 a N3 a týká se požadavků, které plynou ze zamýšleného interaktivního webového zobrazení. Podle těchto dvou skupin požadavků provedu v následujících sekcích této kapitoly výběr vykreslovacího systému pro použití v implementační části.

3.2 Požadavky na vykreslovací systém podle modelových ukázek

V této sekci shrnu vlastnosti, které vybraný vykreslovací systém musí mít, aby bylo možné jednoduše vytvořit modelové ukázky zmíněné v kapitole 2 – a splnit tak požadavky F1, F2, F3 a N2 definované v sekci 3.1. Následně vyhodnotím splnění těchto požadavků jednotlivými systémy a provedu jejich eliminaci v závislosti na těchto zjištěních.

3.2.1 Stanovení požadavků

Požadavky ze sekce 3.1 jsou stanoveny obecně. Sestavím proto podrobnější seznam požadovaných vlastností, které rozdělím do dvou skupin podle priority funkčních požadavků, z nichž jsou odvozeny – na nutné (pro vykreslení 2D ukázek) a výhodné (pro 3D ukázky a vylepšení 2D ukázek). Dále je pro jednodušší vyhodnocení rozdělím podle dotčené oblasti na požadavky na možnosti jazyka s ohledem na nutné výpočty, na vykreslovací možnosti systému a na interaktivní ovladače, které systém nabízí.

Nutné požadavky na matematické výpočty Při tvorbě vizualizace by co největší část matematických výpočtů měla být prováděna zvolenou technologií, nikoli manuálně. Tvorba vizualizace tak bude pohodlnější a zároveň bude ve vizualizaci zachována určitá obecnost, protože nebude závislá na několika málo předem vypočítaných hodnotách. Požadavky jsou proto následující:

- podpora základních matematických funkcí, jako jsou mocnina, exponenciální funkce, logaritmus, či trigonometrické funkce,
- podpora výpočtu n-té derivace funkce v bodě,
- vyhodnocení minima a maxima funkce na daném intervalu (výpočet může být pouze přibližný),
- podpora výpočtu gradientu funkce v bodě.

Nutné požadavky na vykreslovací možnosti Při výběru vykreslovacího systému jsou stěžejní jeho možnosti pro vizualizace. Podstatným kritériem je také náročnost používání jednotlivých funkcionalit. Požadavky jsem proto stanovila následovně:

- vykreslení souřadnicových os procházejících počátkem soustavy souřadnic a nastavení jejich popisků,
- vykreslení grafu funkce – tedy spojnicového grafu, vykreslení více takových grafů v rámci jednoho modelu,

- zvýraznění konkrétních bodů, případně vykreslení grafického elementu (malé kolečko nebo křížek) v místě daného bodu,
- vykreslení geometrických prvků (např. obdélníků) v rámci grafu na konkrétních souřadnicích (možné nahrazení této vlastnosti sloupcovým grafem za cenu nižší flexibility),
- ilustrace funkce dvou proměnných při pohledu „shora“ – například pomocí vrstevnic, případně barevné škály reprezentující hodnoty v konkrétních místech grafu funkce, kombinace tohoto zobrazení se spojnicovým grafem.

Nutné požadavky na interaktivní možnosti Výsledná vizualizace musí reagovat na uživatelský vstup – nějakým způsobem ho zpracovat a aktualizovat vizualizaci. Proto je podstatné, jaké možnosti pro zadávání uživatelského vstupu vykreslovací systémy nabízí. Vybraný systém musí nabízet alespoň:

- výběr číselné hodnoty z konkrétního rozsahu – např. `slider`,
- volba intervalu hodnot, tedy dvojice čísel ze stanoveného rozsahu – např. `slider` pro dvojici hodnot (tzv. `range slider`),
- výběr právě jedné konkrétní možnosti z nabídky několika variant – např. `select` nebo `radio button`.

Výhodné výpočetní možnosti Pro jednodušší výpočet s co nejméně řádky potřebného kódu a pro umožnění 3D vizualizací jsou vhodné některé další vlastnosti zvolené technologie:

- parsování vzorce matematické funkce tak, aby se následně dala tato funkce použít pro výpočty,
- výpočet parciální derivace funkce více proměnných a její další použití při výpočtech,
- výpočet vícenásobné parciální derivace (i podle různých proměnných).

Výhodné vykreslovací možnosti Tyto doplňující vykreslovací možnosti jsou v zásadě dvojího typu – v první řadě se jedná o popisné prvky, které umožní zvýšit informační hodnotu tvořeného modelu, a ve druhé jsou to funkcionality potřebné pro vytvoření 3D vizualizací:

- možnost upravovat, jaké konkrétní hodnoty budou zobrazeny na číselných osách,

- podpora popisných prvků – „vodících čar“ mezi grafem a hodnotou na ose, horizontálních a vertikálních čar, anotací (textových popisků) na libovolném místě grafu,
- vykreslení konkrétní vrstevnice, na které se zvolený bod na vazbě nachází (dostačující by bylo její naznačení například pomocí zvýraznění bodů, které se na ní nacházejí),
- vykreslení povrchových grafů, případně spojnicových grafů ve 3D prostoru a jejich spojování v rámci jednoho modelu,
- vykreslení kvádrů pod grafem funkce dvou proměnných – možnost pro vkládání 3D geometrických objektů (v krajním případě by mohl dostačovat i 3D sloupcový graf),
- popisné prvky ve 3D prostoru.

Výhodné interaktivní možnosti Tyto interaktivní prvky by umožnily intuitivnější a přirozenější práci s vizualizací, nejsou ale pro její funkčnost nezbytné:

- zobrazení tooltipů po najetí myší, základní manipulace s grafem – přibližování, posouvání,
- výběr bodu na grafu funkce pomocí kliknutí do daného místa.

3.2.2 Zhodnocení splnění požadavků

V této části zhodnotím splnění stanovených požadavků. Nejprve vyhodnotím splnění požadavků na výpočetní možnosti programovacími jazyky a dalšími technologiemi. Poté budu sledovat naplnění požadavků na vykreslování a interaktivitu jednotlivými výpočetními systémy.

Výpočetní možnosti V porovnání technologií podle výpočetních možností shrnutém v tabulce 3.1 dopadl nejhůře LaTeX ve spojení s balíčkem PGF – nejedná se o klasický programovací jazyk, a tak mnoho operací není podporovaných. Naopak nejlépe dopadl Python ve spojení s knihovnou SymPy, která podporuje všechny operace a počítání s ní je velmi pohodlné. JavaScript s knihovnou math.js pak podporuje pouze základní operace. Sice dokáže vypočítat vše potřebné, operace se ale často musí odvozovat a práce s ním není tak přímočará.

	Nutné požadavky	Výhodné požadavky
LaTeX (PGF)	Podpora základních funkcí jen s omezenou přesností. Nepodporuje pokročilé operace.	Operace nejsou podporovány.
Python (SymPy)	Podpora všech operací. Přesný výpočet minima i maxima. Přímočarý kód díky podpoře symbolických výpočtů.	Přímočará podpora operací, pohodlný výpočet.
JavaScript (math.js)	Podpora všech základních operací a první derivace. Pokročilé operace je třeba odvodit. Nepřesný výpočet minima a maxima.	Přímočará podpora parsování a parciální derivace, nutno odvodit výpočet vícenásobné parciální derivace.

Tabulka 3.1: Zhodnocení výpočetních možností jednotlivých programovacích jazyků a technologií ve spojení s konkrétními balíčky a knihovnami pro matematické operace (podle informací z dokumentací [3, 10, 13] a postřehů z vlastního testování technologií).

Vykreslovací možnosti Z detailního porovnání vykreslovacích systémů, které je shrnuto v tabulce 3.2, vyplývá, že nutné požadavky splňují všechny systémy, navzájem se však liší náročností používání – nejlépe použitelné jsou systémy Pyplot, Bokeh a Plotly. Doplňující 2D popisné prvky podporují rovněž všechny systémy. Vestavěnou podporu 3D vizualizací mají pouze systémy Pyplot a Plotly.

Interaktivní možnosti Podle detailního porovnání systémů v tabulce 3.3 mají všechny systémy s výjimkou TikZ, který v podstatě interaktivní není, nějakou úroveň podpory interaktivity. Nejlépe jsou na ni připraveny systémy Bokeh a Plotly – jsou přímo zamýšlené pro interaktivní použití, nabízí širokou škálu moderních widgetů a umožňují reagovat na události typu kliknutí nebo pohyb myši. Dobré interaktivní možnosti mají také D3.js a Pyplot.

	Nutné požadavky	Výhodné požadavky
TikZ	Graf funkce přímo ze vzorce. Manuální tvorba os. Manuální vykreslení vrstevnic.	Podporuje 2D popisné prvky. 3D projekce pouze pomocí externího balíčku.
Pyplot	Podporuje všechny požadavky.	Podporuje 2D popisné prvky. Podporuje také 3D povrchové, spojnicové, sloupcové a bodové grafy.
Bokeh	Podporuje všechny požadavky. Nepodporuje klasický vrstevnicový graf – náhrada pomocí obrázku s hodnotami rozlišenými barevnou škálou.	Podporuje 2D popisné prvky. Podpora 3D objektů pouze pomocí externích knihoven.
D3.js	Podporuje všechny požadavky. Náročnější na používání.	Podporuje 2D popisné prvky. 3D objekty pouze pomocí externího pluginu.
Raphaël	Manuální tvorba os. Nutnost přepočítávání souřadnicového systému. Funkce pomocí SVG `path`. Manuální vrstevnice.	Podporuje 2D popisné prvky. 3D objekty pouze pomocí externí knihovny.
Plotly	Podporuje všechny požadavky. Problematický posun os do počátku.	Podporuje 2D popisné prvky. Podporuje také 3D povrchové, bodové a spojnicové grafy. Kvádry je možné reprezentovat pomocí sítí trojúhelníků ve 3D prostoru.

Tabulka 3.2: Zhodnocení vykreslovacích možností jednotlivých systémů (podle informací z kapitoly 1 a vlastních zkušeností z jejich testování).

	Nutné požadavky	Výhodné požadavky
TikZ	Omezená interaktivita.	Podpora chybí.
Pyplot	Podpora potřebných widgetů s výjimkou `range slider`. Zastaralý vzhled widgetů.	Podpora přibližování, posouvání a také reakce na polohu a kliknutí myší.
Bokeh	Podpora všech požadovaných widgetů a mnoha jiných. Moderní vzhled.	Podpora přibližování, posouvání i tooltipů, reakce na kliknutí nebo pohyb myší v grafu.
D3.js	Lze použít HTML formulářové elementy (chybí podpora pro `range slider`).	Podpora přibližování, posouvání i tooltipů, reakce na kliknutí nebo pohyb myší v grafu.
Raphaël	Lze použít HTML formulářové elementy (chybí podpora pro `range slider`).	Umožňuje reagovat na kliknutí nebo pohyb myší v grafu.
Plotly.py	Například v kombinaci s Dash podpora všech požadovaných widgetů a také mnoha jiných. Moderní vzhled.	Podpora přibližování, posouvání, tooltipů, v kombinaci s Dash také reakce na kliknutí nebo pohyb myší v grafu.
Plotly.js	Lze použít HTML formulářové elementy (chybí podpora pro `range slider`).	Podpora přibližování, posouvání, tooltipů, reakce na kliknutí nebo pohyb myší v grafu.

Tabulka 3.3: Zhodnocení interaktivních možností jednotlivých systémů (podle informací z kapitoly 1 a vlastních zkušeností z jejich testování).

3.2.3 Eliminace systémů v závislosti na splnění požadavků

Na základě nedostatečných výpočetních možností systému LaTeX, omezené podpory interaktivity a vysoké náročnosti na vytváření vizualizací musím z dalšího výběru vyřadit systém TikZ.

Zejména kvůli vysoké náročnosti vytváření vizualizací, ale také kvůli nedostatečné kvalitě dokumentace vyřazuji z výběru také knihovny Raphaël a D3.js.

V dalším výběru se tedy budu soustředit na systémy Pyplot, Bokeh a Plotly, přičemž u posledního jmenovaného budu kvůli výpočetním možnostem programovacích jazyků brát v úvahu primárně variantu pro Python.

3.3 Požadavky na vykreslovací systém s ohledem na nasazení v systému MARAST

V této sekci analyzuji možnosti nasazení vizualizace v systému MARAST. Následně stanovím požadavky na vykreslovací systémy, které plynou z funkčních a nefunkčních požadavků ze sekce 3.1 a které se týkají publikování na webu a propojení se systémem MARAST. Vyhodnotím splnění těchto požadavků vykreslovacími systémy v jejich jednotlivých variantách používaných pro zobrazení na webu. Na závěr z dalšího výběru vyřadím systémy, které stanovené požadavky nesplňují.

3.3.1 Analýza možností nasazení

Pro zobrazení vizualizace na stránce v systému MARAST je několik variant, které se dělí do dvou základních skupin – buď bude vizualizace na stránce viditelná přímo, nebo bude dostupná pomocí odkazu a přesměrování na externí webovou stránku.

Zobrazení přímo na stránce Pro uživatele nejjednodušší možností by bylo zobrazení vizualizace přímo na stránce v systému MARAST. Viděli by související kontext, nemuseli by přecházet na jinou stránku, nemuseli by čekat, než se vizualizace načte. Pro takovéto zobrazení jsou v zásadě tři možnosti:

- export vizualizace do statického obrázku (vizualizace přijde o požadovanou interaktivitu),
- export vizualizace do samostatného HTML, CSS a JavaScriptu a vložení tohoto kódu do stránky v systému MARAST,
- zobrazení samostatné webové stránky s vizualizací v rámci stránky v systému MARAST, například pomocí technologie `iframe` [47].

Přesměrování na externí webovou stránku Méně preferovanou možností pak je URL odkaz na stránce v systému MARAST, který uživatele přesměruje na samostatnou webovou stránku s vizualizací. V takovéto situaci by bylo vhodné doprovodit odkaz například statickým obrázkem, aby uživatel věděl, co má po kliknutí na odkaz očekávat. Výhodou by také byla možnost vložit do webové stránky s vizualizací doprovodný text, aby uživatel neztratil kontext, ke kterému se vizualizace vztahuje. Existují různé druhy samostatných stránek, které se mohou po kliknutí na odkaz otevřít:

- samostatná webová stránka obsahující pouze vizualizaci a případný doprovodný text,
- částečné statický Jupyter Notebook, ve kterém bude interaktivní pouze vizualizace a který by kromě vizualizace obsahoval také kód potřebný k jejímu vytvoření (pomocí technologie nbviewer [6]),
- plně interaktivní Jupyter Notebook obsahující interaktivní vizualizaci včetně kódu potřebného k jejímu vytvoření, který by bylo možné měnit (pomocí technologie Binder [7]).

3.3.2 Stanovení požadavků

Z úkolu zobrazit interaktivní vizualizace v rámci webové stránky a z požadavků stanovených v sekci 3.1 (konkrétně F4, N2 a N3) plynou tři základní požadavky na vykreslovací systém – vizualizace si musí při zobrazení na webu zachovat plnou interaktivitu, interaktivní a webové funkce musí být co nejjednodušší na naprogramování a výsledná vizualizace musí zobrazena co nejvíce uživatelsky přívětivě.

Zachování plné interaktivity vizualizace Vedle většinou zabudovaných interaktivních funkcí typu přibližování, posouvání a zobrazování tooltipů je nutné, aby vizualizace dokázala dynamicky přepočítávat vizualizované hodnoty v závislosti na změnách vstupních parametrů prováděných uživatelem. Statické předpočítání vizualizovaných hodnot není dle požadavku F4 vhodné, mimo jiné také kvůli množství kombinací parametrů a s tím souvisejícím velkým objemem dat.

Jednoduchost vytváření interaktivní vizualizace Vytvoření interaktivní vizualizace obvykle úzce souvisí se zvoleným způsobem webového zobrazení. V tomto bodě se proto budu zajímat především o náročnost aktualizace vizualizace po uživatelském zadání nových dat – kolik kódu je potřeba a jak moc je tento kód přímočarý. Účelem je zajistit, aby přidání interaktivity nevedlo k neúměrnému ztížení tvorby vizualizace.

Jednoduchost zobrazení interaktivní vizualizace na webu Pro umožnění zobrazení na webu je občas nutné přidat k vizualizaci nějaký kód navíc. Tento kód typicky určuje vzhled a rozložení webové stránky s vizualizací, může ale mít i jiné funkce. V tomto bodě budu posuzovat pouze kód, který je nutný psát při vytváření každé vizualizace, nikoli jednorázovou konfiguraci.

Uživatelsky přívětivé zobrazení hotové vizualizace Uživatelé musí mít dle požadavku N3 možnost zobrazit ukázku jednoduše a rychle. Tedy ji buď vidět přímo na stránce, kde se nacházejí, nebo být přesměrováni na jinou webovou stránku s vizualizací bez dlouhého čekání. Ukázka by neměla být z různých důvodů nedostupná a její aktualizace při změně parametrů by měla být dostatečně rychlá.

3.3.3 Zhodnocení splnění požadavků

V této části zhodnotím možnosti pro nasazení na webu pro zbývající tři vykreslovací systémy – Pyplot, Bokeh a Plotly. Pro každý systém nejprve shrnu jeho možnosti pro webové zobrazení a poté vyhodnotím, jak tyto jednotlivé varianty splňují požadavky zmíněné v předchozí části.

Pyplot Vizualizaci vytvořenou v systému Pyplot je dle závěrů kapitoly 1 možné na webu zobrazit jako statický obrázek, jako součást samostatné webové stránky (při použití Matplotlibu na webovém aplikačním serveru) nebo jako součást sdíleného Jupyter notebooku.

- **Zachování interaktivity** – Podle kapitoly 1 postrádá export do statického obrázku interaktivitu zcela, použití Matplotlibu na webovém aplikačním serveru pak kvůli pravděpodobnému přenosu vizualizace jako statického obrázku neumožní funkce jako přibližování apod. Těmito možnostmi se proto již dále nebudu zabývat. Plnou interaktivitu je schopno zaručit pouze sdílení Jupyter Notebooku s vizualizací, nevýhodou však zůstane zastaralý vzhled použitých widgetů.
- **Aktualizace vizualizace** – Kód potřebný pro aktualizaci vizualizace není dle závěrů kapitoly 1 zanedbatelný. Pro vytvoření widgetů je třeba manuálně nastavit mnoho parametrů a také samotná aktualizace hodnot je poměrně pracná.
- **Nastavení webového zobrazení** – Webové sdílení Jupyter Notebooku s vizualizací skrz službu Binder by mělo být téměř bezpracné [7].
- **Uživatelská přívětivost** – Vzhledem ke kódu pro aktualizaci vizualizace psaném v Pythonu a z toho plynoucí nutnosti využít službu Binder pro sdílení by bylo nutné se při zobrazení vizualizace spokojit s delší dobou načítání [11].

Po vyhodnocení vlastností není Pyplot a potažmo Matplotlib pro webové zobrazení vizualizace tohoto typu příliš vhodný. Jedinou variantou sdílení na webu je vytvoření vizualizace v rámci Jupyter Notebooků. Jejími nevýhodami jsou především množství kódu doprovázející interaktivní prvky a dlouhé načítání při sdílení v rámci Jupyter Notebooku.

Bokeh Vyjma statického obrázku (kterému se dále nebudu věnovat) lze dle kapitoly 1 vizualizaci vytvořenou v Bokehu zobrazit na webu třemi způsoby – pomocí samostatného HTML dokumentu, Bokeh serveru a Jupyter Notebooku.

- **Zachování interaktivity** – Všechny tři varianty sdílení umožňují dle závěrů kapitoly 1 zachování plné interaktivity.
- **Aktualizace vizualizace** – Kód potřebný pro aktualizaci vizualizace je dle kapitoly 1 kvůli využívání specifické třídy `ColumnDataSource` poměrně zdlouhavý a méně přehledný.
- **Nastavení webového zobrazení** – Zobrazení na webu je téměř bezpracné pro varianty s callbacky v Pythonu (tedy pro Bokeh server a Jupyter Notebook sdílený pomocí služby Binder) [22, 7]. Samostatné dokumenty HTML a Jupyter Notebooky sdílené pomocí služby nbviewer ale vyžadují pro zachování interaktivity callbacky v JavaScriptu [22], které s sebou přinášejí řadu nevýhod. Představují další programovací jazyk, který by musel tvůrce vizualizace ovládat. Jsou zadávány formou textových řetězců a musí se tedy počítat s omezenou podporou ze strany IDE. Znamenaly by také omezené použití matematických knihoven Pythonu jako například SymPy a nutnost spokojit se s méně pokročilými matematickými knihovnami JavaScriptu.
- **Uživatelská přívětivost** – Z předchozí analýzy možností nasazení a závěrů kapitoly 1 plyne, že pro uživatelské zobrazení je nejpohodlnější samostatný HTML dokument, který by se dal vložit přímo do systému MARAST. Podobně vhodnou variantou je také Bokeh server, na kterém bude aplikace buď dostupná ihned po kliknutí na odkaz, nebo vložená do stránky v systému MARAST např. pomocí technologie `iframe`. Ihned po kliknutí na odkaz by bylo možné zobrazit také Jupyter Notebook sdílený pomocí technologie nbviewer. Notebook sdílený pomocí služby Binder se déle načítá a jeho otevření by trvalo delší dobu.

Po zhodnocení je Bokeh server jedinou variantou, která splňuje všechny požadavky – jediný umožňuje využít využít callbacků psaných v Pythonu a zároveň nezpůsobovat zdržení při spouštění ukázky. Jeho nevýhodou však zůstává nepřímočarý kód potřebný pro aktualizaci vizualizace při změně parametrů. Je nutné zmínit také varování z dokumentace [22] před použitím jednoho serveru pro sdílení vícero aplikací od různých tvůrců.

Plotly Vizualizaci vytvořenou v Plotly je na webu možné zobrazit třemi způsoby (vyjma exportu do statického obrázku) – v rámci webové aplikace ve frameworku Dash, v rámci Jupyter Notebooku s využitím tzv. `ipywidgets` a exportovanou do HTML stránky.

- **Zachování interaktivity** – Plnou interaktivitu dle kapitoly 1 poskytují v tomto případě pouze první dvě varianty – samostatný HTML dokument neumožňuje dynamické zpracování uživatelského vstupu a předpočítávání všech potřebných sad hodnot by pro vizualizace s vícero parametry nebylo kvůli množství kombinací schůdné. Proto se touto variantou dále již nebudu zabývat.

- **Aktualizace vizualizace** – Kód potřebný pro aktualizaci vizualizace ve frameworku Dash je dle kapitoly 1 minimální – celou vizualizaci je možné vytvořit znovu s jinými parametry. Kód potřebný pro aktualizaci vizualizace v Jupyter Notebooku by byl dle ukázek v [31] o něco složitější. Při aktualizaci průběhu funkce je např. nutné z modelu vybrat konkrétní datovou řadu podle pořadí jejich vytváření a aktualizovat její hodnoty a parametry – přičemž zejména výběr řady dle indexu by mohl být matoucí a problematický.

- **Nastavení webového zobrazení** – Jak je uvedeno v kapitole 1, pro zobrazení na webu je při použití frameworku Dash potřeba kód navíc pro určení rozložení webové stránky. Při vícero vizualizacích v rámci jedné webové aplikace je také třeba implementovat potřebnou infrastrukturu [32]. Nicméně tuto infrastrukturu postačuje vytvořit pouze jednou a možnost určit rozložení webové stránky s vizualizací lze chápat jako výhodu – na stránku je možné vložit nadpis, doprovodný text popisující kontext ukázky a další hodnoty. Vše je navíc možné provázat s hodnotami v interaktivních widgetech a celá stránka je tak velmi flexibilní. Webové zobrazení vizualizace v rámci Jupyter Notebooku pomocí služby Binder je téměř bezpracné [7].

