České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídící techniky

Bakalářská práce

Interaktivní učebnice teorie řízení v programu Mathematica

Petr Martinec

Vedoucí práce: Ing. František Vacek

Studijní program: Elektrotechnika a informatika, strukturovaný, Bakalářský

Obor: Kybernetika a měření

8. ledna 2010

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 8. 1. 2010

podpis

Poděkování

Děkuji především vedoucímu bakalářské práce za čas, který mi věnoval. Jeho obětavost a podnětné nápady zasluhují uznání.

Abstrakt

Tato práce si klade za cíl vytvoření frameworku pro tvorbu technicky zaměřených učebnic v softwaru Mathematica. Důraz je kladen na nastavení uživatelského rozhraní tak, aby bylo intuitivně použitelné i pro uživatele bez zkušeností se softwarem Mathematica a zároveň aby co nejméně omezovalo toho zkušeného. Zabývá se převodem elektronického dokumentu do tištěné podoby se zachováním vzhledu a vypuštěním syntaxe. Navrhuje systém, jak zapisovat, zpracovávat a používat výrazy, aby byly řetězitelné v rámci notebooku i mezi nimi.

Abstract

The aim of this work is a creation of a framework for technical oriented textbooks in Mathematica software. It emphasizes on the GUI creation with a good accessibility for non-skilled Mathematica users. Nevertheless, it is suitable for skilled ones as well. The work deals with the electronic document transformation into a printable form. It designs the expressions form suitable for Wolfram Mathematica manipulations acroding to text book needs.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Petr Martinec**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Interaktivni učebnice teorie řízeni v programu Mathematica**

Pokyny pro vypracování:

Seznamte se se software Mathematica 7.0.
Navrhněte kostru dokumentu pro interaktivní učebnici.
Zpracujte několik vybraných kapitol z teorie řízení v navrženém dokumentu.

Seznam odborné literatury:

www.wolfram.com

Vedoucí: Ing. František Vacek

Platnost zadání: do konce letního semestru 2009/10

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 25. 5. 2009

Obsah

Seznam obrázků **viii**

Seznam tabulek **ix**

1 Úvod **1**

2 Wolfram Mathematica **3**

2.1 Historie 3

2.2 Struktura 4

2.3 Programovací styly 5

2.4 Novinky ve verzi 6 a 7 6

2.5 Mathematica a eBook 7

3 Realizace **9**

3.1 Základní problémy 9

3.2 Styly 11

3.3 Výpočty 15

3.4 Modely 19

3.5 Kapitoly a jmenné prostory 21

4 Popis složitějších konstrukcí **23**

4.1 Přehled programovacích stylů 23

4.2 Příklady 25

5 Jak používat navržený framework **29**

5.1 Pravidla, která je třeba dodržet 29

5.2 Doporučení 30

6 Závěr **31**

A Obsah přiloženého CD **I**

B Dostupnost frameworku **II**

Seznam obrázků

2.1 Schema kernel, frontend a notebook 5

3.1 Skrývání syntaxe a zobrazení výsledku výpočtu v TraditionalForm 13

Seznam tabulek

Kapitola 1

Úvod

Výuka v odborných technických předmětech je náročným úkolem. Po studentech je požadována kvalitní znalost matematiky, která je mnohdy vyučována paralelně s technickým předmětem. Studentovi často činí potíže prohlédnout spojení technického problému s matematikou na pozadí. Teorii předkládanou na přednáškách může student rozvíjet čtením skript a aplikovat na praktické problémy při cvičeních. V dnešní době se do tohoto řetězce zapojuje i samostudium pomocí internetu. Problémem technických oborů je jejich úzká specifikace. Najít základní informace srozumitelné pro studenta začínajícího v oboru může být složité.

Celý výše popsaný proces vzdělávání studenta může být nesouvislý a propojení některých kroků může činit potíže. Například cvičení mohou kvůli časovému tlaku předbíhat přednášky nebo se během let upraví osnova předmětu a v doporučeném skriptu chybí informace. Stává se, že skriptum pro daný předmět ani neexistuje a studenti používají k učení slajdy z přednášek.

Z těchto důvodů jsem se rozhodl prozkoumat možnost vývoje interaktivní učebnice, která by umožnila spojit celý vzdělávací proces do jedné linie. Z této učebnice by byl tištěný výstup podobný klasickému skriptu. Měla by elektronickou podobu se silným výpočetním pozadím. Uživatelské rozhraní by umožňovalo předkládanou teorii podrobovat pokusům. Uživatel by mohl vstupovat do vzorců a experimentovat s nimi úpravou proměnných a konstant. V této učebnici by se také připravovaly modely a simulace reagující na výsledky odvozené v předchozí teoretické části. Navíc by se tato elektronická verze dala snadno zjednodušit a použít k doprovodu přednášek v podobě slajdů. Díky interaktivitě by tyto slajdy regovaly na dotazy studentů a zobrazovaly důsledky jejich návrhů.

Student by tak dostal vzdělávací pomůcku "vše v jednom". Ve volném čase by mohl snadno přemýšlet nad teorií a aplikovat své poznatky a domněnky na modelech. Učitel

by měl materiál, ze kterého získá přednáškové slajdy i podklady a praktické úkoly na cvičení.

Jako prostředek pro cestu k vytčenému cíli jsem si zvolil software Mathematica. V tomto software bych chtěl připravit framework, který by usnadnil vytvoření popsané učebnice. Mathematica je software pro symbolickou matematiku. Výsledky výpočtů tedy vrací v analytické formě a díky vysokému výpočetnímu výkonu se výborně hodí pro účely popisované učebnice. Navíc je dobře připraven na publikování prováděných výpočtů a jejich výsledků. Dále umožňuje práci s grafickými objekty, což je vhodné pro přípravu modelů.

Použití Mathematicy jako nástroje s sebou přináší i jisté nevýhody. Autor učebnice bude muset ovládat práci s Mathematicou. Příprava takto komplexní učebnice zabere pravděpodobně více času než práce na klasickém skriptu. Aby mohl student využít všech výhod učebnice, bude muset Mathematice alespoň částečně porozumět. Myslím si ale, že výše popsaná přidaná hodnota oproti klasickému skriptu stojí za větší námahu.

Kapitola 2

Wolfram Mathematica

Mathematica je výpočetní software pro symbolickou matematiku. Znamená to, že výpočty, pokud je to možné, nejsou prováděny numericky, ale stále v podobě symbolů. Výpočet probíhá postupnou úpravou výrazů. Základní činností Mathematicy je nahrazování obecnějších symbolů konkrétnějšími. Při úpravě výrazů tedy pracuje stejně jako člověk. Výsledkem výpočtu jsou analytická řešení. Příkladem může být obecný integrál funkce. Výsledkem výpočtu obecného integrálu je předpis primitivní funkce.

Pro řešení běžných problémů je Mathematica vybavena sadou vestavěných funkcí. Z těch základních to jsou funkce pro řešení lineárních a diferenciálních rovnic, algebraické úpravy, Laplaceovu i Fourierovu transformaci atd. Ve verzi sedm je v Mathematice obsaženo více než dva a půl tisíce funkcí a lze předpokládat, že toto číslo se bude dále zvyšovat.

2.1 Historie

Stephen Wolfram, autor softwaru Mathematica, se narodil roku 1959 v Londýně. První vědecký článek publikoval v patnácti letech a titul PhDr. z teoretické fyziky obdržel ve dvaceti letech. Studoval na Oxfordské univerzitě a Caltechu. Během studií i po nich se v osmdesátých letech zaměřil na výzkum celulárních automatů. V roce 1986 začal s vývojem Mathematicy. O rok později založil společnost Wolfram Research a 23. června 1988 vyšla první verze softwaru Mathematica.

Na začátku devadesátých let byla následována druhou verzí, ve které Stephen Wolfram nalezl silný výpočetní nástroj pro svůj předchozí výzkum celulárních automatů. Vy-

tvořil v Mathematice výpočetní framework pro svoje potřeby a pokračoval v intenzivním výzkumu. Výsledkem této mnohaleté práce bylo vydání knihy A New Kind of Science (1200 stran), ve které Wolfram shrnuje výsledky svého předchozího výzkumu celulárních automatů, pomocí kterého nalezl nejenom nový druh otázek, ale i alternativní odpovědi na otázky z fyziky, biologie, kybernetiky a mnoha dalších oborů.

Vývoj software Mathematica pokračoval dále pod hlavičkou Wolfram Research, ze které se postupem času stala úspěšná firma zaměstnávající stovky lidí. S postupujícími verzemi se od hlavní větve začaly oddělovat další nastroje, jako je MathLink, který umožňuje propojit výpočetní výkon Mathematicy s libovolným programem, nebo Web Mathematica, která je určena pro webové publikování. Dalším důležitým nástrojem je Wolfram Mathematica Player, který je přehrávačem pro soubory napsané v Mathematice. Obsahuje ale celé výpočetní jádro, takže umožňuje provádět předem definované výpočty přímo u uživatele. Základní verze přehrávače je nabízena zdarma. Samotná Mathematica byla ve verzi šest rozšířena o možnost online přístupu do znalostních bází Wolfram Research. Data v těchto databázích byla připravena pro zpracování pomocí software Mathematica a jsou neustále aktualizována. Uživatelé Mathematicy tak mají možnost přístupu k datům z oblasti matematiky, geografie, vědy a techniky, ekonomiky a financí a lingvistiky. Vybudováním těchto znalostních bází si Wolfram Research připravil pozici pro svůj, zatím poslední, dlouhodobý projekt s názvem Wolfram—Alfa. Alfa je webový výpočetně-vyhledávací nástroj, jehož jádro je kompletně napsáno v Mathematice (vývojáři uvádí bilióny řádků). Měl by sloužit vědecké komunitě a lidem z průmyslové praxe. Klade si náročný cíl - být univerzálním výpočetním nástrojem obsahujícím všechny dostupné algoritmy a znalosti, které budou dle potřeby kombinovány tak, aby se nalezl požadovaný cíl.

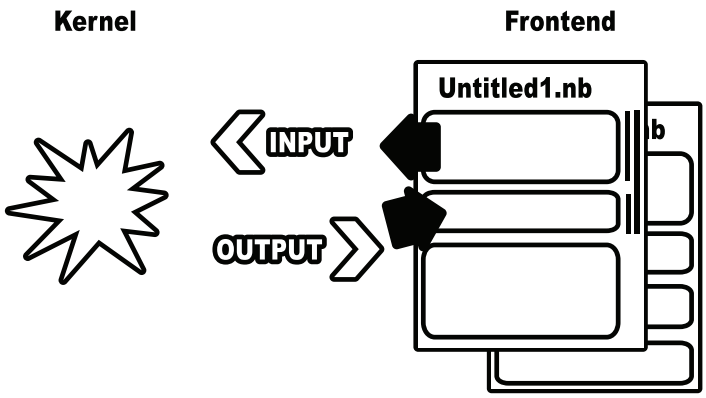
2.2 Struktura

Mathematica je rozdělena na dva samostatné programy. Je to výpočetní jádro - kernel a program pro komunikaci s uživatelem - frontend. Kernel je srdce Mathematicy. Přijímá vstupy (input), provádí výpočty (evaluation) a vrací výsledky (output). Vstupy mohou přicházet z frontendu nebo mohou být propojeny odkudkoliv pomocí MathLinku. Výpočty v Mathematice mohou být paralelní díky tomu, že lze spustit více kernelů. Ty mohou běžet na jednom multiprocesorovém stroji nebo mohou být spouštěny na různých

strojích.

Frontend vizualizuje dokument, do kterého uživatel zapisuje příkazy pro kernel (Notebook, *.nb). Stará se také o správu otevřených notebooků, jako je ukládání, tisk, označování textu, funkce kopírovat a vložit atd. Jeho hlavní funkcí je přijmout od uživatele vstup, odeslat ho kernelu a zobrazit uživateli výstup.

Notebook je rozdělen do buněk. Kernelu je vždy odeslán obsah jedné buňky a pod ní je vytvořena buňka obsahující výsledek zpracování. K notebooku mohou být dále připojeny styly pro formátování.



Obrázek 2.1: Schema kernel, frontend a notebook

2.3 Programovací styly

Jak již bylo řečeno, Mathematica má velké množství vestavěných funkcí, které pokrývají většinu početních nároků uživatele. Může se ovšem stát, že některá funkce bude nedostačující nebo budeme potřebovat kombinovat výsledky výpočtů, zpracovávat série dat atd. Pro tyto případy nám Mathematica nabízí vnitřní programovací jazyk, který uspokojí i toho nejnáročnějšího uživatele. Kromě běžných konstrukcí, jakou jsou cykly a podmínky, známých z procedurálních jazyků (C, Java), nám dává možnost využít vlastností funkcionálního a pravidlového (rule based) programování.

Hlavni filozofie Mahtematicy, kterou Wolfram Research často zdůrazňuje, je, že všechno je symbol. Obrázek vložený v notebooku, pokud je odeslán kernelu, je považován za symbol stejně tak jako π nebo $\frac{3}{5}$. Symboly se používají jako parametry funkcí. Sa-

motné názvy funkcí(Head) jsou také symboly. Dále v textu se již budu držet terminologie Mathematicy a nazývat tyto symboly Head. Symboly, které jsou v Mathematice předpřipravené, začínají velkým písmenem. Je proto výrazně doporučeno, aby uživatelské symboly začínaly malým písmenem.

Hlavní datovou strukturou jsou seznamy (List). Například matice ze zapisuje jako seznam seznamů. Také celá funkce je vnitřně interpretována jako seznam tak, že název funkce - Head je nultým prvkem seznamu a pak následují parametry od indexu jedna. Někomu může tato vlastnost připomínat Lisp. Není to náhoda. Mathematica dále nabízí funkce Map, MapAt, Apply a Lambda (zde je nazývána Function). Mnoho funkcí má atribut Listable, který znamená, že jako parametr funkce může být předán seznam. Příkladem může být funkce Plus - pokud dostane jako parametr dva symboly, vrátí jejich součet, a pokud obdrží dva seznamy, vrátí seznam se součty na jednotlivých pozicích. Funkcionální programování je silnou stránkou Mathematicy a je nutné podotknout, že oproti Lispu přinesla jisté zpřehlednění kódu.

Další vlastností, se kterou se každý uživatel setká, je používání pravidel. Pravidlo je definice pro nahrazovaní symbolů nebo jejich skupin jinými symboly nebo skupinami symbolů. Pomocí pravidla říkáme, že symbol a může být nahrazen symbolem $b+2$.

$$a \rightarrow b + 2$$

. Pravidlo aplikujeme na nějaký výraz pomocí funkce Replace, ReplaceAll, ReplaceRepeated. Často používané funkce mají speciální zkrácený zápis. Například pro ReplaceAll je to /. a //. pro ReplaceRepeated. Zatímco funkce ReplaceAll použije pravidla v argumentu a skončí, funkce ReplaceRepeated používá předaná pravidla do té doby, pokud se výraz, na který jsou aplikována, mění. V pravidlech je možné používat proměnné a je možné testovat, zda má proměnná určité vlastnosti. Pravidlo

$$x_Integer \rightarrow x + 2$$

nahradí pouze celá čísla číslem o dvě vyšším. Používání pravidel je pro Mathematicu nejpřirozenější, protože vystihuje její funkci nahrazování obecnějších symbolů konkrétnějšími.

2.4 Novinky ve verzi 6 a 7

Ve vydání Wolfram Mathematica 6 bylo přidáno několik nástrojů, které kladly důraz na interaktivitu a možnost publikovat výpočty a jejich výsledky. Kromě již zmíněného

napojení na znalostní báze Wolfram Resaerch, to byla funkce Style, která umožňuje měnit nejen formátování, ale i chování buněk notebooku.

Další novinkou byla funkce Dynamic. Dynamic způsobí, že výraz, který obsahuje, je přepočítán pokaždé, když se změní hodnota proměnné ve výrazu, a to i bez toho, aby buňka, která dynamic obsahuje, byla odeslána kernelu. Výborně se hodí pro spolupráci s různými kontrolery, jako je například Slider nebo Navigator. Tyto kontrolery umožňují měnit hodnotu proměnné pomocí myši. Díky Dynamic pak lze tuto hodnotu zobrazovat nebo zobrazovat výsledky výpočtů s danými proměnnými.

Dynamic využívá funkce Manipulate, která byla také přidána ve verzi šest. Tato funkce vytváří dynamický modul(DynamicModule). To způsobí, že lokální proměnné v rámci modulu jsou Dynamic a jejich hodnoty je možné nastavovat pomocí kontrolerů. Tyto změny se automaticky projeví na obsahu, který je parametrem Manipulate. Nejčastěji to bývá Graphics nebo Plot. Typy kontrolerů jsou automaticky voleny v závislosti na inicializačních hodnotách nebo na nastavených hranicích proměnných. Pokud uživatel chce, lze je samozřejmě nastavit i ručně.

2.5 Mathematica a eBook

V nynější podobě verze sedm má Mathematica solidní základ pro plnohodnotné publikování a prezentaci prováděných výpočtů a jejich výsledků. Výrazně k tomu přispěla zmiňovaná funkce Style. Ta umožňuje klasické nastavení písma a odstavce, dále umožňuje formátovaní buněk včetně rámečků a popisků. Dále lze nastavit pravidla pro spojování buněk do skupin (CellGroup) a formátování těchto skupin. Stylem lze i některou buňku nebo jejich skupinu zavřít a tím ji skrýt, nebo ji učinit nedostupnou a zakázat tak její editaci.

Další pomůckou pro prezentaci dat jsou prostředí. Každý notebook může mít nastaveno prostředí pro tisk (PrintStyleEnvironment) a prostředí pro zobrazení ve frontendu (ScreenStyleEnvironment). Základních prostředí je pět: Working, Presentation, Printout, SlideShow a Condensed. Styly pro každé prostředí jsou nastaveny odlišně. Například tiskové prostředí (Printout) má menší písmo než pracovní (Working) a to má menší písmo než prostředí pro prezentaci(Presentation). Uživatel může definovat vlastní styl pro jedno nebo více prostředí.

Mathematica má předpřipravené styly pro články, knihy i prezentace, ale díky prostředím

lze připravit jeden soubor, který můžeme snadno upravit, prezentovat i vytisknout na papír bez nutnosti změny nebo úpravy stylu. Zde je nutné podotknout, že dosáhnout tohoto stavu vyžaduje úsilí a jistou dávku zkušeností.