- **Uživatelská přívětivost** – Z předchozí analýzy vyplývá, že pro uživatelské zobrazení je nejvýhodnější samostatný HTML dokument, který jsem ale již dříve kvůli nedostatečné interaktivitě z výběru vyřadila. Vhodná bude nicméně také webová aplikace vytvořená s pomocí frameworku Dash – vizualizaci bude možné zobrazit buď po přesměrování bez dlouhého čekání na načtení nebo po vložení do stránky např. pomocí `iframe`. Nejméně vhodný bude v tomto případě JupyterNotebook – jelikož jsou callbacky psané v Pythonu, nebude možné použít nbviewer a při použití Binderu by na načtení vizualizace bylo nutné čekat [11].

Zobrazení vizualizace v rámci samostatného HTML dokumentu je tedy nevhodné kvůli omezené interaktivitě a zobrazení v rámci Jupyter Notebooku

kvůli čekání při načítání a málo přehlednému kódu potřebnému pro aktualizaci vizualizace. Nejlépe ze srovnání opět vychází samostatná webová aplikace s využitím frameworku Dash, i když i ta má svou nevýhodu v podobě kódu potřebného pro určení rozložení webové stránky s vizualizací.

3.3.4 Eliminace systémů v závislosti na splnění požadavků

Vhodné způsoby pro webové sdílení, které zachovávají interaktivitu a jsou dostatečně jednoduché na vytváření a uživatelsky přívětivé na zobrazování, jsou v závislosti na předchozím vyhodnocení pouze dva – Bokeh v rámci Bokeh serveru a Plotly pro Python v rámci webové aplikace ve frameworku Dash.

Knihovnu Pyplot jsem musela z výběru odstranit. Bylo by nutné ji používat v rámci Jupyter Notebooku sdíleného pomocí služby Binder, což by přineslo delší načítací dobu. Knihovna na mě také působí dojmem, že v porovnání se zbývajícími dvěma není na interaktivitu na této úrovni tak dobře připravená (interaktivní funkce jsou i poměrně nepřehledně dokumentované). I když by bylo možné všech potřebných funkcí dosáhnout, bylo by to v porovnání s ostatními knihovnami náročnější. Další nevýhodou je zastaralý vzhled widgetů.

Samostatné HTML dokumenty u knihoven Bokeh a Plotly jsem musela z výběru bohužel vyřadit i přes to, že by jejich webové zobrazení bylo velmi uživatelsky přívětivé. V prvním případě by byla taková vizualizace příliš náročná na vytvoření (kvůli nutnosti psát části kódu v JavaScriptu) a ve druhém by nemohla nabídnout požadovanou úroveň interaktivity.

Nakonec použití v rámci Jupyter Notebooků u knihoven Bokeh a Plotly bylo nutné vyřadit buď kvůli dlouhému načítání vizualizace (nutnost použít službu Binder), nebo neúměrné náročnosti pro její vytvoření (nutnost psát callbacky v JavaScriptu). Jupyter Notebooky by navíc vždy obsahovaly také kód potřebný pro vytvoření vizualizace, který by mohl odvádět pozornost od zamýšleného sdělení.

3.4 Výběr systému

V této sekci přistoupím k volbě vykreslovacího systému pro použití v rámci implementační části. Nejprve mezi sebou porovnám výhody obou systémů a poté provedu a zdůvodním konečný výběr.

V předchozích sekcích jsem zúžila výběr pouze na dva systémy a vybrala jsem pro ně také nejvhodnější varianty pro publikování na webu – konkrétně se jedná o knihovnu Bokeh použitou v rámci Bokeh serveru a knihovnu Plotly použitou v rámci webové aplikace s frameworkem Dash.

3.4.1 Porovnání výhod systémů Bokeh a Plotly

Knihovny Bokeh a Plotly jsou si podle předchozích zjištění v mnoha ohledech velmi podobné – obě slouží pro vytváření vizualizací v Pythonu, poskytují široké možnosti pro interaktivitu a nabízí moderní vzhled. Každá má nicméně určité výhodné vlastnosti, které ta druhá postrádá. Tyto výhody shrnuji v tabulce 3.4.

Bokeh	kvalitní dokumentace jednodušší nastavení webového zobrazení bezproblémový posun os do počátku systému souřadnic
Plotly	podpora 3D zobrazení podpora vrstevnicového grafu podpora sazby matematických formulí v LaTeX jednodušší aktualizace vizualizace po změně parametrů přímočaré a flexibilní nastavení dalšího obsahu webové stránky použití jedné aplikace pro vizualizace od vícero autorů

Tabulka 3.4: Porovnání hlavních vzájemných výhod systémů Bokeh a Plotly (informace pochází z analýzy provedené v předchozím textu).

3.4.2 Výběr a odůvodnění

Z tabulky uvedené v předchozí části je vidět, že knihovna Plotly má oproti knihovně Bokeh výhod o něco více. Výhody na straně Plotly považuji navíc za velmi významné – podpora 3D zobrazení, přímočará podpora vrstevnicových grafů, celková jednoduchost interaktivity a flexibilita webového zobrazení jsou důležité a potřebné vlastnosti, které mohou být pro výsledné řešení významnou přidanou hodnotou.

Jedinou významnější nevýhodou této knihovny je dokumentace, která je pro mě osobně v porovnání s dokumentací knihovny Bokeh méně přehledná. Nicméně práce s ní není natolik problematická, aby způsobila výrazně horší použitelnost knihovny.

V další implementační části tedy budu používat knihovnu Plotly pro Python v rámci webové aplikace vytvořené s pomocí frameworku Dash. Vytvořím jednu webovou aplikaci, která bude obsahovat všechny vizualizace – každou odlišenou svým specifickým URL.

Kapitola 4

Implementace

V této kapitole přejdu k popisu implementace webové aplikace s ukázkami definovanými v kapitole 2. Nejprve se budu věnovat technologiím zvoleným pro implementaci. Dále navrhnu obecnou strukturu aplikace, strukturu implementace ukázek a postup jejich vytváření. Poté popíši každou implementovanou ukázku a zmíním nejzajímavější body jejího vypracování. Na závěr zmíním testování aplikace a její nasazení na server a na systém MARAST.

4.1 Zvolené technologie

V této sekci uvedu seznam nejdůležitějších technologií, které budou použity při implementaci. Pro jednotlivé technologie následně shrnu jejich účel v aplikaci a některé jejich důležité funkce.

Pro vývoj jsem zvolila tyto technologie:

- **Python** – programovací jazyk a základ pro veškerou implementaci,
- **SymPy** – knihovna pro matematické výpočty,
- **NumPy** – knihovna pro práci s poli,
- **Plotly** – knihovna pro vytváření interaktivních vizualizací,
- **Dash** – knihovna pro zobrazení vizualizací na webu a pro interaktivní zpracování uživatelských vstupů.

4.1.1 Python

Jak jsem již uvedla v kapitole 1, Python je interpretovaný, dynamicky typovaný programovací jazyk. Jako programovací jazyk pro implementaci ukázek v rámci této práce byl zvolen kvůli svým možnostem pro matematické výpočty a kvůli vysoké úrovni knihoven pro tvorbu interaktivních matematických vizualizací zobrazitelných na webu.

Pro implementaci používám verzi Pythonu 3.9.1, která byla dle [4] v době zahájení implementace nejnovější dostupnou verzí.

4.1.2 SymPy

SymPy je dle poznatků z kapitoly 1 knihovna pro symbolické výpočty v Pythonu. V rámci implementace bude sloužit jak pro převod textových řetězců s matematickými výrazy do jejich programové reprezentace, tak pro provádění všech komplexních matematických operací (zejména derivací).

Převod textových řetězců na SymPy výrazy Jednou z velmi důležitých schopností knihovny SymPy je převod textové reprezentace matematického výrazu do jeho programové podoby tak, že je možné do zpracovaného výrazu dosazovat hodnoty proměnných a zjišťovat hodnotu výsledku.

Tato schopnost je ilustrovaná na funkci `get_function_formula`, jejíž kód je na výpisu 4.1. Tato funkce převádí textový řetězec se vzorcem matematické funkce na lambda funkci, která pro zadané x vypočítá funkční hodnotu $f(x)$ v tomto bodě definičního oboru.

```
1  def get_function_formula(function_string):
2      x = sp.symbols("x")
3      func_expr = sp.sympify(function_string)
4      return sp.lambdify(x, func_expr, "numpy")
5
6  f = get_function_formula("x**2 + 5")
7  result = f(2)       # result = 9
```

Výpis kódu 4.1: Funkce pro převod textového řetězce se vzorcem matematické funkce na lambda funkci pomocí knihovny SymPy, včetně příkladu zavolání a použití (kód zjednodušen pro popisnost).

V uvedené funkci jsou použity tři základní funkce knihovny SymPy, popsané v dokumentaci [10]:

- `symbols` – definice proměnné x pro další výpočty,
- `sympify` – převod textového řetězce do interní SymPy reprezentace,
- `lambdify` – převod z interní reprezentace na lambda funkci, která umožní rychlejší vyhodnocení výsledku při dosazování mnoha různých hodnot.

Běžné dosazování hodnot proměnných do SymPy výrazu je v knihovně dle [10] umožněno pomocí metody `subs`. Tato metoda však dle dokumentace [10] není vhodná pro mnohonásobné opakování s různými hodnotami

proměnných. Proto je v ukázce pomocí funkce `lambdify` ze SymPy výrazu odvozena lambda funkce s operacemi jiné knihovny (v ukázce NumPy), která provádí totožné dosazení, ale pro opakovaný výpočet je z hlediska výkonu vhodnější.

Derivace a určitý integrál Derivace bude v ukázkách potřeba v různých variantách – první derivace funkce jedné proměnné, n-tá derivace funkce jedné proměnné, vícenásobná částečná derivace funkce dvou proměnných a výpočet gradientu. Podle dokumentace [10] se pro tyto výpočty používají funkce `diff` a `derive_by_array`, jejichž konkrétní použití je ilustrováno ve výpisu kódu 4.2.

Podobně existuje dle dokumentace [10] také funkce `integrate`, která dokáže vypočítat mimo jiné určitý (i vícerozměrný) integrál daného výrazu na daném intervalu.

```
1  # první derivace SymPy výrazu "expr" podle proměnné "x"
2  diff_expr = sp.diff(expr, x)
3
4  # třetí derivace SymPy výrazu "expr" podle proměnné "x"
5  diff_expr = sp.diff(expr, x, 3)
6
7  # derivace SymPy výrazu "expr" nejprve podle proměnné "x" a poté podle "y"
8  diff_expr = sp.diff(expr, x, y)
9
10 # výpočet gradientu SymPy výrazu "expr" s proměnnými "x" a "y"
11 gradient = sp.derive_by_array(expr, [x, y])
```

Výpis kódu 4.2: Výpočet derivací a gradientu s pomocí knihovny SymPy.

4.1.3 NumPy

Knihovna NumPy má dle analýzy provedené v kapitole 1 mnoho různých využití, v rámci této implementace ji ale použiji zejména pro práci s poli. Tato pole budou sloužit jako vstupní hodnoty pro vykreslení vytvářených vizualizací.

Důležitá je proto například funkce `arange`, která dle dokumentace [8] dovede vygenerovat sekvenci hodnot v zadaném intervalu a s konkrétním krokem. Například výsledkem volání této funkce v podobě `arange(0, 1, 0.2)` je pole hodnot `[0, 0.2, 0.4, 0.6, 0.8]`.

4.1.4 Plotly

Pro vytváření interaktivních vizualizací jsem v předchozích kapitolách vybrala knihovnu Plotly ve variantě pro Python. Tato knihovna dovede vytvořit vizu-

alizaci, nastavit její vzhled a přidávat do ní jednotlivé prvky.

Vytvoření modelu a nastavení obecných vlastností Ve vizualizacích použiji modul `graph_objects`, ve kterém se dle mého názoru složitější modely složené z více různých datových řad vytvářejí přehledněji.

Nový model – tzv. `figure` – se dle dokumentace [31] tvoří pomocí konstruktoru `Figure()`. Většina úprav vzhledu konkrétní `figure`, od nastavení titulku až po nastavení souřadnicových os, se pak může provést například pomocí metody `figure.update_layout`, která má za tímto účelem velké množství pojmenovaných (keyword) argumentů.

Přidávání datových řad a dalších objektů Datová řada – tzv. `trace` – je základním prvkem modelu. Pro účely této práce může reprezentovat například průběh grafu matematické funkce. Datové řady mohou tvořit také vodící linie, vyznačení bodů a jiné popisné prvky.

Datovou řadu je do modelu dle [31] možné přidat například pomocí metody `figure.add_trace`, která jako argument dostane konkrétní nastavený objekt datové řady. Typy datových řad používaných v implementaci jsou například `Scatter` pro spojnicové a bodové grafy, `Contour` pro vrstevnicové grafy, `Scatter3d` pro spojnicové a bodové grafy ve 3D prostoru a `Surface` pro povrchové grafy. Konstruktory datových řad přijímají opět řadu pojmenovaných argumentů pro nastavení svých vlastností. Nejpodstatnějšími argumenty jsou `x`, `y` a případně `z`, které popisují souřadnice jednotlivých bodů grafu. Platí, že dvojice (resp. trojice) čísel na i-tém indexu v těchto polích tvoří kompletní souřadnice jednoho bodu grafu.

Pro ilustraci použití těchto argumentů slouží kód ve výpisu 4.3 na řádcích 1–3. Tento kód přidává do modelu jednoduchý červený spojnicový graf procházející třemi danými body.

Použití datové řady typu `Scatter` pro vykreslení grafu funkce ilustruje kód ve výpisu 4.3 na řádcích 5–10. Nejprve je získána lambda funkce pro $f(x) = x^2 - 5$ – tato funkce pro zadanou souřadnici na ose x doplní odpovídající souřadnici na ose y tak, aby výsledný bod byl na grafu funkce. Poté je s pomocí NumPy vygenerováno pole `xvalues` – pole bodů z definičního oboru funkce. K těmto hodnotám je doplněno pole `yvalues` odpovídajících funkčních hodnot. Společně tato pole popisují řadu bodů ležících na grafu dané matematické funkce, jejichž propojením je možné graficky znázornit její průběh – granularita tohoto „vzorkování“ je natolik jemná, že se průběh funkce zdá hladký. Na závěr je do modelu přidána datová řada s odpovídajícími argumenty.

Jako popisné prvky pak je možné dle [31] do modelu přidávat například různé tvary pomocí metody `figure.add_shape`, případně textové anotace pomocí metody `figure.add_annotation`.

```
# jednoduchý spojnicový graf procházející body (1, 0), (2, 6) a (3, 4)
fig.add_trace(go.Scatter(x=[1, 2, 3], y=[0, 6, 4], line_color="red",
    mode="lines"))

# graf funkce vykreslený pomocí spojnicového grafu
f = get_function_formula("x**2 - 5")
xvalues = np.arange(1, 3, 0.05)
yvalues = np.array([f(x) for x in XVALUES])
fig.add_trace(go.Scatter(x=xvalues, y=yvalues, line_color="red",
    mode="lines", line_width=1))
```

Výpis kódu 4.3: Přidání jednoduchého spojnicového grafu a grafu funkce do modelu s využitím knihovny Plotly.

4.1.5 Dash

Jak jsem zmiňovala v kapitole 3, Dash je framework pro vytváření webových aplikací v jazyce Python. Tyto aplikace pak mohou být nasazeny na server a zobrazovány v prohlížečích. Dash je přímo dokumentací Plotly [31] doporučován jako nejlepší způsob pro implementaci webových aplikací s modely vytvořenými pomocí této knihovny.

V této implementaci bude mít framework Dash dvě základní funkce – určení rozložení webové stránky a zpracování uživatelského vstupu v rámci interaktivní aktualizace vytvořených modelů.

Určení rozložení webové stránky Pomocí frameworku Dash je dle [32] možno určit rozložení (tzv. „layout“) webové stránky – kromě Plotly modelu může stránka obsahovat také nadpis, doprovodný kontext a v neposlední řadě widgety pro umožnění interaktivity.

Rozložení webové stránky je v tomto frameworku dle [32] definováno pomocí tzv. komponent. V modulu `dash_html_components` nabízí Dash komponenty odpovídající standardním HTML elementům, tedy například `div`, `p` nebo `h1`. V modulu `dash_core_components` se pak nacházejí již předpřipravené komplexnější komponenty, jako například widgety nebo komponenta `Graph` pro zobrazení modelu vytvořeného v Plotly. Každá komponenta má sadu pojmenovaných argumentů, pomocí kterých je možné nastavit například její identifikátor `id`, CSS třídu `className` a potomky `children`. Pomocí polí potomků pak je možné vytvořit celou strukturu webové stránky.

Příklad definice rozložení jednoduché webové aplikace obsahující jednu stránku s nadpisem, vizualizací vytvořenou v Plotly a jedním widgetem typu `slider` je na výpisu 4.4. Základem rozložení aplikace `app` je jeden `div` se stanoveným `id` a třemi potomky – jednou `html` komponentou pro nadpis a dvěma `core` komponentami pro vizualizaci a widget. Vizualizace je v této ukázce vy-

tvořena pomocí funkce `create_figure`, jejíž návratovou hodnotou je objekt `Figure` z knihovny Plotly nastavený dle jejích argumentů.

```
app.layout = html.Div(
    id="page-id",
    children=[
        html.H1(className="headline", children="Page title"),
        dcc.Graph(id="graph", figure=create_figure(2)),
        dcc.Slider(id="point-slider", min=1, max=5, value=2, step=0.5)])
```

Výpis kódu 4.4: Rozložení jednoduché webové aplikace definované ve frameworku Dash.

Zpracování uživatelského vstupu Pro zpracování uživatelského vstupu z widgetů nabízí dle [32] Dash anotaci `@app.callback`, ve které je možné definovat vstupy a výstupy aktualizace. Takto anotovaná funkce pak na základě vstupů vypočítá novou hodnotu a tu předá výstupům.

Konkrétní příklad aktualizace vizualizace vytvořené v Plotly je na výpisu 4.5 a navazuje na rozložení webové stránky popsané ve výpisu 4.4. Funkce `update_figure` přijímá jako vstup `value` atribut komponenty s identifikátorem `point-slider` a její návratová hodnota se zapíše do atributu `figure` komponenty s identifikátorem `graph`. Tato funkce tedy přijme uživatelem zvolenou novou hodnotu parametru vizualizace, vytvoří novou Plotly `Figure` s aktualizovanou vizualizací (pomocí stejné funkce jako při inicializaci – viz výše) a tuto `Figure` vrátí jako výsledek, aby se zobrazila v rámci komponenty `Graph`.

```
@app.callback(
    Output("graph", "figure"),
    Input("point-slider", "value"))
def update_figure(point_value):
    return create_figure(point_value)
```

Výpis kódu 4.5: Anotovaná funkce pro aktualizaci webové stránky podle uživatelského vstupu ve frameworku Dash.

4.2 Návrh řešení

Na základě předchozí analýzy provedené v sekci 3.3 jsem ověřila, že pro implementaci jednotlivých modelových ukázek bude nejvhodnější vytvořit samostatnou webovou aplikaci, která bude obsahovat jednotlivé vizualizace.

V této sekci proto navrhnu obecnou strukturu výsledné aplikace. Určím uspořádání adresáře se zdrojovými kódy a vysvětlím účel nejdůležitějších souborů v něm. Následně popíši obecnou strukturu kódu, který bude sloužit pro definici vizualizací. Budu se věnovat i dynamické správě obsahu. Na závěr shrnu navržený postup pro vytvoření nové vizualizace.

4.2.1 Struktura webové aplikace

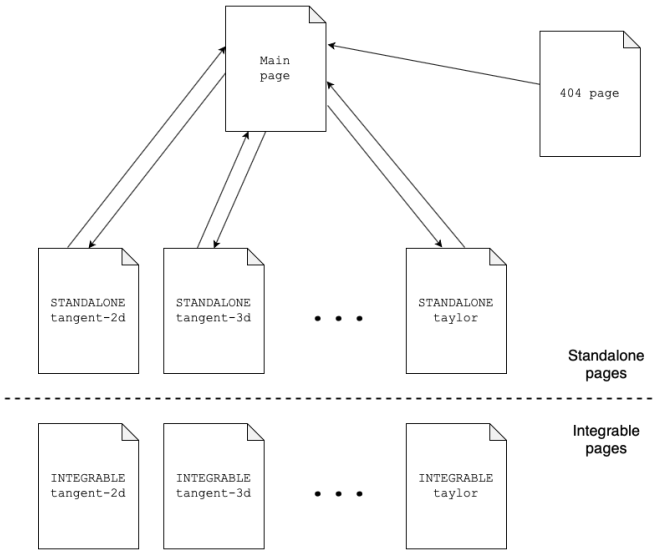
Z analýzy v sekci 3.3 vyplynulo, že existují dvě možnosti pro propojení systému MARAST se vznikající aplikací. První způsob je skrze URL odkazy, které uživatele přesměrují na konkrétní vizualizaci v aplikaci. V tomto případě pak bude vhodné, pokud bude odkaz doplněn alespoň statickým obrázkem a zároveň bude stránka aplikace obsahovat vedle samotného modelu také doprovodný kontext (název vizualizace, její popis atd.). Druhý způsob počítá se zobrazením vizualizace v rámci stránky v systému MARAST pomocí HTML elementu `iframe`. V tomto případě by naopak bylo ideální, pokud by stránka s vizualizací neobsahovala nadbytečný kontext a byla zobrazitelná v elementu `iframe` s přijatelnou výškou a šířkou.

Jelikož nechci omezit způsob propojení mezi vznikající aplikací a systémem MARAST pouze na jednu z těchto dvou možností a naopak bych ráda pro oba způsoby propojení nabídla vhodné podmínky, budou pro každou vizualizaci vytvořeny dvě stránky – jedna samostatná s kontextem a vzhledem samostatné webové stránky a jedna integrovatelná, která nebude obsahovat kontext a bude vhodná pro zobrazení v rámci elementu `iframe`. Obě stránky přitom budou mít unikátní URL adresy.

Webová aplikace se tedy bude skládat z těchto stránek.

- **Hlavní stránka** bude centrálním rozcestníkem, který bude obsahovat popis účelu aplikace a odkazy na jednotlivé samostatné stránky s ukázkami. Její adresa bude `/`.
- **Samostatná stránka s ukázkou** bude vytvořena pro každou vizualizaci. Bude obsahovat nadpis, popis ukázky, samotnou ukázku s widgety a volitelné doprovodné prvky. Bude vzhledově přizpůsobena pro zobrazení na širších obrazovkách. Bude obsahovat tlačítko pro návrat na hlavní stránku. Její adresa bude ve tvaru `/app/<jmeno-vizualizace>`.
- **Integrovatelná stránka s ukázkou** bude také vytvořena pro každou vizualizaci. Oproti samostatné stránce bude obsahovat pouze to nejnutnější – samotný model s ukázkou a doprovodnými widgety. Bude nutné, aby byla zobrazitelná bez scrollování v oknech o rozměru cca 800 px na šířku, 650 px výšku. Nebude obsahovat odkaz pro přesměrování na hlavní stránku. Její adresa bude ve tvaru `/model/<jmeno-vizualizace>`.
- **Chybová stránka** bude zobrazena, pokud bude zadaná URL adresa neznámá. Bude obsahovat odkaz pro návrat na hlavní stránku.

Tato struktura je zachycena na obrázku 4.1. Na tomto obrázku jsou znázorněny jednotlivé stránky, které budou součástí webové aplikace, a jejich vzájemné propojení pomocí odkazů.



Obrázek 4.1: Navržená struktura webové aplikace.

Jedním z hlavních požadavků na navržené řešení je, aby proces přidání nové vizualizace byl co nejjednodušší a pokud možno neobsahoval kromě vytvoření vizualizace příliš mnoho kroků navíc.

Z tohoto důvodu bude webová aplikace tvořena hotovým jádrem – hlavní stránkou s rozcestníkem, šablonovými funkcemi pro vytváření dvou typů stránek, hotovými pomocnými funkcemi, dynamickou správou obsahu a podobně. K tomuto jádru bude možné postupně přidávat jednotlivé vizualizace, které se díky pevně dané struktuře a dynamické správě obsahu stanou automaticky plně funkčními součástmi aplikace. Cílem návrhu řešení je, aby pro přidání vizualizace do aplikace stačilo do odpovídající složky přidat soubor, který bude mít požadovanou strukturu a který bude vizualizaci definovat. Veškeré propojení tohoto souboru s aplikací a generování dvojice různých stránek bude pak probíhat automaticky.