Kapitola 3

Realizace

3.1 Základní problémy

Již na počátku práce se objevilo několik zásadních problémů, které bylo třeba vyřešit, aby byly splněny základní vytčené cíle. Hlavně šlo o promítání změn ve vzorcích a výpočtech do celého textu, předpoklad čtenářovy nízké úrovně znalostí Mathematicy, zobrazení na PC a tisk na papír v podobě klasického skripta. Tyto problémy by se daly rozdělit do několika skupin.

První skupinou jsou problémy s výpočty. To jsou hlavně algebraické úpravy výrazů a výpočty rovnic. Tyto potíže pramení z nejednoznačnosti matematické symboliky. Problémem bylo používání znaménka rovná se. Dříve jsem si neuvědomoval, že běžně používáme stejný znak pro tři matematické operace: definici, porovnávání a dosazení. Člověk bez větších problémů z kontextu pozná, o kterou operaci se jedná, ale stroj není zatím tohoto rozhodnutí stoprocentně schopen, proto byla v Mathematice zvolena syntaxe pro definici následovně:

$$a = b + 2$$

. Tato formule definuje nový symbol a jako součet symbolu b a čísla dvě. Pro porovnávání je použit symbol ==:

$$a == b + 2$$

. Pro dosazení se používá pravidla(Rule):

$$a \to b + 2$$

. Pravidlo nedefinuje žádný nový symbol. Pouze říká, jaký symbol má být nahrazen danou skupinou symbolů. Pro samotné dosazení se použije funkce Replace, ReplaceAll

nebo ReplaceRepeated. Podobná nejednoznačnost platí i v případě závorek. Stejné symboly používáme pro seznam parametrů funkce *f(x,y)* i upřednostňování operací *2(a+b)*. Mathematica má pro tyto situace opět rozdílné znaky. Seznam parametrů funkce se píše do hranatých závorek

$$f[x, y]$$

a upřednostňování operací se děje klasicky pomocí kulatých závorek

$$2(a + b)$$

. Matehmatica dále ještě používá složené závorky pro zápis seznamu (List) a dvojité hranaté závorky *[[n]]* pro index pořadí, kde n je pořadí prvku v seznamu.

$$\{a, b, c, d\}[[2]]$$

Výsledek zpracování tohoto výrazu bude druhý prvek v seznamu $\{a,b,c,d\}$, tedy symbol b.

Další skupinou jsou problémy se zobrazením vstupu a výstupu a problémy s tiskem, které jsem řešil pomocí tvorby vlastní šablony stylů. Do této skupiny patří hlavně práce s TraditionalForm, což je zobrazení výstupu ve formě, která odpovídá zaběhlým standardům pro zápis matematických formulí a je tedy pro každého snadno čitelná. Tento zápis ale není možné odesílat kernelu ke zpracování. Kernel přijímá pouze kódy v syntaxi Mathematicy. Toto omezení pramení z nejednoznačnosti symbolů používaných v matematice, která byla popsána výše.

Třetí skupina problémů se týká vytváření modelů a simulací. Zde se mi osvědčilo používání funkce Manipulate ve spojení s funkcí Graphics a jejími primitivy nebo funkcemi pro kreslení grafů.

Na první pohled se zdálo, že problémy s promítáním změn ve vzorcích a výpočtech by se daly vyřešit použitím funkce Dynamic. Toto zdání se ale ukázalo jako mylné. Problém je v tom, že Head Dynamic je nepoužitelný pro výpočty a dosazování. Pokud jsem vytvořil pravidlo, které bylo celé ve funkci Dynamic, pak bylo nepoužitelné pro dosazování. Sama Mathematica ho už nepovažovala za pravidlo, protože nemělo Head Rule, ale Dynamic. Pokud jsem vložil Dynamic na levou a pravou stranu pravidla a nikoliv okolo celého, bylo takové pravidlo nepoužitelné pro větší řetězení. Navíc toto řešení výrazně ubíralo na čitelnosti.

```
Dynamic[p] + 2 Dynamic[a] /. Dynamic[a] -> Dynamic[b]

p + 2a /. a -> b
```

Podobně je tomu i u funkce Solve, která slouží k řešení algebraických rovnic. Rovnice obalená funkcí Dynamic není považována za správně zapsanou rovnici, protože její Head není Equal, ale opět Dynamic. [Více viz 3.2]

3.2 Styly

Formátování notebooku v Mathematice je řešeno pomocí stylů. Každá buňka v notebooku má přiřazen styl. V každém stylu je definováno, jak buňka vypadá (velikost, barva a typ písma, zarovnání, okraje atd.) a jak se chová (slučování s okolními buňkami do skupin, reakce na myš, označování nebo nahrazování určitého textu). Styly se zapisují do šablon (Stylesheet). Šablona je standardní notebook, který obsahuje buňky, ve kterých jsou funkce StyleData. Dokumentace funkce StyleData se v nápovědě k Mathematice nedá najít. Podle mých zkušeností se chová tak, že první parametr je název stylu a druhý parametr je název prostředí (Environment). Funkce StyleData je celým obsahem buňky. Takto zapsané buňce můžeme nastavit libovolné vlastnosti pomocí Option Inspectoru (File → Option Inspector, Ctrl+Shift+O). Také lze pomocí příkazu Show Expression(Cell → Show Expression, Ctrl+Shift+E) zobrazit zápis buňky v syntaxi Mathematicy a jako druhý až n-tý parametr funkce Cell zapsat vlastnosti a jejich hodnoty jako pravidla. Definice standardního stylu Text potom vypadá:

```
Cell[StyleData["Text"],
    CellMargins->{{60, 10}, {7, 7}},
    LineSpacing->{1, 3},
    MenuPosition->1400,
    MenuCommandKey->"7"
]
```

Šablona může podědit styly od jiné šablony. V odvozené šabloně je pak možné přidat další styly nebo ty poděděné upravit. Upravuje se také pomocí funkce StyleData. Použije se stejné jméno případně i prostředí stylu a přidají se vlastnosti, které je třeba změnit nebo přidat. Dědění šablon se provádí funkcí StyleData. Celá buňka vypadá následovně:

```
Cell[StyleData[StyleDefinitions -> "Core.nb"]]
```

Mathematica má vlastní systém šablon. Základem je šablona Core.nb. V systémech Windows ji lze po standardní instalaci nalézt v adresáři:

```
C:\Program Files\Wolfram Research\Mathematica\7.0\backslash\FrontEnd\StyleSheets\.
```

Ve stejném adresáři se nalézají i ostatní systémové šablony. Tato šablona definuje pět základních prostředí pro notebook, základní vlastnosti notebooku, buňky vstupu a výstupu atd. Od této šablony dědí šablona Detault.nb, která definuje styly pro text, nadpisy, sekce, matematické formule atd. pro základní prostředí notebooku.

Pro svůj účel jsem se rozhodl vytvořit vlastní šablonu eBookStyle.nb. Chtěl jsem minimalizovat množství dostupných stylů a mít přehled o tom, co který z nich dělá a proč je tak definován. Na druhou stranu mně připadalo důležité zachovat základní vzhled a chování notebooku. Proto jsem se rozhodl vyjít z úplně nejzákladnější šablony Core.nb. Tím moje šablona dostala základní nastavení pro notebook a styly pro vstupní a výstupní buňky. Dále jsem z Default.nb přenesl styly pro text, nadpisy a číslované formule.

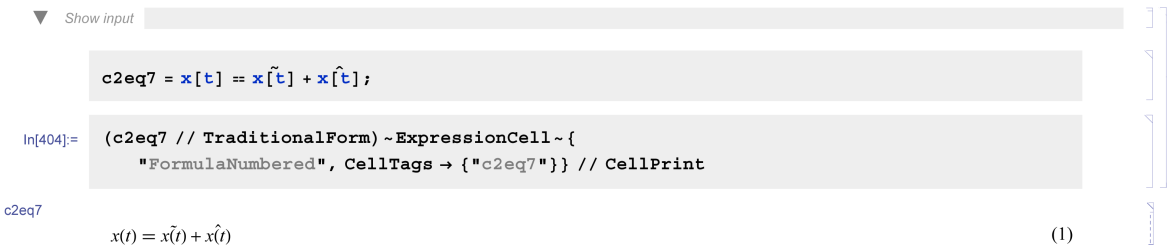
Tato šablona také řeší skrývání syntaxe Mathematicy před uživatelem. Pro skrývání jsem se rozhodl ze dvou důvodů. Za prvé abych zpříjemnil práci s učebnicí uživatelům neznalým Mathematicy, kteří tak uvidí pouze zápisy výpočtů a jejich výsledky v TraditionalForm, tedy tak, jak jsou zvyklí. Za druhé kvůli čtení tištěného materiálu, při kterém studium syntaxe výpočtu pozbývá smysl a navíc její tištění zabírá místo. Čtenář tak bude mít pocit, že čte klasický studijní text. Zároveň musí být syntaxe snadno dostupná pro čtenáře, kteří chtějí výpočty zkoumat a s příklady experimentovat.