4.2.2 Návrh struktury adresáře se zdrojovými kódy

Výsledná aplikace bude obsahovat různé stránky, které musí být rozlišeny pomocí URL adres. Dokumentace frameworku Dash [32] popisuje vhodnou strukturu adresáře se zdrojovými kódy takové aplikace. Tuto strukturu zachovám podle doporučení, ale jednotlivé soubory a složky přejmenuji, aby byl

```
assets ..... složka pro CSS a JavaScriptové podpůrné soubory
exceptions ..... balíček pro deklaraci vlastních výjimek
smoketests ..... balíček s tzv. smoke testy ověřujícími spustitelnost
tests ..... balíček s unit testy ověřujícími funkčnost pomocných funkcí
utils ..... balíček pro deklaraci pomocných funkcí
visualizations ..... balíček s jednotlivými vizualizacemi
    __template__.py ..... šablona použitelná pro vytvoření vizualizace
    integral_2d.py ..... vizualizace definice určitého integrálu
    tangent_2d.py ..... vizualizace tečny ke grafu funkce
    …
app_setup.py ..... vytvoření a nastavení aplikace
app.py ..... definice obecného rozložení a spuštění aplikace
index.py ..... definice rozložení hlavní stránky s rozcestníkem
requirements.txt ..... seznam knihoven potřebných pro vývoj
```

Obrázek 4.2: Struktura adresáře se zdrojovými kódy (zjednodušeno pro přehlednost).

adresář přehlednější. Zároveň několik souborů a složek přidám, abych pokryla další potřeby aplikace. Návrh struktury adresáře je naznačen na obrázku 4.2.

Dvojice souborů `app_setup.py` a `app.py` bude stěžejní pro nastavení a spuštění celé aplikace. První z nich bude sloužit k vytvoření aplikace a bude používán i pro její import do jednotlivých vizualizací. Ve druhém bude mimo jiné definováno obecné rozložení aplikace a routing (párování mezi URL a zobrazenou vizualizací a typem stránky) a bude sloužit ke spouštění aplikace. Důležitou složkou pak bude `visualizations`, která bude obsahovat jednotlivé vizualizace.

4.2.3 Návrh struktury vizualizace

Pro jednoduchost vytváření budou dvě stránky – samostatná a integrovatelná – pro jednu vizualizaci generovány dynamicky z jednoho souboru popisujícího tuto vizualizaci. Aby tato dynamická správa obsahu byla možná a aby byla zachována jednotnost, bude mít tento soubor pevně definovanou strukturu a určené povinné atributy a funkce. Každý soubor popisující vizualizaci bude rozdělen na čtyři části:

- První část kódu bude sloužit k definici matematických funkcí – například pro výpočet rovnice tečny nebo Taylorova polynomu.
- Druhá část bude sloužit pro vlastní vykreslení modelu. Budou do ní patřit:
  - podpůrné vykreslovací funkce – například vykreslení obdélníku ve vizualizaci definice integrálu,

  - povinná funkce `create_figure`, která bude přijímat parametry modelu (např. vybrané body a intervaly) a vrátí objekt `Figure` s vytvořeným modelem.

- Třetí část bude určená pro definici rozložení webové stránky. Bude obsahovat tyto atributy:
  - `title` (povinný) s titulkem stránky používaným mj. v rozcestníku,
  - slovník `default_args` (povinný) s počátečními hodnotami parametrů funkce `create_figure`,
  - `headline` (povinný) s nadpisem stránky,
  - textový řetězec nebo pole textových řetězců `annotation` (povinný) s popisem tématu, jehož se vizualizace týká, a s popisem interaktivních možností vizualizace,
  - pole komponent `before_layout` (volitelné) definující doplňující komponenty zobrazené před částí `layout`,
  - pole komponent `after_layout` (volitelné) definující doplňující komponenty zobrazené za částí `layout`,
  - pole komponent `layout` (povinné) obsahující ústřední komponenty vizualizace – tedy zejména komponentu pro zobrazení Plotly modelu a widgety pro interaktivní manipulaci s ním.
- Čtvrtá a poslední část bude obsahovat Dash callbacky pro aktualizaci vizualizace na základě uživatelských vstupů.

Samostatná stránka s vizualizací pak bude tvořena nadpisem, anotací s popisem vizualizace, vlastní ukázkou (grafem a widgety) a doplňujícími komponentami umístěnými před a za touto ukázkou. Integrovatelnou stránku pak bude tvořit pouze ukázka bez kontextu – tj. pouze komponenty definované v poli `layout`.

Díky tomuto rozdělení obsahu do povinných atributů bude jednodušší dodržet jednotný vzhled a strukturu vizualizací v obou typech stránek. Zároveň se výrazně zjednoduší také tvorba vizualizace, jelikož bude možné všechny informace jednoduše zapsat do jednoho souboru, ze kterého budou dynamicky vygenerovány dvě různé stránky.

Aby bylo jednodušší dodržet definovanou strukturu a nevynechat definici povinných prvků, bude složka `visualizations` obsahovat šablonový soubor `__template__.py`, který tuto strukturu pomocí komentářů popíše, bude obsahovat všechny povinné části a bude možné ho použít jako základ při vytváření nové vizualizace.

Při vytváření vizualizací nebude vyžadována implementace podle návrhových vzorů, aby byla zachována jednoduchost tvorby a aby nebylo nutné rozdělovat definici vizualizace do vícero souborů.

4.2.4 Dynamická správa obsahu

Pro každou vytvořenou vizualizaci musí být vygenerovány dvě stránky a odkaz na samostatnou stránku musí být umístěn v rozcestníku na hlavní stránce. Pro všechny vizualizace v různých typech stránek musí dále existovat mapování na unikátní URL. Samostatné stránky také musí mít přiřazený odpovídající titulek zobrazený na záložce v prohlížeči.

Aby byl průběh vytváření vizualizace co nejjednodušší, budou všechny tyto činnosti řešeny dynamicky podle obsahu složky `visualizations`. Tento přístup nicméně vyžaduje dodržení povinné struktury kódu definujícího vizualizaci – je třeba vyplnit povinné atributy a implementovat povinné funkce zmíněné v předchozí sekci.

Zobrazení v rozcestníku Aby bylo možné vytvořit rozcestník, je třeba nejprve získat seznam vizualizací a poté ke každé vizualizaci zjistit její titulek – atribut `title`.

Pro zjištění obsahu adresáře bude možné využít funkci `listdir` z modulu `os`, která dle [48] umožňuje vypsat obsah adresáře. Ten bude následně zpracován a vznikne z něj seznam textových řetězců s názvy všech modulů v rámci daného balíčku.

Tyto moduly bude pak možné dynamicky importovat pomocí modulu `importlib` a jeho funkce `import_module` [49]. Z importovaného modulu pak bude možné přečíst jeho atribut `title`.

Mapování URL a vizualizace U vícestránkových aplikací je dle [32] nutné zajistit párování mezi jednotlivými stránkami a URL adresami. Toho je možné docílit pomocí komponenty `Location`, jejíž atribut `pathname` obsahující aktuální adresu je předán jako vstup do callbacku. Výstupem tohoto callbacku pak je obsah stránky odpovídající aktuální adrese. Toto párování může být provedeno například pomocí `if-else` výroků, ale pak by do něj bylo třeba každou vizualizaci manuálně přidat.

Mapování proto bude provedeno dynamicky. Na základě aktuální adresy budou zjištěny typ stránky – samostatná nebo integrovatelná – a název požadované vizualizace. Modul s touto vizualizací bude importován pomocí dynamického importu zmíněného výše. Podle požadovaného typu stránky se vybere šablonová funkce, která z modulu získá hodnoty požadovaných atributů a doplní je do kostry stránky. Návratovou hodnotou šablonové funkce bude strom komponent definující rozložení požadované webové stránky. Mapování také zahrne podmínku pro zobrazení hlavní stránky. Samozřejmostí pak bude odchycení chyb – při neznámé vizualizaci nebo neznámém typu stránky bude zobrazena stránka s informací o nenalezení obsahu.

Nastavení titulku na záložce prohlížeče Dle dokumentace frameworku Dash [32] je možné dynamicky nastavit různé titulky pro různé stránky po-

mocí metody `clientside_callback`. Tento callback má argumenty definující vstupy (aktuální URL) a výstupy (prázdný element `div`), ale především má argument s JavaScriptovou funkcí zapsanou pomocí textového řetězce, která na základě URL na vstupu nastaví webovému dokumentu odpovídající titulek.

Aby nebylo po přidání každé vizualizace potřeba tuto JavaScriptovou funkci upravovat, bude její text také generován dynamicky v závislosti na obsahu balíčku `visualizations` a atributech `title` jednotlivých vizualizací v něm.

4.2.5 Postup vytvoření nové vizualizace

Díky dynamické správě vizualizací popsané v předchozí části této sekce bude vytvoření nové vizualizace velmi jednoduché. Pomocí zkopírování šablonového souboru `__template__.py` a jeho přejmenování se vytvoří nový soubor ve složce `visualizations`. V tomto souboru se vytvoří vizualizace s odpovídající strukturou a atributy. Dále již nebude potřeba nic nastavovat – vizualizace bude automaticky plnohodnotnou součástí aplikace.

S vytvořením vizualizace pomůže i podrobný manuál (viz příloha D) popisující detailněji proces vytvoření vizualizace, nejčastější používané funkce a metody použitých knihoven, případně řešení nejčastějších problémů. Elektronická forma manuálu bude doplněna o odkazy na příslušná místa v dokumentaci tam, kde je to potřeba.

4.3 Implementace modelových ukázek

V této sekci se budu věnovat implementaci modelových ukázek popsaných v kapitole 2. U každé ukázky popíši její vlastnosti a přiložím její screenshot z hotové aplikace. Dále zmíním zajímavá místa její implementace s ohledem na matematické výpočty, případně na vykreslení.

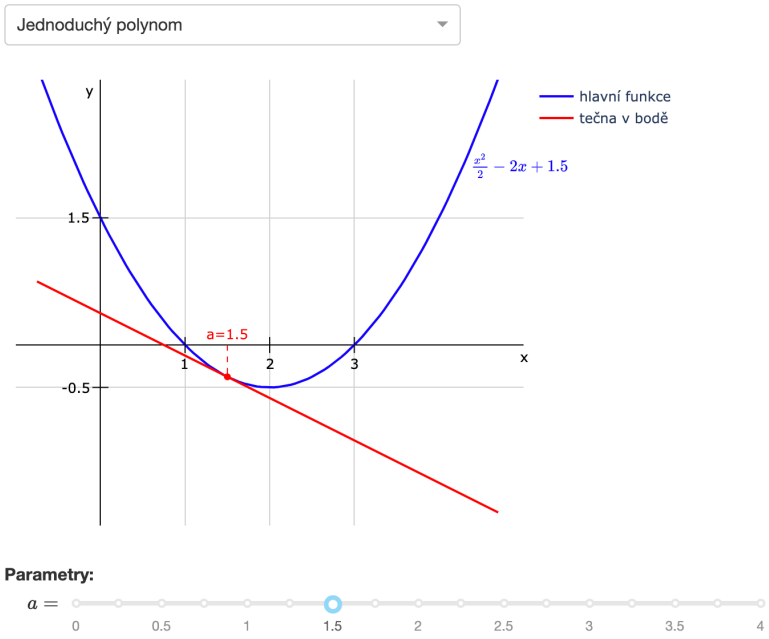
4.3.1 Tečna ke grafu funkce jedné proměnné

Tato vizualizace zobrazuje graf funkce jedné proměnné. Umožní uživateli zvolit bod z jejího definičního oboru a vykreslí tečnu ke grafu funkce procházející tímto bodem.

Hotová vizualizace je na obrázku 4.3. V modelu jsou zobrazeny dvě funkce – hlavní (modře) a její tečna (červeně). Navíc jsem přidala několik popisných prvků – rovnici hlavní funkce a indikátor zvoleného bodu. Pomocí posuvníku pod modelem je možné vybírat bod ze stanoveného rozsahu, jímž má tečna probíhat. Dále je model možné posouvat a přibližovat.

Na rozdíl od návrhu jsem parametrizovala také funkci, k jejímuž grafu je hledána tečna – pomocí rozbalovací nabídky je tedy možné zvolit jednu ze tří různých funkcí, která bude následně v modelu zobrazena. Toto rozšíření vyply-

nulo z diskuze o možných vylepšeních, která proběhla v rámci uživatelského testování (viz 4.4.3) a bylo aplikováno také u některých dalších ukázek.



Obrázek 4.3: Výsledná vizualizace tečny ke grafu funkce jedné proměnné.

Zajímavé body implementace matematických výpočtů Pro vykreslení průběhu tečny je stěžejní funkce `get_tangent_formula`, která z textového řetězce se vzorcem funkce a z bodu v jejím definičním oboru vytvoří lambda funkci reprezentující tečnu ke grafu dané funkce v daném bodě a vrátí ji jako výsledek. Tato lambda funkce dovede k souřadnici na ose x doplnit odpovídající souřadnici na ose y tak, aby výsledný bod ležel na grafu tečny. Kromě toho provádí tato funkce kontrolu výsledného výrazu a pokud není validní (pokud například derivace v daném bodě neexistuje), vyvolá výjimku. Celý kód této funkce je na výpisu 4.6.

Zajímavé body implementace vizualizace O samotné vykreslení tečny se následně stará úsek kódu na výpisu 4.7. Nejprve je s využitím výše zmiňované funkce získána lambda funkce pro tečnu funkce v bodě. Dále je vypočítáno pole `tvalues`, což jsou souřadnice na ose y odpovídající bodům na ose x z pole `XVALUES`. Společně tato pole definují body na grafu tečny, jejichž propojením je možné tečnu znázornit. Následně již může být vykreslen průběh tečny pomocí metody `add_trace`.

V případě, že při získávání lambda funkce pro tečnu nastane výjimka `NoResultException`, není graf funkce vykreslen a místo toho je na odpovída-

```
def get_tangent_formula(function_string, point):
    x = sp.symbols("x")
    func_expr = sp.sympify(function_string)
    diff_expr = sp.diff(func_expr, "x")
    expr = func_expr.subs(x, point) + diff_expr.subs(x, point) * (x - point)

    if is_expr_invalid(expr):
        raise NoResultException

    return sp.lambdify(x, expr, "numpy")
```

Výpis kódu 4.6: Matematický základ pro vykreslení průběhu tečny ve vizualizaci tečny ke grafu funkce jedné proměnné (kód zjednodušen pro popisnost).

jící místo v modelu přidaná textová anotace informující o nemožnosti vykreslit tečnu v tomto bodě.

```
try:
    t = get_tangent_formula(FUNCTION, point)
    tvalues = np.array([t(x) for x in XVALUES])
    fig.add_trace(go.Scatter(x=XVALUES, y=tvalues, name="tečna v bodě",
        mode="lines", line_color="red", hoverinfo="none"))
except NoResultException:
    fig.add_annotation(x=point, y=0, yanchor="bottom",
        text="Tečnu nelze vykreslit", showarrow=False, font_color="red")
```

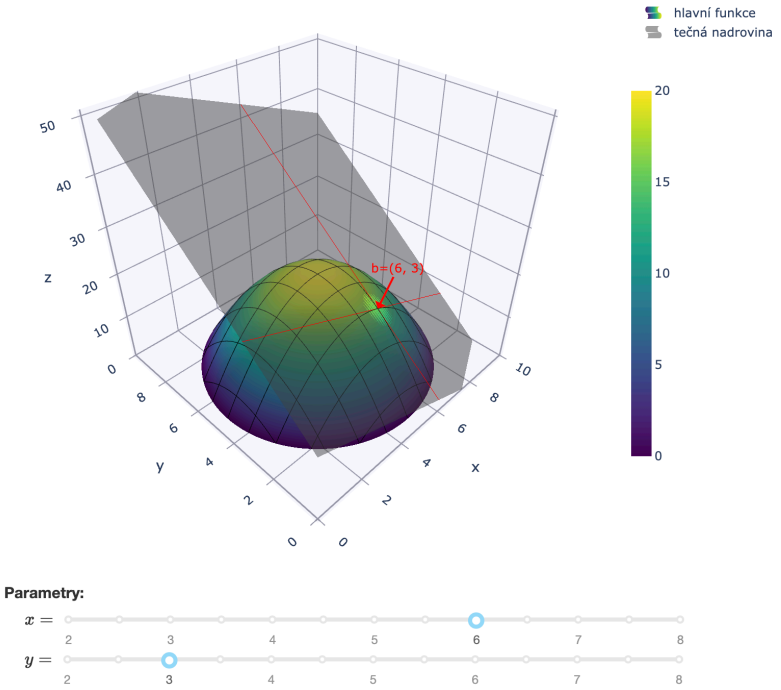
Výpis kódu 4.7: Vykreslení průběhu tečny ve vizualizaci tečny ke grafu funkce jedné proměnné (kód zjednodušen pro popisnost).

4.3.2 Tečná nadrovina ke grafu funkce dvou proměnných

V této vizualizaci je zobrazen graf funkce dvou proměnných s tečnou nadrovinou vykreslenou v uživatelem zvoleném bodě definičního oboru této funkce.

Hotová vizualizace je na obrázku 4.4. V modelu jsou pomocí povrchových grafů zobrazeny dvě funkce dvou proměnných – hlavní funkce a její tečná nadrovina. Hlavní funkce je obarvena pomocí gradientu, který indikuje hodnotu funkce v jednotlivých bodech definičního oboru. Tečnou nadrovinu jsem ponechala jednobarevnou a částečně průhlednou, aby byla vizualizace přehlednější. Červeně jsou na tečné nadrovině vyznačeny tečny ke grafu funkce ve směru souřadnicových os x a y. Z popisných prvků jsem kvůli zachování přehlednosti přidala pouze šipku ukazující na zvolený bod, v němž je tečná

nadrovina vykreslena, a jeho souřadnice. Pomocí dvou posuvníků pod modelem je možné zvolit konkrétní bod $b = (x, y)$, ve kterém bude tečna vykreslena. Celý graf je možné posouvat, přibližovat a otáčet.



Obrázek 4.4: Výsledná vizualizace tečné nadroviny ke grafu funkce dvou proměnných.

Zajímavé body implementace matematických výpočtů Pro výpočet vzorce tečné nadroviny slouží funkce `get_tangent_formula`. Ta z textového řetězce se vzorcem funkce a souřadnic zvoleného bodu definičního oboru vypočítá rovnici popisující tečnou nadrovinu v tomto bodě. Jako výsledek vrací lambda funkci dvou proměnných, která pro zadané souřadnice na osách x a y vypočte odpovídající funkční hodnotu v tomto bodě – tedy souřadnici bodu na ose z tak, aby výsledný bod ležel na hledané tečné nadrovině. Celý kód této funkce je na výpisu 4.8. Nejdůležitější v této funkci jsou řádky 5–7, kde je s pomocí knihovny SymPy tvořen vlastní výraz popisující tečnou nadrovinu.

Zajímavé body implementace vizualizace Vykreslení tečné nadroviny jako takové pak probíhá pomocí kódu na výpisu 4.9. Pomocí dříve zmíněné funkce je získána lambda funkce pro výpočet funkčních hodnot v jednotlivých bodech definičního oboru. Vypočítané hodnoty jsou následně uloženy v matici `tvalues` – ta k dvojici hodnot z polí `XVALUES` a `YVALUES` reprezentujících nějaký bod definičního oboru funkce drží právě odpovídající funkční hodnotu.

```
def get_tangent_formula(function_string, point_x, point_y):
    x, y = sp.symbols("x y")
    func_expr = sp.sympify(function_string)
    subs = [(x, point_x), (y, point_y)]
    expr = sp.diff(func_expr, "x").subs(subs) * (x - point_x)
           + sp.diff(func_expr, "y").subs(subs) * (y - point_y)
           + func_expr.subs(subs)
    return sp.lambdify((x, y), expr, "numpy")
```

Výpis kódu 4.8: Matematický základ pro vykreslení průběhu tečné nadroviny ve vizualizaci tečné nadroviny ke grafu funkce dvou proměnných (kód zjednodušen pro popisnost).

Poté lze vykreslit tečnou nadrovinu pomocí metody `add_trace` a povrchového grafu. Zvýraznění tečen ve směru souřadnicových os x a y je dosaženo pomocí zobrazení vrstevnic v daných bodech v těchto směrech (ve výpisu řádky 4–9).

```
t = get_tangent_formula(FUNCTION, point_x, point_y)
tvalues = np.array([[t(x, y) for x in XVALUES] for y in YVALUES])
fig.add_trace(go.Surface(x=XVALUES, y=YVALUES, z=tvalues,
    contours={
        "x": {"show": True, "start": point_x, "end": point_x+1, "size": 1,
              "color": "red", "highlight": False},
        "y": {"show": True, "start": point_y, "end": point_y+1, "size": 1,
              "color": "red", "highlight": False},
        "z": {"highlight": False}},
    opacity=0.5, colorscale=["#444444", "#444444"], name="tečná nadrovina",
    showscale=False, hoverinfo="none", showlegend=True))
```

Výpis kódu 4.9: Vykreslení průběhu tečné nadroviny ve vizualizaci tečné nadroviny ke grafu funkce dvou proměnných (kód zjednodušen pro popisnost).

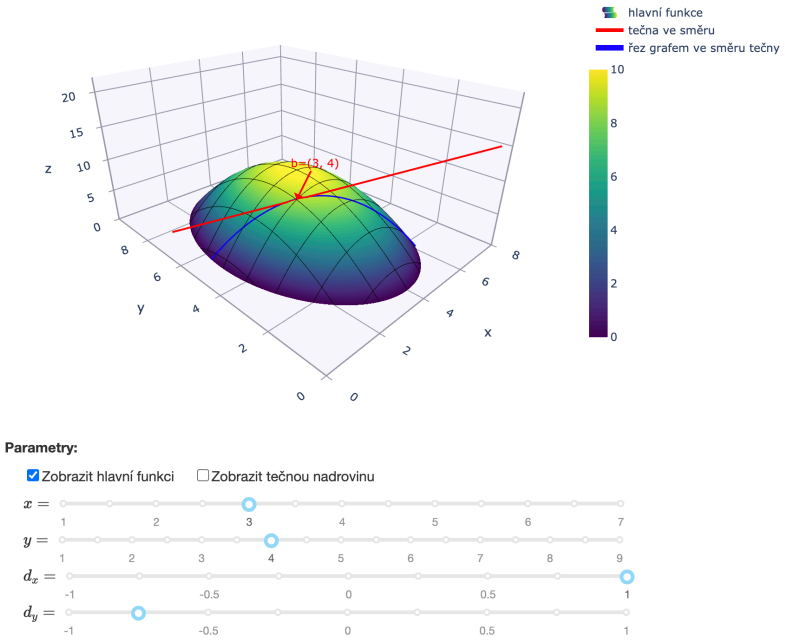
4.3.3 Tečna ke grafu funkce dvou proměnných ve směru

Vizualizace obsahuje graf funkce dvou proměnných a jeho tečnu v uživatelem zvoleném bodě a směru.

Hotová vizualizace je na obrázku 4.5. Pomocí povrchového grafu je v modelu zobrazena hlavní funkce obarvená pomocí gradientu pro popis funkční hodnoty v daném bodě. V uživatelem určeném bodě a směru je k této funkci vykreslena tečna (červeně). Dále jsem do modelu přidala řez grafu hlavní

funkce ve směru tečny procházející stejným bodem (modře) – aby vizualizace upozornila na vztah mezi tečnou a hlavní funkcí zúženou na požadovaný směr. Z popisných prvků jsem opět použila pouze textovou anotaci se šipkou označující zvolený bod.