Po několika marných pokusech se skrýváním obsahu a zavíráním buňky pomocí vlastnosti CellOpen, jsem dospěl ke konceptu skupiny buněk, kterou lze zavírat a otevírat. Myšlenka je taková, že buňky se vstupem pro kernel jsou spojeny do skupiny, která je zavřená, takže buňky nejsou vidět. Tyto buňky také vygenerují výstup v TraditionalForm, který se do této skupiny nepřipojí. Tohoto chování je dosaženo tak, že celý proces zpracování dat je rozdělen do dvou fází. Skupina vstupních buněk odešle data ke zpracování kernelu, který provede výpočet. Výsledky jsou uloženy do pojmenovaných proměnných. Výstup je potlačen pomocí středníku za jednotlivými výrazy(CompoundExpression). Ve druhém kroku, v další vstupní buňce, je pomocí funkce ExpressionCell vygenerována buňka, do které je vložena hodnota pojmenované proměnné v TraditionaForm. Tato buňka má styl číslované formule (FormulaNumbered) a má přidělen tag (CellTags), který je shodný s názvem proměnné, kterou buňka zobrazuje. Generovaná buňka je zobrazena uživateli pomocí funkce CellPrint. Dále je zde buňka se stylem ViewableInput, která spojí do skupiny všechny buňky stylu Input následující bezprostředně po sobě. Vytvoří jejich nadpis a zobrazí tlačítko pro otevření skupiny (GroupOpener). Naopak buňka stylu FormulaNumbered, ve které je výstup v TraditionalForm určený pro uživatele, se k této

skupině nepřipojí. A je tedy běžně viditelná. Tohoto chování jsem dosáhl pomocí vlastnosti CellGroupingRules.

Vlastnost CellGroupingRules se ukázala jako velice užitečná, ale nápověda k Mathematice o ní pouze udává, že není plně zapracována do systému a může byt změněna. Podařilo se mi najít diskuzní příspěvek P. J. Hintona z Mathematica Programming Group (Wolfram Research) ze 7. února 2000 [1], ve kterém popisuje, jak tato vlastnost funguje. Může nabývat pěti hodnot: InputGrouping, OutputGrouping, GraphicsGrouping, TitleGrouping a SectionGrouping. Nejzajímavější jsou poslední dvě, které se definují s číselnou hodnotou od nuly do sta. Pokud za sebou následují dvě buňky se stejnou hodnotou vlastnosti CellGroupingRules, pak spolu utvoří skupinu. V případě TitleGrouping a SectionGrouping vytvoří skupinu pouze pokud má předcházející buňka nižší nebo stejnou číselnou hodnotu. Pokud mají buňky rozdílná pravidla pro slučování do skupin, pak platí, že TitleGrouping se spojí s jakoukoliv buňkou, která za ní následuje. SectionGrouping se chová stejně, ale nespojí se TitleGrouping buňkou. Zbylé tři možnosti se s ostatními nespojují.

V šabloně eBookStyle.nb jsem dodefinoval CellGroupingRules pro styl Input na hodnotu SectionGrouping, 50, pro styly FormulaNumbered a Output na hodnotu {*"SectionGrouping", 40*} a pro styl ViewableInput na hodnotu {*"SectionGrouping", 40*}. To vytvořilo požadovaný výsledek.



Obrázek 3.1: Skrývání syntaxe a zobrazení výsledku výpočtu v TraditionalForm

Bohužel má toto nastavení nepříjemné chování. Za předpokladu, že výstupem jsou dvě a více buněk stylu FormulaNumbered nebo Output a za nimi následuje další buňka stylu Input, je poslední buňka výstupu spojena do skupiny s následující buňkou Input. To má za následek, že při znovuzpracování výstupu není poslední buňka vstupu nahrazena novým výstupem, ale je zachována, a nový výstup je vložen před ni.

[1] http://forums.wolfram.com/mathgroup/archive/2000/Feb/msg00126.html

K situaci, kdy za výstupní buňkou (styl Output nebo FormulaNumbered) přímo následuje buňka vstupní (styl Input), by v učebnici nemělo běžně docházet díky uvozující buňce (ViewableInput). Toto může nastat při ladění a testování výpočtů. Z vlastní zkušenosti musím říci, že pokud situace nastane, popsané chování je nepříjemné. Naneštěstí se mi nepodařilo najít výhodnější řešení pohodlného skrývání kódu. Myslím ale, že přidaná hodnota tohoto řešení převyšuje nepříjemnosti, na které běžný i mírně pokročilý uživatel nenarazí.

Pro usnadnění zobrazování kódu jsem pro styl ViewableInput nastavil vlastnost WholeCellGroupOpener na hodnotu True. To zajistí, že po kliknutí na buňku s tímto stylem se celá skupina buněk zavře nebo otevře. S touto vlastností souvisí problém. Pokud má ve stylu buňka nastaveno WholeCellGroupOpener → True, tak při každém kliknutí na oblast buňky v šabloně vyvolá Mathematica chybu a ukončí se. Ztratíte tím tak všechnu neuloženou práci. Toto chování jsem pozoroval ve verzi 7.0.1.

Jak už jsem zmínil, každá výstupní buňka má přidělen tag (CellTags). Tag slouží k identifikaci buňky v rámci notebooku. Buňka může mít několik tagů a různé buňky můžou mít stejný tag. Pro interaktivní text jsem potřeboval zajistit, aby byl každý tag jednoznačný. Je to kvůli odkazování na jednotlivé výsledky z textu. To se děje pomocí funkce Hyperlink, kterou lze odkazovat jak do internetu na konkrétní URI, tak i na notebook v operačním systému. Pokud je zadán i tag, je odkázáno na buňku s daným tagem. Jednoznačnost tagu jsem zajistil díky tomu, že výsledek má stejný tag jako je jméno proměnné, jejíž obsah zobrazuje. O tom, jak je zajištěna jednoznačnost jmen proměnných, se zmíním v následující části.

Posledním problémem, který souvisí se styly, i když jen okrajově, byla nutnost přepočítání celého notebooku. Tato potřeba vznikla kvůli tomu, že nelze snadno použít funkce Dynamic, jak jsem popsal na začátku kapitoly. Aby se změna provedená ve vzorci nebo výpočtu projevila v celém notebooku, je potřeba, aby se po provedení změn přepočítal celý notebook. Toho lze dosáhnout pomocí položky menu Evaluate Notebook (Evaluation → Evaluate Notebook). Kvůli usnadnění práce pro uživatele nezběhlé v Mathematice jsem vedle každého políčka stylu Input vložil tlačítko s nápisem Evaluate, které funguje stejně jako položka menu Evaluate notebook. Vložení tlačítka je zapracováno přímo v definici stylu Input v šabloně eBookStyle. Zobrazení tlačítka je provedeno pomocí vlastnosti CellFrameLabels, která umožňuje připojit popisek ke každému rámečku buňky (levý, pravý, horní a dolní). Jako popisek lze vložit celou buňku. Definice celého tlačítka vypadá takto:

```
Cell[
```

```
BoxData[
    ButtonBox[
        "Evaluate",
        Evaluator -> Automatic,
        ButtonFunction -> (
            SelectionMove[ButtonNotebook[], All, Notebook];
            SelectionEvaluate[ButtonNotebook[]];
            SelectionMove[ButtonNotebook[], After, ButtonCell]),
        ]
    ]
]
```

Tlačítko je umístěno v buňce (Cell). Pokud buňka obsahuje nějaká grafická data, musí být tato data obalena funkcí BoxData. Funkce ButtonBox vykresluje samotné tlačítko. První parametr je nápis na tlačítku, druhý (Evaluator) je jméno kernelu,, který bude zpracovávat funkci daného tlačítka. To je vhodné pro paralelní zpracování náročnějších dat. Třetím a nejdůležitějším parametrem je funkce (ButtonFunction), která se vykoná po stisku tlačítka. Zde jsou to postupně příkazy: pro označení celého notebooku (SelectionMove), pro zpracování označeného obsahu kernelem (SelectionEvaluate), pro přesunutí kurzoru za buňku, ve které bylo stisknuto tlačítko (SelectionMove).

3.3 Výpočty

Vzhledem k rozsáhlé návaznosti výpočtů bylo třeba vymyslet, jak používat proměnné a konstanty tak, aby pisatel nebyl nijak omezován v pojmenovávání veličin a hodnot. Protože Mathematica sdílí symboly mezi všemi notebooky, bylo potřeba zajistit tuto jednoznačnost pro všechny kapitoly, tedy skrze celou učebnici. V žádném případě nebylo možné používat konstrukce jako je tato.

```
F = m a;
a = v'[t]
```

Tímto způsobem by pisatel brzy vyčerpal písmena na značení proměnných. Určitě by se také vyskytly problémy ve výpočtech.

Jak jsem již popisoval výše, navrhl jsem systém jednoznačných jmen proměnných, do kterých se ukládají výsledky jednotlivých výpočtů. Jména těchto proměnných se skládají z čísla kapitoly a pořadového čísla výpočtu. Tím je zajištěna jejich jednoznačnost. Konkrétní jméno může být c2eq12 (chapter 2, equation 12). Čísla výpočtů nemusí souhla-

sit s tím, jak jdou po sobě v textu. Jde pouze o to, aby byla unikátní. Jména proměnných začínají malým písmenem, protože všechna jména systémových symbolů začínají velkým písmenem.

Další otázkou bylo, jak zapisovat rovnice a definice do těchto proměnných. Nalezl jsem dvě možnosti, jak dosáhnout řetězení výpočtů. Obě tyto možnosti souvisí s nejednoznačností používaní znaménka rovná se, jak už jsem popisoval.