Pomocí posuvníků pod modelem je možné vybrat bod $b = (x, y)$, kterým má tečna procházet, a její směr $d = (d_x, d_y)$. Kromě toho je možné pomocí zaškrtávacího políčka schovat graf hlavní funkce a ponechat zobrazený pouze řez a tečnu – což ještě lépe ilustruje jejich vztah. Druhým zaškrtávacím políčkem je možné naopak zobrazit tečnou nadrovinu v daném bodě – a ověřit si tak, že tečná nadrovina skutečně vzniká sjednocením tečen v daném bodě v různých směrech. Celý graf je pak možné posouvat, přibližovat a otáčet.



Obrázek 4.5: Výsledná vizualizace tečny ke grafu funkce dvou proměnných ve směru.

Zajímavé body implementace matematických výpočtů Pro výpočet rovnice tečny ve směru slouží funkce `get_tangent_fomula`, která je v mírně zjednodušené podobě na výpisu 4.10. Stavím na parametrickém vyjádření přímky – hledaná tečna bude mít tvar $t(k) = \boldsymbol{B} + k * \boldsymbol{D}$, kde $\boldsymbol{B} = (b_x, b_y, b_z)$ je bod, jímž tečna prochází, a $\boldsymbol{D} = (d_x, d_y, d_z)$ je její směr. Bod $\boldsymbol{B}$ je přitom lehce dopočítatelný a z vektoru $\boldsymbol{D}$ znám první dvě složky – dopočítat je třeba pouze jeho vertikální sklon. Ten je dán derivací funkce v bodě a ve směru (záměrně jsem na žádném místě nepoužila normalizaci směrového vektoru, aby

do výpočtu nebyla zanesena větší nepřesnost). Vlastní výpočet třetí složky směrového vektoru je ve výpisu na řádcích 5–7.

Konkrétní body na přímce bych následně mohla dopočítávat pomocí dosazování různých hodnot parametru k, ale špatně by se zajišťovalo pokrytí celého rozsahu zobrazeného v grafu. Využívám proto faktu, že pro bod na grafu tečny s konkrétní hodnotou souřadnice na ose x mohu ze znalosti b_x a d_x dopočítat hodnotu parametru k a z něj i hodnoty souřadnic na osách y a z. Je nicméně potřeba ošetřit případ, kdy je $d_x = 0$ a parametr k tak z hodnoty souřadnice na ose x není možné dopočítat – pak je pro výpočet k nutné použít požadovanou souřadnici na ose y.

Z těchto důvodů tedy provádím převod na řádku 9, kdy hodnotu parametru k vyjádřím pomocí proměnné x, eventuálně y pokud je $d_x = 0$. Návratovou hodnotou pak jsou tři lambda funkce, které přijímají dva parametry, ale uplatní se vždy jenom jeden z nich, podle toho, jaké parametrické vyjádření bylo použito pro parametr k (buď se k počítá z požadované souřadnice na ose x, nebo na ose y). S pomocí těchto tří funkcí je následně možné vykreslit tečnu v daném bodě i směru.

```
1  def get_tangent_formula(function_string, point_x, point_y,
2                          direction_x, direction_y):
3      x, y, k = sp.symbols("x y k")
4      f, f_expr = get_function_formula(function_string)
5      point_z = f(point_x, point_y)
6      grad = get_gradient_in_point(f_expr, point_x, point_y)
7      direction = [direction_x, direction_y]
8      direction_z = sum([grad[i]*direction[i] for i in range(len(grad))])
9
10     k = (x - point_x)/direction_x if direction_x != 0
11         else (y - point_y)/direction_y
12     return (sp.lambdify((x, y), point_x + direction_x * k, "numpy"),
13         sp.lambdify((x, y), point_y + direction_y * k, "numpy"),
14         sp.lambdify((x, y), point_z + direction_z * k, "numpy"))
```

Výpis kódu 4.10: Matematický základ pro vykreslení tečny ke grafu funkce dvou proměnných ve směru (kód zjednodušen pro popisnost).

Zajímavé body implementace vizualizace Důležitý úsek kódu při tvorbě této ukázky je na výpisu 4.11 – slouží k vykreslení tečny ve směru a řezu hlavní funkce.

Nejprve pomocí výše zmíněné funkce `get_tangent_formula` získám tři lambda funkce, pomocí nichž vygeneruji souřadnice bodů popisujících hleda-

nou tečnu. Samotné vykreslení je již jednoduché – s použitím 3D spojnicového grafu.

Pro řez hlavní funkce použiji stejné souřadnice na osách x a y, jako pro vykreslení tečny – obě funkce mají stejný směr a procházejí stejným bodem, navzájem se proto budou lišit pouze souřadnicí na ose z, kterou zde dopočítávám pomocí lambda funkce f, která popisuje hodnoty hlavní funkce. Pro vykreslení následně opět používám 3D spojnicový graf.

```
x_func, y_func, z_func = get_tangent_formula(FUNCTION,
    point_x, point_y, direction_x, direction_y)
xvalues = np.array([x_func(x, y) for x, y in zip(XVALUES, YVALUES)])
yvalues = np.array([y_func(x, y) for x, y in zip(XVALUES, YVALUES)])
zvalues = np.array([z_func(x, y) for x, y in zip(XVALUES, YVALUES)])
fig.add_trace(go.Scatter3d(x=xvalues, y=yvalues, z=zvalues, mode="lines",
    name="tečna ve směru", line_color="red", line_width=4,
    showlegend=True, hoverinfo="none"))

func_projection_zvalues = np.array([f(xvalues[i], yvalues[i]) for i
    in range(len(XVALUES))])
fig.add_trace(go.Scatter3d(x=xvalues, y=yvalues, z=func_projection_zvalues,
    mode="lines", name="řez grafem ve směru tečny", line_color="blue",
    line_width=4, showlegend=True, hoverinfo="none"))
```

Výpis kódu 4.11: Vykreslení průběhu tečny ke grafu funkce dvou proměnných ve směru (kód zjednodušen pro popisnost).

4.3.4 Vykreslení Taylorova polynomu

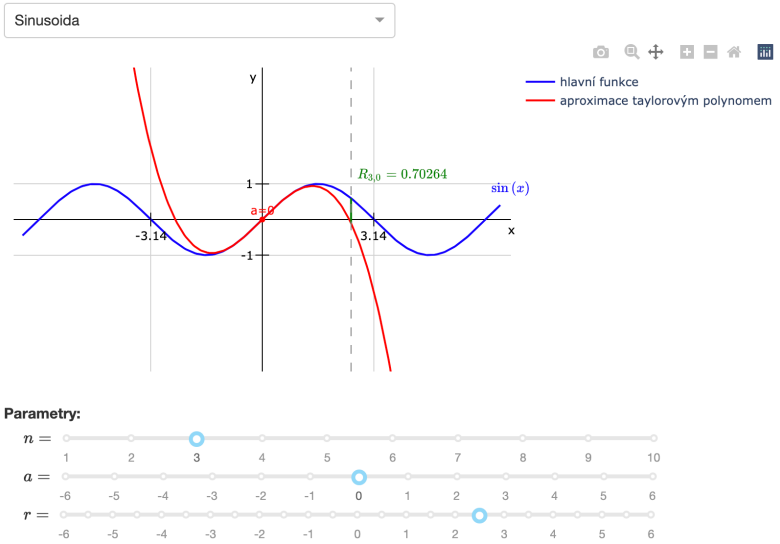
Tato vizualizace zobrazuje graf funkce jedné proměnné. Uživateli umožňuje zvolit bod a z definovaného rozsahu a stupeň n Taylorova polynomu $T_{n,a}$, který následně podle zadaných parametrů vykreslí.

Hotová vizualizace je na obrázku 4.6. Vizualizace obsahuje dvě matematické funkce – hlavní (modře) a její aproximaci pomocí Taylorova polynomu se zvolenými parametry (červeně). Z popisných prvků pak obsahuje textovou anotaci se vzorcem hlavní funkce a indikátor zvoleného bodu a. Pomocí dvojice posuvníků je možné volit různé body a a stupně polynomu n a pozorovat efekt, který tyto parametry mají. Pomocí nástrojů modelu je také možné graf přibližovat a posouvat.

Vedle toho jsem do modelu oproti návrhu z kapitoly 2 přidala třetí posuvník, který slouží pro výběr bodu r. V tom je následně graficky znázorněná odchylka funkce a její aproximace – Taylorův zbytek $R_{n,a}$. Zobrazená je také její vypočítaná hodnota. Lze tak porovnat přesnou hodnotu odchylky v různých

bodech definičního oboru, případně pozorovat změny odchylky při změnách stupně polynomu.

Tuto vizualizaci jsem také (podobně jako výše tečnu ke grafu funkce jedné proměnné) rozšířila o možnost volit z několika různých funkcí, které mohou být v ukázce aproximovány Taylorovým polynomem.



Obrázek 4.6: Výsledná vizualizace Taylorova polynomu.

Zajímavé body implementace matematických výpočtů Pro výpočet Taylorova polynomu slouží funkce `get_taylor_polynom`, která jako své parametry přijímá textovou reprezentaci vzorce hlavní funkce, centrální bod a Taylorova polynomu a jeho stupeň n. Celý kód této funkce je na výpisu 4.12. Nejdůležitější jeho částí je `for` cyklus na řádcích 5–7 – zde dochází k postupnému sestavení sumy tvořící Taylorův polynom. Tento výraz je převeden do lambda funkce, která je vrácena jako výsledek a která k zadané hodnotě na ose x vypočítá odpovídající souřadnici na ose y tak, aby výsledný bod ležel na grafu hledaného Taylorova polynomu.

Zajímavé body implementace vizualizace Pro umožnění výběru z několika různých funkcí jsem implementovala speciální objekt `Function` pro funkci jedné proměnné (podobně jsem pro potřeby jiných ukázek implementovala i `Function3D` pro funkci dvou proměnných). Tento objekt je jakousi obálkou, která sdružuje především vzorec funkce a její jméno a dále různé vlastnosti modelu specifické pro vykreslení konkrétní funkce – například rozsahy zobrazení souřadnicových os, či význačné body zobrazené na osách.

Kód inicializační metody této třídy je na výpisu 4.13. Metoda přijímá šestici povinných parametrů (např. vzorec funkce a její jméno) a libovolný

```
def get_taylor_polynom(function_string, center, degree):
    x = sp.symbols("x")
    func_expr = sp.sympify(function_string)
    expr = 0
    for k in range(degree + 1):
        expr += sp.diff(func_expr, "x", k).subs(x, center) /
            sp.factorial(k) * (x - center)**k
    return sp.lambdify(x, expr, "numpy")
```

Výpis kódu 4.12: Matematický základ pro vykreslení aproximace funkce pomocí Taylorova polynomu (kód zjednodušen pro popisnost).

další počet pojmenovaných argumentů, které ukládá v podobě slovníku. Díky tomu je možné v závislosti na požadavcích vizualizace parametrizovat libovolné vlastnosti – mimo jiné například hodnoty zobrazené na posuvnících.

Vizualizace obsahuje konstantu s polem objektů `Function`. Pomocí rozbalovací nabídky je vybrán index v tomto poli odpovídající požadované funkci. Zvolený objekt `Function` je pak použit pro parametrizaci modelu.

```
def __init__(self, formula, name, xrange, yrange, xticks, yticks, **kwargs):
    self.formula = formula
    self.name = name
    self.xrange = xrange
    self.yrange = yrange
    self.xticks = xticks
    self.yticks = yticks
    self.other = kwargs
```

Výpis kódu 4.13: Inicializační metoda třídy sdružující informace o nastavení modelu pro zobrazení dané funkce (kód zjednodušen pro popisnost).

4.3.5 Určitý integrál funkce jedné proměnné

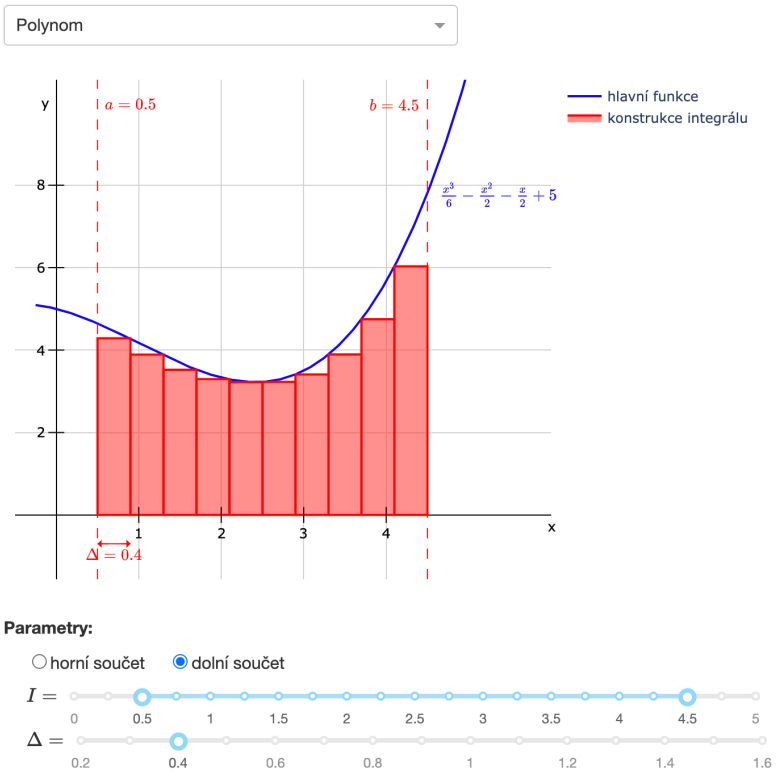
V rámci této vizualizace je v modelu vykreslen graf funkce jedné proměnné. Na základě uživatelem zvolených krajních bodů intervalu $I = \langle a, b \rangle$, velikosti částečného intervalu Δ a typu součtu (horní nebo dolní) je potom v modelu vykreslena konstrukce určitého integrálu na tomto intervalu a s částečnými intervaly dané velikosti.

Hotová vizualizace je na obrázku 4.7. V modelu je modře vykreslená integrovaná funkce, červeně čárkovaně krajní body zvoleného intervalu a pomocí červených obdélníků je naznačena konstrukce určitého integrálu. Model dále

obsahuje popisné prvky, jako například vzorec integrované funkce, popisky krajních bodů intervalu a v levé dolní části také naznačenou zvolenou velikost částečného intervalu.

Pomocí přepínačů je možné vybírat, zda bude vizualizace zobrazovat horní nebo dolní součet. Dále je možné pomocí posuvníků vybrat krajní body intervalu I a velikost částečného intervalu Δ. Celý graf lze také přibližovat a posouvat.

Také u této ukázky jsem oproti návrhu přidala parametrizaci integrované funkce – je možné volit mezi několika funkcemi z nabídky, z nichž některé mají jak kladné, tak i záporné funkční hodnoty.



Obrázek 4.7: Výsledná vizualizace konstrukce určitého integrálu funkce jedné proměnné.

Zajímavé body implementace matematických výpočtů Pro určení výšky jednotlivých obdélníků ve vizualizaci je důležitý výpočet infima, případně suprema z funkčních hodnot na daném intervalu. Ukázka je určena pro spojité funkce s konečnými funkčními hodnotami, místo infima a suprema bude proto postačovat minimum a maximum. Knihovna SymPy nabízí možnost vypočítat minimum a maximum funkčních hodnot na daném intervalu, ale tento

výpočet je zdlouhavý a zejména pro malé hodnoty Δ neúměrně prodlužuje dobu překreslení vizualizace.

Proto jsem se rozhodla zvolit méně náročnou variantu výpočtu – z intervalu je rovnoměrně vybrána konečná množina bodů x (dostatečně velká) a na minimum se testují pouze funkční hodnoty $f(x)$ v těchto bodech. Touto úpravou se výpočet značně zrychlil a při dostatečné jemnosti „vzorkování“ navíc není nepřesnost na vizualizaci patrná. Zrychlení výpočtu nicméně přineslo další omezení na integrovanou funkci – její průběh musí být dostatečně plynulý. Navíc se předpokládá, že při implementaci vizualizace její tvůrce výsledek zkontroluje (což je obecně doporučeno v manuálu pro tvorbu vizualizací, viz příloha D) a vybere tak pouze funkce vhodné pro tuto aproximaci suprema a infima.

Zajímavé body implementace vizualizace Podstatnou částí kódu při vykreslování této vizualizace je funkce `draw_rectangles` používaná pro vykreslení konstrukce integrálu (viz kód na výpisu 4.14). Na vstupu tato funkce dostává popis požadované konstrukce – velikost částečného intervalu Δ, zvolený interval I a typ součtu. Kromě toho dostává také referenci na model, do kterého je třeba konstrukci vykreslit a lambda funkci pro výpočet funkčních hodnot integrované funkce f.

Nejprve jsou vygenerovány dělící body daného intervalu. Pro každý počáteční bod částečného intervalu je poté určen koncový bod – poslední částečný interval může být menší než Δ, pokud není velikost intervalu I dělitelná Δ beze zbytku. Podle vybraného typu součtu je určena výška konkrétního obdélníku. Následně je obdélník vykreslen s využitím jiné pomocné funkce.

```
1  def draw_rectangles(fig, f, delta, interval, sum_type):
2      steps = np.arange(interval[0], interval[1], delta)
3      for i in steps:
4          i_end = min(i+delta, interval[1])
5          height = 0
6          if sum_type == "LOWER":
7              height = get_min(f, [i, i_end])
8          elif sum_type == "UPPER":
9              height = get_max(f, [i, i_end])
10         draw_rectangle(fig, i, i_end, height,
11             showlegend=bool(i == steps[0]))
```

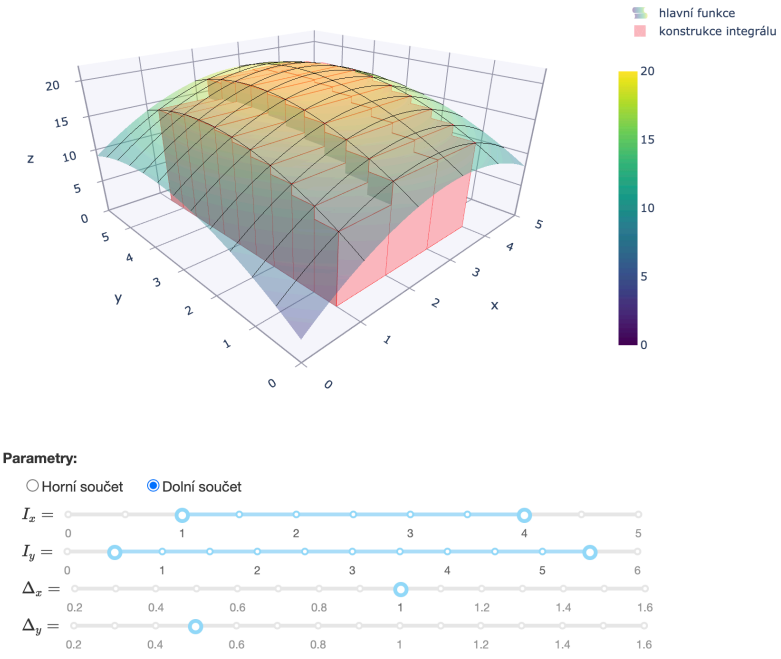
Výpis kódu 4.14: Funkce pro vykreslení konstrukce určitého integrálu funkce jedné proměnné.

4.3.6 Určitý integrál funkce dvou proměnných

Vizualizace ukazuje graf funkce dvou proměnných. Podle uživatelem zvolených parametrů – intervalů $I_x = \langle a_x, b_x \rangle$ a $I_y = \langle a_y, b_y \rangle$, velikostí částečných intervalů Δ_x a Δ_y a typu použitého součtu (horní nebo dolní) – jsou poté v modelu vykresleny kvádry znázorňující konstrukci určitého integrálu funkce dvou proměnných.

Vizualizace je na obrázku 4.8. V modelu je pomocí barevného gradientu vykreslená hlavní integrovaná funkce. Barevný gradient naznačuje funkční hodnotu v daném bodě definičního oboru funkce. Funkci jsem záměrně ponechala částečně průhlednou, aby byla viditelná konstrukce integrálu pod ní. Zobrazila jsem na ní vrstevnice s rozestupem, který odpovídá zvoleným hodnotám Δ_x a Δ_y, aby v modelu byla lépe zdůrazněna velikost těchto částečných intervalů. Světle červenou barvou pak jsou vyznačeny jednotlivé kvádry tvořící konstrukci integrálu. Popisné prvky jsem v modelu kvůli zachování přehlednosti záměrně vynechala.

Pomocí přepínačů je možné vybírat, zda bude vizualizace zobrazovat horní nebo dolní součet. Dále je možné pomocí posuvníků vybrat krajní body intervalů I_x a I_y a velikost částečných intervalů Δ_x a Δ_y. Grafem je možné otáčet a posouvat, je možné ho i přibližovat.



Obrázek 4.8: Výsledná vizualizace konstrukce určitého integrálu funkce dvou proměnných.

Zajímavé body implementace matematických výpočtů Pro výpočet výšky jednotlivých kvádrů, která je dána infimem, případně supremem funkčních hodnot na daném intervalu, jsem použila podobné zjednodušení jako v případě určitého integrálu funkce jedné proměnné – pouze přibližný, ale zato výrazně rychlejší výpočet minima, resp. maxima, a to pomocí výběru určitých bodů v daném součinu částečných intervalů a porovnání funkčních hodnot v těchto bodech. Opět ale bylo nutné omezit vizualizaci na spojité funkce s konečnými funkčními hodnotami a s dostatečně plynulým průběhem.

Zajímavé body implementace vizualizace Pro vykreslení jednotlivých kvádrů byly použity sítě trojúhelníků – tzv. `Mesh`. Každý kvádr je tvořený samostatnou sítí. K jeho vytvoření slouží funkce `create_block` na výpisu 4.15. Na vstupu této funkce jsou kromě reference na model, do kterého funkce vykresluje, především souřadnice levého dolního předního rohu kvádru a jeho rozměry – `width` pro velikost na ose x, `depth` pro osu y a `height` pro osu z.

Při vytváření těchto sítí je nejprve nutné nadefinovat souřadnice vrcholů všech trojúhelníků v síti – viz pole `x_data`, `y_data` a `z_data` na řádcích výpisu 2–4, kde trojice i-tých položek v každém poli reprezentuje souřadnice jednoho vrcholu popořadě na osách x, y a z. Dále je nutné vybrat, mezi kterými trojicemi vrcholů mají být trojúhelníky vykreslené – to určují pole `i_data`, `j_data` a `k_data` na řádcích 6–8. Trojice čísel na i-tém indexu v těchto polích reprezentuje indexy vrcholů tvořících trojúhelník – index vrcholu přitom odpovídá pořadí definice vrcholu v předchozí trojici polí. Poté zbývá už jen přidání dané datové řady do modelu a nastavení jejích parametrů, jako je barva, jméno a podobně.

```
1  def create_block(fig, x, y, z, width, depth, height):
2      x_data = [x, x+width, x, x+width, x, x+width, x, x+width]
3      y_data = [y, y, y+depth, y+depth, y, y, y+depth, y+depth]
4      z_data = [z, z, z, z, z+height, z+height, z+height, z+height]
5
6      i_data = [0, 1, 0, 2, 1, 3, 2, 3, 4, 5, 0, 1]
7      j_data = [1, 4, 2, 4, 3, 5, 3, 6, 5, 6, 1, 2]
8      k_data = [4, 5, 4, 6, 5, 7, 6, 7, 6, 7, 2, 3]
9
10     fig.add_trace(go.Mesh3d(x=x_data, y=y_data, z=z_data,
11         i=i_data, j=j_data, k=k_data,
12         color="lightpink", flatshading=True,
13         name="konstrukce integrálu", hoverinfo="none"))
```

Výpis kódu 4.15: Vykreslení jednoho kvádru v rámci konstrukce určitého integrálu funkce dvou proměnných (kód zjednodušen pro přehlednost – byla mj. odebrána datová řada zajišťující červené obtažení hran).