Jeden způsob je pomocí rovnic. Jednotlivým proměnným jsem přiřazoval rovnice. Tyto rovnice nebo jejich soustavy jsem potom řešil pomocí funkce Solve, která řeší algebraické rovnice. Výsledky funkce Solve jsem opět upravil na rovnice a přiřadil další proměnné. Tento způsob řešení fungoval uspokojivě. Celý proces vypadá následovně:

```
c1eq1 = Ft == g m phi[t];
c1eq2 = Fp == v[t] k;
c1eq3 = -F == Fp+Ft;
Solve [{c2eq1, c2eq2, c2eq3}, {F, Ft, Fp}];
c2eq4 = F == (F/.Flatten[%])
```

Výsledek uložený v proměnné c2eq4 je opět rovnice. Tuto rovnici je možné čtenáři zobrazit v TraditonalForm:

$$F = -g\, m\, phi(t) - k\, v(t)$$

Vzhledem k tomu, jak Mathematica pracuje, a lze se domnívat, že i kvůli výkonnosti, není tento proces úplně ideální. Dalším způsobem, jak dosáhnout stejného cíle, je použití pravidel v kombinaci s funkcí ReplaceRepated(//.). Postupuje se podobně, pouze se do proměnné neuloží rovnice, ale pravidlo. Místo funkce Solve je použita funkce ReplaceAll, která se postará o dosazení do základní rovnice.

```
c2eq1 = Ft -> g m phi[t];
c2eq2 = Fp -> v[t] k;
c2eq3 = -F -> Fp + Ft;
c2eq4 = c2eq3 //. {c2eq1, c2eq2};
c2eq4 /. Rule -> Equal // TraditionalForm
```

Ve čtvrtém kroku funkce ReplaceRepeated použije pravidla *c2eq1* a *c2eq2* a nahradí symboly *Ft* a *Fp* podle příslušných pravidel. Výsledek uložený v proměnné c2eq4 je tentokrát pravidlo. Pokud chceme čtenáři předložit pravidlo v TraditionalForm, musíme provést ještě záměnu šipky za symbol rovná se. To proto, aby dostal výstup opět ve formě, kterou dobře zná.

$$-F = g\, m\, phi(t) + k\, v(t)$$

Samozřejmě pro řešení algebraických rovnic je nutné stále používat funkci Solve. Při vyjadřování hodnoty proměnné, například m z poslední rovnice, pravidla nepomohou. Funkce ReplaceAll nebo ReplaceRepeated pouze dosazuje symboly za jiné symboly. Výhodou je, že výsledky funkce Solve jsou již připraveny ve formě pravidel, takže se dají snadno použít pro další nahrazování.

Osobně výrazně doporučuji pro dosazování používat pravidla. Díky zkušenostem, které jsem nabyl při přípravě a testování, se pravidla ukázala jako přehlednější a rychlejší k používání. A vhledem k tomu, jak je Mathematica navržena, také předpokládám, že používání pravidel pro dosazování je původní záměr tvůrců.

Další problém, který se při řešení příkladů objevil, bylo řešení maticových rovnic. Mathematica nemá vestavěnou funkci pro práci s maticovými rovnicemi. Mathematica matice vnitřně interpretuje jako seznamy seznamů. Matice je pro ni symbol a s těmi lze provádět libovolné operace. Proto můžeme použít funkci Solve i pro řešení lineárních rovnic. Pro skalární součin Mathematica používá funkci Dot (zkratka $\cdot$). Problém je v tom, že nezná inverzní funkci k Dot, kterou by mohla používat při řešení rovnic. Do výsledků, k jejichž dosažení bylo potřeba násobit inverzní maticí, funkce vkládá funkci *InverseFunction*.

$$\left\{ y(t) \to \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .\text{InverseFunction}[\text{Dot}, 2, 2] \left[\begin{pmatrix} 0 & \frac{1}{l} \\ -g & -\frac{k_p}{m} \end{pmatrix}, x'(t) \right] \right\}$$

InverseFunction může mít tři parametry. Prvním je funkce, ke které je inverze tvořena. U funkcí více proměnných následují ještě dva parametry. Prvním z nich je pozice proměnné, na kterou se inverze aplikuje, a druhým je počet proměnných celkem. V výše zmíněném příkladě se maticově dělí (inverzní operace k maticovému násobení) vektor *x'(t)*. Inverzní operace k *Dot* se tedy provádí na druhé proměnné ze dvou.

Jediným řešením této situace, ke kterému jsem se dopracoval, bylo vytvořit pravidlo, které nahradí inverzní funkci. Bylo tedy nutné dodefinovat násobení inverzní maticí. Pravidlo vypadá následovně:

```
InverseFunction[Dot, 2, 2][aa_?MatrixQ, bb_] :> Inverse[aa].HoldForm[bb]
```

Pravidlo nejprve testuje, jestli je první parametr předávané funkce Matice(MatrixQ). Pokud ano, udělá jeho inverzi, tedy inverzní matici, a vynásobí ji druhým parametrem. Matoucí může být použití funkce RuleDelayed(:→). Tato funkce pracuje stejně jako Rule, ale pravá strana pravidla je vyhodnocena až poté, co jsou dosazeny oba parametry. Je to pro případ, že by v globálním kontextu byl již definován symbol *aa* nebo *bb*. Výsledek po

použití pravidla vypadá dle očekávaní:

$$y(t) = \begin{pmatrix} -\frac{lk_p}{gm} & -\frac{1}{g} \\ l & 0 \end{pmatrix} .x'(t)$$

S používáním matic v Mathematice souvisí ještě jedna nepříjemnost - značení matic velkým písmenem. Jak jsem již zmiňoval, není dobré pojmenovávat uživatelské symboly tak, aby měly první písmeno velké. Tady se dostávám k velkému sporu s tím, že jsem chtěl čtenáři předkládat všechny výpočty a symboly ve formě, na jakou je zvyklý. Řešení by se dalo najít pomocí vlastnosti InputAutoReplacements, kterou může mít buňka nebo celý notebook. Tato buňka okamžitě nahradí definovaný text po vložení. V učebním textu jsem při testování použil značení dvěma malými písmeny místo jednoho velkého. Například místo A jsem použil aa.

Poslední částí týkající se výpočtů, která je pro výuku teorie dynamický systémů důležitá, je Laplaceova transformace. Není třeba se učit slovníky, ani se zabývat algebraickými úpravami zlomků. Mathematica vyřeší vše za čtenáře. Laplaceova transformace zde funguje výborně jak na obecných vzorcích, jako je například ověření linearity:

```
HoldForm[LaplaceTransform[k f[t],t,s]] == LaplaceTransform[k f[t],t,s]
HoldForm[LaplaceTransform[f[t] + g[t],t,s]] == LaplaceTransform[f[t] + g[t],t,s]
```

$$\mathcal{L}_t[kf(t)](s) = k\left(\mathcal{L}_t[f(t)](s)\right)$$

$$\mathcal{L}_t[kf(t)](s) = k\left(\mathcal{L}_t[f(t)](s)\right)$$

tak i na konkrétních výpočtech. Příkladem může být vytváření Laplaceova obrazu diferenciální rovnice. Zde musíme dát pozor na to, jak Mathematica transformuje obecnou funkci. Vzhledem k tomu, že tvar funkce je neznámý, je transformace ponechána v nezměněném stavu. Toho lze výborně využít k nahrazení obrazu vlastním znakem.

$$\text{c3eq1} = -\frac{g\phi(t)}{l} - \frac{k_p\phi'(t)}{m} - \phi''(t) = -\frac{F_v}{lm};$$

c3eq2=LaplaceTransform[c3eq1, t, s];

$$\text{c3eq3} = \left\{LaplaceTransform[\phi[t], t, s] \rightarrow \Phi(s), \phi(0) \rightarrow \frac{\pi}{2}, \phi'(0) \rightarrow 0\right\};$$

c3eq2/.c3eq3

V tomto kódu se nejprve připraví seznam nahrazovacích pravidel. Laplaceův obraz funkce $\phi(t)$ je nahrazen symbolem $\Phi(s)$ a jsou přidána pravidla pro počáteční podmínky.