4.3.7 Vizualizace Lagrangeovy metody pro vázané extrémy

V této vizualizaci je pomocí vrstevnic vyznačen graf funkce dvou proměnných a pomocí spojnicového grafu graf vazební funkce. Uživatel vybírá bod z definičního oboru vazební funkce – vybere hodnotu proměnné x a odpovídající hodnota y je dopočítána automaticky. Ve vybraném bodě jsou vykresleny gradienty funkce i vazby a je možné porovnat jejich vzájemný směr.

Vizualizace je na obrázku 4.9. Pomocí vrstevnicového grafu je v modelu vyznačen graf hlavní funkce – pro vrstevnice je použit barevný gradient, který naznačuje funkční hodnotu v daném bodě definičního oboru. Červeně je v modelu vykreslena vazební funkce. V uživatelem vybraném bodě jsou vykresleny dvě šipky znázorňující gradienty hlavní funkce (modře) a vazby (červeně). Z popisných prvků je přidáno vyznačení vybraného bodu, dvě vodící linie z tohoto bodu k souřadnicovým osám, popisky s hodnotami souřadnic zvoleného bodu a především také aktuální vrstevnice, na které se zvolený bod nachází, vykreslená modře čárkovaně.

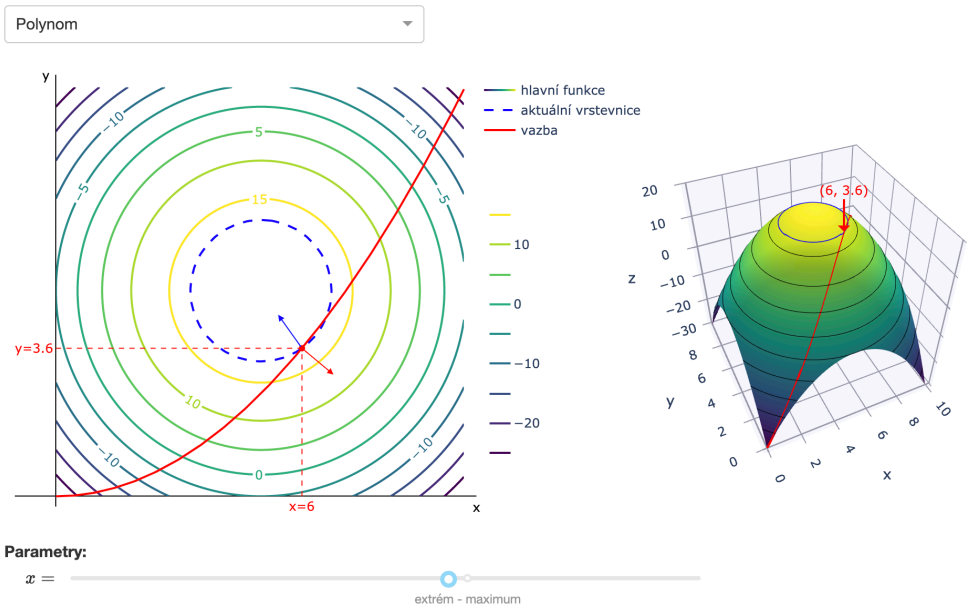
Pomocí posuvníku v dolní části modelu je možné vybírat bod na vazební funkci, ve kterém budou zobrazeny gradienty. Pro zdůraznění speciálních bodů (např. bodů podezřelých z extrémů), jsem ponechala posuvník bez zobrazených číselných hodnot a zvýraznila jsem pouze manuálně určené speciální body. Díky tomu uživatel bude vědět, na které místo posuvníku je potřeba kliknout, hledá-li konkrétní chování gradientů.

Je také možné zobrazit tooltip se souřadnicemi zvoleného bodu a tooltip s funkční hodnotou aktuální vrstevnice při najetí myší nad tyto prvky. Celým grafem je možné posouvat nebo jej přibližovat.

Oproti návrhu je opět možné volit z několika různých dvojic funkcí a vazeb. Dále jsem v pravé části obrazovky přidala také menší 3D graf, který obsahuje prostorové znázornění hlavní funkce pomocí povrchového grafu obarveného stejným gradientem jako vrstevnice. Na tento graf je modře promítnuta aktuální vrstevnice a červeně vazba – tedy vyznačení, ve kterých bodech se extrém hledá. Červená šipka pak označuje aktuálně vybraný bod. Díky tomuto grafu si je možné jednodušeji představit vertikální uspořádání bodů při pohybu po vazbě. S tímto grafem je možné otáčet a posouvat a je ho možné přibližovat.

Zajímavé body implementace matematických výpočtů Pro výpočet směru gradientů v kódu používám funkci `get_gradient_in_point`. Ta na vstupu dostává SymPy výraz a souřadnice bodu. S využitím SymPy funkce `derive_by_array` je vypočítán obecný gradient výrazu, do něhož jsou dosazeny souřadnice bodu a jako výsledek je následně vráceno pole hodnot typu `float`. Kód této funkce je na výpisu 4.16. Před vykreslením jsou oba gradienty navíc normalizovány.

Zajímavé body implementace vizualizace Prvním zajímavým bodem při vykreslení této vizualizace je vykreslení samotných gradientů. Jelikož má



Obrázek 4.9: Výsledná vizualizace Lagrangeovy metody pro vázané extrémy.

```
1  def get_gradient_in_point(expr, point_x, point_y):
2      x, y = sp.symbols("x y")
3      grad = sp.derive_by_array(expr, [x, y])
4      subs = {x: point_x, y: point_y}
5      in_point = [formula.subs(subs) for formula in grad]
6      return [float(expr) for expr in in_point]
```

Výpis kódu 4.16: Funkce pro výpočet gradientu funkce dvou proměnných v bodě.

Plotly omezené možnosti pro vykreslování šipek, musela jsem pro gradient využít textovou anotaci (která má ve výchozím stavu šipku vedoucí k popisovanému místu na grafu) s prázdným textem. Kód funkce `draw_gradient` je na výpisu 4.17. Na vstupu přijímá funkce referenci na model, do nějž mají být gradienty vykresleny, souřadnice počátečního a cílového bodu (přičemž cílový je označen šipkou) a barvu šipky. Tato funkce do modelu přidá odpovídajícím způsobem nastavenou anotaci – parametry `x` a `y` označují souřadnice bodu na který šipka ukazuje, parametry `ax` a `ay` označují pozici koncové části šipky.

Druhým zajímavým bodem je vykreslení aktuální vrstevnice. Pro její vykreslení jsem využila knihovnu scikit-image – kolekci algoritmů pro zpracování obrázků [50]. Konkrétně jsem využila funkci `find_contours` z modulu

```
def draw_gradient(fig, from_x, from_y, to_x, to_y, color):
    fig.add_annotation(x=to_x, y=to_y,
        ax=from_x, axref="x", ay=from_y, ayref="y",
        text="", arrowhead=2, arrowcolor=color, arrowsize=1.2)
```

Výpis kódu 4.17: Funkce pro vykreslení gradientů funkce a vazby ve vizualizaci Lagrangeovy metody pro vázané extrémy.

`measure`, která je schopná z dvourozměrného pole hodnot a hledané funkční hodnoty vrátit pole souvislých vrstevnic s danou funkční hodnotou, kde každá vrstevnice je reprezentována polem souřadnic bodů na vrstevnici [50].

Kód pro vykreslení vrstevnic je na výpisu 4.18. Při vykreslení jednotlivých vrstevnic (ve `for` cyklu na řádku 2 a dále) je třeba nejprve přepočítat souřadnice bodů na vrstevnici (řádky 3 a 4) – hodnoty vrácené funkcí `find_contours` odpovídají indexům v polích matice `fvalues`. Indexy hodnot v matici ale neodpovídají hodnotám proměnných x a y použitým pro jejich výpočet, proto je nutné je přepočítat. Následně už stačí pouze přidat do modelu datovou řadu a odpovídajícím způsobem ji nastavit.

```
contours = measure.find_contours(fvalues, point_z)
for contour in contours:
    cyvalues = [c[0]*STEP+YVALUES[0] for c in contour]
    cxvalues = [c[1]*STEP+XVALUES[0] for c in contour]
    fig.add_trace(go.Scatter(x=cxvalues, y=cyvalues, mode="lines",
        name="aktuální vrstevnice",
        line_color="blue", line_dash="dash",
        hovertemplate = f"f(x,y)={round(point_z, 2)}<extra></extra>"))
```

Výpis kódu 4.18: Vykreslení aktuální vrstevnice v ukázce Lagrangeovy metody pro vázané extrémy.

4.4 Testování aplikace

Aplikace je testována jak manuálně, tak také automatizovaně. Automatizované jsou především testy funkcí používaných napříč celou aplikací. Oproti tomu manuálně je nutné testovat jednotlivé vizualizace. Kromě těchto opakovaných metod proběhlo jednorázově také zjednodušené uživatelské testování.

4.4.1 Automatizované testy

Automatizované testy v aplikaci jsou dvojího druhu – unit testy pro ověření správnosti funkcionalit používaných napříč aplikací a tzv. smoke tesy, které si kladou za cíl ověřit, že je aplikace spustitelná a může být tedy nasazena na server. Obě sady testů jsou spuštěny jako součást automatizovaného nasazení tak, aby zabránily nasazení chybného software.

Unit testy Předmětem testování pomocí unit testů jsou funkce a třídy z balíčku `utils`, které jsou používány v mnoha vizualizacích. Jsou proto kritickou součástí aplikace, jež musí pro pohodlné tvoření dalších vizualizací fungovat bezchybně. Otestovány jsou všechny funkce v tomto balíčku kromě těch, které pracují s operačním systémem a dynamickými importy.

Pro testování jsem využila modul `unittest`, který je součástí standardní instalace Pythonu. Dle [51] se jedná o testovací framework, který se chová podobně jako hlavní testovací frameworky v jiných jazycích. Umožňuje testy automatizovat a agregovat. Disponuje také automatizovaným vyhledáváním testů například ve specifikovaném adresáři. Obsahuje řadu ověřovacích metod, které kontrolují výstup kódu – například `assertEqual`, `assertTrue`, případně `assertRaises`.

Jako příklad uvádím ve výpisu kódu 4.19 jednu z metod testujících funkci `normalize_vector` pro normalizaci vektoru zadaného jako pole číselných hodnot. V rámci testu nejprve získám vektor vrácený touto funkcí. Poté ověřím, zda jeho jednotlivé složky odpovídají očekávaným hodnotám – jelikož se jedná o nepřesnou floating point reprezentaci desetinných čísel, je nutné používat metodu `assertAlmostEqual`, která porovnává na rovnost, ale akceptuje při tom malé odchylky způsobené právě touto nepřesnou reprezentací. Na závěr testu je ještě provedena kontrola, zda je Eukleidovská norma vráceného vektoru skutečně rovná jedné.

```
1  def test_normalize_vector(self):
2      normalized = normalize_vector([2, 5, 4, 2])
3      self.assertAlmostEqual(normalized[0], 2/7)
4      self.assertAlmostEqual(normalized[1], 5/7)
5      self.assertAlmostEqual(normalized[2], 4/7)
6      self.assertAlmostEqual(normalized[3], 2/7)
7      self.assertAlmostEqual(math.sqrt(sum([i**2 for i in normalized])), 1)
```

Výpis kódu 4.19: Test funkce pro normalizaci vektorů (kód upraven pro přehlednost).

Spuštění sady unit testů nad konečnou podobou implementace proběhlo bez nalezených chyb.

Smoke testy Pro bezpečnější proces automatizovaného nasazení je vhodné ověřit, že je aplikace spustitelná a neobsahuje vážné chyby, než je publikována na veřejně dostupném serveru. Tento účel pokrývají takzvané smoke testy.

Součástí automatizovaného testování této aplikace jsou dva testy tohoto typu – první kontroluje, zda je aplikace spustitelná a druhý kontroluje přítomnost povinných atributů ve vizualizacích. Tyto testy nekontrolují, zda aplikace funguje správně, ani zda je zobrazen správný obsah – slouží pouze k nalezení nejzávažnějších problémů. Přesto jsou nedílnou součástí testovacího procesu, díky níž není možné na server nasadit například nespustitelnou verzi.

První test aplikaci sestaví, spustí a ihned zase ukončí. Tím se ověří, že aplikace neobsahuje vážné chyby či syntaktické nedostatky. Kvůli tomuto testu bylo nutné přistoupit u smoke testů k pokročilejší metodě testování popsané v dokumentaci frameworku Dash [32]. Tato metoda využívá testovací nástroj `pytest` (rozšířený o konkrétní funkcionality potřebné pro testování aplikací psaných ve framewroku Dash) a Selenium WebDriver. Pomocí těchto technologií je pak simulována interakce uvnitř prohlížeče. Samotný kód testu je velmi jednoduchý (viz výpis 4.20). Aplikace je v něm opravdu pouze načtena a spuštěna.

```
1 def test_smoke_build(dash_duo):
2     app = import_app("app")
3     dash_duo.start_server(app)
```

Výpis kódu 4.20: Smoke test spuštění aplikace.

Druhý test kontroluje, zda všechny vizualizace obsahují povinné atributy a zda jsou tyto atributy správného typu. Tento test doplňuje první test spustitelnosti o odhalení dalších závažných chyb ve struktuře vizualizací.

Smoke testy spuštěné nad konečnou verzí aplikace proběhly bez chyb.

4.4.2 Manuální testy

Manuální testy pokrývají funkčnost jednotlivých vizualizací. Testování vizualizací není automatizováno, jelikož by požadavek na vytvoření automatizovaných testů ke každé vizualizaci byl v rozporu s požadavkem na co nejjednodušší, nejrychlejší a nejpřímočařejší tvorbu vizualizace – vytvoření těchto testů by bylo zdlouhavé a neúměrně náročné. Místo toho jsem sestavila obecný testovací scénář, který pomůže manuálně ověřit, že je vytvořená vizualizace funkční.

Scénář je v plném rozsahu dostupný v příloze D.4. Je doporučeno jeho provedení v různých prohlížečích a na různých operačních systémech. Kontroluje mimo jiné také následující prvky aplikace:

- obsah hlavní stránky s rozcestníkem,

- obsah a interaktivitu samostatné stránky s vizualizací,
- obsah a interaktivitu integrovatelné stránky s vizualizací,
- základní responzivitu obou stránek.

Pomocí tohoto scénáře byla ověřena funkčnost všech vytvořených vizualizací na následujících konfiguracích:

- **MacOS** verze 11.1 – prohlížeče Chrome (verze 89), Safari (verze 14) a Firefox (verze 87),
- **Windows 10** – prohlížeče Chrome (verze 90), Firefox (verze 88) a Edge (verze 90),
- **Linux (Kubuntu)** verze 18.10 – prohlížeče Chrome (verze 90) a Firefox (verze 88).

Většina nalezených nedostatků byla kosmetického rázu a týkala se zejména nekonzistencí mezi prohlížeči a operačními systémy. Všechny takovéto nedostatky byly opraveny.

Jediný problém, který se opravit nepodařilo, se projevuje v prohlížeči Firefox napříč operačními systémy. Anotace v Plotly modelu, které využívají sazby pomocí LaTeX, jsou v tomto prohlížeči nesprávně pozicovány a většinou se proto objeví na jiných místech modelu nebo vůbec. Dle nedostatků reportovaných v GitHub repozitáři Plotly v lednu 2021 [52, 53] se nejspíš jedná o projev známého bugu, u kterého se mi nepodařilo dohledat zmínku o jeho opravení.

Další objevený nedostatek, který ale byl v rámci možností opraven, se projevil ve vrstevnicích povrchových grafů. Na operačních systémech MacOS a Windows tyto vrstevnice nereagovaly na atribut `width`, udávající jejich šířku. Na Linuxu však tento atribut funguje a ve výchozím stavu jsou na něm proto v grafech vrstevnice mnohem výraznější. Tato nekonzistence může být opravena manuálním nastavením nejmenší možné hodnoty atributu `width`. I po opravě jsou ale vrstevnice na operačním systému Linux mírně výraznější.

4.4.3 Uživatelské testování

V rámci vývoje první verze aplikace proběhlo jednorázově zjednodušené uživatelské testování. Aplikace byla představena vedoucímu této práce, který si měl možnost vyzkoušet její ovládání a zhodnotit možnosti nabízené jednotlivými ukázkami.

V rámci tohoto testování bylo objeveno několik nedostatků v návrhu uživatelského rozhraní. Jako významný nedostatek označil uživatel například podobu posuvníků pro výběr jedné hodnoty. Kvůli obarvení části posuvníku mezi jeho levým okrajem a zvolenou hodnotou byl příliš podobný posuvníku

pro výběr dvojice hodnot (intervalu) a nebyl proto uživatelsky přívětivý. Dále uživatel identifikoval několik nedostatků v textových popiscích v rámci aplikace.

Uživatel také navrhl některé možnosti pro vylepšení stávající podoby aplikace. Nejvýraznější změnou bylo parametrizování některých ukázek tak, aby bylo možné vybírat z několika základních funkcí pro další aproximaci, integraci či hledání tečen. Dále bylo navrženo několik vylepšení vizuální stránky ukázek – uživatel doporučil například zvýraznění hran kvádrů při vizualizaci konstrukce určitého integrálu funkce dvou proměnných.

Nedostatky nalezené při uživatelském testování byly ve své většině opraveny. Jediným neopraveným problémem bylo neúplné zobrazení textových anotací v prohlížeči Firefox, které je ale dle závěrů manuálního testování aplikace způsobeno známým bugem (viz 4.4.2).

Navržená vylepšení se podařilo implementovat bezezbytku.

4.5 Nasazení aplikace

V této sekci se budu věnovat nasazení hotové aplikace. Popíši „proof of concept“ nasazení na platformu Heroku automatizované pomocí GitLab CI/CD, které jsem provedla pro účely této práce. Dále zmíním doporučení pro reálné nasazení v systému MARAST – jak aplikaci nasadit, jak nasazení automatizovat a jak aplikaci se systémem MARAST propojit.

4.5.1 Nasazení pro účely práce

Pro účely této práce jsem hotovou aplikaci nasadila na webový server pomocí služby Heroku, aby byla veřejně dostupná a testovatelná. Dále jsem nasazování automatizovala pomocí GitLab CI/CD tak, aby každá úprava `master` větve v repozitáři aplikace spustila nasazení a publikování této upravené verze.

Nasazení na Heroku Heroku je dle svých webových stránek [54] cloudová platforma pro nasazení a provozování moderních aplikací. Kromě Pythonu podporuje i další jazyky, například Node.js, Ruby nebo Javu. Tuto platformu jsem zvolila mimo jiné také proto, že ji dokumentace frameworku Dash [32] přímo doporučuje pro bezplatné nasazování aplikací.

Dle stránek Heroku [54] je bezplatné nasazení aplikace na této platformě vhodné zejména pro prototypy, nekomerční aplikace a soukromé projekty. Bezplatně nasazená aplikace má totiž omezený počet hodin, kdy může být spuštěná – konkrétně maximálně 550 hodin měsíčně. Kromě toho se navíc aplikace po 30 minutách nečinnosti uspává a její opětovné nastartování může chvíli trvat. Z těchto důvodů není tento typ nasazení trvalým řešením, ale slouží zejména jako „proof of concept“ pro účely této práce.

Pro nasazení na Heroku by aplikace dle [54] měla obsahovat několik souborů:

- `requirements.txt` obsahující seznam knihoven Pythonu (včetně jejich verzí) potřebných pro její spuštění,
- `runtime.txt` s uvedenou požadovanou verzí Pythonu,
- `Procfile` s konkrétním příkazem pro spuštění aplikace.

Pro nasazení aplikace na platformu Heroku pak je dle [54] potřeba na Heroku vytvořit účet a mít lokálně instalovaný verzovací systém Git a Heroku CLI. Po vytvoření Heroku aplikace a propojení lokálního adresáře se vzdáleným („remote“) Git repozitářem na Heroku je poté možné vyvíjenou aplikaci publikovat. Pomocí `git push` do tohoto vzdáleného repozitáře jsou také publikovány jakékoli změny aplikace.

GitLab CI/CD Aby byl proces publikování změn v aplikaci ještě jednodušší, rozhodla jsem se ho automatizovat pomocí GitLab CI/CD. Automatizovanému nasazení předchází spuštění automatizovaných testů, aby se zajistila základní kontrola kvality a zabránilo se nasazení nefunkční aplikace.

Automatizované testy se provádí po jakémkoli nahrání změn do repozitáře. Testy se provádí ve dvou tzv. úkolech (jobs) – jeden slouží pro unit testy a druhý pro smoke testy.

Při jakýchkoli úpravách `master` větve (tedy i při merge jednotlivých větví) se pak spustí celý proces nasazení, který je rozdělen do dvou fází. V první fázi jsou spuštěny automatizované testy. Pokud jakákoli z testovacích sad selže, je proces nasazení zastaven. Pokud obě sady dopadnou bez chyb, přistoupí se k další fázi, ve které je provedeno nasazení na Heroku – stav `master` větve je synchronizovaný se stavem nasazené aplikace. V této fázi je také provedeno automatizované generování obrázků, které se na GitLabu uloží v podobě tzv. „artefaktů“. Generování obrázků se budu více věnovat v následující sekci.

Soubor, který definuje průběh procesu CI/CD se jmenuje `.gitlab-ci.yml`. Tento soubor obsahuje definovaný `image`, což je dle [55] základ pro Docker container, ve kterém jsou spouštěny jednotlivé skripty. Proces CI/CD v této práci využívá Docker image `python:3.9-slim-buster`, který je dle [56] jedním z oficiálních pro Python. Dále soubor obsahuje definování dvou fází pro testování a nasazení. Poté obsahuje čtyři úkoly provádějící jednotlivé úkony procesu nasazení.

Po nasazení se využívá úkol s názvem „production“ na výpisu 4.21, který instaluje potřebné závislosti a provede nasazení na Heroku. Využívá také GitLab „environment variables“.

4.5.2 Doporučení pro nasazení v systému MARAST

Jak jsem již zmínila v předchozí sekci, aktuální varianta nasazení – bezplatně na platformě Heroku – není pro reálné použití příliš vhodná. Do budoucna je

```
production:
    stage: deploy
    script:
    - apt-get update
    - apt-get install -y ruby-dev ruby-json
    - apt-get install -y git curl
    - gem install dpl-heroku
    - dpl --provider=heroku --app=$HEROKU_APP_PRODUCTION
        --api-key=$HEROKU_API_KEY
    only:
    - master
```

Výpis kódu 4.21: Definice GitLab CI/CD úkolu pro nasazení implementované aplikace.

například možné použít některou placenou platformu, případně nasadit aplikaci na soukromý server. Dále se v této sekci budu zabývat složitější, ale také pravděpodobnější variantou, a to sice právě nasazením na soukromý server. Zároveň shrnu také možnosti propojení nasazené aplikace a serveru MARAST.

Nasazení na server V první řadě bych doporučila rozšíření procesu nasazení o druhý, testovací server. Provedené změny by se nenasazovaly ihned na produkční server, ale místo toho by se nasadily na tento testovací server, kde by bylo možné ověřit jejich správné fungování. Na tomto serveru by se také lépe prováděly manuální testy, jelikož by se aplikace snáze spouštěla v různých prostředích. Teprve po potvrzení funkčnosti na testovacím serveru by se pak provedlo nasazení na produkci. Nasazení na testovací server by navíc bylo vhodné zapojit do procesu CI/CD zmíněného v sekci 4.5.1, aby bylo prováděno automatizovaně.

Pro nasazení aplikace na server je dle [57] doporučováno několik vrstev technologií. Základem je operační systém, na něm je postavena infrastruktura sestávající z webového serveru, WSGI serveru a samotné aplikace. Webový server dle [58] přijímá HTTP dotazy klientů a odpovídá na ně. Známé webové servery jsou například Nginx nebo Apache. WSGI server pak dle [59] tvoří prostředníka mezi webovým serverem a aplikací v Pythonu, přijímá požadavky z webového serveru a předává je aplikaci, která je na něm spuštěná. Tento server podporuje škálování a flexibilitu. Známým WSGI serverem je například Gunicorn (který je v používán i v současném nasazení pro spuštění aplikace na Heroku).

Pro nasazení aplikace na server je dále možné použít například Docker containers. Dle [60] umožňuje Docker jednodušší nasazení napříč prostředími, protože aplikaci „zabalí“ společně s operačním systémem a všemi nutnými zá-

vislostmi (včetně webového a WSGI serveru). Dle [61] container může vycházet z nějakého již existujícího image – lze použít například nějaký z oficiálních pro Python. V rámci containeru se pak provede instalace závislostí a spuštění serveru s aplikací.

Nasazení na server lze pomocí CI/CD automatizovat mnoha způsoby. Například podle [62] se lze ze skriptu prováděného během procesu CI/CD připojit na vzdálený server pomocí SSH a vygenerovaného klíče (je ale přitom třeba dbát na bezpečnost). Po připojení na vzdálený server lze mimo jiné také stáhnout obsah repozitáře z GitLabu, sestavit Docker container nebo, jak navrhuje [62], je možné využít tzv. GitLab Container Registry. Dle [63] se jedná o prostor, kam může každý projekt ukládat Docker images a ze kterého mohou být tyto images stahovány. Stránka [62] navrhuje postup, kde je Docker image obsahující aplikaci a její závislosti sestaven a nahrán do Container Ragistry v rámci CI/CD procesu. V rámci následujícího úkolu tohoto procesu pak je možné se vzdáleně připojit na server, z něj stáhnout Docker image s aplikací z Container Registry a spustit jej.

Po nasazení na server doporučuji dále provést zátěžové testy, aby se ověřilo, že zvolená konfigurace zvládne obsloužit všechny uživatele pokoušející se k ní připojit.

Propojení se systémem MARAST Díky zdvojení stránek tvořených pro jednu vizualizaci je možné využít obě dříve zmiňované varianty propojení.

Se samostatnými stránkami je možné aplikaci využívat jako zcela soběstačnou samostatnou webovou stránku, na kterou je možné se odkazovat pomocí URL adres vizualizací. Tyto odkazy je možné doplnit obrázky vytvořenými generátorem zmiňovaným v sekci 4.6.

Druhou možností pak je zobrazení webové stránky s aplikací v rámci systému MARAST pomocí HTML elementů `iframe` s využitím integrovatelného typu stránky. Ten zobrazuje pouze model bez doprovodného kontextu a navíc je optimalizován tak, aby byl zobrazitelný a použitelný bez scrollování i v oknech se šířkou 800 px a výškou 650 px.

4.6 Generování obrázků z vizualizací

Případný odkaz na aplikaci je vhodné doplnit alespoň statickým obrázkem vizualizace. Proto jsem vytvořila skript `generate_images.py`, který se nachází v adresáři se zdrojovými kódy. Obrázky se pak pomocí tohoto skriptu dají vygenerovat buď manuálně, nebo je možné využít jejich automaticky generovanou verzi, která je součástí nastaveného GitLab CI/CD.

Při spuštění bez parametrů skript vygeneruje a uloží do složky `images` v adresáři se zdrojovými kódy statické obrázky všech vizualizací ve složce `visualizations` ve formátu SVG. Jiný formát je možné zvolit pomocí přepínače `-f`, podporované jsou například formáty JPG, PNG nebo PDF. Pomocí

přepínače `-n` a jména vizualizace (totožné se jménem souboru s vizualizací bez přípony `.py`) je možné zvolit konkrétní vizualizaci ze složky `visualizations`, jejíž statický obrázek se má vygenerovat. Například vygenerování obrázku ve formátu JPG z vizualizace tečny ke grafu funkce jedné proměnné je možné takto: `python generate_images.py -n tangent_2d -f jpg`.

Pro generování obrázků se využívá dynamické získání seznamu vizualizací a dynamický import požadovaného souboru, které byly popsané v sekci 4.2.4. Pro vytvoření objektu `Figure` s vizualizací se využije povinně definovaná funkce `create_figure` a její výchozí argumenty `default_args` – pokud je tedy třeba změnit argumenty, se kterými je obrázek vygenerován, postačí změnit hodnoty výchozích parametrů. Pro export do statického formátu pak slouží metoda `figure.write_image` poskytnutá knihovnou Plotly. Kód funkce pro export obrázku je na výpisu 4.22.

```
1  def export_image(name, format_suffix):
2      module = importlib.import_module(f"visualizations.{name}")
3      fig = module.create_figure(**module.default_args)
4      fig.write_image(f"images/{name}.{format_suffix}")
```

Výpis kódu 4.22: Funkce pro vygenerování obrázku z vizualizace.

Skript pro vygenerování obrázků je součástí GitLab CI/CD. Po úspěšném provedení testů se vedle nasazení spustí také generování obrázků ve formě tzv. „artefaktů“. Tyto obrázky se dle dokumentace [64] na GitLabu uloží a je možné je z něj následně stáhnout. Skript jsem nastavila tak, aby tyto obrázky byly na GitLabu uloženy po dobu jednoho týdne. Dle dokumentace [64] je nicméně možné manuálně uložit artefakty do doby, dokud nebudou vytvořeny nové.

Závěr

V rámci této diplomové práce jsem sestavila systém technologií pro implementaci interaktivních grafických ukázek matematických výpočtů a jejich publikování na webu a využila jsem toto navržené řešení při implementaci prototypů ukázek. Pro implementaci jsem po analýze zvolila systém Plotly pro Python pro tvorbu vizualizací ve spojení s frameworkem Dash pro jejich webové sdílení. Implementovala jsem samostatnou webovou aplikaci, jejíž jsou jednotlivé interaktivní ukázky součástí. Navrhla jsem dále možnosti pro její propojení se systémem MARAST.

Naplnění cílů

V úvodu práce jsem stanovila jeden hlavní a tři dílčí cíle této práce. Všechny tyto cíle jsem v práci splnila.

V kapitole 1 jsem provedla analýzu dostupných vykreslovacích systémů, například Bokehu, Plotly nebo D3.js. Stanovila jsem kritéria hodnocení a vyhodnotila jejich splnění jednotlivými systémy. Všechny systémy jsem také otestovala pomocí implementace jednoduché ukázky. Tím jsem splnila první dílčí cíl.

Dále jsem v kapitole 2 sestavila seznam modelových příkladů vhodných pro interaktivní ukázky při výuce. Zahrnula jsem mimo jiné tečnu ke grafu funkce nebo konstrukci určitého integrálu. Seznámila jsem se s matematickou teorií, která je základem pro tvorbu těchto ukázek. Navrhla jsem podobu ukázek včetně jejich vstupů, výstupů a parametrů. Druhý dílčí cíl byl tedy také splněn.

V kapitole 3 jsem stanovila požadavky na zvolený vykreslovací systém a provedla jsem jeho výběr – zvolila jsem systém Plotly v kombinaci s frameworkem Dash. Následně jsem pomocí tohoto systému provedla v kapitole 4 implementaci modelových ukázek jako součástí samostatné webové aplikace. Aplikaci jsem otestovala automatizovaně i manuálně a provedla jsem její zkušební nasazení. Proces nasazování jsem dále automatizovala pomocí GitLab

CI/CD. Na závěr jsem navrhla možnosti propojení hotové aplikace se systémem MARAST – pomocí URL odkazů, případně HTML elementů `iframe`. Těmito kroky jsem naplnila také třetí dílčí cíl.

Hlavním cílem práce bylo sestavit systém technologií, které umožní vytvářet a sdílet interaktivní ukázky na webu, využít tento systém při implementaci modelových příkladů a zaměřit se na jednoduchost postupu pro vytváření vizualizací. Tento cíl jsem splnila díky volbě technologií s požadovanými vlastnostmi, implementaci webové aplikace s jednotlivými ukázkami a sestavení jednoduchého postupu pro vytvoření a publikování nové vizualizace, který vedle samotného vytvoření vizualizace neklade na jejího autora téměř žádné další nároky. Zvolila jsem navíc technologie s kvalitní dokumentací, která ještě více usnadní postup vytváření vizualizace.

Možnosti dalšího rozšíření

Přestože je vytvořená webová aplikace již v současnosti velmi dobře použitelná za požadovaným účelem, nabízí stále další možnosti rozšíření. Tím nejdůležitějším je implementace pokročilejší responzivity – tedy možnosti ji zobrazovat a používat také na mobilních zařízeních. V současnosti je aplikace dimenzovaná pro použití pouze na zařízeních s dostatečnou velikostí displeje.

Dále by bylo možné implementovat také dodatečnou vrstvu zabezpečení tak, aby ukázky mohli zobrazovat pouze přihlášení uživatelé – studenti a pracovníci Fakulty informačních technologií ČVUT.

Aplikaci je také možné rozšiřovat přidáváním nových interaktivních ukázek.

Zhodnocení výsledků a využitelnost v praxi

Výsledná aplikace splňuje cíle práce a účel, za nímž byla vytvořena. Po jejím propojení se systémem MARAST bude studentům pomáhat pochopit látku matematických předmětů, ujasnit si souvislosti a zasadit získané znalosti do kontextu. Bude tak přínosem pro stávající i budoucí studenty Fakulty informačních technologií ČVUT.

Literatura

1. KALVODA, Tomáš; KLOUDA, Karel. *MARAST* [online] [cit. 2020-04-29]. Dostupné z: `https://marast.fit.cvut.cz`.
2. *An introduction to LaTeX* [online] [cit. 2020-09-30]. Dostupné z: `https://www.latex-project.org/`.
3. TANTAU, Till. *The TikZ and PGF Packages, Manual for version 3.1.6* [online]. 2020-09-28 [cit. 2020-09-30]. Dostupné z: `https://mirrors.nic.cz/tex-archive/graphics/pgf/base/doc/pgfmanual.pdf`.
4. *Python* [online]. Python Software Foundation, © 2020 [cit. 2020-10-03]. Dostupné z: `https://www.python.org`.
5. *jupyter* [online]. Project Jupyter, © 2020 [cit. 2020-10-17]. Dostupné z: `https://jupyter.org`.
6. *nbviewer* [online]. 2020 [cit. 2020-10-17]. Dostupné z: `https://nbviewer.jupyter.org`.
7. *binder* [online]. The Binder Team, © 2017 [cit. 2020-10-17]. Dostupné z: `https://mybinder.org`.
8. *NumPy* [online]. NumPy, © 2020 [cit. 2020-10-03]. Dostupné z: `https://numpy.org`.
9. *SciPy* [online]. SciPy, © 2020 [cit. 2020-10-03]. Dostupné z: `https://scipy.org`.
10. *SciPy* [online]. SymPy Development Team, © 2020 [cit. 2020-10-03]. Dostupné z: `https://www.sympy.org/en/index.html`.
11. *How to reduce mybinder.org repository startup time* [online]. 2020-07-09 [cit. 2020-03-05]. Dostupné z: `https://discourse.jupyter.org/t/how-to-reduce-mybinder-org-repository-startup-time/4956`.
12. *MDN web docs* [online]. Mozilla a samostatní přispěvatelé, © 2020 [cit. 2020-10-24]. Dostupné z: `https://developer.mozilla.org/en-US/`.

13. JONG, Jos de. *math.js* [online]. © 2020 [cit. 2020-10-24]. Dostupné z: https://mathjs.org.

14. *Canvas API* [online]. MDN web docs, © 2005–2019 [cit. 2021-01-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

15. *SVG: Scalable Vector Graphics* [online]. MDN web docs, © 2005–2019 [cit. 2020-01-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/SVG.

16. *TikZ package* [online]. Overleaf, © 2020 [cit. 2020-09-26]. Dostupné z: https://www.overleaf.com/learn/latex/TikZ_package.

17. CREMER, Jacques. *A very minimal introduction to TikZ* [online]. 2011-03-11 [cit. 2020-09-26]. Dostupné z: http://cremeronline.com/LaTeX/minimaltikz.pdf.

18. *Pgfplots package* [online]. Overleaf, © 2020 [cit. 2020-09-30]. Dostupné z: https://www.overleaf.com/learn/latex/pgfplots_package.

19. *Does TikZ support interactive animation?* [Online]. TeX - LaTeX Stack Exchange, 2016-02-13 [cit. 2020-09-30]. Dostupné z: https://tex.stackexchange.com/questions/98446/does-tikz-support-interactive-animation.

20. *Matplotlib* [online]. The Matplotlib development team, © 2020 [cit. 2020-10-03]. Dostupné z: https://matplotlib.org/index.html.

21. *What is MATLAB?* [Online]. The MathWorks, Inc., © 2020 [cit. 2020-10-03]. Dostupné z: https://uk.mathworks.com/discovery/what-is-matlab.html.

22. *Bokeh* [online]. Bokeh contributors, © 2020 [cit. 2020-10-17]. Dostupné z: https://bokeh.org.

23. BOSTOCK, Mike. *D3.js - Data-Driven Documents* [online]. © 2020 [cit. 2020-10-24]. Dostupné z: https://d3js.org.

24. *d3-3d* [online]. Niekes, © 2020 [cit. 2020-10-24]. Dostupné z: https://github.com/Niekes/d3-3d.

25. *An Intro to Raphaël* [online] [cit. 2020-10-31]. Dostupné z: http://raphaeljs.com.

26. BARANOVSKIY, Dmitry. *Raphaël—JavaScript Library* [online] [cit. 2020-10-31]. Dostupné z: https://dmitrybaranovskiy.github.io/raphael/.

27. WILLIAMS, James. *Introduction to Raphaël.js* [online]. 2013-10-29 [cit. 2020-10-31]. Dostupné z: https://www.html5rocks.com/en/tutorials/raphael/intro/.

28. DAWBER, Damian. *An Introduction to the Raphael JS Library* [online]. Envato Pty Ltd., 2009-10-14 [cit. 2020-10-31]. Dostupné z: https://code.tutsplus.com/tutorials/an-introduction-to-the-raphael-js-library--net-7186.

29. *Raphaël3d* [online] [cit. 2020-10-31]. Dostupné z: https://mech.fsv.cvut.cz/~stransky/en/software/raphael3d/.

30. *Plotly Open Source Graphing Libraries* [online]. Plotly, © 2021 [cit. 2021-02-03]. Dostupné z: https://plotly.com/graphing-libraries/.

31. *Plotly Python Open Source Graphing Library* [online]. Plotly, © 2021 [cit. 2021-02-03]. Dostupné z: https://plotly.com/python/.

32. *Dash User Guide* [online] [cit. 2021-02-03]. Dostupné z: https://dash.plotly.com.

33. *Plotly JavaScript Open Source Graphing Library* [online]. Plotly, © 2021 [cit. 2021-02-03]. Dostupné z: https://plotly.com/javascript/.

34. *Simple Widget Introduction* [online]. Project Jupyter, © 2017 [cit. 2021-02-17]. Dostupné z: https://ipywidgets.readthedocs.io/en/stable/examples/Widget%5C%20Basics.html.

35. *Manim Documentation* [online]. The Manim Community Dev Team, © 2020 [cit. 2021-01-27]. Dostupné z: https://docs.manim.community/en/v0.2.0/.

36. *Altair: Declarative Visualization in Python* [online]. Altair Developers, © 2019 [cit. 2021-01-27]. Dostupné z: https://altair-viz.github.io/index.html.

37. *Vega – A Visualization Grammar* [online] [cit. 2021-01-27]. Dostupné z: https://vega.github.io/vega/.

38. *Vega-Lite – A Grammar of Interactive Graphics* [online] [cit. 2021-01-27]. Dostupné z: http://vega.github.io/vega-lite/.

39. *Chart.js* [online] [cit. 2021-01-28]. Dostupné z: https://www.chartjs.org.

40. *JSXGraph* [online] [cit. 2021-02-03]. Dostupné z: https://jsxgraph.org/wp/index.html.

41. KALVODA, Tomáš; VAŠATA, Daniel. *Základy matematické analýzy – studijní text* [online]. 2020-11-19 [cit. 2020-11-21]. Dostupné z: https://courses.fit.cvut.cz/BI-ZMA/latex/lectures/bi-zma-skripta.pdf.

42. KLOUDA, Karel; KALVODA, Tomáš; STAROSTA, Štěpán. *Analýza I: funkce více proměnných* [online]. 2020-11-03 [cit. 2020-11-14]. Dostupné z: https://courses.fit.cvut.cz/MI-MPI/latex/lectures/czech/mi-mpi-prednaska-11-analyza-I-slides.pdf.

43. HRABÁK, Pavel; KALVODA, Tomáš; PETR, Ivo. *Základy matematické analýzy – Limita funkce* [online]. 2020-11-12 [cit. 2020-11-14]. Dostupné z: https://courses.fit.cvut.cz/BI-ZMA/latex/lectures/bi-zma-05-limita-funkce.pdf.

44. HRABÁK, Pavel; KALVODA, Tomáš; PETR, Ivo. *Základy matematické analýzy – Derivace funkce* [online]. 2020-11-12 [cit. 2020-11-14]. Dostupné z: https://courses.fit.cvut.cz/BI-ZMA/latex/lectures/bi-zma-07-derivace.pdf.

45. KLOUDA, Karel; KALVODA, Tomáš; STAROSTA, Štěpán. *Analýza IV: integrace* [online]. 2020-12-01 [cit. 2021-02-05]. Dostupné z: https://courses.fit.cvut.cz/MI-MPI/latex/lectures/czech/mi-mpi-prednaska-14-analyza-IV-integrace-slides.pdf.

46. KLOUDA, Karel; KALVODA, Tomáš; STAROSTA, Štěpán. *Analýza III: vázané extrémy* [online]. 2020-11-24 [cit. 2020-12-06]. Dostupné z: https://courses.fit.cvut.cz/MI-MPI/latex/lectures/czech/mi-mpi-prednaska-13-analyza-III-vazane-extremy.pdf.

47. *<iframe>: The Inline Frame element* [online]. MDN web docs, © 2005–2021 [cit. 2021-02-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe.

48. *os — Miscellaneous operating system interfaces* [online]. Python Software Foundation, © 2021 [cit. 2021-04-02]. Dostupné z: https://docs.python.org/3/library/os.html.

49. *importlib — The implementation of import* [online]. Python Software Foundation, © 2021 [cit. 2021-04-02]. Dostupné z: https://docs.python.org/3/library/importlib.html.

50. *scikit-image* [online]. scikit-image development [cit. 2021-03-26]. Dostupné z: https://scikit-image.org/.

51. *unittest — Unit testing framework* [online]. Python Software Foundation, © 2021 [cit. 2021-04-22]. Dostupné z: https://docs.python.org/3/library/unittest.html.

52. *Firefox Latex/Mathjax Rendering Broken* [online]. © 2021 [cit. 2021-04-22]. Dostupné z: https://github.com/plotly/plotly.js/issues/5391.

53. *LaTeX/MathJax broken using Firefox v84.0.1* [online]. © 2021 [cit. 2021-04-22]. Dostupné z: https://github.com/plotly/plotly.js/issues/5374.

54. *Heroku* [online]. Salesforce.com, © 2021 [cit. 2021-04-07]. Dostupné z: https://www.heroku.com.

55. *Run your CI/CD jobs in Docker containers* [online]. GitLab Docs [cit. 2021-04-24]. Dostupné z: https://docs.gitlab.com/ee/ci/docker/using_docker_images.html.

56. *Python - Docker Official Images* [online]. Docker Hub, © 2021 [cit. 2021-04-24]. Dostupné z: https://hub.docker.com/_/python.

57. *Deployment* [online]. Full Stack Python, 2021 [cit. 2021-04-30]. Dostupné z: https://www.fullstackpython.com/deployment.html.

58. *Web Servers* [online]. Full Stack Python, 2021 [cit. 2021-04-30]. Dostupné z: https://www.fullstackpython.com/web-servers.html.

59. *WSGI Servers* [online]. Full Stack Python, 2021 [cit. 2021-04-30]. Dostupné z: https://www.fullstackpython.com/wsgi-servers.html.

60. *Docker* [online]. Full Stack Python, 2021 [cit. 2021-04-24]. Dostupné z: https://www.fullstackpython.com/docker.html.

61. FAJAR, Ridwan. *Build Python Web Application using Flask and Docker* [online]. Medium, 2020-04-18 [cit. 2021-04-24]. Dostupné z: https://medium.com/swlh/build-python-web-application-using-flask-and-docker-1e38cfa12f22.

62. NÖTHIGER, Mike. *How To Set Up a Continuous Deployment Pipeline with GitLab CI/CD on Ubuntu 18.04* [online]. DigitalOcean, LLC., © 2021 [cit. 2021-04-24]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-continuous-deployment-pipeline-with-gitlab-ci-cd-on-ubuntu-18-04.

63. *GitLab Container Registry* [online]. GitLab Docs [cit. 2021-04-24]. Dostupné z: https://docs.gitlab.com/ee/user/packages/container_registry/.

64. *Job artifacts* [online]. GitLab Docs [cit. 2021-04-24]. Dostupné z: https://docs.gitlab.com/ee/ci/pipelines/job_artifacts.html.

Dodatek A

Seznam použitých zkratek

API Application Programming Interface

CI/CD Continuous Integration and Continuous Deployment

CLI Command Line Interface

CSS Cascading Style Sheets

ČVUT České vysoké učení technické

GIF Graphics Interchange Format

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

JPG Joint Photographic Experts Group

JSON JavaScript Object Notation

MARAST Matematika radostně

MATLAB Matrix Laboratory

MDN Mozilla Developer Network

MP4 MPEG-4 Part 14

MPI Matematika pro informatiku

PDF Portable Document Format

PGF Portable Graphics Format

PNG Portable Network Graphics

SSH Secure Shell

SVG Scalable Vector Graphics

URL Uniform Resource Locator

VML Vector Markup Language

W3C World Wide Web Consortium

WSGI Web Server Gateway Interface

ZMA Základy matematické analýzy

Dodatek B

Instalační příručka

B.1 Použití online verze výsledné aplikace

Na adrese `https://fit-cvut-interactive-math.herokuapp.com` je možné spustit a používat nasazenou verzi aplikace.

Upozorňuji na možné delší trvání prvotního spuštění aplikace, které je způsobeno vlastnostmi platformy Heroku popsanými v kapitole 4.

B.2 Spuštění výsledné aplikace z přiložených souborů

Pro lokální spuštění aplikace je potřeba mít nainstalovaný Python alespoň verze 3.9. Dále doporučuji mít instalovaný správce balíčků Pip a modul `venv` pro tvorbu virtuálních prostředí, případně nějaké jejich ekvivalenty. Poté je obecný postup následující.

1. V terminálu se přesuňte do adresáře se zdrojovými kódy aplikace.
2. Vytvořte virtuální prostředí ve složce `env` například pomocí příkazu `python3 -m venv env`.
3. Virtuální prostředí následně aktivujte, například pomocí zadání příkazu `source env/bin/activate` (po ukončení práce s aplikací je ho možné vypnout příkazem `deactivate`).
4. Nainstalujte potřebné balíčky uvedené v souboru `requirements.txt`, například příkazem `pip3 install -r requirements.txt`.
5. Spusťte aplikaci pro lokální zobrazení příkazem `python3 app.py`.
6. Zobrazte aplikaci v prohlížeči na adrese `http://127.0.0.1:8050/`.

Tento obecný postup byl testován na MacOS, instalace balíčků byla navíc otestována i pomocí Docker image `ubuntu:latest`. V závislosti na konfiguraci prostředí (operačním systému a již instalovaných balíčcích a programech) se mohou některé kroky tohoto postupu lišit. Zejména příkazy v krocích 2, 3 a 5 budou závislé na operačním systému. Dále krok 4 může vyžadovat instalaci dalších balíčků a programů, například kompilátoru `gcc` a balíčku `python3-wheel`.

B.3 Spuštění testů

Pro spuštění unit testů není třeba nic dodatečně instalovat. Postup jejich spuštění je následující.

1. V adresáři se zdrojovými kódy aktivujte virtuální prostředí, stejně jako při spouštění aplikace.
2. Pomocí příkazu `python3 -m unittest discover -s tests` poté sadu testů spusťte, detailní výpis lze aktivovat přepínačem `-v`.