Potom jsou tato pravidla aplikována na transformovanou rovnici. Výsledek je pro čtenáře snadno čitelný.

$$s^2\Phi(s) - \frac{\pi s}{2} = \frac{g\Phi(s)}{l} - \frac{k_p\left(s\Phi(s) - \frac{\pi}{2}\right)}{m}$$

3.4 Modely

Nedílnou součástí navrženého konceptu je možnost vytváření modelů a simulací. Pro tyto případy byla v Mathematice 6 přidána funkce Manipulate. Tato funkce vytváří dynamický modul(DynamicModule) a ke všem lokálním proměnným vytvoří kontrolery, pomocí kterých lze snadno měnit hodnoty proměnných myší. Uživatel tak o Matehematice nemusí vůbec nic vědět a za pomoci standardních ovládacích prvků (slider, combobox, checkbox atd.) je schopen měnit parametry prováděné simulace. Prvním parametrem Manipulate je výraz, který bude zobrazen uživateli společně s ovládacími prvky. Nejčastěji to je funkce Graphics, která umožňuje práci s grafickými primitivy, nebo některá z funkcí pro kreslení grafů (Plot, ParametricPlot, Plot3D atd.). Druhý až n-tý parametr je definice proměnných. Definice proměnné je seznam, který obsahuje tři hlavní prvky: jméno, minimální hodnotu a maximální hodnotu. Mohou následovat další parametry, které upravují chování proměnné a jejího kontroleru. První parametr, jméno proměnné, může být opět seznam, který obsahuje jméno proměnné, počáteční hodnotu a její popisek v seznamu ovládacích prvků. Definice proměnné představující gravitační zrychlení, které je možno nastavit od hodnoty nula až po $20[s^{-2}]$ pomocí slideru, může vypadat například následovně:

$$\{\{g, 9.81, \text{“Gravitační zrychlení } [s^{-2}]\text{”}\}, 0, 20, \text{ControlType} \rightarrow \text{Slider}\}$$

Při zobrazování některých složitějších modelů, jsem potřeboval provést přípravné výpočty před zobrazením samotného modelu. Tyto případy jsem vyřešil pomocí funkce CompoundExpression(zkratka ;). Tato funkce je Mathematicou brána jako jeden výraz. Navíc poslední provedený příkaz je návratová hodnota celého výrazu. Pokud je za posledním příkazem uveden středník, je vrácena hodnota Null. Díky tomu jsem mohl provést libovolné množství přípravných výpočtů oddělených středníkem a nakonec uvést funkci pro zobrazení dat (Graphics, Plot, atd.) bez středníku. Tato funkce byla předána Manipulate jako první parametr a tím zobrazena uživateli.

```
Manipulate[(
    s=a b /2;
    Graphics[{
        Polygon[{{0,0},{a,0},{0,b}}],
        Text["S = "<>ToString[s]<>" m^2",{2,2}]
        }]
    ),
    {a,1,5},
    {b,1,5}
]
```

V tomto příkladě funkce Manipulate zobrazuje trojúhelník s proměnnou délkou odvěsen. Nejprve je vypočítán obsah trojúhelníku, který je potom zobrazen uživateli pomocí funkce Text.

S přípravnými výpočty také souvisí problém lokálních proměnných. Proměnná s v předchozím příkladě je definována jako globální. Používání globálních proměnných jsem se chtěl vyhnout. Jediné proměnné, které mohou zůstat globální, jsou jednoznačně identifikovatelné proměnné s názvy komplikovanými tak, že uživatel prakticky nemá šanci je omylem přepsat.

Tento problém jsem vyřešil pomocí funkce Modul. Tato funkce má dva parametry. V prvním parametru se definuje seznam lokálních proměnných. Druhým je obecný výraz, který tyto proměnné používá. Nadefinované proměnné jsou platné pouze v tomto výrazu. Při použití Module vše funguje jako v předchozím příkladě, ale kolem výrazů CompoudExpression je vložena funkce Module s nadefinovanou proměnnou s. Tímto způsobem sem proměnná s stane lokální v modulu.

```
Manipulate[
    Module[{s},
        s=a b /2;
        Graphics[{
            Polygon[{{0,0},{a,0},{0,b}}],
            Text["S = "<>ToString[s]<>" m^2",{2,2}]
            }]
    ],
    {a,1,5},
    {b,1,5}
]
```

Později jsem objevil další možnost řešení popsaného problému, která je asi efektivnější. Efektivita je nesporná minimálně v psaní a čitelnosti kódu. Celé zjednodušení spočívá v tom, že proměnná se nadefinuje ve funkci Manipulate, což zapříčiní, že je lokální. Dále je potřeba zajistit, aby se nezobrazovala v seznamu ovládacích prvků. To se děje díky

vlastnosti ControlType, kterou je třeba nastavit na hodnotu None. Výsledek je stejný jako při použití funkce Module.

```
Manipulate[(
   s = a b/2;
   Graphics[{
      Polygon[{{0, 0}, {a, 0}, {0, b}}],
      Text["S=" <> ToString[s] <> "m^-2", {2, 2}]
      }]
   ),
   {a, 1, 5},
   {b, 1, 5},
   {s, ControlType -> None}
   ]
```

3.5 Kapitoly a jmenné prostory

Struktura interaktivní učebnice, dle mého návrhu, je dělena do kapitol. Tyto kapitoly by měly být logickými celky, které uzavřeně shrnují část problematiky, tak jako je tomu u klasické učebnice. Zároveň by pisateli i čtenáři měla být poskytnuta možnost se mezi těmito celky snadno pohybovat a přebírat výsledky i vzorce mezi nimi. Toho je dosaženo pomocí systému globálních proměnných s jedinečnými jmény jak v rámci kapitoly, tak i mezi kapitolami samotnými. Tím není uživatel nijak omezován a může bez problémů využívat výsledky a vzorce napříč celou učebnici.

Mathematica používá systém kontextů (Context). Kontext je podobný jmenným prostorům známým například z C++ nebo XML. Každý symbol vytvořený v Mathematice je vytvořený v nějakém kontextu. Základní systémové symboly jsou uloženy v kontextu *System‘*. Uživatelem definovaný symbol se standardně uloží do kontextu *Global‘*. Každý balíček, který je používán, by měl mít vlastní kontext. Právě používané kontexty je možné zjistit pomocí proměnné *$ContextPath*. Kontexty jsou prohledávány v pořadí, v jakém jsou uvedeny v této proměnné. Proměnná *$ContextPath* nemá atribut Protected, takže lze bez problémů kontexty přidávat a odebírat. Není žádný problém vytvořit si symbol ve vlastním kontextu *MyContext‘x* = *y*, ani kontexty řetězit *Global‘MyContext‘f* = *g*.

Použití kontextů pro potřeby interaktivní učebnice jsem zvažoval, ale nakonec jsem se rozhodl pro systém globálních proměnných. Po zkušenostech s prací ve třech kapitolách jsem nenarazil na žádné problémy. Jediným problémem, který by mohl nastat, je, že by

uživatel nadefinoval stejné jméno proměnné, které je již použito v učebnici, a tím by mohl narušit funkčnost kapitoly. Pro větší bezpečnost by se dal nadefinovat samostatný kontext pro každou kapitolu jako subkontext celé učebnice. Tím by byl zmiňovaný problém vyřešen. Otázkou zůstává, zda by režie kolem správy kontextů nečinila potíže. V případě, že by zmiňovaný problém nastal, není, vzhledem k nahrazování a pravidlům, náročným úkolem do systému proměnných zabudovat kontexty.

Kapitola 4

Popis složitějších konstrukcí

V této kapitole bych se chtěl věnovat výhodám, které přináší spojení funkcionálního a pravidlového programování, které Mathematica nabízí. Chtěl bych se podělit o zkušenosti, které jsem během své práce nabyl. Doufám že moje zjištění budou případnému čtenáři k užitku. Popis této funkcionality provedu na příkladech použitých v mé práci. Je to script pro generování obsahu učebnice, který je uložen v souboru generatorObsahu.nb a jednodušší funkce použité v kapitole o řízení lineárních systémů (3_RizeniLinearnichSystemu.nb).

4.1 Přehled programovacích stylů

Základem funkcionálního programování v Mathematice jsou funkce Function(&), Map(/@) a Apply(@@). Function implementuje anonymní funkci, která je v LISPu známá jako lambda funkce. Každá z těchto funkcí má pro snazší použití zkrácený zápis, který je uveden v závorce. Všechny se chovají podobně jako v LISPu.

Funkce Function využívá nepojmenované parametry, které jsou značeny #, případně #n, kde n je pořadové číslo argumentu od jedné. Dále je ještě možné použít znak ##, který značí sekvenci všech parametrů. Příklad použití Function může být:

```
((#1+#2)#3)&[1,2,3]
```

Výsledkem je číslo 9. Funkce nejprve vezme první dva argumenty (1 a 2), ty sečte a vynásobí je třetím (3).

Funkce Map vezme funkci předanou jako první parametr a aplikuje ji na všechny prvky výrazu, který je druhým elementem této funkce. Tato funkce má ještě třetí parametr,

kterým se dá specifikovat úroveň, na kterou se funkce aplikuje. Bohužel při specifikaci úrovně nelze použít zkrácenou verzi.

```
Abs /@ {-1, -2, -3}
Map[Abs, {-1, {-2}, {-3, -4}, -5}, {2}]
```

V prvním případě bude výsledkem seznam {*1,2,3*} a ve druhém to bude {*-1,* {*2*}*,* {*3, 4*}*, -5*}.

Funkce Apply je z předchozích nejjednodušší. Vezme symbol který dostane, jako první parametr a nehradí jím Head výrazu, který má jako druhý parametr.

```
Plus @@ {1, 2, 3}
Apply[Plus, {1, {2, 3}, 4}, {1}]
```

Výsledek prvního zpracování bude číslo 6, protože Head List bude nahrazen za Head Plus. Druhým výsledkem bude seznam {*1, 5, 4*}, protože záměna Head proběhne pouze u vnořeného seznamu {*2,3*}.

Principy pravidlového programování jsem shrnul již dříve. Jen pro přehlednost tedy zopakuji základní zásady. Základním stavebním kamenem je pravidlo Rule(→) nebo RuleDelayed(:→). Pravidla jsou aplikována pomocí funkcí ReplaceAll(/.) a ReplaceRepeated(//.). Rozdíl je v tom, že ReplaceAll použije seznam pravidel a vrátí výsledek, zatímco ReplaceRepeated Aplikuje pravidla, dokud se výraz, ve kterém se nahrazuje, mění.