Pro spuštění smoke testů je třeba mít instalovaný WebDriver například pro prohlížeč Chrome. Pro jeho instalaci následujte pokyny na stránkách `http://chromedriver.chromium.org/getting-started`. Složku se spustitelným souborem nezapomeňte zahrnout do systémové `PATH`. Poté postupujte podle následujících pokynů.

1. V adresáři se zdrojovými kódy aktivujte virtuální prostředí, stejně jako při spouštění aplikace.
2. Spusťte testy pomocí příkazu `pytest -k smoke -W ignore`.

B.4 Generování obrázků

Generování obrázků je možné manuálně pomocí skriptu `generate_images.py`. Postup je následující.

1. V adresáři se zdrojovými kódy aktivujte virtuální prostředí, vytvořené při spouštění aplikace.
2. Pomocí příkazu `python3 generate_images.py` spusťte generování obrázků. V základním stavu se vygenerují obrázky ze všech vizualizací ve formátu SVG. Pro bližší specifikování výstupu můžete použít následující přepínače:
   - `-n <jmeno-vizualizace>` pro určení jedné vizualizace pro vytvoření obrázku, jméno vizualizace je totožné s názvem souboru s vizualizací bez koncovky `.py`, tedy například `tangent_2d`,

- `-f <format>` pro určení formátu obrázků, podporované jsou například formáty SVG, JPG nebo PNG.

3. Vygenerované obrázky jsou uloženy do složky `images` v adresáři se zdrojovými kódy.

Je také možné využít automaticky generovaných obrázků, které jsou výstupem GitLab CI/CD ve formě artefaktů.

B.5 Spuštění testovacích ukázek

V rámci seznámení s vykreslovacími systémy v kapitole 1 jsem každý systém otestovala vytvořením jednoduchého příkladu. Zdrojové kódy těchto testů jsou součástí přiloženého média. Test každého systému má vlastní složku, která vedle zdrojových souborů a screenshotu výsledku obsahuje také `README.md` – soubor popisující obsah dané složky a obsahující návod ke spuštění vytvořené ukázky.

Dodatek C

Ukázka výsledné aplikace

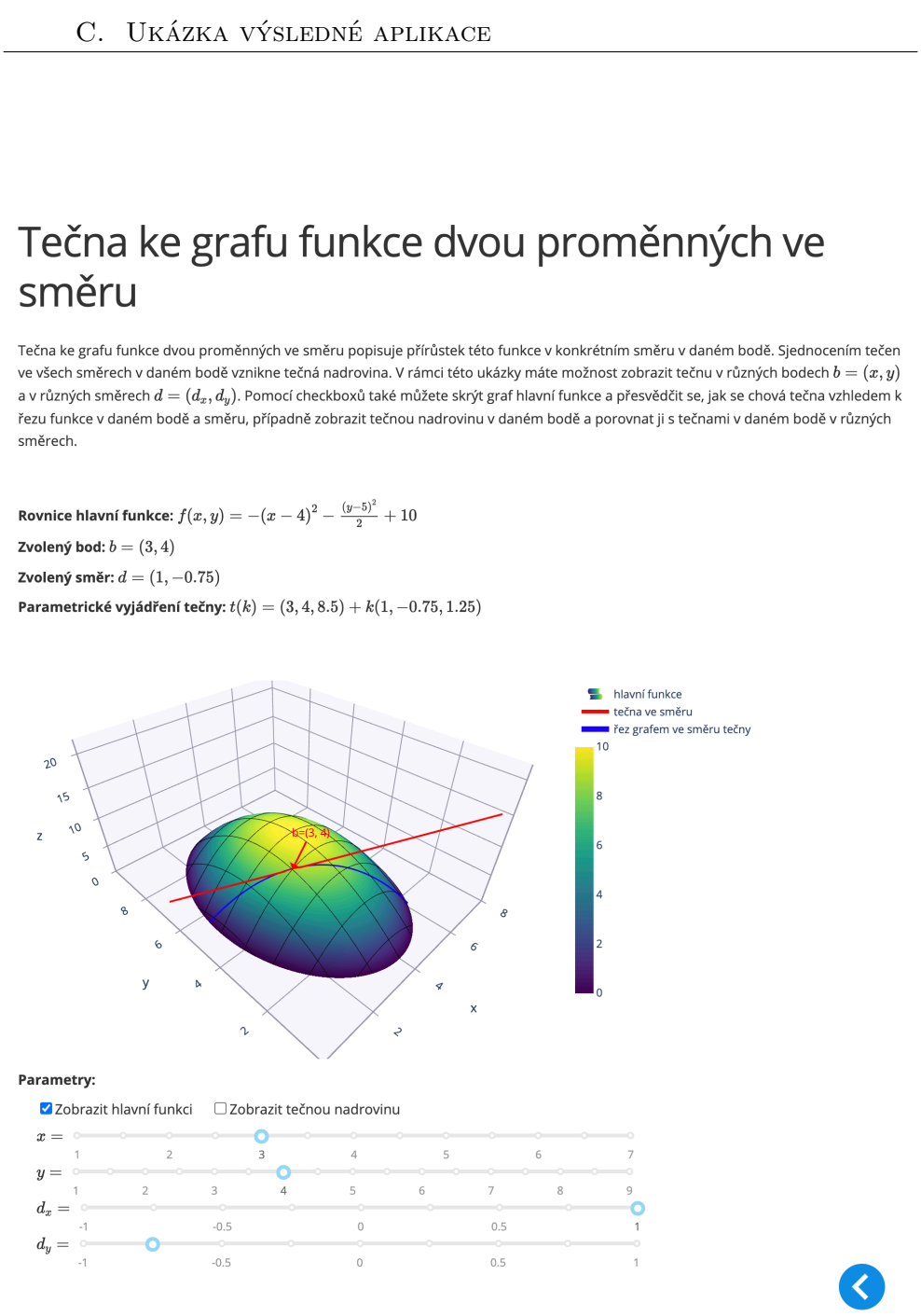
Interaktivní grafický výstup z výpočtů

Na této stránce naleznete interaktivní modelové ukázky vybrané z látky vyučované na Fakultě informačních technologií ČVUT v předmětech zaštítěných Katedrou aplikované matematiky.

Ukázky:

[ZMA] Konstrukce určitého integrálu funkce jedné proměnné
[MPI] Konstrukce určitého integrálu funkce dvou proměnných
[MPI] Lagrangeova metoda pro vázané extrémy
[ZMA] Tečna ke grafu funkce jedné proměnné
[MPI] Tečná nadrovina ke grafu funkce dvou proměnných
[MPI] Tečna ke grafu funkce dvou proměnných ve směru
[ZMA] Taylorův polynom

Obrázek C.1: Hlavní obrazovka s rozcestníkem.



Obrázek C.2: Webová stránka s vizualizací tečny ke grafu funkce dvou proměnných ve směru.

Konstrukce určitého integrálu funkce jedné proměnné

Geometrickou motivací pro výpočet určitého integrálu je výpočet obsahu plochy pod grafem funkce. Konstrukce určitého integrálu je založena na aproximaci této plochy pomocí součtu obsahů obdélníků nad částečnými intervaly. Jejich výška je dána infimem případně supremem funkčních hodnot dané funkce na daném částečném intervalu.

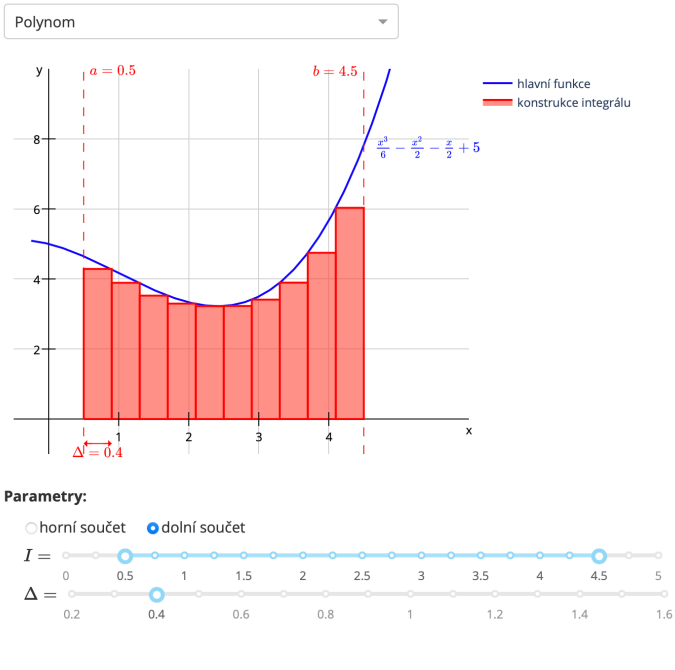
V této ukázce je možné si vyzkoušet, jak se hodnota součtu zpřesňuje se zmenšující se velikostí částečného intervalu Δ, a to jak pro dolní, tak pro horní součet. Také je možné zobrazit konstrukci pro různé intervaly $I = (a, b)$.

Zvolený interval: $I = (0.5, 4.5)$

Zvolená velikost částečného intervalu: $\Delta = 0.4$

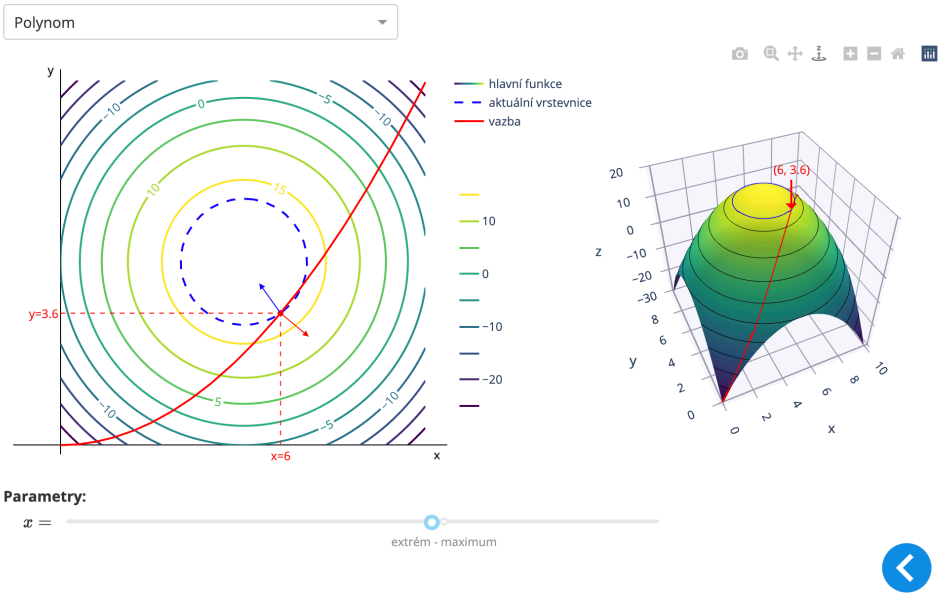
Hodnota integrálu: $\int_{0.5}^{4.5} f = 16.917$

Hodnota součtu ploch obdélníků: $\underline{\int_{0.5}^{4.5}} f = 15.813$



Parametry:

Obrázek C.3: Webová stránka s vizualizací konstrukce určitého integrálu funkce jedné proměnné.



Obrázek C.4: Webová stránka s vizualizací Lagrangeovy metody.

Dodatek D

Manuál pro tvorbu vizualizací

Tento manuál obsahuje několik obecných doporučení pro vývoj a stručný popis struktury aplikace. Jeho největší část je věnována podrobnému návodu pro vytvoření nové vizualizace. Na závěr je připojena sekce o testování vizualizací se scénářem pro manuální ověření funkčnosti vizualizace a sekce s řešením častých problémů.

D.1 Obecná doporučení pro psaní kódu

Aplikace je implementována v jazyce Python. Oficiální sada pokynů pro psaní kódu v Pythonu zvaná PEP 8 udává konvence, které je dobré dodržovat. V této aplikaci jsou striktně dodržovány zejména následující pokyny:

- odsazení kódu pomocí čtyř mezer, víceřádkový seznam argumentů funkce by měl být zarovnán s otevírací závorkou,
- obklopení definic tříd a funkcí dvěma prázdnými řádky,
- jednotné používání uvozovek pro textové řetězce (v této aplikaci jsou to dvojité – `"` – uvozovky),
- používání malých písmen a podtržítek pro pojmenovávání modulů a pouze malých písmen pro pojmenovávání balíčků,
- používání `CamelCase` stylu pro pojmenovávání tříd,
- používání `snake_case` stylu pro pojmenovávání funkcí a proměnných, případně velkých písmen pro pojmenovávání konstant.

Pro usnadnění dodržování těchto a dalších pravidel obsahuje základní instalace také knihovnu pycodestyle, která kontroluje dodržování konvencí podle PEP 8.

D.2 Obecná struktura aplikace

Navenek se aplikace skládá z úvodní stránky s rozcestníkem, stránky s hláškou o nenalezení požadovaného obsahu a dvou stránek pro každou vytvořenou vizualizaci. První z těchto stránek (tzv. samostatná stránka) je plnohodnotnou součástí webové aplikace a má fungovat samostatně při otevření v prohlížeči. Druhá stránka (tzv. integrovatelná) slouží pro zobrazení vizualizace v rámci elementu `iframe`. Stránky se navzájem liší obsahem, první obsahuje doprovodný kontext, druhá většinou pouze Plotly model a widgety.

Díky automatické správě obsahu není nutné vytvářet stránky manuálně – každá vizualizace je definována v jediném souboru s definovanou strukturou, která bude popsána níže. Obě stránky jsou následně vygenerovány automaticky.

Celkově je kód složen z hotového jádra, které zahrnuje například úvodní stránku s rozcestníkem, dynamickou správu obsahu a pomocné funkce. K tomuto jádru se přidávají jednotlivé vizualizace způsobem popsaným v dalších částech tohoto manuálu.

Adresář se zdrojovými kódy aplikace je strukturován následovně:

- `assets` – složka pro CSS a JavaScriptové podpůrné soubory
- `exceptions` – balíček pro deklaraci vlastních výjimek
- `smoketests` – balíček s tzv. smoke testy ověřujícími spustitelnost
- `tests` – balíček s unit testy ověřujícími funkčnost pomocných funkcí
- `utils` – balíček pro deklaraci pomocných funkcí
- `visualizations` – balíček s jednotlivými vizualizacemi
  - `__template__.py` – šablona použitelná pro vytvoření vizualizace
  - `integral_2d.py` – vizualizace definice určitého integrálu
  - `tangent_2d.py` – vizualizace tečny ke grafu funkce
  - …
- `app_setup.py` – vytvoření a nastavení aplikace
- `app.py` – definice obecného rozložení a spuštění aplikace
- `index.py` – definice rozložení hlavní stránky s rozcestníkem
- `requirements.txt` – seznam knihoven potřebných pro vývoj

Pro spuštění a nastavení aplikace jsou nejdůležitější soubory `app_setup.py` a `app.py`. První z nich slouží k vytvoření aplikace a je používán i pro její import do jednotlivých vizualizací. Ve druhém je mimo jiné definováno obecné rozložení aplikace a routing (párování mezi URL a zobrazenou vizualizací) a slouží ke spouštění aplikace.

Pro doplňování aplikace o nové vizualizace jsou nejdůležitější tyto složky: `visualizations`, která obsahuje jednotlivé vizualizace, `utils`, ve které jsou definované pomocné funkce, a `exceptions`, která sdružuje nově definované výjimky.

D.3 Postup vytvoření nové vizualizace

Pro rozšiřování stávající aplikace doporučuji vyvíjet lokálně v aplikaci instalované a spuštěné pomocí instalační příručky v sekci B.2. Vývoj skrz GitLab rozhraní a CI/CD nedoporučuji zejména kvůli pomalé odezvě a dlouhému čekání na zobrazení výsledku.

Pro vývoj je potom potřeba mít lokálně naklonovaný GitLab repozitář se zdrojovými kódy. Dále pro implementaci nové vizualizace (ale také pro úpravy a opravy aplikace) doporučuji vyvíjet v samostatné větvi.

D.3.1 Obecný postup vytvoření nové vizualizace

Po naklonování repozitáře, instalaci a spuštění je možné přidávat do aplikace nové vizualizace. Obecný postup je následující.

1. Pro novou vizualizaci vymyslete krátké unikátní `id` zapsané pomocí malých písmen s pomlčkami – například `tangent-2d` pro tečnu ke grafu funkce dvou proměnných

2. Zkopírujte soubor `__template__.py` ve složce `visualizations` na nový soubor v téže složce a přejmenujte jej, jako nový název použijte zmíněné `id`, kde nahradíte pomlčky podtržítky – např. `tangent_2d.py`

3. Implementujte novou vizualizaci. Během implementace můžete sledovat průběžný výsledek na adrese `http://127.0.0.1:8050/app/<id>` pro samostatnou stránku a `http://127.0.0.1:8050/model/<id>` pro integrovatelnou stránku.

   a) Vytvořte funkci `create_figure`. Tato funkce může mít libovolný počet argumentů, které jsou parametry vizualizace a ovlivňují její vzhled – například bod z definičního oboru, ve kterém má být vykreslena tečna. Návratovou hodnotou této funkce musí být nastavený objekt `Figure`. Podrobněji se touto funkcí budu zabývat v další části manuálu.

   b) Vyplňte atribut `title`. Tento titulek bude zobrazen v rozcestníku a také jako text na záložce prohlížeče. Dodržujte přitom prosím konvenci existujících titulků – jako první uveďte kód předmětu, ke kterému se vizualizace vztahuje, v hranatých závorkách, poté název vizualizace – například „[ZMA] Tečna ke grafu funkce jedné proměnné“.

   c) Vyplňte atribut `default_args`. Jedná se o slovník – tedy seznam dvojic klíč-hodnota – kde klíč je textový řetězec odpovídající jménu argumentu funkce `create_figure` a hodnota je jeho výchozí hodnota, se kterou bude vizualizace vykreslena ve výchozím stavu. Je nutné ve slovníku uvést všechny argumenty funkce `create_figure`.

d) Vyplňte atributy `headline` s nadpisem stránky a `annotation` s popiskem ukázky. Popisek může být buď jeden textový řetězec nebo pole několika textových řetězců, které se zobrazí jako samostatné odstavce. Měl by obsahovat shrnutí tématu, kterým se vizualizace zabývá, a stručný popis jejích interaktivních možností. Nadpis a popisek vizualizace jsou viditelné pouze na samostatné stránce.

e) Vyplňte atribut `layout` udávající rozložení centrální části vizualizace – tj. části s Plotly grafem a widgety. Tato část je je jedinou částí zobrazenou na integrovatelné stránce. Musí obsahovat pouze nezbytné komponenty. Měla by se bez scrollování vejít do šířky 800 px a výšky nejlépe 650, maximálně 700 px. Pro vytváření identifikátorů – `id` – jednotlivých elementů prosím dodržujte konvenci `<id-vizualizace>__<jmeno-elementu>` – identifikátory elementů musí být unikátní napříč celou aplikací, pomocí prefixu s identifikátorem vizualizace to je možné jednoduše zajistit. Definici rozložení se budu dále podrobněji věnovat v jiné části této sekce.

f) Dále je možné vyplnit atributy `before_layout` a `after_layout` (volitelně) obsahující doprovodné komponenty zobrazené na samostatné stránce buď před částí `layout`, nebo za ní. Pro identifikátory elementů prosím dodržujte stejná pravidla jako při definici atributu `layout`.

g) Definujte callbacky pro aktualizaci parametrů vizualizace na základě uživatelského vstupu. Postupu vytvoření callbacku se budu věnovat níže v této sekci.

h) Na počátek souboru s implementací doplňte dokumentační řetězec (může být v češtině) se jménem vizualizace a s omezeními kladenými na vykreslované funkce (např. spojitá, diferencovatelná).

4. Hotovou vizualizaci manuálně otestujte – nejlépe podle scénáře pro manuální testování definovaného níže. Zkontrolujte obě stránky s hotovou vizualizací ve více různých prohlížečích a vyzkoušejte je zadáváním různých hodnot parametrů.

5. Po otestování nahrajte změny do GitLab repozitáře a vytvořte pro vývojovou větev `merge request`.

6. Po schválení `merge requestu` a sloučení stávající větve s `master` větví se výsledek automaticky nasadí na server a bude veřejně dostupný.

D.3.2 Matematické výpočty ve vizualizacích

Pro komplexní matematické operace a parsování textových řetězců s matematickými výrazy slouží v implementaci knihovna SymPy.

Základní operace Základní používané operace této knihovny jsou:

- `symbols` – definice proměnných pro další výpočty,
- `sympify` – převod textového řetězce do interní SymPy reprezentace,
- `lambdify` – převod z interní reprezentace na lambda funkci, která umožní rychlejší vyhodnocení výsledku při dosazování mnoha různých hodnot,
- `subs` – dosazení výrazu nebo číselné hodnoty místo proměnné.

Pomocí těchto operací je například možné z textového řetězce se vzorcem funkce (např. `x**2 - 2`) získat lambda funkci, která přijímá parametr `x` a vrací hodnotu výrazu po dosazení tohoto parametru. Tuto operaci v aplikaci zastává funkce `get_function_formula`, jejíž kód a použití jsou pro ilustraci na následujícím výpisu.

```
1  def get_function_formula(function_string):
2      x = sp.symbols("x")
3      func_expr = sp.sympify(function_string)
4      return sp.lambdify(x, func_expr, "numpy"), func_expr
5
6  f, f_expr = get_function_formula("x**2 - 2")
7  res = f(5)        # res = 23
```

Specializované operace Knihovna SymPy dále nabízí celou řadu operací například pro výpočet derivací (`diff`), gradientu (`derive_by_array`), či integrálu (`integrate`). Tyto operace jsou přehledně a detailně popsány v dokumentaci knihovny.

Výpis výrazů v latexu Pomocí SymPy funkce `latex(expr)` je možné převést SymPy výraz do textového řetězce zapsaného v LaTeX. Pro vypsání takového řetězce ve formátované podobě přímo v Plotly modelu je třeba jej obklopit znakem dolaru – `$ latex(expr) $`, pro vypsání v doprovodném textu v Dash komponentě je naopak nutné ho obklopit lomítky a závorkami – `\( latex(expr) \)`.

Pomocné funkce Aplikace obsahuje v modulu `utils.math_utils` několik definovaných funkcí, které zjednodušují často prováděné operace. Patří sem například:

- `get_function_formula` pro získání lambda funkce z matematického výrazu o jedné proměnné (proměnná musí být x),

- `get_3D_function_formula` pro získání lambda funkce z matematického výrazu o dvou proměnných (vždy x a y),
- `is_expression_invalid` pro testování validity výrazu,
- `get_gradient_in_point` pro výpočet gradientu výrazu o dvou proměnných (x a y) v daném bodě,
- `normalize_vector` pro normalizaci číselného vektoru v bodě.

Všechny funkce obsahují dokumentační textové řetězce, které blíže vysvětlují jejich účel a použití.

D.3.3 Implementace funkce pro vytvoření modelu

Nyní se budu podrobněji věnovat funkci `create_figure` pro vytvoření modelu. Jak jsem zmínila již dříve, tato funkce může mít libovolné množství argumentů. Jednotlivé argumenty pak reprezentují parametry vizualizace a ovlivňují její vzhled a nastavení. Při implementaci se využívá knihovna Plotly pro Python, dodatečné informace lze najít v její dokumentaci.

Předávání argumentů do metod z knihovny Plotly Nejprve je důležité zmínit systém pro předávání argumentů do metod knihovny Plotly. Celý model je na pozadí reprezentován pomocí sytému slovníků (dvojic klíč-hodnota), které mohou být vzájemně zanořené. Například datová řada – `trace` – má atribut `line`, jehož hodnota je slovník s atributem `color`, který určuje barvu této datové řady. Při vytváření datové řady mohu předat konstruktoru `Scatter()` pojmenovaný (keyword) argument `line` jehož hodnotou bude daný slovník: `Scatter(line={"color": "blue"})`. Je ale také možné využít podtržítkovou notaci a každý „skok“ ve slovnících o úroveň níž nahradit právě podtržítkem. Pak bude totožný příkaz zapsaný takto: `Scatter(line_color="blue")`. Podtržítková notace se vyplatí zejména pokud je nastavováno pouze několik málo argumentů ze zanořeného slovníku.