Důležitou součástí používání pravidel jsou vzory (Patterns). Vzory umožňují, aby pravidla byla obecnější. Umožňují používat lokální proměnné a navíc testovat jejich hodnoty. Ve skutečnosti toho dokážou mnohem více, ale pro přiblížení to stačí. Dalo by se říci, že pravidla a vzory jsou základem Mathematicy.

Základním prvkem vzorů je Blank(_). Blank umí nahradit libovolný symbol. Dále jsou BlankSequence(__) a BlankNullSequence(___). První nahrazuje jeden nebo více symbolů a druhý žádný nebo více symbolů. Před Blank jde předřadit jméno, do kterého se uloží symbol vyhovující vzoru. Ten lze pak použít na pravé straně pravidla. Tyto proměnné jsou lokální pouze pro dané pravidlo.

$$\mathrm{x_} \rightarrow x^2$$

Za Blank nebo jeho dva příbuzné může být doplněn libovolný Head. Ten potom reaguje pouze na symboly s tímto Head.

$$_\mathrm{Integer}$$

Nahrazuje pouze celá čísla typu Integer. Podobnou funkci jako připojení Head za Blank má PatternTest(?). Ten vyhodnotí vzor jako padnoucí, pokud symbol vyhoví testu za otazníkem. Například vzor:

$$_?\mathrm{NumberQ}^{_\mathrm{Integer}}$$

vyhoví libovolnému číslu umocněnému na celočíselný exponent. Další konstrukcí, která omezuje symboly vyhovující vzoru, je Condition(/;). Condition, jak název napovídá, je podmínka, kterou musí splnit pojmenovaný vzor, aby vyhověl. Vzor

$$\mathrm{a_NumberQ}^{\mathrm{x_Integer}}/;x > 2$$

vyhoví libovolnému číslu umocněnému na celočíselný exponent, který je větší než 2. Další užitečnou funkcí jsou alternativy (Alternatives, |). Ty umožňují, že pravidlo vyhoví jedné z možností.

$$a|b \rightarrow x$$

4.2 Příklady

Pro představení funkcionálního programování v praxi si vypůjčím příklady z mé práce. Konkrétně to jsou funkce, které pracují na pozadí nástroje pro hledání regulátoru systému metodou GMK.

Začnu funkcí pracující na pozadí nástroje pro hledání regulátoru systému metodou GMK. Tato funkce vykresluje póly a nuly na větve grafu. Je součástí vlastnosti Epilog parametrického grafu (ParametricPlot). Po vykreslení samotného grafu jsou vykreslena grafická primitiva, která jsou hodnotou této vlastnosti.

```
Epilog->{
   Point[({Re[s /. #], Im[s /. #]} &) /@ c3eq16 /. k -> kk],
   Red, Opacity[0.5], PointSize[Large],
   Point[({Re[s /. #], Im[s /. #]} &) /@ c3eq17 /. k -> kk]
   }
```

V proměnné *c3eq16* jsou umístěny póly zpětnovazebního systému ve formě seznamu pravidel. Jsou to výsledky funkce Solve pro $s = 0$.

$$\{\{s \rightarrow x_1\}, \{s \rightarrow x_2\}, \{s_3\}, \ldots\}$$

Proměnná *c3eq17* je stejná, jen obsahuje nuly systému. Tato pravidla se díky funkci Map stanou parametry anonymní funkce.

```
({Re[s /. #], Im[s /. #]} &)
```

Funkce použije pravidla k nahrazení za *s*, čímž získá jeho hodnotu. Z této hodnoty si vezme reálnou a imaginární část (funkce Re a Im), které vloží do seznamu. Tento seznam reprezentuje bod v Gaussově rovině. Protože původní pravidla byla uzavřena do seznamu, je díky namapování řádkové funkce na tato pravidla vytvořen seznam bodů (seznam seznamů). V tomto seznamu se *k*, které symbolizuje zesílení regulátoru, nahradí proměnnou *kk*. Ta je nadefinována ve funkci Manipulate, která celý graf zobrazuje, je tedy nastavována uživatelem. Seznam bodů, ve kterém jsou v tomto kroku jen číselné hodnoty, je předán funkci Point, která je vykreslí. Mezi kreslením nul a pólů se provede změna vzhledu bodů, aby od sebe byly vizuálně odděleny.

Příklad, ve kterém jsem použil kombinaci vzorů a funkcionálního programování, je script na generování obsahu. Tento script projde všechny notebooky ve složce, načte z nich nadpisy a podnadpisy a odkazy na ně uloží do souboru Obsah.nb.

```
generateTblOfContents[l_List] :=
 If[l == {}, Return[{}],
   Module[{nb},
     nb = NotebookOpen[First[l], Visible -> False];
     SelectionMove[nb, All, Notebook];
     Join[
     (ExpressionCell[
        Hyperlink[#[[1]],
        {"." <> $PathnameSeparator <> #[[3]], #[[4]]}],"ContentsTbl" <> #[[2]]] &)
     /@
     (Cases[NotebookRead[nb],
        Cell[x_,z_?(# == "Title" || # == "SubTitle" ||
            # == "SubSubTitle" &), ___, Rule[CellTags, y_], ___] |
          Cell[x_,z_?(#=="Title" || #=="SubTitle" || #=="SubSubTitle" &), ___] -> {
           x, z, FileNameTake[First[l]], y}, Infinity
        ] /. {x_, z_, name_} -> {x, z, name, None}),
    (NotebookClose[nb]; generateTblOfContents[Rest[l]])
    ]
   ]
  ]
```

Funkce generateTblOfContents přijímá jako parametr seznam souborů ve složce. Názvy souborů jsou včetně cesty na disku. Předpokládá se, že v seznamu jsou pouze notebooky, které reprezentují kapitoly knihy. Pokud je seznam prázdný, tato funkce skončí díky počáteční podmínce. Pokud jsou v seznamu nějaké soubory, funkce vytvoří modul s lokální proměnnou *nb*, která reprezentuje ukazatel na notebook. Vezme první notebook v seznamu, otevře ho tak, aby zůstal uživateli skrytý, a označí celý obsah.

```
Module[{nb},
  nb = NotebookOpen[First[l], Visible -> False];
  SelectionMove[nb, All, Notebook];
```

Funkce Cases vrátí ze seznamu, který dostane jako první parametr, seznam položek, které vyhoví vzoru ve druhém parametru. V mém případě jsou seznamem všechny buňku v otevřeném notebooku, které jsou načteny funkcí NotebookRead. Jak už jsem popsal výše, funkce je v Mathematice vnitřně reprezentována jako seznam. Vzor, kterému musí buňka vyhovět, má dvě části oddělené pomocí funkce Alternatives (|). Funkce Cell, která vytváří buňku, má tři části. Prvním parametrem je obsah buňky. Druhým je její styl a dále nasledují vlastnosti buňky oddělené čárkou. Vzor se zaměřuje na tři hodnoty této funkce. První (x) je obsah buňky, což je v tomto případě text nadpisu. Druhá (z) je styl buňky. Vzoru vyhoví pouze ty buňky, které mají style Title, SubTitle nebo SubSubTitle. Toho je docíleno pomocí funkce PatternTest. Znak || znamená logickou funkci *nebo*. V první části pravidla je ještě hledána třetí hodnota (y). Je to hodnota vlastnosti CellTags. Vlastnost je nalezena pomocí BlankNullSequence. Této části vzoru vyhoví pravidlo, před kterým a za kterým je žádný až nekonečno výrazů oddělených čárkami a které má na levé straně symbol CellTags. Druhá část celého vzoru je stejná, pouze se nezajímá o vlastnost CellTags.

```
Cell[x_,z_?(# == "Title" || # == "SubTitle" ||
    # == "SubSubTitle" &), ___, Rule[CellTags, y_], ___] |
  Cell[x_,z_?(# == "Title" || # == "SubTitle" || # == "SubSubTitle" &), ___] -> {
    x, z, FileNameTake[First[l]], y}
```

Z buněk, které vyhoví vzoru, jsou hodnoty (x,z,y) vloženy do seznamu. Tento seznam obsahuje text nadpisu, styl buňky, jméno souboru bez cesty (FileNameTake vybere název souboru z cesty) a pokud buňka vyhověla první části vzoru, zahrnuje i tag buňky. Na výstupu funkce Cases tak vznikne seznam seznamů. Funkce Cases má ještě jeden parametr. Tím je hloubka, ve které se mají vzory aplikovat. Hodnota Infinity vyzkouší všechny vrstvy. Aby se zajistila stejná délka seznamů ve výstupním seznamu, je na výstup funkce Cases aplikováno pravidlo, které do seznamů neobsahujících tag přidá místo tagu hodnotu None.

Na vytvořený seznam je "namapována" funkce, která vytvoří buňku symbolizující položku obsahu. V této buňce je vytvořen odkaz na buňku notebooku, ze které pocházejí data v seznamu. Styly, které mají přiřazeny buňky obsahu, se jmenují stejně jako styly nadpisů, jen mají jinou předponu (*ContentsTblTitle, ContentsTblSubTitle a ContentsT-*

blSubSubTitle). Na této funkci může být zarážející, že není použito standardní číslování parametrů *#1,#2,#3,....* To je způsobeno tím, že funkci je předán pouze jeden parametr a tím je seznam hodnot. Proto funkce pro přístup k těmto parametrů používá indexů hodnot v seznamu.

```
(ExpressionCell[Hyperlink[
    #[[1]],{"."<> $PathnameSeparator <> #[[3]], #[[4]]}],
    "ContentsTbl" <> #[[2]]] &)
```

Výsledný seznam buněk je spojen s výsledkem rekurzivního volání funkce generateTblOfContents. Parametrem rekurzivního volání funkce je seznam souborů, ze kterého je odebrán první prvek pomocí funkce *Rest*. Takto jsou postupně zpracovány všechny soubory v seznamu. Výsledkem celé funkce generateTblOfContents je seznam buněk obsahu. Další funkce vytvoří notebook, přiřadí mu šablonu eBookSytal.nb a tento seznam do něho vloží. Tím je generování obsahu ukončeno.

Kapitola 5

Jak používat navržený framework

Navržený framework je poměrně jednoduchý a uživateli zběhlému v Mathematice by nemělo činit potíže se v něm orientovat. Snažil jsem se, abych tvůrcům učebnice nechával volné ruce a dával jim jen minimum omezení. Proto je třeba omezení, která jsou nastavena, přísně dodržovat, aby nedocházelo ke snížení funkčnosti výsledné učebnice.

5.1 Pravidla, která je třeba dodržet

Výsledná učebnice je členěna do kapitol jako klasická kniha. Každá kapitola musí být uložena v samostatném notebooku. Název notebooku se skládá z čísla kapitoly, podtržítka a názvu kapitoly (například *2_PopisLinearnichDynamickychSystemu.nb*). Notebook začíná buňkou, která obsahuje název kapitoly. Buňka je stylu *Title* a musí mít nastavena tag (*CellTags*), který je unikátní v celé kapitole. Doporučuji jako tag používat text nadpisu bez mezer a diakritiky.

Strukturu kapitoly je možné rozdělit na části pomocí podnadpisů (styl *SubTitle*) a nadpisů nižší úrovně (*SubSubTitle*). Tyto nadpisy také musí mít nastaven unikátní tag.

Vstupní buňce (styl *Input*) nebo jejich sérii musí bezprostředně předcházet buňka stylu *VeiwableInput*. Tato buňka spojí vstupní buňky do skupiny. Díky ní lze tuto skupinu otevírat a zavírat. Tím se zajistí skrývání syntaxe Mathematicy.

Výstupy by měly být uživateli zobrazovány v buňce se stylem *FormulaNumbered* v *TraditionalForm*. Toho je dosaženo pomocí speciální konstrukce, která buňce přidělí i tag, který by měl být opět unikátní. Doporučuji jako tag používat jméno proměnné, kterou buňka zobrazuje, protože to musí být unikátní bezpodmínečně.

```
(c2eq29 // TraditionalForm)~ExpressionCell~{"FormulaNumbered", CellTags -> {"c2eq29"}}
//CellPrint;
```

Tag u buňky s výstupem pro uživatele slouží k tomu, aby se na ni dalo odkazovat z textu. Odkaz se vytvoří položkou menu *Hyperlink* (*Insert* → *Hyperlink*) nebo funkcí *Hyperlink*. Tagy uvádějte vždy, i když máte pocit, že na buňku nebudete odkazovat. Nikdy nevíte, který ze spoluautorů to bude chtít udělat.

Pravidla pro proměnné jsou spolu s nadpisy nejdůležitější zásadou. Proměnné, které jsou globální, musí mít speciální název. Tento název se skládá z písmene *c* (z anglického chapter), čísla kapitoly, písmen *eq* (z anglického equation) a pořadového čísla proměnné (například c3eq32). Žádný jiný globální symbol nesmí být definován. Pro práci s lokálními proměnnými slouží funkce *Module*.

Všechny notebooky s kapitolami jsou uloženy v jednom adresáři. V tomto adresáři je i soubor *generatorObsahu.nb*. Ten podle nadpisů a jejich tagů vygeneruje obsah všech kapitol.

5.2 Doporučení

V této části bych chtěl zmínit několik postupů, které se mi při testování frameworku osvědčily.

Všechny vzorce a definice proměnných doporučuji ukládat do globálních proměnných ve formě pravidel. Tato pravidla potom vzájemně dosazovat pomocí funkce *ReplaceAll* nebo *ReplaceRepeated*.

```
c4eq1 = F -> m a;
c4eq2 = a -> x'[t];
c4eq1 /. c4eq2 //TraditionalForm
```

$$F \to mx'(t)$$

Při práci s funkcí *Manipulate* doporučuji definovat případné lokální proměnné přímo v této funkci. Každé z proměnných, která nemá být zobrazena uživateli, stačí nastavit vlastnost *ControlType* na hodnotu *None*.

Kapitola 6

Závěr

Po dokončení této práce mohu odpovědně říci, že se mi podařilo dosáhnout naprosté většiny vytčených cílů . Mathematica se ukázala jako velice silný nástroj, který výrazně předčil má očekávání. Symbolické výpočty, které provádí, mi pomohly i při řešení problémů mimo rámec této práce. Při testování navrženého frameworku mě překvapila kompaktnost a efektivita práce při spojení funkcionálního a pravidlového programování, kterou Mathematica nabízí.

Framework, který jsem v rámci této práce navrhl, funguje při dodržení pravidel popsaných v poslední kapitole velice dobře. Podařilo se mi zajistit řetězení výpočtů. Není také problémem postupnými komentovanými kroky odvodit diferenciální rovnici popisující chování nějakého systému, tuto rovnici vyřešit a její řešení použít na přípravu modelu. Návaznost jednotlivých výpočtů umožňuje provést změnu kdekoliv v postupu a tato změna se automaticky projeví i ve výsledném modelu. Díky symbolickým výpočtům mohou být vzorce složeny z proměnných a konstant bez přiřazených hodnot. Tyto hodnoty jsou vloženy až do výsledného modelu, kde mohou být za pomoci myši snadno měněny. Výsledky všech výpočtů i zápisy vzorců jsou předkládany v *TraditionalForm*, nikoliv v syntaxi Mathematicy, tedy podle zásad zapisování matematických formulí. Aby se uživatel v učebnici snadno orientoval a mohl s modely experimentovat, nemusí o Mathematice nic vědět .

Zkušenější uživatel má snadný přístup ke kódu, ve kterém jsou výpočty zadávány. Může tak snadno upravovat samotné vzorce a ověřovat svoje nápady. Výsledky experimentů se budou stále projevovat v celé učebnici.

Při tisku učebnice do papírové podoby se mi podařilo dosáhnout vzhledu srovnatelného se skriptem. Do textu není tištěna žádná syntaxe, ani popisky buňek nebo jiné znaky, které by poukazovaly na to, že by výstup pocházel z Mathematicy. Na papír se

vytiskne pouze text, vzorce a výsledky v *TraditionalForm* a případný grafický výstup.

Díky systému globálních proměnných lze z připravené učebnice do nového notebooku vybrat jakékoliv vzorce nebo výsledky. Do notebooku v prezentačním prostředí tak lze snadno vložit vybrané vzorce a modely a doplnit k nim komentáře. Příprava prezentace nebo přednášky z hotové kapitoly proto zabere velice krátkou dobu.

Případný zájemce o pokračování v této práci by se mohl zamyslet nad možností úpravy stylů buněk tak, aby se jen některé zobrazily pouze v určitých prostředích, například prezentačním. Potom by prezentace k přednášce mohla vzniknout pouhým přepnutím prostředí.

Během psaní této práce jsem si také uvědomil, že zařazení vnořených kontextů (*Context*) do systému proměnných, jak popisuji v kapitole 3.5, by mohlo mít další pozitiva. Přiřazením nejvyšší priority kontextu učebnice by se dalo dosáhnout stavu, kdy by bylo možné používat libovolné značení proměnných. Matice by se pak mohly značit velkými písmeny a stejně tak i obrazy Laplaceových funkcí.

Přestože mým cílem nebyl vytvořit učebnici, při testování vyvíjeného frameworku vznikal text o lineárních systémech. Dovedu si představit, že podhled akademického pracovníka může být od toho studentského odlišný. Proto jsem chtěl popsat danou problematiku z pohledu studenta ve formě studenti studentům. Navrhl jsem strukturu kapitol a poskytnul tak ostatním studentům platformu, kde by se mohli podělit o své zkušenosti. Věřím, že spolužáci tuto nabídku přijmou a text se rozroste o další kapitoly.

Literatura

Wagner,Davivd B.; (1996), *Power Programing with Mathematica, The Kernel*, New York: McGraw-Hill

Wolfram Research, Inc.;(2010), *Wolfram Mathematica 7 Documentation Center*, http://reference.wolfram.com/mathematica/guide/Mathematica.html

Stephen Wolfram, LLC;(2010), *About Stephen Wolfram*, http://www.stephenwolfram.com/about-sw/

Wolfram Research, Inc.;(2010), *MathGroup Archive 2000, Re: Cell Grouping Question*, http://forums.wolfram.com/mathgroup/archive/2000/Feb/msg00126.html

Vacek, František;(2010), *Mathematica Programing Tutorial*, http://sourceforge.net/projects/mathcontrolbook/

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD:

- Adresář kapitoly: Sada notebooků s testovacím textem frameworku.
- Adresář BP_martipe3: Tato práce ve formátu pdf.

Příloha B

Dostupnost frameworku

Framework, který je předmětem této práce je dostupný na adrese *http://sourceforge.net/projects/mathcontrolbook/*.