Vytvoření modelu a jeho nastavení V první části funkce `create_figure` by měl být vytvořen model – Plotly objekt `Figure` – a nastaveny jeho základní vlastnosti, jako jsou rozměry, barva pozadí a vlastnosti souřadnicových os.

Nový model je možné vytvořit pomocí `plotly.graph_objects.Figure()`. Jeho vlastnosti jako je barva pozadí, velikost okrajů a podobně nastavuje jeho metoda `update_layout`. Konkrétní přehled všech vlastností, které je pomocí této metody možné nastavit, je možné dohledat v dokumentaci.

Pro nastavení souřadnicových os je možné použít funkce z balíčku `utils`, konkrétně pro 2D model `utils.axes_utils.setup_axes` a pro 3D model `utils.axes_3d_utils.setup_3d_axes`. Pomocí těchto funkcí je možné nastavit mimo jiné rozsahy souřadnicových os a hodnoty vyznačené na osách.

Jejich rozhraní je detailně popsáno pomocí dokumentačních textových řetězců v kódu.

Získávání dat pro vykreslení funkcí V knihovně Plotly je možné funkci $f(x)$ vykreslit pomocí datové řady (tzv. `trace`). Datová řada pro 2D model očekává dvojici polí `x` a `y`, kde položky `x[i]` a `y[i]` reprezentují souřadnice nějakého bodu na grafu funkce f a platí, že $y[i] = f(x[i])$. Propojením těchto bodů je možné graficky znázornit graf funkce.

Pole `x` je možné získat příkazem `x = np.arange(START, END, STEP)` pomocí NumPy, kde `START` a `END` určují rozsah vzorkování a `STEP` jeho jemnost. Právě dostatečně velká jemnost (tedy dostatečně malý krok) vzorkování zajistí vykreslení hladkého průběhu funkce – doporučuji `STEP` o velikost 0,02–0,05.

Pro získání pole `y` je nejprve potřeba získat programovou reprezentaci vizualizované funkce f, která dokáže pro daný bod z definičního oboru x vrátit odpovídající hodnotu $y = f(x)$. Tuto reprezentaci lze získat pomocí funkce `utils.math_utils.get_function_formula`, která dokáže převést textový řetězec se vzorcem funkce (např. `x**2-2`), na lambda funkci s požadovanými vlastnostmi. Kromě lambda funkce vrací zmíněná funkce také SymPy výraz, který je možné použít například pro vypsání matematického vzorce funkce pomocí LaTeX. Je také možné napsat vlastní pomocnou funkci pro převod v podobném tvaru, která bude vracet například rovnici tečny ke grafu funkce v bodě – doporučuji inspirovat se již hotovými funkcemi v kódu.

Pomocí mapovací funkce je možné získat pole hodnot `y`. Lze využít například tzv. „list comprehension“, která umožní vytvořit jeden nový seznam na základě iterace přes nějaký již existující. Příkaz pro vygenerování hodnot pole `y` tedy může vypadat následovně: `y = np.array([f(x_i) for x_i in x])`, získaný seznam je poté nutné převést na NumPy pole.

Datová řada pro 3D model vyžaduje tři pole `x`, `y` a `z`, která mají obdobný význam a také se obdobně vytvoří. Lze vygenerovat pole hodnot `x` a `y` pomocí funkce `np.arange` a dopočítat odpovídající hodnoty v poli `z`. Pro převod textového řetězce se vzorcem funkce dvou proměnných na lambda funkci existuje funkce `utils.math_utils.get_3D_function_formula`. Takto potom `z = np.array([[f(x_i, y_i) for x_i in x] for y_i in y])` může vypadat vytvoření pole `z`.

Vykreslování funkcí Matematické funkce se do modelů vykreslují pomocí datových řad. Mezi základní typy datových řad patří `Scatter` pro 2D bodové a spojnicové grafy, `Surface` pro povrchové grafy a `Scatter3d` pro 3D bodové a spojnicové grafy.

Datová řada se přidává do modelu pomocí metody `figure.add_trace`. Jako argument dostává tato metoda objekt libovolného typu datové řady: `figure.add_trace(plotly.graph_objects.Scatter(...))`.

Základními argumenty všech datových řad jsou `x`, `y` a případně `z` pro

předání souřadnic bodů, jak jsem popsala výše. Dalšími důležitými argumenty jsou:

- **name** – jméno datové řady, zobrazí se v legendě,
- **mode** – u datových řad `Scatter` a `Scatter3d` umožňuje výběr mezi spojnicovým (hodnota `"lines"`) a bodovým (hodnota `"markers"`) grafem,
- **line_color, line_width, line_dash** – určují barvu (standardizovaný název nebo HEX kód barvy – např. `"blue"`), tloušťku (číslo – např. `1`) a čárkování (délka čárek v pixelech – např. `"5px"`) linie,
- **showlegend** – booleovská hodnota určující, zda bude prvek zobrazen v legendě nebo nikoli,
- **hoverinfo** – pomocí hodnoty `"none"` lze vypnout tooltip zobrazovaný po najetí myší nad datovou řadu,
- **hovertemplate** – určuje vzhled tootlipu zobrazeného po najetí myší, pro detailní popis viz dokumentace Plotly,
- **opacity** – průhlednost na škále od 0 (zcela průhledné) do 1 (zcela neprůhledné), vhodné zejména u 3D modelů,
- **contours** – určuje vzhled vrstevnic povrchových grafů, jedná se o slovník s vícero atributy, viz dokumentace Plotly.

Kompletní seznam argumentů a jejich popis je dostupný v dokumentaci Plotly.

Přidávání textových anotací a dalších popisných prvků Přidávání textových anotací je možné pomocí metody `figure.add_annotation` pro 2D modely nebo pomocí funkce `utils.axes_3d_utils_add_annotation` pro 3D modely (Plotly bohužel jednoduchou metodu pro přidání anotace do 3D modelu neposkytuje).

Základními argumenty jsou `x` a `y` (případně `z`) určující polohu ukotvení anotace a `text` určující její znění. Další důležité atributy jsou:

- **xanchor, yanchor** – určují horizontální (`"left"`, `"center"`, `"right"`) a vertikální (`"top"`, `"middle"`, `"bottom"`) polohu textu anotace vzhledem k jejímu kotvícímu bodu,
- **xshift, yshift** – určuje posun anotace horizontálně nebo vertikálně o daný počet pixelů,
- **ax, ay, axref, ayref** – společně určují délku a směr šipky, viz dokumentace Plotly,

- **showarrow** – booleovská hodnota určující, zda bude zobrazena šipka nebo nikoli,
- **arrowcolor** – určuje bravu šipky,
- **font_color** – určuje barvu písma (standardizovaný název nebo HEX kód barvy – např. `"blue"`).

Anotace mohou obsahovat výrazy v LaTeX, takovýto výraz lze vyznačit pomocí symbolů `$`.

Další popisné prvky, například vodící čáry, označení bodů a různé tvary je možné tvořit pomocí datových řad (v takových případech je vhodné vypnout zobrazení v legendě a tooltipy na najetí myší). Dále existují metody `figure.add_shape` pro přidání tvarů, `figure.add_vline` a `figure.add_hline` pro přidání nekonečných čar a `figure.add_vrect`, `figure.add_hrect` pro přidávání nekonečných pruhů. Jejich atributy a vlastnosti jsou dobře popsány v dokumentaci Plotly.

Ošetřování chyb Může se stát, že pro nějakou konkrétní kombinaci parametrů nemají matematické výpočty smysl. Proto jsem implementovala výjimku `NoResultException`, která by tyto stavy měla reprezentovat. Je vhodné výjimku zachytit a do modelu vypsat místo problematického obsahu informaci o špatné kombinaci parametrů.

Parametrizace hlavní zobrazované funkce V několika existujících ukázkách (např. tečna ke grafu funkce jedné proměnné nebo Taylorův polynom) je možné volit z několika nabízených hlavních funkcí. Každá takováto funkce může mít např. vlastní rozsahy os, které mají být zobrazeny, a hodnoty na těchto osách. Je tedy potřeba parametrizovat mnohem více prvků, než samotný vzorec funkce.

Z těchto důvodů byla vytvořena dvojice tříd `utils.function.Function` a `utils.function_3d.Function3D`, jež slouží jako obálky na všechny parametry, které se mění spolu s modelovanou funkcí. Mají instanční proměnné (vzorec funkce, její jméno, zobrazený rozsah na souřadnicových osách, hodnoty zobrazené na souřadnicových osách aj.), které jsou pro parametrizaci potřeba prakticky vždy. Dokáží ale uchovávat také další hodnoty pomocí slovníku `other`. Při inicializaci přijímají povinné a nepovinné argumenty, ukládané do instančních proměnných a přístupné pomocí tečkové notace. Přijímají také ale libovolné další pojmenované argumenty, které jsou následně uloženy právě do slovníku `other` a přístupné jsou např. jako `function.other["argument"]`.

Každá parametrizovaná ukázka obsahuje konstantu s polem tvořeným objekty `Function` (případně `Function3D`) a pomocí komponenty `Dropdown` umožňuje vybrat konkrétní index v tomto poli. Objekt `Function` na daném indexu je poté použit pro parametrizaci.

Plně parametrizovaná je například ukázka tečny ke grafu funkce jedné proměnné – zde není pevně stanovený zobrazený rozsah na ose x, protože jsou parametrizované hodnoty zobrazované na widgetech `Slider`. Naopak např. ukázka Taylorova polynomu je parametrizována pouze částečně, protože neupravuje hodnoty na widgetech `Slider` a požaduje proto, aby všechny funkce byly zobrazeny v daném rozsahu na ose x – aby widgety `Slider` dávaly smysl. Pro implementaci parametrizace hlavní zobrazované funkce doporučuji prostudovat zdrojové kódy k těmto dvěma ukázkám.

D.3.4 Definice rozložení webové stránky

V této části se zaměřím na definici rozložení webové stránky s vizualizací pomocí frameworku Dash. Tato definice může být uplatněna při definici atributů `layout`, `before_layout` a `after_layout`.

Obecně je webová stránka v tomto frameworku tvořena pomocí komponent dvou typů – `html` komponent z modulu `dash_html_components` a `core` komponent z modulu `dash_core_components`. Všechny komponenty mají atributy definované pomocí pojmenovaných argumentů (paří sem například `id` nebo `className`) a mohou mít potomky (textový řetězec, číslo nebo pole vnořených komponent) předávané pomocí argumentu `children`.

Následující úsek kódu ukazuje příklad aplikace `app`, jejíž atribut `layout` určující rozložení této aplikace má hodnotu tvořenou `html` komponentou `Div` s definovaným identifikátorem `id` a seznamem potomků `children`. Těmito potomky jsou `html` komponenta `H1` pro nadpis, a dvě `core` komponenty pro model (`Graph`) a widget (`Slider`).

```
1  app.layout = html.Div(
2      id="page-id",
3      children=[
4          html.H1(className="headline", children="Page title"),
5          dcc.Graph(id="graph", figure=create_figure(2)),
6          dcc.Slider(id="point-slider", min=1, max=5, value=2, step=0.5)])
```

Je důležité, aby byla u komponent dodržována konvence pro tvoření hodnot atributů `id` a `className`. Tato hodnota by vždy měla vypadat obecně jako `<id-vizualizace>__<pojmenovani-komponenty>`, kde „id-vizualizace“ je `id` zmiňované v podsekci D.3.1. Příklad validní hodnoty je „tangent-2d___point--slider“. Tato konvence je důležitá jak z hlediska zachování jednotnosti napříč vizualizacemi, tak pro zachování unikátnosti identifikátorů a tříd – aby se mezi sebou vizualizace neovlivňovaly.

Důležité html komponenty V modulu `dash_html_components` se nacházejí komponenty, které odpovídají běžným HTML elementům, jako jsou například `div`, `p` nebo `h1`, s tím rozdílem, že se píší s prvním velkým počátečním

písmenem – tedy `Div`, `P` a `H1`. Tyto komponenty mají mimo jiné atributy `id` a `className` pro identifikaci a nastavení CSS třídy, `style` jako slovník pro nastavení vzhledu elementu (klíče ve slovníku odpovídají CSS vlastnostem zapsaným pomocí `camelCase` konvence, tedy např. `marginTop`) a `children` pro nastavení potomků. Plný seznam komponent a jejich atributů může být dohledán v dokumentaci frameworku Dash, nejčastěji používanými komponentami jsou:

- `Div` pro obecný blok,
- `P` pro text,
- `H1` až `H6` pro nadpisy,
- `A` pro odkazy (doporučuji použití pouze pro odkazy mimo aplikaci, pro odkazy uvnitř aplikace je vhodnější `core` komponenta `Link`).

Důležité core komponenty Modul `dash_core_components` pak obsahuje složitější hotové komponenty, které výrazně ulehčují práci s vytvářením webové stránky. Patří sem zejména widgety pro zpracování uživatelských vstupů, komponenta pro zobrazení modelu a komponenta pro odkazy v rámci aplikace. Stejně jako `html` komponenty mají i `core` komponenty atributy `id`, `className`, `style` a `children`, kromě nich ale obvykle mají i řadu jiných atributů popisujících jejich chování. Plný seznam komponent a jejich atributů může být dohledán v dokumentaci frameworku Dash, v následujícím výčtu zmíním pouze nejčastěji používané komponenty a jejich důležité atributy.

- `Graph` – zobrazení modelu vytvořeného pomocí knihovny Plotly.
  - `figure` – definice modelu k zobrazení, může mu být předán např. Plotly objekt `Figure`, který je výsledkem funkce `create_figure`.
  - `responsive` – boolean udávající, zda má být graf responzivní (tedy zmenšující se podle velikosti stránky), či nikoli. Doporučuji uvádět hodnotu `True` a dále v argumentu `style` definovat výšku a maximální šířku vizualizace.
- `Link` – navigace mezi stránkami v rámci aplikace.
  - `href` – URL adresa stránky, na kterou má proběhnout přesměrování.
  - `children` – například text zobrazený v odkazu.
- `Loading` – zobrazení indikátoru načítání, má specifický způsob použití – viz dokumentace nebo hotové ukázky.

Nejčastěji používanými widgety pak jsou následující komponenty (pro výčet jejich atributů a příklady použití viz dokumentace):

- `Slider` – posuvník pro výběr hodnoty z rozsahu,
- `RangeSlider` – posuvník pro výběr dvojice hodnot, tj. krajních bodů intervalu,
- `Checklist` – sada zaškrtávacích políček,
- `RadioItems` – sada přepínačů pro výběr jedné možnosti z nabídky,
- `Dropdown` – rozbalovací nabídka pro výběr jedné z více hodnot.

D.3.5 Definice callbacků pro aktualizaci vizualizace

Zpracování uživatelského vstupu a aktualizaci vizualizace na základě nových hodnot je možné provádět v tzv. callbacích. Callbackem je jakákoli funkce anotovaná pomocí `@app.callback`. V rámci této anotace je možné definovat vstupy aktualizace (třída `Input`) a její výstupy (třída `Output`), těmito vstupy i výstupy jsou vždy konkrétní atributy konkrétních komponent. Na obecné úrovni aktualizační funkce přijímá jako argumenty hodnoty vstupů a jako návratové hodnoty vrací hodnoty výstupů.

Pro jednoduchou aktualizaci vizualizace na základě jednoho parametru je možné použít následující callback:

```
1  @app.callback(
2      Output("graph", "figure"),
3      Input("point-slider", "value"))
4  def update_figure(point_value):
5      return create_figure(point_value)
```

Jako vstup tato aktualizační funkce obdrží hodnotu atributu `value` komponenty s identifikátorem `point-slider`. Vytvoří nový model (pomocí funkce `create_figure`) s aktualizovaným parametrem a vrátí objekt `Figure`, který bude předán do `figure` atributu komponenty s identifikátorem `graph`.

Mohou samozřejmě existovat i složitější callbacky s vícero vstupy a výstupy, jako například:

```
1  @app.callback([
2      Output("graph", "figure"),
3      Output("loading-output", "children")],
4      Input("a-slider", "value"),
5      Input("n-slider", "value"))
6  def update_figure(a, n):
7      return create_figure(a, n), True
```

Výstupy je třeba zapsat v rámci jednoho pole a návratovou hodnotou pak musí být n-tice (`tuple`) s odpovídajícím počtem prvků. Vstupy nemusí být součástí pole, předají se jako argumenty aktualizační funkci ve stejném pořadí, v jakém jsou v anotaci zapsány za sebou.

D.3.6 Nastavení vzhledu komponent

Pro nastavení vzhledu Dash komponent je možné použít několik metod. Styl lze například zapsat přímo do atributu `style` dané komponenty. Tento přístup ale doporučuji pouze pokud je styl unikátní a neopakuje se u vícero komponent – pro obarvení textu jednoho elementu na červenou barvu je to vhodný přístup, ale pro obarvení všech nadpisů už nikoli.

Pro opakující se komponenty jsem připravila několik globálních CSS tříd, které je možné na tyto elementy aplikovat. Plný seznam těchto tříd je možné zjistit ze souboru `assets/styles.css`, nejdůležitějšími třídami jsou:

- `headline` pro nadpisy stránek,
- `annotation` pro texty v úvodu stránek,
- `label` pro štítky,
- `widget`, `widget-container`, `widget-label` a `function-select` pro widgety.

Všechny opakující se třídy jsou součástí předdefinovaného rozložení stránky v šabloně `__template__.py` pro vytváření nových vizualizací.

V případě nutnosti definice nové globální třídy používané napříč aplikací je vhodné její definici vložit právě do souboru `assets/styles.css`. Pokud bude třída specifická pouze pro jednu vizualizaci, doporučuji založení nového souboru ve složce `assets` pojmenovaného `<id-vizualizace>.css` a používání pojmenovávací konvence zmíněné v podsekci D.3.4.

D.4 Testování

Automatizované testy je potřeba doplňovat pouze pro funkce z balíčku `utils`. Vytvořené vizualizace dostačuje testovat manuálně.

Při testování doporučuji dodržovat následující scénář:

1. Otevřete aplikaci na hlavní stránce a ověřte, že je vizualizace součástí seznamu ukázek. Zkontrolujte zobrazený text.
2. Klikněte na odkaz na vizualizaci, budete přesměrováni na stránku s vizualizací (konkrétně na samostatnou stránku). Zkontrolujte titulek na záložce prohlížeče.

3. Zkontrolujte obsah stránky:
   - nadpis a anotace jsou zobrazeny a mají správný text,
   - pokud je součástí anotace a doprovodných textů LaTeX, je zobrazen správně,
   - model v Plotly obsahuje všechny vytvořené prvky a jejich nastavení odpovídá požadavkům,
   - funkce a jiné matematické objekty, jejichž poloha a vzhled je dána výpočty, se v modelu zobrazují na očekávaných pozicích a mají očekávaný tvar či průběh,
   - widgety jsou zobrazeny, obsahují správné možnosti, jsou správně nastaveny,
   - jakékoli dodatečně prvky se zobrazují a chovají dle očekávání.
4. Zkontrolujte interaktivitu stránky:
   - s modelem je možné interagovat pomocí vestavěných nástrojů – například je možné jej přiblížit nebo posunout,
   - interakce s widgety způsobí přepočítání modelu, přepočítaný model obsahuje všechny požadované prvky a vypadá požadovaným způsobem (např. funkce mají správný průběh, jsou zvýrazněny správné body),
   - je možné interagovat s různými widgety postupně, hodnoty nastavené v předchozích krocích jsou zachovány, dokud nejsou manuálně změněny,
   - během přepočítávání modelu je zobrazena indikace načítání,
   - jakékoli další přidané prvky, které mají reagovat na změny parametrů, se chovají dle očekávání.
5. Zkontrolujte, že je stránku možné zobrazit jak na obrazovkách o šířce 600 px, tak na obrazovkách širších než 1 000 px. Zobrazení na úzkých obrazovkách nemusí být úplně přesné, důležité je, aby byl obsah stránky čitelný.
6. Zkontrolujte, že se v dolní části stránky nachází tlačítko pro návrat na domovskou stránku. Zkontrolujte, že jste na ni po kliknutí skutečně přesměrováni.
7. Dále zkontrolujte chování integrovatelné stránky s vizualizací – na adrese `http://127.0.0.1:8050/model/<id-vizualizace>` zkontrolujte obsah zobrazené stránky:
   - stránka obsahuje Plotly model, widgety a všechny ostatní komponenty definované jako součást atributu `layout`,

- model v Plotly obsahuje všechny vytvořené prvky a jejich nastavení odpovídá požadavkům,
- funkce a jiné matematické objekty, jejichž poloha a vzhled je dána výpočty, se v modelu zobrazují na očekávaných pozicích a mají očekávaný tvar či průběh,
- widgety jsou zobrazeny, obsahují správné možnosti, jsou správně nastaveny,
- jakékoli dodatečně prvky se zobrazují a chovají dle očekávání.

8. Ověřte interaktivitu integrovatelné stránky:
   - s modelem je možné interagovat pomocí vestavěných nástrojů – například je možné jej přiblížit nebo posunout,
   - interakce s widgety způsobí přepočítání modelu, přepočítaný model obsahuje všechny požadované prvky a vypadá požadovaným způsobem (např. funkce mají správný průběh, jsou zvýrazněny správné body),
   - je možné interagovat s různými widgety postupně, hodnoty nastavené v předchozích krocích jsou zachovány, dokud nejsou manuálně změněny,
   - během přepočítávání modelu je zobrazena indikace načítání,
   - jakékoli další přidané prvky, které mají reagovat na změny parametrů, se chovají dle očekávání.
9. Zkontrolujte, že je stránka zobrazitelná bez scrollování na obrazovkách o šířce maximálně 800 px a výšce nejlépe 650 px, maximálně však 700 px.
10. Uvedené kroky zopakujte pokud možno v několika různých prohlížečích a pokud je možnost, i na různých operačních systémech.

Při testování v prohlížeči Firefox může dojít k tomu, že se budou špatně zobrazovat některé anotace. Jedná se o známý problém knihovny Plotly, kterému se více věnuji v následující sekci.

D.5 Řešení problémů

V této sekci bych ráda zmínila několik častých problémů, se kterými se můžete během vytváření vizualizace setkat.

Chyba typu „not JSON serializable“ Tato chyba obvykle vzniká při předávání výsledků získaných v knihovně SymPy do `Figure` z knihovny Plotly. Výsledky totiž nejsou klasické číselné hodnoty z Pythonu (`float` nebo `int`), ale jsou interní třídou SymPy, která není serializovatelná. Je proto nutné je před předáním do Plotly konvertovat na obyčejná čísla například pomocí `float(x)`.

Chyby vznikající použitím špatných argumentů v Plotly metodách Tyto chyby způsobí pád a ukončení celé aplikace. Vrací ale přesnou indikaci chybné hodnoty včetně manuálu se seznamem možných hodnot.

Chyby při testování responzivity na malých obrazovkách Při zmenšování stránky s vizualizací může při nedůkladně implementované responzivitě docházet k tomu, že levá část stránky se postupně schovává za okraj prohlížeče a není zobrazitelná. To je způsobeno kombinací knihovny Dash a faktu, že některý element na stránce má šířku větší, než je šířka obrazovky.

Pokud je tímto elementem model, doporučuji nastavit u Dash komponenty `Graph` s modelem atribut `responsive` na hodnotu `True` a do atributu `style` definovat výšku modelu a její maximální šířku – viz hotové příklady.

Pokud se jedná o jiný element, doporučuji použít CSS media queries, které zajistí, že na menších obrazovkách bude šířka například 100% – viz hotový CSS stylesheet `styles.css`.

Nevykreslení anotací ve Firefoxu V prohlížeči Firefox se kvůli známé chybě špatně vykreslují anotace využívající sazbu pomocí LaTeX. Pokud je anotace kritická a musí být zobrazitelná v každém prohlížeči, doporučuji vyzkoušet různá nastavení atributů `xanchor` a `yanchor` této anotace.

Dodatek E

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `trials` ..... zdrojové kódy testů vykreslovacích systémů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF