evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vědecké výpočty v matematické biologii

Jiří Hřebíček, Miroslav Kubásek, Lukáš Kohút, Luděk Matyska, Lucia Tokárová, Jaroslav Urbánek

Leden 2012

Příprava a vydání této publikace byly podporovány projektem ESF č. CZ.1.07/2.2.00/07.0318 **„Víceoborová inovace studia Matematické biologie“** a státním rozpočtem České republiky.

Předmluva

Publikace „***Vědecké výpočty v matematické biologii***“ vznikla v souvislosti s řešením projektu ESF č. CZ.1.07/2.2.00/07.0318 „VÍCEOBOROVÁ INOVACE STUDIA MATEMATICKÉ BIOLOGIE“, který si klade za cíl inovovat náplň a provázanost předmětů z kmenového souboru předmětů oboru „Matematická biologie“, studijního programu „Biologie“ Přírodovědecké fakulty Masarykovy university (MU) v Brně profilujících studijní obor a zajišťovaných pracovníky Institutu biostatistiky a analýz (IBA) MU v Brně.

Oproti koncepci doposud používaného učebního textu „***Jiří Hřebíček, Jan Žižka: Vědecké výpočty v biologii a biomedicíně***“ [1] zaměřeného spíše na související problematiku, jako jsou výpočetní systém Maple a jeho aplikace v biologii, informační a komunikační technologie v biomedicíně, metody umělé inteligence, dolování dat a strojového učení, je tato publikace orientována jiným směrem. Čtenář se seznámí se současným pojetím výpočetní vědy a vědeckých výpočtů, jejich historií, používanými informačními technologiemi, problematikou výpočetního prostředí a aplikačním software Maple, MATLAB a SPSS včetně jejich použití tak, aby je mohli využít i studenti dalších oborů studijního programu „Biologie“. Doporučujeme však, aby se nejdříve seznámili s publikacemi „***Jiří Hřebíček, Zdeněk Pospíšil a Jaroslav Urbánek: Úvod do matematického modelování s využitím Maple“*** [2] a ***„Ivanka Horová, Jiří Zelinka: Numerické metody“*** [3]**.**

Studentům, kteří mají více znalostí v biologických oborech a méně v oblasti informatiky (typicky studenti příbuzných biologických či zdravotnických oborů, případně dalších programů MU), tento text bude pomáhat jak v pochopení terminologie potřebné pro vědecké výpočty, tak v tom, kde nalézt a používat zdroje informací o vědeckých výpočtech, jejich algoritmech a aplikacích. K tomu budou sloužit všechny kapitoly této publikace. K hlubšímu osvojení terminologie v oblasti informatiky doporučujeme seznámit se s druhou kapitolou publikace „***Jiří Hřebíček, Miroslav Kubásek: Environmentální informační systémy“*** [4]**.** U studentů s vyšším stupněm informatických vědomostí (například v oboru „Matematická biologie“, případně v dalších studijních oborech a programech MU) tomu nebude třeba.

Publikace pomůže zkvalitnit výuku jednoho z kmenových předmětů oboru „Matematická biologie“ s využitím nových a aktualizovaných informací a doufáme, že přispěje i dalším vysokoškolským učitelům ke zlepšení jejich pedagogických znalostí a dovedností v oblastech matematické a výpočetní biologie a ekologie. Využívá k tomu veřejných webových služeb a portálů se zaměřením na vědecké výpočty, grid a cloud computing a aplikace Maple, MATLAB a SPSS, které jsou v ní přehledně uvedeny.

Tato publikace je dílem níže uvedeného autorského kolektivu, kde Jiří Hřebíček je editorem celé publikace, vytvořil Předmluvu, kapitoly 1, 2, Summary a podílel se na kapitolách 3 až 7, Miroslav Kubásek se podílel na kapitolách 4.2, 6.2 a 7.3, Lukáš Kohút se podílel na kapitolách 6.3 a 8.2, Luděk Matyska se podílel na kapitolách 3, 4.1 a 4.4, Lucie Tokárová vytvořila kapitolu 4.4 a Jaroslav Urbánek se podílel na kapitolách 4.3 a vytvořil kapitoly 5 a 8.1.

V Brně 31. ledna 2012

Jiří Hřebíček, Miroslav Kubásek, Lukáš Kohút,
Luděk Matyska, Lucia Tokárová, Jaroslav Urbánek

ISBN 978-80-7204-781-9

1 Úvod

Cílem, vědeckých výpočtů respektive výpočetní vědy je řešení problémů reálného světa s využitím informačních a komunikačních technologií (ICT), respektive výpočetní architektury (software, hardware a komunikací). Využití ICT je nezbytné, protože současné vědecké problémy jsou často pro teoretické nebo experimentální řešení příliš složité nebo časově velmi náročné. Člověk sice nemůže být nahrazen „počítačem“, avšak lidský intelekt může (při korektně vedených krocích) počítač řídit tak, že lze zkoumat i takové vědecké problémy, které by jinak nebylo možné ani analyzovat, ani řešit.

Pro vědecké výpočty vývoj prostředků ICT, jako jsou aplikační, komunikační a systémový software, výkonný hardware včetně výkonných clusterů, komunikačních technologií pro cloud a grid computing, Web 2.0, viz [4], pokračuje vysokým tempem. Možnost modelovat, počítat a numericky simulovat studované problémy (systémy) proniká do nejrůznějších oborů vědy a techniky.

Aplikační software se obvykle spouští na ICT (na počítačích) s různými množinami vstupních parametrů tak, aby bylo nalezeno řešení zkoumaných problémů (systémů). Složité modely vyžadují rozsáhlé množství náročných výpočtů (obvykle jde o numerické výpočty v pohyblivé řádové čárce). Proto jsou vědecké výpočty často prováděny na superpočítačích nebo pomocí distribuovaných výpočtů v počítačových sítích (*grid computing*[1]), anebo na samostatných „vzdálených“ výpočetních platformách (*cloud computing*[2]).

Současná věda je charakterizována enormním nárůstem schopností měřit a sbírat data v tak rozsáhlých objemech, které dříve nebyly myslitelné. To ovšem vede k požadavkům na vývoj a nasazení nových metod a ICT schopných tato data ukládat, přenášet a dále analyzovat, případně využívat je jako vstupní informace pro matematické modelování [2].

Významnou oblastí využití ICT, ve které je potřeba zpracovat obrovské množství dat a ve které je největší potenciál pro využití superpočítačů, grid i cloud computingu, je výzkum lidského genomu v bioinformatice [1]. V tomto oboru se provádějí především operace nad velkými databázemi. Proto se pracuje s největšími databázovými systémy využívajícími paralelní architektury postavené na nejmodernějších procesorech.

Tato situace se zpracováním dat při analýze genomu v bioinformatice se opakuje i v dalších vědních oborech (viz např. data monitorovaná v životním prostředí, kde počet senzorů narůstá geometrickou řadou a jednotlivé senzory jsou schopny sbírat stále větší počet různých dat [4]), pro něž bude třeba vyvinout nové algoritmy i prostředky ICT.

Ve vědeckých výpočtech se často modelují proměnlivé podmínky a situace reálného světa. Jde například o modelování vývoje počasí, šíření znečištění v ovzduší nebo v oceánech, šíření vln tsunami, deformace kloubů kostry člověka při dlouhodobém zatížení, deformace karosérie automobilu při nárazu, deformace různých jiných zařízení, pohyb hvězd ve vesmíru, šíření karcinomů v tkáni apod.

V odborné literatuře jsou vědecké výpočty a výpočetní věda děleny na další disciplíny [5], které poukazují na šíři jejich bádání. Jsou to například: výpočetní biologie[3], bioinformatika[4], chemoinformatika[5], chemometrika[6], výpočetní chemie[7], výpočetní matematika[8], výpočet-

[1] http://en.wikipedia.org/wiki/Grid_computing
[2] http://en.wikipedia.org/wiki/Cloud_computing
[3] http://www.iscb.org/iscb-aboutus, http://en.wikipedia.org/wiki/Computational_biology
[4] http://en.wikipedia.org/wiki/Bioinformatics
[5] http://en.wikipedia.org/wiki/Cheminformatics
[6] http://en.wikipedia.org/wiki/Chemometrics
[7] http://en.wikipedia.org/wiki/Computational_chemistry

ní statistika[9], vysoce náročné výpočty (*high performance computing*)[10], výpočetní fyzika[11], výpočetní ekonomie[12], finanční modelování[13] apod.

Na vysokých školách se vědecké výpočty nejčastěji studují v rámci studijních programů aplikované matematiky a informatiky nebo v rámci standardní výuky matematiky, přírodních a technických věd a programování. Avšak na mnoha univerzitách ve světě je v současnosti koncipována celá řada i dalších bakalářských a magisterských studijních programů v oborech výpočetní vědy[14]. Některé školy také nabízejí i doktorské studium v oblasti výpočetní vědy, výpočetní techniky, výpočetní vědy a techniky nebo vědeckých výpočtů[15].

Tato publikace se snaží reagovat jak na současný vývoj vědeckých výpočtů respektive výpočetní vědy, tak i na metody jejich výuky na vysokých školách. Proto jsou v ní uvedeny jak odkazy na současnou dostupnou literaturu o výpočetní vědě a vědeckých výpočtech, tak na informační zdroje na internetu, neboť problematika výpočetní vědy je velmi rozsáhlá. Popsat ji až do detailů v plné šíři v tomto textu není možné. Text však poskytuje čtenáři možnost seznámit s problematikou současných vědeckých výpočtů pro obor matematické biologie a biomedicíny jak z odborné literatury, tak i z nejrůznějších informačních zdrojů na internetu[16], kde využívá i internetových encyklopedií (např. Wikipedia[17], cs.Wikipedia[18] apod.), pokud se jedná o ověřené informace. Dále si publikace klade za cíl, aby si čtenář mohl vyzkoušet s příslušným aplikačním software (Maple, MATLAB, SPSS), jak vědecké výpočty v reálných podmínkách na počítačích fungují případně, kde mohou být prospěšně využity.

V dalším stručně shrňme obsah této publikace. Po vysvětlujícím úvodu je v její druhé kapitole uvedena koncepce výpočetní vědy a s ní související terminologie v oblasti vědeckých výpočtů. Je objasněno, co jsou vědecké výpočty či výpočetní věda v současném pojetí, a současně jsou zmíněny i některé jejich aplikace v oblasti biologie a biomedicíny.

Třetí kapitola je věnována historii vědeckých výpočtů od jejich počátku ve čtyřicátých letech minulého století až po současný vývoj, včetně vývoje v České republice (ČR) a na Masarykově univerzitě v Brně.

Dnešní vývoj výpočetního prostředí (algoritmy, metody a ICT) přináší nové možností pro výpočetní vědu a vědecké výpočty, ale také určitá omezení. Této problematice je věnována čtvrtá kapitola. Jde zejména o grid a cloud computing, využívání mobilních platforem, analýzu chyb ve výpočtech a aplikaci numerických metod.

Pátá kapitola je věnována vlivu neurčitosti v modelech, datech a počítačovém zpracování. V této kapitole je rovněž diskutována problematika citlivosti, zejména pak globální citlivosti řešení dané aplikace na jejich parametrech.

V šesté kapitole je stručně popsán nejvíce používaný aplikační software ve vědeckých výpočtech (Maple, MATLAB, SPSS) v oboru „Matematická biologie“ mimo tabulkový procesor Microsoft Excel, jehož využití je ve výuce a pri výpočtech omezeno zejména jeho nu-

---

[8] http://en.wikipedia.org/wiki/Computational_mathematics
[9] http://en.wikipedia.org/wiki/Computational_statistics
[10] http://en.wikipedia.org/wiki/High_performance_computing
[11] http://en.wikipedia.org/wiki/Computational_physics
[12] http://en.wikipedia.org/wiki/Computational_economics
[13] http://en.wikipedia.org/wiki/Financial_modeling
[14] http://www.siam.org/students/resources/cse_programs.php
[15] http://www.siam.org/students/resources/report.php
[16] http://www.brothersoft.com/downloads/biology.html,
http://www.bestfreewaredownload.com/s-cxlvwegy-biology-c-75-science-freeware.html
[17] http://en.wikipedia.org/wiki/Main_Page
[18] http://cs.wikipedia.org/wiki/Wikipedie

merickou nestabilitou. V sedmé kapitole jsou pak diskutovány možnosti efektivního využití tohoto softwaru v paralelních výpočtech.

K výkladu některých částí textu ve čtvrté, páté a sedmé kapitole jsou připojeny řešené příklady pomocí aplikačního software Maple a MATLAB. Na nich může čtenář konfrontovat pozitiva a negativa tohoto software, případně získat inspiraci či volbu pro jeho samostatné používání.

V současné době existuje mnoho vhodných algoritmů a svobodný (nebo open source) software/freeware[19], který s využitím dostupného licencovaného software na MU umožňuje studentům i pedagogům zaměřit se na vlastní řešení biologických nebo biomedicínských problémů (obecně aplikací), a tím neztrácet čas programováním algoritmů pro danou aplikaci.

Proto je v osmé kapitole popsáno ve dvou případových studiích využití aplikačního software Maple, MATLAB a SPSS. Výpisy programů Maple a MATLAB jsou pak vzhledem ke svému rozsahu uvedeny v příloze této publikace zvlášť.

[19] http://en.wikipedia.org/wiki/List_of_open_source_bioinformatics_software

2 Vědecké výpočty a výpočetní věda

V této kapitole uvedeme terminologii, která souvisí s vědeckými výpočty a výpočetní vědou s využitím pojmů zavedených v [1], [3] a [4].

Věda je systematický způsob poznání skutečnosti. Předmětem vědeckého poznání mohou být objekty a procesy živé i neživé přírody nebo lidské společnosti[20]. K vymezení pojmu věda však existuje řada dalších přístupů.

Obecně lze vědu charakterizovat jako nepřetržitý společensky podmíněný proces systematického racionálního poznávání přírody, společnosti a myšlení, při němž dochází ke stále pravdivějšímu odrazu objektivní reality ve vědomí, ke stále hlubšímu pronikání od povrchových jevů k jejich vnitřní podstatě. To umožňuje v současné době stále dalekosáhlejší využívání a ovládání přírodních a společenských procesů a stále účinnější praktické přetváření světa člověkem. Věda představuje nejvyšší formu poznávací teoretické činnosti, jejíž výsledky jsou nenahraditelnou kulturní hodnotou[21].

Jako celostní soustava zahrnuje věda nauky o přírodě i o společnosti, také filozofii jako racionální světový názor, teorie a metody, metodiky a techniky, metodologie, základní i aplikovaný výzkum. Vědu lze rozdělit na skupiny přírodovědných, technických, zemědělských, ekonomických, sociálních, lékařských a jiných vědeckých disciplín.

V užším slova smyslu se vědou nazývá její určité odvětví, obor, disciplína, které charakterizuje specifický předmět poznání a metoda, popřípadě technika výzkumu.

Světoznámý vědec K. F. Popper uvádí v [6], že od novověku je věda založena na představě zákonitého, v některých případech matematizovatelného chování reálného světa, které se odhaluje hypotézou, zobecňuje v teorii a ověřuje pozorováním s opakovatelným experimentem. Věda nastoluje požadavek obecného poznání, na základě něhož lze ve zdánlivě nepřehledném světě oddělit podstatné od nepodstatného a určit obecně platné zákony, pomocí nichž lze úspěšně předvídat a případně zjištěné skutečnosti vhodnými prostředky modelovat. Každá skutečnost je více či méně verifikovanou hypotézou. Mění se každým novým poznatkem.

Současná věda využívá čtyř druhů základních přístupů k poznání reálného světa. Jde o vědecký přístup založený ***na pozorování, na experimentech, na teorii*** a ***na vědeckých výpočtech*** (nazývaný také výpočetní vědou). Někdy se přístup založený na pozorování spojuje s přístupem založeným na experimentech. Dříve, než se budeme podrobněji zabývat objasněním problematiky vědeckých výpočtů, respektive výpočetní vědy, shrňme nejprve první tři přístupy, které byly základem metod poznání po staletí.

2.1 Vědecký přístup založený na pozorování: Pozorovací věda

Vědecký přístup založený na pozorování je základem všech věd a základem všech způsobů, jak poznávat objektivní realitu. Věda začíná a často končí pozorováním. V zásadě lze říci, že, ***pozorovací věda*** (*observation science*) je taková oblast vědy, která používá celou řadu zařízení od pouhého oka až po speciální mechanizmy a zařízení ke sběru dat, například i v souvislosti s určitým zvláštním objektem nebo jevem. Pozorování jsou často, ale ne vždy, prováděna na základě žádaných odpovědí na konkrétně stanovenou otázku. Pro zlepšení kvality pozorování bývá pozorovací věda spojena s výstupy přístrojů, jako jsou například mikro-

[20] http://cs.wikipedia.org/wiki/V%C4%9Bda
[21] http://www.cojeco.cz/index.php?detail=1&id_desc=102538&title=v%ECda&s_lang=2

skopy, teleskopy, družice, senzory a jiná zařízení. Pozorování vytvářejí základ primární vědecké metody, která je cílena k odůvodnění pozorovaného jevu.

U přístupu založeného na pozorování v procesu shromažďování pozorovaných dat vědci uplatňují nejčastěji několik technologií, metod a nástrojů. Pomocí úvah založených na důkazech navrhují možná vysvětlení pro konkrétní soubor shromážděných dat. V tomto vývoji si kladou otázky, jako jsou například:

1. Co se můžeme dozvědět, co se skutečně dozvídáme z těchto pozorování?
2. Jak jsme si jistí, že se určitě jedná o správné vysvětlení?
3. Proč bychom měli brát v úvahu další aspekty?

Pozorovací proces, jako je tomu u většiny ostatních procesů ve vědě, často směřuje k provedení nebo navržení dalších pozorování, která mají být realizována i s případným nasazením jiných zařízení nebo možnostmi dalších úhlů pohledu.

2.2 Vědecký přístup založený na experimentech: Experimentální věda

Vědecký přístup založený na experimentech se ve své podstatě zabývá pozorováním experimentů, „důkazy“ získanými při použití pozorovacích technik při realizaci experimentů a následným prováděním specifických testů, které se týkají některých aspektů v rámci získaných výstupů těchto pozorování. ***Experimentální věda*** (*experimental science*) se soustřeďuje především na provádění měření logickým a systematickým způsobem. Také se zabývá hodnocením a vyšetřováním vztahů mezi příčinou a důsledkem jevů. Pomocí identifikace a izolace (tedy řízením) specifických proměnných, které charakterizují určitý jev, může vědec (experimentátor) testovat vliv těchto (jedné nebo více) proměnných na sledovaný jev. Jedním z problémů experimentální vědy je případ, kdy je často velmi obtížné zajistit, aby testované proměnné neměnily své přirozené chování. V experimentální vědě vědci primárně sledují tři typy proměnných:

1. Nezávisle proměnné, které jsou zkoumány a/nebo řízeny experimentátorem.
2. Závisle proměnné, které se měří nebo počítají.
3. Dodatečné proměnné, které nejsou řízeny a které mohou, ale nemusejí mít vliv na výsledek experimentu.

Pro své průkopnické užívání kvantitativních experimentů, jejichž výsledky matematicky analyzoval, k nejznámějším experimentátorům patří Galileo Galilei (1564 –1642), toskánský astronom, filosof a fyzik, těsně spjatý s vědeckou revolucí. Tyto metody neměly tehdy v evropském myšlení pevnou tradici. Zmiňme pravděpodobně největšího experimentátora, který Galileovi předcházel, Williama Gilberta, který kvantitativní postupy nepoužíval. Galileo svým dílem také přispěl k odmítnutí slepé důvěry k autoritám (např. církvi) nebo k některým myslitelům (jako byl Aristoteles) ve výsledcích vědy. Podílel se i na oddělení vědy od filosofie a náboženství. To jsou hlavní důvody, pro které je považován za „otce vědy“.

2.3 Vědecký přístup založený na teorii: Teoretická věda

Vědecký přístup založený na teorii je nejvíce esoterický[22] z výše zmíněných čtyř přístupů k vědě. Z tohoto důvodu nelze jednoduše definovat, co je teoretická věda. K nejčastěji uváděným odpovědím na otázku „Kdo je představitel teoretické vědy?“ je jméno Alberta Einsteina.

[22] Esoterismus (z řeckého εσωτερικός, esoterikós, „vnitřní“, „uzavřený“) je označení pro souhrn vědomostí určených výhradně pro úzký okruh zasvěcených, intelektuálně vyspělých nebo privilegovaných lidí, viz http://cs.wikipedia.org/wiki/Esoterismus.

Je to klasický příklad člověka, který tvořil skutečnou teoretickou vědu v odvětví fyziky. Teoretická věda byla vždy významnou součástí celkového výzkumného procesu. Ve srovnání s přístupy založenými na pozorování a experimentech však složitost teoretické vědy má za následek nedostatek její výjimečnosti.

Teoretické vědy jsou primárně založeny na matematických základech. Teoretičtí vědci se často pokoušejí matematicky vyjádřit některé pozorovatelné nebo i nepozorovatelné jevy. Použitím různých matematických metod, tedy teoretických pohledů „dokazují" platnost určité hypotézy nebo teorie. Často je tato matematická teorie vyšetřována (ověřována) pomocí pozorování a/nebo experimentů pro další opodstatnění nebo odmítnutí závěrů teoretiků.

Typický koncový produkt teoretické vědy je jeden nebo častěji řada přiměřeně jednoduchých matematických popisů nějakého jevu, objektu nebo události. Ovšem dosáhnout jich velmi často vyžaduje enormně náročný aparát. Tyto popisy například mohou být cíleny k predikci nových událostí a jevů, které jsme doposud nepozorovali, anebo jejichž existenci jsme zatím experimentálně neprokazovali.

Příklad 2.1: Většina lidí je seznámena s Einsteinovou rovnicí $E = mc^2$ udávající teoretický popis množství energie E obsažené v určitém množství látky. Množství energie je určeno součinem hmotnosti m této látky a druhou mocninou rychlosti světla c. Tento vynikající objev, tedy teoretický popis množství energie látky byl později dokázán i cestou pozorování a užitím experimentální techniky.

2.4 Vědecký přístup založený na vědeckých výpočtech: Výpočetní věda

Ze všech jmenovaných čtyř přístupů k získání vědeckého poznání jde o nejnovější přístup založený na vědeckých výpočtech. K tomu přispělo a umožnilo to výrazné zdokonalení hardware a software (ICT) v průběhu posledních třicet let.

Výpočetní věda, někdy známá jako modelování a simulace nebo vědecké výpočty, je používána v přírodovědných, technických, sociálních a ekonomických vědách, k nimž patří např. bioinformatika, výpočetní biologie, výpočetní matematika, výpočetní chemie, výpočetní fyzika, výpočetní inženýrství, výpočetní mechanika, výpočetní ekonomie aj.

V oblasti výpočetní vědy se vědci zabývají specifickými aplikacemi pro daný vědní obor, pro něž jsou definovány algoritmy, které jsou pak prováděny pomocí určité počítačové architektury, tedy užitím prostředků ICT. Tyto aplikace se odkazují na danou oblast vědy, a/nebo na konkrétní řešený problém. Později zmíníme podrobněji některé aplikace ve výpočetní vědě.

Pro účely výpočetní vědy můžeme definovat ***algoritmus*** jako jednoznačně určenou posloupnost konečného počtu elementárních kroků vedoucí k řešení daného problému, přičemž musí být splněna jeho následující kritéria (vlastnosti algoritmu)::

- ***Hromadnost a univerzálnost*** – algoritmus musí vést k řešení celé třídy úloh vzájemně se lišících pouze konkrétními vstupy. Musí řešit úlohu pro libovolnou přípustnou kombinaci vstupních dat a musí pokrývat všechny situace, které mohou při výpočtu nastat.
- ***Konečnost*** – algoritmus musí skončit po konečném počtu kroků.
- ***Rezultativnost*** – algoritmus při zadání vstupních dat vždy vrátí nějaký výsledek (může se jednat pouze o chybové hlášení).
- ***Správnost*** – výsledek vydaný algoritmem musí být správný, získaný korektními kroky.

- ***Determinovanost*** (jednoznačnost) – v každém kroku algoritmem popsaného postupu musí být jednoznačně určeno, co je výsledkem tohoto kroku a jak má algoritmus dále pokračovat. Důsledkem této vlastnosti je fakt, že pro stejná vstupní data vydá algoritmus vždy stejný výsledek.

Je nutné si však uvědomit, že obecně existují algoritmy, které jsou nekonečné, nedeterministické, nedávají vždy stejné výsledky. Vyskytují se například významné nedeterministické algoritmy, jako jsou Monte Carlo metody, které s využitím skutečně náhodných generátorů dají při opakování mírně odlišné výsledky. Je třeba sledovat princip metody.

Výpočetní věda přinesla nový pokrok ve vědecké práci, bádání a myšlení. Je syntézou informatiky a matematických metod pro aplikace při řešení rozsáhlých a složitých problémů v nejrůznějších vědních oborech. Výpočetní věda využívá jak zlepšení v oblasti počítačového hardware, tak pravděpodobně ještě důležitějšího zlepšení v oblasti počítačového software a matematických metod. Umožňuje tak vědcům zkoumat skutečnosti, které pro bádání v minulosti byly příliš komplikované vzhledem ke složitosti jejich matematického vyjádření, rozsáhlých a vysoce náročných výpočtů, časové neúnosnosti nebo kombinace všeho zmíněného. Rovněž umožňuje vědcům vytvářet predikční modely předpovídající, co by se mohlo stát ve výzkumu v laboratoři. Výpočetní věda může tedy predikovat budoucí události – např. konkrétně může odhalit negativa či pozitiva ve vývojových trendech biologických či medicínských jevů aj. Jako taková je výpočetní věda komplementární k předchozím třem přístupům k vědě.

Zkoumaný vědecký problém ve výpočetní vědě musí být vyjádřen matematicky pomocí algoritmu. Vědci, často ve spolupráci s matematiky, tak musejí definovat matematický model pro vyřešení problému ve svém oboru. V tomto okamžiku se zpravidla teoretická a výpočetní věda setkávají. Většina matematických modelů používá aproximací reálných problémů a/nebo zavedení určitých předpokladů, aby byl zjednodušen matematický základ problému a aby mohl být příslušný algoritmus s výše zmíněnými vlastnostmi zkonstruován. Mohou k tomu použít již existující algoritmus, nebo existující algoritmus upravit, anebo vytvořit algoritmus zcela nový. Matematický model musí být testován, jak dobře vyjadřuje modelovaný problém. Toto vyhodnocování se děje v celém procesu modelování jako zpětná vazba pro postupné vylepšování modelu.

Jakmile je vhodný algoritmus sestaven, je převeden do jednoho nebo více počítačových programů (software) a implementován na jeden nebo více typů počítačů (hardware), obecně ICT. Kombinace software a hardware je označována jako ***výpočetní architektura*** (krátce architektura).

K vývoji implementace může být použit některý ze systémů, které obsahují rozsáhlé algoritmické bloky, vyvinuté dříve a optimalizované na modelování určitého jevu. Mezi typické příklady vhodného aplikačního software [1] patří například systémy Maple, MATLAB, SPSS, Statgraphics aj., v nichž lze i komplexní nový model a jeho algoritmus vytvořit formou určitého „skládání“ předdefinovaných bloků. V jiném případě může jeden nebo více programátorů ve spolupráci s matematiky a vědci speciálních odborností vyvíjet žádaný software zcela od začátku, a to s využitím vhodných programovacích jazyků, jako jsou C++, Java, Fortran [7], Python [8], případně jiné.

Pokud jde o volbu hardware, může být software realizován na stolním nebo přenosném počítači s operačním systémem Windows, Linux nebo Macintosh, pokud pro tuto realizaci je dostatečně výkonný (například kapacitně apod.). Pro rozsáhlejší a výpočetně náročnější software jsou často používány vysoce výkonné počítače (od výkonných serverů po superpočítače). Takové počítače mají obvykle větší počet procesorů či jader (CPU). Lze na nich spustit i složitý software efektivně a s nízkými provozními náklady. V České republice existuje něko-

lik center podporujících výkonné výpočty (v Brně, Praze, Ostravě a Plzni), v nichž lze zpracovávat výpočetně náročné úlohy pomocí zmíněných složitých software.

Vědci, studenti i výzkumní pracovníci mohou využít architekturu (ICT) například k provádění experimentů, jež:

1. by mohly být jinak příliš nebezpečné, kdyby se prováděly v laboratoři. Avšak použitím ICT bez ohrožení mohou například předvídat, jak se nový lék může chovat v lidském těle. To jim umožňuje snížit, i když jistě ne zcela odstranit, i počet pokusů na zvířatech, které by jinak, bez nasazení vyvinutého software v oblasti výpočetní farmakologie musely být skutečně provedeny;
2. se dějí příliš rychle nebo příliš pomalu. Například modely změn globálního klimatu umožní vědcům v oblasti životního prostředí spouštět prediktivní modely na mnoho roků do budoucnosti a zjistit tím, jak by mohlo mít minulé, současné a budoucí lidské chování vliv na klimatické podmínky na Zemi;
3. by mohly být jinak příliš nákladné (například pokusy v laboratoři, v přírodě, ve výrobě lékařských materiálů, nástrojů apod.). Zvláště v oblasti přírodních a technických věd existuje řada experimentů, které vyžadují nákladné vybavení. Užitím ICT některé z nich lze dobře simulovat pomocí vhodně zkonstruovaných software. I když to nenahradí vlastní přístroj, učiteli či studentovi může být poskytnut návod, jak jej ovládat a co od něj očekávat. Např. letecké simulátory jsou dalším dobrým příkladem využití simulačního software jako úsporného způsobu výcviku pilotů. Letecké počítačové simulátory jsou výrazně levnější, než je výcvik ve skutečném letounu, nemluvě o bezpečnosti pilota;
4. jsou řešitelné pouze pomocí ICT. Mnoho problémů v astrofyzice, jako je například formování galaxií, nelze snadno pozorovat a už vůbec je nelze experimentovat. Vytváření unikátních obrazů vesmírných těles pro další pozorování by bez využití ICT byly zcela nemyslitelné[23]. Výpočetní modely, založené na srozumitelné matematice, dovolují astrofyzikům testovat celou řadu parametrů modelů a scénářů při formování galaxií, vesmírných jevů apod. Stejně tak třeba nelze experimentálně ověřit, zda kontejner jaderného reaktoru vydrží roztavení jádra, avšak výpočetní modely to mohou potvrdit nebo vyvrátit.

I když výpočetní modely nemohou v absolutní míře nahradit bádání v laboratořích, stávají se nedílnou součástí celkového vědeckého výzkumu a mohou vlastní laboratorní výzkum významně urychlit a zlevnit.

2.5 Dva pohledy na výpočetní vědu a vědecké výpočty

Existuje mnoho definic výpočetní vědy – většina z nich ji popisuje jako interdisciplinární přístup k řešení složitých problémů, který používá teorie a poznání z různých odvětví vědy (aplikace), informatiky a matematiky. Musíme zdůraznit, že výpočetní věda není totožná s informatikou[24]. Výpočetní věda je více metodologie, která umožňuje studium různých objektů a jevů v různých odvětvích vědy. Stejně jako ostatní tři přístupy k vědě, je to jednak způsob, jak bádat, jednak vědní disciplína sama o sobě.

[23] http://www.zam.fme.vutbr.cz/~druck/

[24] Informatiku chápeme jako vědní obor zabývající se shromažďováním, klasifikací, ukládáním, výběrem a šířením zaznamenané znalosti. Je to obor zabývající se zákonitostmi vzniku, sběru, přenosu, třídění, strukturou, vlastnostmi, zpracováním, ukládáním a vyhledáváním dat a informací, a to převážně se zaměřením na využití informačních technologií. [9]

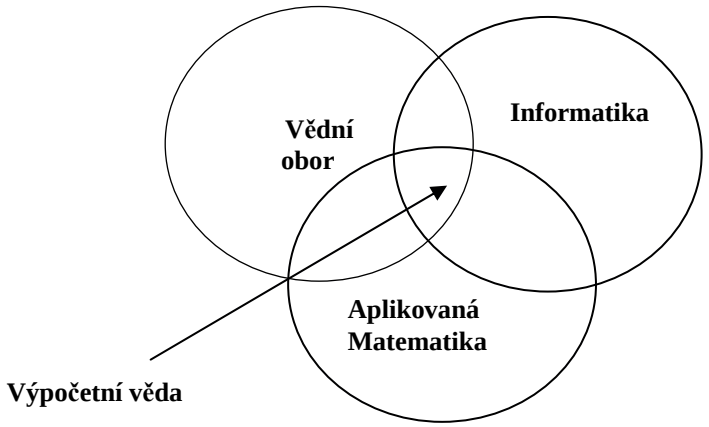
Například zvažme výpočetní vědu v analogii s asociativním zákonem v matematice, kdy je „operováno“ s třemi pojmy – výpočty, věda, výzkum:

1. výpočty, (věda, výzkum);
2. (výpočty, věda), výzkum.

Z pohledu prvního přístupu vědci mohou používat výpočty jako vědeckou metodu bádání. Výpočty se tak používají k provádění vědeckého výzkumu. Z pohledu druhého přístupu vědci však také zkoumají, jak využít výpočetní vědu jako nástroj pro vyšetřování a zkoumání. Proto oba přístupy lze shledat ekvivalentními.

Totéž platí o ostatních třech výše jmenovaných vědeckých přístupech. Někteří vědci používají stávající ICT, techniky a nástroje ve vědeckém bádání u všech přístupů: pozorování, experimentování, teorie, zatímco jiní vědci vyvíjejí nové ICT, postupy a nástroje pro tyto oblasti. Jinými slovy, všechny čtyři výše uvedené typy bádání jsou jak procesy, tak produkty ve velkém schématu vědeckého bádání.

Existují však i jiné způsoby, jak reflektovat výpočetní vědu. Výpočetní věda je často definována jako věda, která je průnikem příslušného vědního oboru, informatiky a matematiky, viz obrázek 2.1.



Obr. 2.1 *Výpočetní věda jako průnik tří různých vědních oborů.*

Z dosud řečeného shrňme: výpočetní věda není totožná s informatikou (která se zabývá vytvářením ICT, tj. software a/nebo vývojem nového hardware). Spadá do ní modelování a počítačová či numerická simulace, které se zaměřují na vytváření a používání výpočetních modelů pro určité metody pozorování, provádění experimentů a vytváření nebo testování nových teorií. Tyto vysoce náročné výpočty nebo výzkumy jsou v současnosti prováděny na velkých, velmi výkonných počítačích, výpočetních gridech, cloudech nebo superpočítačích. Mnoho současných a budoucích výzev[25] k řešení vědeckých problémů bude výrazně závislé na aplikaci této nové metodologie.

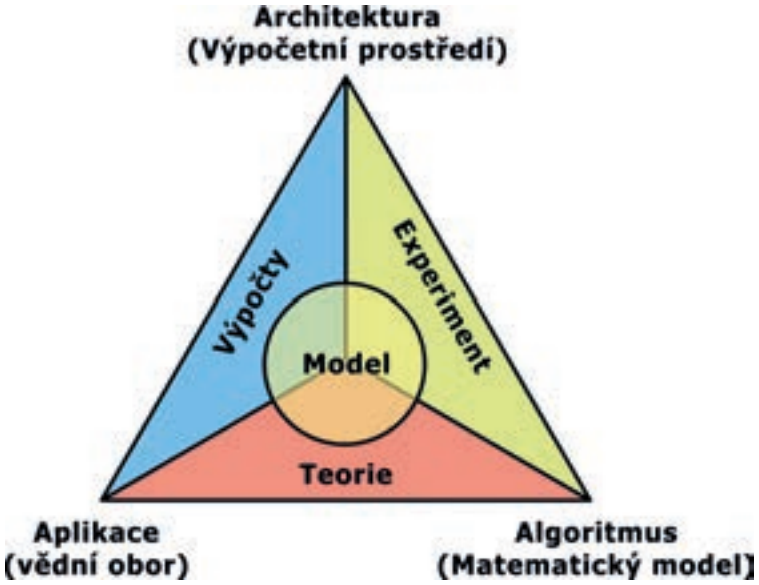
Jiný způsob chápání výpočetní vědy je pohled ze tří souvisejících hledisek***: aplikace, algoritmus*** a ***architektura.***

Aplikace popisuje konkrétní řešený vědecký problém v daném odvětví vědy. Například v biologii můžeme uvažovat aplikaci pro kategorizace všech genů v lidském genomu, v meteorologii aplikaci pro předpovídání počasí aj. Aplikaci lze popsat pomocí algoritmu většinou založeném na matematickém modelu. K jeho realizaci použijeme architektury, v tomto případě ICT, tj. hardware a software, která umožní vytvořit program pro výpočet i velmi komplikovaných problémů.

[25] Výzvou se zde rozumí obecný termín odkazující se na problémy, jejichž řešení je velmi obtížné.

Na obrázku 2.2 je zobrazen tento pohled na výpočetní vědu. Výsledkem je ***výpočetní model***, který zahrnuje experiment, teorii a výpočty. S výpočetním modelem, můžeme opět provádět numerické tzv. „What if“[26] experimenty, testovat matematické aproximace problému a různé teorie a studovat objekty a jevy, které je obtížné, ne-li nemožné, studovat pomocí pozorování nebo experimentálně.

Výzvou k bádání pro vědce ve výpočetní vědě je často vyskytující se skutečnost, že buď celý algoritmus není pro konkrétní případ znám, anebo že je tak složitý, že i moderní superpočítače nejsou schopny takový algoritmus pro řešení problémů akceptovat.



Obr. 2.2 *Výpočetní model jako průnik tří hledisek.*

Příkladem prvního problému je předpověď počasí. U numerické předpovědi počasí mají vědci povědomí o procesech, které se dějí v atmosféře, ale nemají již k těmto představám vyvinuté správné algoritmy. Výpočetním meteorologům se daří modelování velkých účinků počasí v horních vrstvách atmosféry. Avšak pro místní povětrnostní podmínky, které občany nejvíce zajímají, to zatím nedokáží zpracovat, protože odpovídající algoritmy jsou matematicky příliš složité.

Příkladem druhého problému je řešení Schrödingerovy rovnice[27], ke kterému jsou vyvinuté algoritmy a příslušný software. I současné superpočítače schopné řešit tiliony výpočtů za sekundu však mají stále problém s numerickým výpočtem řešení Schrödingerovy rovnice pro systémy s více atomy.

V dalším budeme diskutovat tři výše uvedená hlediska výpočetní vědy (aplikace, algoritmy, architektura).

2.5.1 Aplikace

Výpočetní věda je z pohledu aplikace podporována koncepty a teoriemi z oblasti matematiky (algoritmy) a informatiky (architektura). Pro jakýkoli problém ve výpočetní vědě se

[26] What-if analýza, tj. analýza „Co se stane, když“ je používaná většinou při numerické simulaci sloužící k odhalení, jak je model citlivý na odhad vstupních parametrů, jež by měly ležet v nějakém vhodném intervalu. [2]

[27] http://cs.wikipedia.org/wiki/Schr%C3%B6dingerova_rovnice

vědou samou o sobě stává otázka: „Co je ve vědecké události nebo problému zajímavé?“, anebo při zkoumání složitého systému: „Jaké má hranice, jaké elementy nebo faktory jsou jeho součástí, které z nich lze, resp. nelze eliminovat, jaké předpoklady je třeba stanovit pro popis jeho chování, co víme o jiných systémech, které s daným problémem mají určitou podobnost, již máme zájem studovat, tj. jaký je cíl našeho bádání, jaké máme k dispozici dostupné prostředky k jeho realizaci apod.?“

Aplikace tedy popisuje problém, který chceme řešit. Uveďme některé příklady aplikací v několika vědních oborech:

- chemie: určení elektronové struktury látky, studium dynamiky chemických procesů;
- biologie: studium genetiky a genomiky;
- fyzika: problémy v astrofyzice a kosmologii, vlastnosti elementárních částic;
- matematika: modelování ve výpočetní geometrii, teorie čísel;
- medicína: výpočetní epidemiologie, počítačově podporované navrhování léků, genetika;
- věda o životním prostředí: modelování kvality ovzduší a numerické předpovědi počasí.

2.5.2 Algoritmus

Aplikace nebo řešený problém dovoluje výzkumnému pracovníkovi přejít k další části úkolu: nalézt vhodný algoritmus (tj. stanovit postupné kroky k řešení problému podle výše jmenovaných pravidel). Ve výpočetní vědě je algoritmus reprezentován matematickým modelem, který může být vytvořen tak, aby modeloval chování systému s využitím jeho parametrů. Vědci ve výpočetní vědě musejí často použít jednu nebo více numerických metod [3] nebo tzv. numerických „receptů“[28] [9], aby začali řešit vytvořený matematický model. Mnoho numerických metod je příliš složitých k tomu, aby přibližné řešení bylo možné spočítat jedním výpočtem. Vyžadují nejčastěji opakované výpočty (iterace), aby dospěly k alespoň nějakému vhodnému řešení. Toto použití opakovaných výpočtů vedoucích k „zpřesnění“ přibližného řešení je známé v matematice jako konvergence. Takový postup je často používán ve vědeckých výpočtech při řešení nelineárních problémů.

Musíme zvolit i vhodné ICT, abychom implementovali vytvořený algoritmus nebo matematický model na vhodný počítač. Výpočetní vědci potřebují vědět, jak vybrat vhodné ICT, tj. architekturu pro implementaci algoritmu, a vytvořit tak výpočetní model.

Ve výpočetní vědě je výzkum zaměřen rovněž na vývoj teorií, algoritmů a nástrojů týkajících se zpracování dat, získávání informací a integraci informací na podporu rozhodování v reálném čase. Algoritmy a matematické metody užívané ve vědeckých výpočtech mají různý charakter [3], [10], [11], [12].

Poznamenejme, že komplexní proces: aplikace – algoritmus – architektura budeme ilustrovat na jednoduchých problémech v dalších kapitolách, kde bude nezbytné uvažovat i vliv výpočetního prostředí.

Vzhledem k tomu, že reálná čísla v počítači jsou uložena v pohyblivé řádové čárce [3], nemusí (při provádění aritmetických operací) z důvodu zaokrouhlovací chyby u mantisy zůstat zachována přesnost výpočtu. Zaokrouhlovací (round-off) chyba vzniká „přirozeně“ v závislosti na uložení čísla v počítači, podrobněji viz [3].

Jednou ze současných výzev výpočetní vědy je rozvíjení schopnosti „***vidět svět jako matematický algoritmus***“, který lze implementovat ve výpočetním prostředí. Výpočetní vědci přemýšlejí o zkoumaném vědním oboru (biologie, chemie, fyzika, mechanika atd.) v intencích

[28] http://www.nr.com/

matematické terminologie. Je to zcela odlišný způsob myšlení, který vychází při řešení problémů z pozorování nebo experimentů.

Celosvětově podporovaným trendem je zajištění podmínek, aby budoucí vědci, matematici a inženýři měli příležitosti rozvíjet toto matematické a počítačové myšlení a dovednosti v celém komplexu. Typickým představitelem těchto snah je SIAM - pracovní skupina pro výuku vědeckých výpočtů (*SIAM Working Group on CSE Education*)[29], která doporučuje, aby kromě základů v matematice a informatice měli absolventi vysokých škol v oboru vědeckých výpočtů důkladné vzdělání v aplikační oblasti (např. v některé technické nebo přírodovědné disciplíně). Takovým absolventům by získané matematické znalosti měly stačit k modelování technických a vědeckých problémů. Znalost informatiky, a zejména numerických algoritmů, navrhování software a vizualizace, jim umožňují efektivní využití počítačů. Absolventi vědí, jak najít a využít software (softwarové balíky) pro řešení určitého úkolu. Studenti v oboru vědeckých výpočtů mají zaměřeny bakalářské a magisterské práce interdisciplinárně v matematice, informatice a aplikační oblasti. Je žádoucí, aby absolvent v oboru vědeckých výpočtů byl vyškolen pro komunikaci v týmu s inženýry nebo fyziky a/nebo počítačovými vědci nebo matematiky pro řešení složitých praktických problémů.

2.5.3 Architektura

Jakmile je vytvořen algoritmus, lze používat širokou paletu výpočetních nástrojů (ICT), abychom generovali jeho výstup pro nejrůznější identifikované podmínky. Výpočetní model umožňuje provádět počítačové simulace, tj. provádět „what-if“ analýzy a scénáře změnou vstupních parametrů a počátečních podmínek pro studované jevy nebo objekty.

S pokračujícím vývojem ICT se musí přizpůsobit také použití architektury a navrhování její koncepce (nebo vzorů) k využití jejich nových vylepšení. Tyto nové ICT musejí být škálovatelné (tj. použitelné v řešení rozsáhlých komplexních problémů, schopné zpracovat rozsáhlé množství dat) a přesné (tj. z důvodu možnosti dostatečně přesně předvídat a odhalit jevy a objekty, o něž se zajímáme). Další důležité otázky, které je třeba zvážit pro použití vhodné architektury v souvislosti s přechodem počítačových výrobců na vícejádrové procesory, jsou programování, programovací jazyky a jejich kompilátory, plánování a přidělování zdrojů v reálném čase a vývoj flexibilního softwarového prostředí.

V oblasti vizualizace řešení problémů lze využít specifický výzkum v odvětvích diskrétní matematiky, výpočetní geometrie, robustních geometrických výpočtů, teorii grafů, geometrie těles, interaktivní grafiky, dvou a třírozměrných vizualizačních nástrojů aj. Zvláštní důraz při použití architektury je nutné klást na to, aby numericky náročné simulace a vizualizace rozsáhlých datových struktur byly rychlejší a výpočetně efektivnější, také i interaktivnější pro své uživatele.

Dnešní softwarové systémy mají jedinečné vlastnosti. Jsou velké, komplexní a decentralizované. Kromě toho současná generace software často pracuje v heterogenním prostředí infrastruktury serverů a mobilních platforem. Při výpočtech na počítačových sítích (*grid*) musejí být kladeny na software vysoké nároky: ***vysoká spolehlivost***, ***přizpůsobivost*** a ***flexibilita*** s měnícím se výpočetním prostředím, stejně jako ***odolnost*** vůči potenciálním útokům hackerů.

Výzkum ve výpočetní vědě vede na návrh, vývoj a ověřování robustních a odolných software, a to zejména pro mobilní platformy. Další generace software musejí poskytnout plnou odolnost proti chybám, musejí se bránit účinně proti zneužití a útokům a musejí být schopny „léčení“ a „zotavení“ z chyb v zájmu udržitelnosti vývoje informačních služeb [13].

[29] http://www.siam.org/students/resources/report.php

2.6 Výpočetní simulace

Výpočetní (nebo počítačovou či numerickou) simulací ve vědeckých výpočtech rozumíme běh aplikačního programu, pomocí kterého simulujeme chování abstraktního modelu konkrétního systému. Tradičně se formální modelování systému provádí prostřednictvím matematického modelu, který se pokouší nalézt analytické řešení problému. To umožňuje predikovat chování systému na základě jeho parametrů, počátečních, okrajových podmínek. Výpočetní simulace je často užívána jako doplnění nebo nahrazení modelovaného systému, u něhož nalezení uzavřené analytické formy řešení není možné.

Existuje mnoho typů počítačové simulace. Společným rysem, který všechny simulace sdílejí je výpočetní experimentování, v němž se vygeneruje vzorek reprezentativních scénářů pro zkoumaný model, který se propočítá. Vygenerovat úplný počet všech možných stavů modelu, spočítat je, je často nemožné.

Výpočetní simulaci dělíme vzhledem k různým kritériím na:

- ***stochastickou*** (užívající generátorů náhodných čísel a metodu Monte Carlo) nebo ***deterministickou***. Stochastická simulace je typicky využívána pro diskrétní systémy, kde se události vyskytují s určitou pravděpodobností a nemohou být popsány přímo např. pomocí diferenciálních rovnic. Jevy v této kategorii zahrnují např. genetický drift, biochemické nebo genové sítě s malým počtem molekul. Deterministické simulační modely se snaží explicitně reprezentovat základní mechanizmus a typicky sestávají z vzájemně propojených systémů diferenciálních a diferenčních rovnic. Používají se v environmentálních, technických a společenských vědních oborech, jako jsou globální změny klimatu, deformace materiálu a ekonomické prognózy;
- ***statickou a dynamickou*** (v závislosti na čase). Ve statické simulaci se používají rovnice definující rovnovážné vztahy mezi prvky modelovaného systému. Hledají se podmínky, při nichž je sledovaný systém v rovnováze. Tento způsob je často využíván při simulaci dynamických fyzikálních systémů, aby se zjednodušila sama primární simulace, než se přistoupí k dynamice simulace;
- ***spojitou a diskrétní***. Spojitá dynamická simulace je řešena většinou pomocí aplikačních programů pro numerické řešení diferenciálně algebraických rovnic nebo diferenciálních rovnic (parciálních nebo obyčejných). Speciálním typem diskrétní simulace, která nezávisí na modelu s příslušnou rovnicí, je „agent based“ (zprostředkovaná) simulace. V této formě simulace nejsou zahrnuty individuální entity modelu (např. molekuly, buňky, stromy nebo zákazníci) a reprezentovány přímo, nýbrž zprostředkovaně (např. jejich hustotou nebo koncentrací). Vnitřní stav modelu je řízen pomocí množiny chování nebo pravidel, která určují agentův (zprostředkovatelův) stav, který je aktualizován od jednoho časového kroku k následujícímu;
- ***lokální a distribuovanou*** (je řešena na počítačových sítích a prostřednictvím Internetu).

3 Stručná historie vědeckých výpočtů

Vědecké výpočty se zabývají vytvářením matematických modelů a algoritmů s využitím ICT k analýze a řešení specifických vědeckých problémů. V praxi je pro ně/ni typické použití modelování a počítačové simulace a dalších forem výpočtů pro zkoumané problémy v různých vědních oborech. V této kapitole stručně shrneme vývoj vědeckých výpočtů (i výpočetní vědy) od jejich počátku až do současnosti se zaměřením na klasické ICT.

3.1 Vývoj v zahraničí

Počátek vědeckých výpočtů lze datovat od 40. let minulého století v souvislosti s vývojem balistických raket a atomových zbraní. Počítače ukázaly své možnosti a potenciál pro využití jak obvyklých, tak nově vyvinutých matematických metod, které vyžadovaly náročné výpočty. Již v roce 1943 američtí vědci J. P. Eckert a J. V. Mauchly vyvinuli první programovatelný elektronkový počítač ENIAC, který byl využíván k výpočtu balistických drah raket, atomové a jeho následník pak i vodíkové bomby, ve výzkumu pro návrhy draků letadel ve větrném tunelu a při určování předpovědí počasí. Tento vývoj byl financován jako součást válečného úsilí Spojených států amerických (USA).

Vědecké výpočty se i nadále nejvíce rozvíjely v USA, kde měly podporu americké vlády (tato důvěra pramenila z prokázané užitečnosti počítačů ve druhé světové válce). Národní vědecká nadace USA (*NSF – National Science Foundation*)[30] začala v roce 1950 financovat výzkum, který umožnil zásadní vývoj v této oblasti. V té době firma IBM vyvinula model IBM 7030 prvních paralelních počítačů, které překonávaly problémy s operacemi v operační paměti pomocí tzv. *pipeliningu*[31]. V roce 1964 vyvinul geniální konstruktér Seymour Cray vektorový počítač Cray-1 se špičkovým výkonem 133 MegaFlops[32], který jako první použil funkční víceprocesový paralelismus. Tyto počítače byly sestaveny speciálně pro náročné vědecké výpočty při použití tehdy nejmodernějších, ale velmi drahých technologií. Jejich složitá architektura byla náročná na chlazení a samy počítače byly velmi drahé. Například Cray-1 stál více než 8,8 miliónů dolarů. Existovaly však i jiné americké vektorové počítače, například vyvinuté firmou CDC (počítač CDC 6600 rovněž vyvinul Seymour Cray předtím, než se osamostatnil a založil vlastní firmu), dále takzvané minisuperpočítače firem jako CONVEX, KSR a Alliant.

V letech 1970 až 1980 se výpočetní věda zasloužila o superpočítačovou revoluci (vznikly nové vysoce výkonné, víceprocesorové a paralelní počítače) a začala již mít dopad do celé řady dalších vědních oborů, do kosmického a leteckého průmyslu. Sehrála například významnou roli při komercionalizaci návrhů supersonických letadel. Byla to opět americká nadace NSF, která prostřednictvím svého úřadu pro náročné vědecké výpočty financovala od roku 1984 vytvoření pěti superpočítačových center (San Diego, Illinois, Pittsburgh, Cornell a Princeton), která měla stěžejní úlohu na poli náročných výpočtů, počítačové vizualizace, grafiky a vývoje sítě NSFNET. Tato síť urychlila tempo dalšího technického zdokonalo-

[30] http://www.nsf.gov/

[31] Pipelining (proudové zpracování) - technika práce procesoru koncipovaná tak, aby v době, kdy jedna část procesoru provádí určitou fázi jedné instrukce, mohla jiná část procesoru pracovat na jiné fázi jiné instrukce.

[32] Výkon počítačů pro náročné výpočty se měří ve Flops (*Floting point operations per second*), tedy v počtu operací v pohyblivé řádové čárce za sekundu. Používají se násobné předpony soustavy SI: Mega=10^6, Giga=10^9, Tera=10^{12}, Peta=10^{15}.

vání při propojování novějších rychlejších superpočítačů prostřednictvím novějších rychlejších linek, rozšiřovaných a zdokonalovaných znovu a znovu v letech 1986, 1988, 1990,…

Koncem osmdesátých let existovalo více než deset v podstatě výhradně amerických firem, které nabízely superpočítače založené na myšlence paralelních výpočtů a poskytujících již v té době tisíce a výjimečně i desítky tisíců procesorů. Prudký růst výkonu „běžných" procesorů (zejména pak procesorů rodiny x86) však způsobil, že žádná z těchto firem nepřežila devadesátá léta a systémy s podobným i vyšším počtem procesorů se objevovaly opět až v posledním desetiletí dvacátého století, kdy se prakticky zastavil dříve zcela pravidelný dvojnásobný růst výkonu procesorů realizovaný každých osmnáct měsíců (Moorův zákon).

V devadesátých letech nastal rozkvět výpočetní vědy a prudký rozvoj paralelních výpočtů, včetně rozšíření počítačů a jejich výpočetní mohutnosti, vývoje v jejich nasazení a důležitosti. Výpočetní věda se tak stala intelektuální disciplínou, výkonnou a nepostradatelnou analytickou metodou. Výroba superpočítačů japonskými firmami NEC, Hitachi, Fijitsu a německou firmou Siemens-Nixdorf vytvořila konkurenční prostředí, a tím jisté zlevnění superpočítačů. Ve druhé polovině devadesátých let dokonce vytlačily z trhu vektorové modely firmy Cray, v té době dočasně vlastněné firmou SGI.

Vědci na univerzitě v Tokiu sestavili v roce 1996 tehdy nejrychlejší počítač GRAPE-4 určený pro vědecké výpočty. Počítač dosáhl 1,08 Tflops (trilión operací v pohyblivé řádové čárce za sekundu). Japonští vědci Junichiro Makino a Makoto Taiji na něm provedli simulaci komplexních interakcí mezi astronomickými objekty, jako jsou hvězdy a galaxie. Tento typ simulace se označuje jako problém „*N* těles", protože chování každého z ***N*** objektů je ovlivněno všemi ostatními objekty. Tento problém lze řešit pouze numericky. Takový výpočet byl velmi náročný na strojový čas. Počítač GRAPE-4 byl 100krát rychlejší než nejrychlejší počítače v předchozích deseti letech. Dosud se pro problém ***N*** těles používaly heuristické efektivní algoritmy. GRAPE-4 používal 1692 procesorů, z nichž každý dosahoval rychlosti 640 Mflops. Tokijští vědci chtěli sestavit počítač s rychlostí 10^{15} operací za sekundu, v němž by bylo použito 20 tisíc procesorů s rychlostí 50 Gflops.

USA v té době zavedly celní opatření, kterým prakticky zabránily dovozu superpočítačů z Japonska. To způsobilo v USA postupné zastarávání superpočítačů, a vedlo tak zdánlivě paradoxně k širšímu rozšíření superpočítačů ve světě (japonské firmy ztrátu amerického trhu musely kompenzovat přiměřeně agresivní prodejní politikou jak v samotném Japonsku, tak zejména v Evropě). Navíc dostupnost vysokorychlostních sítí zjednodušila vzdálený přístup k superpočítačům a mnoho výpočtů je od té doby prováděno v jiných zemích než v těch, kde žije vědec takovéto výpočty provádějící.

USA samozřejmě na tento vývoj reagovaly podporou vývoje alternativních superpočítačů, například heterogenní paralelně vektorový počítač SV1 (a hlavně budoucí SV2) firmy SGI (tyto systémy byly založeny na technologii, kterou SGI získala koupí firmy Cray). V té době se za velký paralelní[33] počítač považovaly systémy s více než 64 procesory. Označení superpočítač tak získalo mnohem více počítačů. Cena velkých superpočítačových systémů a současně růst propustnosti lokálních sítí však v devadesátých letech vedly k vývoji jiného typu systémů, které se postupně začaly rovněž označovat jako superpočítače. Šlo o stovky osobních počítačů či pracovních stanic propojených lokální sítí tvořící tzv. počítačový cluster (dříve se bylo možné setkat i s pojmem „počítačová farma"). Mezi první takový počítač patřil systém Beowulf[34], vyvinutý profesorem Thomasem Sterlingem.

[33] Masivně paralelní je více než 64 procesorový počítač s distribuovanou pamětí, v němž má každý procesor přímý přístup jen ke své paměti a o data z paměti jiného procesoru musí požádat, nejčastěji formou zprávy

[34] http://www.beowulf.org/

V roce 1997 NSF změnila podporu financování superpočítačových center v USA na financování Partnerství pro rozvinutou výpočetní infrastrukturu (PACI) pro aliance více než padesáti akademických pracovišť v USA. Cílem bylo vybudovat prototyp další generace informační a výpočetní infrastruktury pro vysoce náročné výpočty, a současně explicitně zapojit existující i budoucí uživatele takové infrastruktury.

Za vládní podpory byl v USA od roku 1998 do 2004 realizován projekt pro vývoj masivně paralelních superpočítačů na bázi standardních procesorů firem Intel, IBM a SGI. Jsou to např. superpočítače IBM ASCI White, Intel ASCI Red, IBM ASCI Blue-Pacific, SGI ASCI Blue Mountain. Největší z těchto systémů mají více než sto tisíc procesorů, jejich synchronní efektivní využití je velmi složitý problém, V rámci projektu PACI byla proto intenzivně podporována snaha vyvinout efektivní výpočetní modely pro určité konkrétní úlohy a získat z masivně paralelního superpočítače více než obvyklých 10 % teoretického instalovaného výkonu (jen pro ilustraci složitosti problému – vývoj jednoho algoritmu pro řešení konkrétních geofyzikálních problémů - tedy ne pro řešení triviální úlohy, který lineárně škáluje, tj. je stále urychlován přidáním procesorů až do 150 tisíc procesorů, trval zhruba dvacetičlennému týmu dva roky). V roce 2005 IBM rozšířila Blue Gene rodinu počítačů o Blue Brain se snahou o vytvoření výpočetního modelu nervového chování v neokortexu, což je nejsložitější část mozku.

V roce 2007 byl spuštěn v CERNu *Large Hadron Collider*[35] největší světový urychlovač částic. Jím generovaná data požadují vyšší výpočetní výkon, než byla žádána kdykoli předtím.

V současné době je nejvýkonnější počítač K, zkonstruovaný japonskou firmou Fujitsu, který spojuje 68 544 SPARC64 VIIIfx procesorů, každý s osmi jádry, v celkovém počtu 548 352 jader, což je téměř dvakrát tolik jako jakýkoli jiný systém v žebříčku superpočítačů TOP500 v roce 2011. Počítač K je také silnější, než je následujících pět systémů na kombinovaném seznamu počítačů TOP500.

Žebříček TOP500[36] nejvýkonnějších počítačů světa je vydáván již dvacet let dvakrát ročně při příležitosti konference ICS[37] (*International Conference on Supercomputing*), která se koná každoročně v červnu v Německu, a konference Supercomputing, která se koná rovněž každým rokem v listopadu v USA. Pořadí počítačů je primárně stanoveno podle jejich výkonu při řešení konkrétních problémů lineární algebry. Žebříček vydávají univerzity (univerzita v Mannheimu a univerzita v Konxwille, Tennessee, ta ve spolupráci s Oak Ridge National Laboratory) na základě dat, která poskytnou majitelé (provozovatelé) počítačů (v žebříčku proto chybí některé počítače využívané armádami apod., neboť v jejich případě není žádoucí, aby tyto systémy byly veřejně známy). Státy jako Japonsko a USA, v poslední době i Čína, pečlivě sledují „svou“ pozici a soupeří o to, kdo vlastní počítač, který stojí na „úplné špičce“.

Tato pozice se velmi rychle, mění. Ještě v listopadu 2010 byl na prvním místě čínský superpočítač využívající k enormnímu výkonu grafických akcelerátorů. V červnu 2011 se však na první místo dostal zmíněný japonský systém. Lze očekávat, že i ten bude velmi brzy nahrazen. Evropa se svými superpočítači pohybuje v první desítce tohoto žebříčku.

V roce 2011 poprvé všech deset nejvýkonnějších superpočítačů dosáhlo výkonu nad několik petaflopů (10^{15} operací v pohyblivé řádové čárce za sekundu). Spojené státy mají pět systémů s tímto výkonem, Japonsko a Čína po dvou a Francie jeden.

Ve světě už vývoj pokračuje dál. Po dosažení petaflopu se vyvíjí nová generace superpočítačů řádu Exascale (překročení exaflopu čili výkonu v řádu milionů teraflopů, 10^{18} operací za sekundu). Výzkum financuje americká armádní agentura DARPA. Výsledky očekává při-

[35] http://press.web.cern.ch/public/en/LHC/LHC-en.html
[36] http://www.top500.org/
[37] http://ics-conference.org/

bližně v roce 2018. Technický ředitel SGI Lim Goh k tomuto vývoji říká[38]: *„**Když se podíváte na vývojářské priority pro Exascale, obecně jde hlavně o vyřešení otázek napájení, chlazení a odolnosti proti výpadku komponenty. Je také potřeba využít chytřejších propojovacích architektur – musíme si uvědomit, že v Exascale již hovoříme až o miliardách paralelně vykonávaných a spolupracujících vláknech výpočtu**.“*

3.1.1 Virtuální superpočítač World Community Grid

V listopadu roku 2004 byl spuštěn virtuální superpočítač World Community Grid[39], což je největší humanitární výpočetní síť světa, která používá nevyužitý výpočetní výkon soukromých a firemních počítačů při řešení nejobtížnějších společenských problémů světa. Po celém světě se používá více než miliarda počítačů, z nichž každý se může potenciálně připojit ke světovému veřejnému gridu. Výpočetní sítě jsou rychle se rozvíjející technologií, která spojuje výkon tisíců a miliónů jednotlivých počítačů do mohutného virtuálního systému s velmi vysokou výpočetní kapacitou. Technologie gridu umožňuje dosahovat takového výpočetního výkonu, který mnohem více přesahuje možnosti největších superpočítačů světa.

Společnost IBM poskytla své informační technologie a prostředky k provozu projektu Help Defeat Cancer[40] (Pomozte porazit rakovinu) v síti World Community Grid. Na tomto projektu, který uplatní možnosti superpočítačů v boji s rakovinou, spolupracuje od roku 2006 s vědci z Univerzity lékařství a stomatologie New Jersey – Lékařské školy Roberta Wood Johnsona a Institutu rakoviny v New Jersey. Help Defeat Cancer je třetím projektem, který využije výpočetní výkon virtuálního superpočítače World Community Grid. Projekt poskytne vědeckým pracovníkům příležitost simultánně analyzovat velké počty mikroskopických vzorků rakovinných tkání (TMA), a tak umožní za kratší dobu uskutečnit větší množství experimentů. Pro lepší představu World Community Grid umožňuje projektu Help Defeat Cancer analyzovat za jediný den takový počet vzorků, na jejichž analýzu by běžný počítač potřeboval zhruba sto třicet let.

Poznámka: Nevyužitý čas svého počítače může prostřednictvím projektu World Community Grid darovat každý, kdo si z jeho webových stránek stáhne bezplatný software a zaregistruje se. Více než tři sta tisíc jednotlivců nyní prostřednictvím sítě World Community Grid poskytuje výkon zhruba čtyřem stům tisícům počítačů pro rozvoj výzkumu rakoviny. K síti World Community Grid se mohou připojit počítače s operačními systémy Windows, Linux i Mac OS.

3.1.2 Berkeley Open Infrastructure for Network Computing

Berkeley Open Infrastructure for Network Computing[41] (BOINC) je open source software pro distribuované výpočty. Umožňuje provozovat projekty, jako je např. SETI@Home[42], jenž využívá internetového připojení počítačů pro hledání mimozemské inteligence (SETI). Tohoto projektu se může zúčastnit každý uživatel, bude-li používat open source software, který stahuje a analyzuje data z radiového teleskopu. Existuje malá, ale významná pravděpodobnost, že právě počítač připojeného uživatele zachytí slabý šum mimozemské civilizace. SETI@Home je jedním nejznámějších projektů využívající BOINC.

[38] http://www.sgi.com/global/cz/press/news/110722.html
[39] http://www.worldcommunitygrid.org/
[40] http://pleiad.umdnj.edu/%7Ewill/IBM/index.html
[41] http://boinc.berkeley.edu/
[42] http://setiathome.berkeley.edu/

BOINC je vyvíjen skupinou pocházející z kalifornské univerzity v USA. BOINC je navržen jako otevřená struktura pro každého, kdo se chce zapojit do projektu distribuovaných výpočtů. Záměrem BOINCu je umožnit badatelům různých oborů, například molekulární biologie, klimatologie nebo astrofyziky, jednoduchý přístup do výpočetní sítě osobních počítačů na celém světě s velmi vysokým výkonem. BOINC je založen na myšlence, která předpokládá, že naprostá většina počítačů na světě „běží“ nevyužita. Moderní operační systémy dokáží tento nevyužitý výkon spotřebovat, aniž by došlo k výraznému zpomalení aplikací, které uživatel používá. Většina projektů využívajících BOINC je neziskových a závisejí, pokud ne zcela, tak převážně na dobrovolnících[43]. To ale neznamená, že BOINC nemůže být použit pro zisk. BOINC sestává ze serveru a klientů, kteří spolu komunikují při distribuci pracovních jednotek. Každý klient pak jednu jednotku zpracuje a vrátí ji server, aby si posléze vyžádal další.

Z různých statistických důvodů BOINC zahrnuje i systém ohodnocení uživatelů. Pro ohodnocování uživatelů se používá jednotka ***cobblestone*** pojmenovaném po Jeffu Cobbovi z projektu SETI@Home. Uživatel získá sto cobblestonů, pokud jeho počítač s následujícími parametry pracoval jeden den: jedna miliarda operací za sekundu s čísly v pohyblivé řádové čárce (založeno na benchmarku Whetstone[44]); jedna miliarda operací za sekundu s celými čísly (založeno na benchmarku Dhrystone[45]). V případě, že má uživatel rychlejší počítač, cobblestony mu přibývají rychleji a naopak.

Tyto dobrovolné aktivity jsou založeny na důvěře těm, kdo poskytují své počítače. Systém hodnocení uživatelů však ukázal, že pro jisté skupiny i jednotlivce je důležitější dostat se do čela pomyslných žebříčků, než skutečně poskytovat nějaké hodnotné výsledky. Programy využívající systém BOINC a takové, které se opírají o dobrovolné poskytování zdrojů, proto musejí počítat s takovým nekorektním (podvodným) chováním. Standardním způsobem ochrany je provádění stejných výpočtů na několika zcela nezávislých systémech. Distribucí úloh se musí zajistit, že jeden počítač a nejlépe ani skupina nějak blízkých počítačů bude počítat vždy jen jednu variantu. Srovnáním výsledků lze poměrně snadno odhalit problém a nedůvěryhodné výsledky vyloučit.

3.1.3 Koordinované gridy

Kromě výše uvedených snah využít „spící“ kapacitu primárně osobních počítačů zapojených do sítě se v posledních více než deseti letech vyvíjí úsilí vytvořit spolehlivější řízenou distribuovanou infrastrukturu, která bude poskytovat garantované výsledky (kde nebude např. nutné provádět výpočty opakovaně pouze z důvodu nedůvěry majitelům počítačů). Součástí těchto aktivit je vývoj speciálního programového vybavení – middleware, který „skryje“ složitost distribuované infrastruktury. Umožní uživateli s ní pracovat jednotným způsobem bez ohledu na to, kde se fyzicky nachází konkrétní počítač či cluster, který uživatelova úloha nakonec využije (to v ideálním případě určuje infrastruktura, nikoliv uživatel). V USA probíhal a stále probíhá vývoj middleware Globus. Ve státech Evropské unie (EU) je to pak primárně middleware gLite. Je vyvinut v rámci série projektů EGEE a dnes dále rozvíjen v rámci projektu EMI. Tento middleware je využíván v největším takto koordinovaném gridu, který je spravován v rámci projektu EGI InSPIRE. Tento grid dnes propojuje více než dvě stě tisíc jader. Zpřístupňuje úložnou kapacitu v řádu stovek petabytů a každodenně se na něm spočítají stovky tisíc úloh.

[43] http://petrdrhlik.webzdarma.cz/veda_a_technika/distribuovane-vypocty/strucne-popisy-jednotlivych-projektu-boinc.htm
[44] http://freespace.virgin.net/roy.longbottom/whetstone.htm
[45] http://en.wikipedia.org/wiki/DMIPS

3.2 Vývoj v České republice

I když tým profesora Antonína Svobody[46] vyvinul v roce 1956 první český samočinný počítač SAPO na Ústav matematických strojů ČSAV a následně elektronkový počítač EPOS 1 a později v roce 1961 tranzistorový počítač EPOS 2, tak výkonná výpočetní technika se nakupovala v sedmdesátých a osmdesátých letech minulého století nekoordinovaně především v zahraničí (USA, Velká Británie, Německo, Francie), neboť počítače vyvíjené v zemích RVHP byly velmi málo spolehlivé. Zpravidla počítače ani výkonem neodpovídaly počítačům, které byly k dispozici mimo země RVHP.

Situace se změnila po roce 1989. Ovšem teprve v roce 1994 se podařilo zkoordinovat snahy vysokých škol a Akademie věd (AV) ČR. Pod hlavičkou Fondu rozvoje vysokých škol (FRVŠ) vznikl projekt na vybavení akademické a výzkumné komunity v ČR skutečně výkonnou výpočetní technikou. Do tohoto projektu byla pod koordinací Masarykovy univerzity (MU) v Brně zapojena většina velkých vysokých škol, zejména Univerzita Karlova (UK) v Praze a České vysoké učení technické (ČVUT) v Praze, Západočeská univerzita (ZČU) v Plzni, Vysoké učení technické (VUT) v Brně, Univerzita Palackého (UP) v Olomouci a Vysoká škola báňská – Technická univerzita (VŠB-TU) v Ostravě.

Na všech těchto školách vnikly na jejich centrálních pracovištích skupiny nebo i značně samostatná oddělení věnovaná podpoře, rozvoji a provozu výkonné výpočetní techniky. Takto i na MU v Brně vzniklo již v roce 1994 Superpočítačové centrum Brno (SCB) jako součást Ústavu výpočetní techniky (ÚVT).

Na všech jmenovaných školách (v menší míře i na dalších) tak v průběhu let 1994 až 1998 byly instalovány výkonné výpočetní počítače, Některé z nich se dokonce na jedno období umístily v seznamu TOP500 (např. počítač na ZČU v Plzni, který v době instalace byl na přibližně padesátém místě tohoto seznamu). Další výkonná technika byla instalována i na AV ČR, (např. první a dosud jediný superpočítač Cray, jehož darovaná verze Cray Y několik let ještě sloužila v ČR) a v Českém hydrometeorologickém ústavu (ČHMÚ) v Praze a dalších akademických institucích.

V roce 1996 vznikl pod koordinací MU projekt ***MetaCentrum***[47], který sdružoval výše uvedená akademická pracoviště a s využitím vysokorychlostní akademické sítě vytvořil první distribuované výpočetní prostředí – Grid – v ČR (dokonce z prvních i ve světovém měřítku). MetaCentrum je od roku 1999 součástí aktivit sdružení CESNET[48]. Toto sdružení založily vysoké školy a AV ČR v roce 1996. Jeho hlavním cílem bylo provozovat a rozvíjet páteřní akademickou počítačovou síť ČR. Současná generace této sítě se nazývá CESNET2 a nabízí na páteřních trasách přenosové rychlosti v řádu desítek gigabitů za sekundu. Od roku 2011 je financování sdružení CESNET plánováno z projektů ***Velká infrastruktura CESNET*** a ***Rozšíření národní informační infrastruktury pro VaV v regionech (eIGeR)*** (poslední jmenovaný je financován z Operačního programu Výzkum a vývoj pro inovace[49], VaVaI), které jsou věnovány vybudování komplexní e-infrastruktury. Cílem pětiletého projektu Velká infrastruktura CESNET je rekonstrukce národní výzkumné sítě CESNET2 na tzv. velkou infrastrukturu – široké spektrum zařízení, vybavení, zdrojů, ale i služeb určených pro výzkum. Jeho úkolem je zajištění podmínek pro efektivní spolupráci rozsáhlých vědeckých týmů, a to jak osob, tak experimentálních zařízení, jejichž jednotlivé části mohou být umístěny i v různých zemích. Součástí je tak i další koordinace rozvoje distribuované výpočetní infrastruktury (gridu) a počátek rozvoje úložné infrastruktury v ČR. Projekt eIGeR je komplemen-

[46] http://www.odbornecasopisy.cz/index.php?id_document=36326
[47] http://www.metacentrum.cz/cs/
[48] http://www.cesnet.cz
[49] http://www.msmt.cz/strukturalni-fondy/op-vavpi

tární k projektu Velká infrastruktura CESNET. Zajišťuje zejména rozsáhlé investice do sítě a úložných kapacit v prvních dvou letech realizace velké infrastruktury CESNET.

MetaCentrum provozuje a spravuje vlastní i svěřené výpočetní prostředky a datová úložiště akademických center AV ČR, Jihočeské univerzity (JČU) v Českých Budějovicích, MU v Brně, Mendelovy university (MENDELU) v Brně, UK v Praze, VUT v Brně, ZČU v Plzni atd. Cílem aktivity MetaCentrum (CESNET) je podpora rozvoje vědy a výzkumu v ČR koordinací ***národní gridové infrastruktury*** (NGI), která propojuje výzkumné týmy a instituce v různých regionech ČR a umožňuje jejich napojení do mezinárodního výzkumného prostředí. MetaCentrum vytváří zmíněný národní Grid, v této fázi v podstatě virtuální počítač, který umožňuje efektivní sdílení výpočetní techniky instalované v rámci předcházejících superpočítačových i jiných projektů různých vysokých škol a ústavů AV ČR. Současně umožňuje řešit úlohy v oblasti vědeckých výpočtů, které svými požadavky (na paměť, výkon centrálního procesoru aj.) přesahují možnosti jednotlivých dílčích výpočetních center (uzlů národního gridu).

Hlavní uzly MetaCentra v Plzni, Praze a Brně byly propojeny na „úrovni L2“, tj. vytvořily virtuální lokální síť (VLAN). V ní je možné vytvářet další úroveň virtuálních sítí, které tvoří základ propojení virtuálních clusterů více přizpůsobených aktuálním potřebám uživatelů a jejich skupin. Výpočetní zdroje MetaCentra jsou virtualizovány, což postupně umožní uživatelům spouštět plnohodnotné virtuální počítače, a tak získat prakticky úplnou kontrolu nad výpočetním prostředím, které využívají. MetaCentrum začíná nabízet i tzv. ***Cloud rozhraní*** a ve spolupráci s Centrem CERIT-SC postupně rozvíjí akademickou (výzkumnou) Cloud infrastrukturu v ČR.

Propojení národní gridové infrastruktury do mezinárodního kontextu je realizováno formou zapojení do klíčových projektů EU. Aktuálně je MetaCentrum zapojeno do celoevropského projektu EGI InSPIRE[50], který buduje a provozuje již výše zmíněný světově největší koordinovaný grid EGI. Kromě toho je MetaCentrum zapojeno i do vývoje gridového middleware pro toto prostředí (projekt EMI[51]). Stejně i do skutečně globální spolupráce prostřednictvím projektu CHAIN [52], který je věnován celosvětové harmonizaci gridových a souvisejících infrastruktur (zejména pak v rozvojových a příbuzných oblastech, jako je jižní Amerika, Afrika, Asie včetně Indie a Číny). Na koncepční úrovni bylo nejdůležitější zapojení do projektu EU EGI_DS (*European Grid Initiative Design Study*)[53], kde MetaCentrum hrálo roli koordinátora celého projektu.

3.2.1 Superpočítačové Centrum Brno a CERIT-SC

Jak je zmíněno výše, Superpočítačové Centrum Brno vzniklo v roce 1994 v rámci ÚVT MU v Brně a bylo pověřeno koordinací národních superpočítačově orientovaných aktivit v rámci projektu FRVŠ. Hlavním cílem SCB bylo poskytovat uživatelům z MU přístup k výkonné výpočetní technice umožňující účinně řešit náročné vědecké výpočty z různých oblastí. Kromě běžného provozu spravované infrastruktury se SCB podílelo na jejím dalším rozvoji a růstu výpočetních a datových kapacit, a to jak v rámci ČR, tak i v mezinárodním měřítku. SCB bylo iniciátor MetaCentra a celou dobu představovalo jeho hlavní uzel jak výkonem, tak především odborným zázemím.

[50] http://www.egi.eu/
[51] http://www.emi.eu/
[52] http://www.chain-project.eu/
[53] http://www.egi-eu.eu/

V roce 2011 se SCB transformovalo na Centrum CERIT-SC[54] (CERIT Scientific Cloud), které je národním centrem poskytujícím flexibilní úložné a výpočetní kapacity a zajišťujícím související výzkum a vývoj. CERIT-SC přebírá péči o cloud infrastrukturu a studuje její možnosti (i omezení) pro uspokojení zejména nestandardních požadavků na infrastrukturu vědeckých výpočtů.

Centrum CERIT-SC předpokládá, že ze současných přibližně pěti set padesáti jader a sto padesáti terabyte úložného prostoru, které poskytovalo SCB, bude do roku 2013 instalováno více než tři tisíce jader a více než čtyři petabyte úložného prostoru. Tyto kapacity budou dostupné na otevřeném principu (open access).

Centrum CERIT-SC je součástí národní e-infrastruktury složitého systému vzájemně propojených síťových, výpočetních a úložných kapacit a souvisejících služeb pro výzkumnou komunitu ČR. Tato e-infrastruktura je explicitně popsána ve vládou schválené Cestovní mapě velkých infrastruktur výzkumu, vývoje a inovací a tvoří ji tři organizace, sdružení CESNET, Centrum CERIT_SC a superpočítačové centrum excelence IT4Innovations.

3.2.2 Superpočítačové Centrum excelence IT4Innovations

V rámci rozvoje národní e-infrastruktury je z prostředků Operačního programu Výzkum a vývoj pro inovace budováno v Ostravě jako součást VŠB-TU Ostrava superpočítačové centrum IT4Innovations[55]. Jeho cílem je, aby se ČR objevila na mapě států se skutečně výkonnou superpočítačovou technologií dostupnou akademické a vývojové komunitě. Mezi lety 2012 až 2015 centrum má být vybaveno superpočítači, z nichž nejvýkonnější by se měl v době instalace (rok 2014) umístit mezi padesátým až stým místem na žebříčku TOP500 nejvýkonnějších superpočítačů světa. IT4Innovations je zapojeno do mezinárodní spolupráce PRACE[56], jejímž cílem je v EU instalovat a provozovat superpočítačové systémy nejvyšší kategorie (exascale).

[54] http://www.cerit-sc.cz
[55] http://www.it4innovations.cz/
[56] http://www.prace-project.eu/?lang=en

4 Vývoj a omezení vědeckých výpočtů

V této kapitole se budeme zabývat otázkami, jak se odráží současný vývoj ICT ve vědeckých výpočtech, jaké chyby ve vědeckých výpočtech mohou nastat, jaký vliv bude mít výpočetní architektura a co míníme výpočetní náročností. Odpovědi na tyto otázky mají obecnější platnost než jen pro matematickou biologii, ale platí i pro ostatní vědní obory.

4.1 Výpočetní prostředí

S pokračujícím vývojem ICT se mění použití výpočetní architektury ve vědeckých výpočtech. Výpočetní platformy (statické nebo mobilní) musejí být použitelné pro řešení rozsáhlých komplexních problémů a schopné přijímat, zadávat a vizualizovat rozsáhlé množství dat. Současně musejí být rychlé a přesné, aby dostatečně přesně předvídaly a odhalily v reálném čase jevy, o něž se zajímáme, a identifikovaly je v objektech, které zkoumáme. Další důležité otázky, které je třeba zvážit u použití vhodné výpočetní platformy, jsou přechod na ***vícejádrové procesory*** a související ***paralelizace***, ***mobilní platformy*** (chytré telefony, tablety apod.), kde se mění jak programování, použití operačních systémů, vhodných programovacích jazyků a jejich kompilátorů, a dále rozsáhlé paralelní a distribuovaná prostředí s plánováním a přidělováním zdrojů v reálném čase. Výpočetní platforma tedy není dána jen hardware, ale kombinací hardware a několika vrstev programového prostředí, od operačního systému až po aplikační software.

Jednoduché výpočetní prostředí je tvořeno jedním počítačem. Zatímco v minulosti se dařilo výrobcům procesorů pravidelně zvyšovat výkon samotných procesorů[57] prostým růstem frekvence, na jejíž hladině pracovaly, v posledních letech se tento trend zastavil a frekvence se ustálila (hlavním důvodem je problém s odváděním tepla, kde již nestačí prosté vyzařování a chlazení vzduchem; využití jiných principů je sice možné, ale ekonomicky nereálné – např. chlazení kapalným dusíkem).

Místo změny frekvence jsme proto v posledních letech svědky růstu počtu ***výpočetních jader*** v rámci jednoho procesoru. Současný vývoj jde směrem k integraci více jader, tedy více procesorů do jediného čipu. Tento trend můžeme pozorovat u procesorů pro osobní počítače. Procesory se tedy dělí na jednojádrové a vícejádrové. Zvyšování počtu jader je v podstatě vynuceno fyzikálními omezeními. Integrací většího počtu jednodušších jader je teoreticky možné dosáhnout při stejné výrobní technologii na stejné ploše křemíku mnohem vyšší výpočetní výkon, než použitím jediného složitého jádra. Zatímco ještě před několika lety byly vrcholem dvoujaderné procesory, dnes jsou běžně k dispozici procesory o dvanácti jádrech a v laboratořích je možné se setkat s procesory vybavenými i stovkou jader. To samozřejmě vyžaduje změnu způsobu využití – výkon vícejaderných procesorů řádně využijeme jen tehdy, dokážeme-li skutečně všechna jádra zapojit do řešení problému. Operační systémy i překladače a knihovny proto musejí podporovat přímou paralelizaci programů. Aplikační software pak musí být paralelizován a musí pracovat s více jádry/procesory současně, jak je to ukázáno v sedmé kapitole.

Počet procesorů a jader, které je možné umístit do jednoho počítače, je omezený. Další růst výkonu využitelného pro řešení problémů je proto spojen s propojováním počítačů do vyšších celků pomocí rychlé propojovací sítě. Příkladem jsou *clustery* propojené InfiniBan-

[57] Procesor je v současnosti velmi složitý sekvenční integrovaný obvod umístěný na základní desce počítače.

dem[58], sítí s velmi nízkou latencí přenosu a přenosovou kapacitou v řádu desítek gigabitů za sekundu současně v obou směrech. Hierarchicky ještě vyšší kategorii pak představují gridy, jimž je věnována následující kapitola.

4.1.1 Grid computing

Grid (ve významu mřížka) představuje počítače propojené přes internet. Označení grid vychází z topologických vlastností takového systému, obdobně jako v případě elektrické distribuční sítě (označované odedávna právě jako grid) a jde o systém, který znamená něco více, než je jen prostý součet jednotlivých částí. Jednu z prvních definic pojmu grid (výpočetní grid) vytvořili I. Foster a C. Kesselman [14] v roce 1999: *„**Výpočetní grid je hardwarová a softwarová infrastruktura, která poskytuje spolehlivý, standardizovaný, všudypřítomný a levný přístup ke špičkovým výpočetním službám***“. Další a rozsáhlejší definici přináší např. Grid Café[59].

Jak se později ukázalo, tato definice však nepostihuje skutečně klíčové vlastnosti gridu, neboť opomíjí způsob vytváření gridu. Základním principem je skutečnost, že uživatel (skupina, instituce, …) má vlastní výpočetní prostředky a ty do gridu zapojuje.

Když pak požaduje výpočetní výkon, zadá úlohu do gridu. Ten automaticky rozhodne, kde se výpočet skutečně provede. To může být na vlastních výpočetních prostředcích, které uživatel zapojil, ale stejně tak to může být na počítačích, které jsou na druhém konci světa. Uživatel také tak může (dočasně, resp. občas) čerpat větší výpočetní výkon, než je kapacita jeho vlastních počítačů; naopak v době, kdy je sám nepoužívá, je může poskytnout k dispozici jiným uživatelům.

Grid, resp. jeho programová vrstva (nazývaná gridový middleware) se stará o přijímání úloh. Rozhoduje, kde se spustí, včetně zohlednění toho, jaké a kolik dat bude třeba, a zda tedy nebude vhodné úlohu spustit „poblíž“ (ve smyslu snadné dostupnosti přes počítačovou síť), anebo zda bude výhodnější (efektivnější) přesunout data k vhodným výpočetním prostředkům (případně pak zase výsledky přesunout na určené místo).

Původ gridů lze nalézt v počátcích distribuovaného clusterového počítání. Zatímco clusterová řešení umožňují paralelní zpracování výpočetních úloh, vyrovnávání zátěže jednotlivých individuálních výpočetních zdrojů a toleranci chyb v jedné lokalitě, grid computing vychází z distribuovaného clusterového řešení (tedy situace, kdy disponujeme více clusterů na různých místech), a současně jej povyšuje na novou kvalitativně vyšší úroveň.

Infrastruktura gridu je na rozdíl od superpočítačů a clusterů specifická i tím, že ji vlastní, spravuje a využívá větší počet organizací. Už se nejedná o monolitickou strukturu z hlediska hardwaru a softwaru, ale obecně o infrastrukturu sestávající z různých typů operačních systémů i síťových technologií. Takže grid ve své podstatě může zahrnovat nejrůznější typy systémů: od stolních počítačů až po superpočítače a clustery, úložná zařízení a databáze, senzory i vědecké přístroje.

Grid je v podstatě virtuální počítač (meta-počítač), jehož všechny uzly mají vlastní správu zdrojů a příslušnou alokační politiku. Jednotlivými typy uzlů mohou být superpočítače, osobní počítače, PDA počítače, nízkoúrovňové senzory či dále hierarchicky členěné clustery. Hlavním rozdílem oproti čistě clusterovému přístupu je rozsáhlá distribuovanost a decentralizace správy.

Na celém světě existují desítky a možná i stovky nejrůznějších „Gridů“ velmi rozmanitého určení i rozsahu (viz např. reference na Grid Cafe[60]). Evropa se v současné době může

[58] http://www.top500.org/2007_overview_recent_supercomputers/infiniband
[59] http://www.gridcafe.org/
[60] http://www.gridcafe.org/grid-powered-project.html

chlubit nejrozsáhlejším a nejvýkonnějším gridem, který je pod názvem „EGI Grid“[61] provozován v rámci projektu EGI inSPIRE[62] 7. Rámcového programu Evropské komise. Tento grid má aktuálně téměř dvě stě padesát tisíc jader a více než sto petabyte úložné kapacity v discích a sto petabyte v magnetických páskách. EGI Grid má zapojené zdroje v padesáti zemích. Pokrývá nejen celou Evropu, ale zasahuje i do rozvojových zemí jižní Ameriky, Afriky či Asie. Hlavním uživatelem tohoto gridu je komunita částicových fyziků (high energy physics) soustředěná kolem centra CERN[63].

Obecně jsou EGI Grid i většina ostatních gridů používány pro podporu výzkumu a vývoje (tzv. e-Science[64]), umožňují tedy řešit problémy, které by nebylo možné řešit bez vhodné koordinace různých zdrojů dat i výpočetní síly. Například biologové používají gridy pro simulování vlivu tisíců molekulárních léků s cílem najít molekuly schopné blokovat konkrétní nemoci proteinů. Vědci zabývající se naukou o zemi používají gridy pro sledování úrovně ozonu pomocí družic, ukládání a analýzu stovek gigabytů dat denně z nejrůznějších senzorů. Inženýři používají gridy například ke studiu alternativních paliv, jako je energie z jaderné syntézy. Umělci zase používají gridy při vytváření komplexní animace pro celovečerní projekce. Sociální vědci využívají gridů ke studiu společenského života včel, složení naší společnosti, tajemství minulosti apod.

ČR je zapojena do EGI Gridu prostřednictvím sdružení CESNET, jehož oddělení MetaCentrum[65] zastává roli Národní gridové iniciativy (NGI, National Grid Initiative) a koordinuje národní gridovou infrastrukturu jak je uvedeno v předchozí kapitole. Do ní jsou zapojeny tisíce jader a stovky terabyte úložných kapacit přibližně deseti různých organizací. Z nichž nejvýznamnějšími jsou MU v Brně a její Centrum CERIT-SC, ZČU v Plzni, UK v Praze, Fyzikální ústav AV ČR a samozřejmě i sdružení CESNET, které rovněž poskytuje významné výpočetní a úložné kapacity do národního gridu. Gridová infrastruktura budovaná MetaCentrem je bezplatně (po registraci) k dispozici všem zaměstnancům a studentům vysokých škol a AV ČR, stejně jako pracovníkům dalších výzkumných ústavů (ve všech případech ovšem pouze pro výzkumné a vývojové prostředí, nikoliv jako komerční výpočetní a úložná platforma). Národní gridová infrastruktura poskytuje výpočetní kapacity především formou dávkového zpracování, část zdrojů je však k dispozici i interaktivně.

4.1.2 Cloud computing

Cloud computing je model zpřístupnění výpočetních a úložných kapacit založený důsledně na internetových technologiích a virtualizaci hardware. Podobné snahy se objevovaly již v minulých letech – nakonec služby např. firmy Google jsou důsledně zpřístupněny přes internet, s využitím možností lokálního prohlížeče.

Cloud computing však směřuje dále a nabízí kompletní virtualizovaný hardware, na kterém si uživatelé mohou spouštět v podstatě cokoliv. Samozřejmě je možné tyto služby řetězit, takže první uživatel si od poskytovatele cloudu vyžádá virtuální počítač. Na něm nainstaluje nějaký operační systém a kupříkladu webový server. Další uživatelé již přistupují ke službám, které tento webový server poskytuje.

Cloud computing vznikl jako komerční, nikoliv výzkumná aktivita. Jedná se o velké výpočetní zdroje, které se však používají pouze nárazově (např. Amazon, aby byl schopen uspokojit například i předvánoční nárazovou poptávku, musí mít dostatečný výpočetní výkon, kte-

[61] http://www.egi.eu/
[62] https://wiki.egi.eu/wiki/EGI-InSPIRE:Main_Page
[63] http://www.cern.ch/
[64] http://esciencenews.com/
[65] http://meta.cesnet.cz

rý však většinu roku není plně využíván). Majitelé takových zdrojů začali tento „nadbytečný“ výkon nabízet k dispozici jiným uživatelům. Virtualizace hardware jim umožňuje poskytovat „prázdné“ virtuální počítače (které se však zejména v oblasti vlastního výpočetního výkonu od fyzických počítačů prakticky neliší) a veškerou správu obsahu přesunují na zákazníka. Ten oplátkou dostává počítač, za který platí jen tehdy, když jej skutečně používá (potřebuje-li jej například tři hodiny denně, nemusí se starat o jeho vypnutí a zapnutí nebo placení elektřiny v době, kdy stroj běží „naprázdno“).

Cloud computing je ideální např. pro začínající firmu, která bude na internetu nabízet nějaké služby. Na začátku provozu nelze odhadnout, jaký bude zájem: pokud firma předpokládá zájem velký, nakoupí mnoho počítačů (aby docílila uspokojení), ale ty při nižším zájmu o služby nemusejí být využity, a prvotní investice byla tak zbytečná. Naopak, pokud firma poptávku podcení, pak nemusí stačit rostoucímu zájmu uživatelů, kteří stihnou „odejít“ jinam dříve, než se firmě podaří pořídit nové kapacity.

S využitím cloud computingu může firma v první fázi pořídit jen menší výpočetní kapacitu (kterou navíc platí jen v rozsahu, v jakém si ji objednala, nikoliv celé náklady spojené s pořízením a provozem hardware – např. nemusí řešit problém, kam servery umístit) a podle potřeby ji navyšovat prakticky v reálném čase (poněkud idealizovaně se předpokládá, že poskytovatel cloud technologií stojí o krok před svými uživateli a má vždy k dispozici dostatečný volný výkon na uspokojení nových požadavků – pokud má poskytovatel již dostatečný počet zákazníků s měnícími se požadavky, pak by tento model měl platit i při poměrně malé rezervě na straně poskytovatele). Mluvíme tak o „***elasticitě***“ cloudu, kdy se cloud jakoby přizpůsobuje tomu, co uživatelé potřebují/žádají.

Poskytování pouze prázdného virtualizovaného počítače (a související úložné kapacity) však poměrně brzy přestalo stačit a cloud computing se postupně vyvinul do celé řady modelů:

- ***Infrastruktura jako služba*** (*Infrastructure as a Service* – IaaS). V tomto případě poskytovatel služeb se zavazuje poskytnout infrastrukturu. Základním typem je poskytnutí onoho virtualizovaného počítače. IaaS však zahrnuje i složitější případy (např. poskytnutí celého virtuálního clusteru). Hlavní výhodou tohoto přístupu je fakt, že se o veškeré problémy s hardware (umístění, elektřina, …) stará poskytovatel. Pro některé uživatele je toto ale nesnadno akceptovatelný model, protože vytváří na poskytovateli cloud computingu příliš vysokou závislost (nejen např. v okamžiku zhroucení firmy, ale i při nečekaném zvýšení ceny, která na rozdíl od ceny elektřiny není hlídaná). Problémem může být i to, že cloud je „někde v internetu“ – zákony či vnitropodniková pravidla např. mohou zakazovat přenášet zpracovávaná data mimo zemi původu apod. Model IaaS je vhodný zejména pro ty uživatele, kteří vlastní software (či jejich licence), ale nechtějí se starat o hardware, a současně nemají výše uvedená omezení. Příkladem IaaS cloudů jsou Amazon WS[66], Rackspace[67] nebo Windows Azure[68]. Zkratka IaaS může také znamenat integrace jako služba (Integration as a Service);
- ***Platforma jako služba*** (*Platform as a Service* – PaaS). Poskytovatel v modelu PaaS poskytuje kompletní prostředky pro podporu celého životního cyklu tvorby a poskytování webových aplikací a služeb plně k dispozici na internetu, bez možnosti stažení software. To zahrnuje různé prostředky pro vývoj aplikace jako IDE nebo API, ale např. také pro údržbu. Nevýhodou tohoto přístupu je zpravidla proprietární uzamčení, kdy může každý poskytovatel používat např. jiný programovací jazyk. Pří-

[66] http://aws.amazon.com/
[67] http://www.rackspace.com/
[68] http://www.microsoft.com/cze/azure/

kladem poskytovatelů PaaS jsou Google App Engine nebo Force.com (Salesforce.com);

- ***Software jako služba*** (*Software as a Service* – SaaS). Aplikační software je licencován jako služba pronajímaná uživateli. Uživatelé si tedy kupují přístup k aplikačnímu software, ne samotný software. Model SaaS je ideální pro ty uživatele, kteří potřebují jen běžné aplikační software a požadují přístup odkudkoliv a kdykoliv. Příkladem může být známá sada aplikací Google Apps[69], nebo v logistice známý systém Cargopass[70].

Model cloud computingu není omezen pouze na interakce „nějaký" poskytovatel a uživatel. Je přirozeně možné budovat vlastní cloud a ten poskytovat buď pouze interně, nebo i dalším subjektům. Z tohoto pohledu můžeme zavést další klasifikace cloud computingu:

- ***veřejný*** (*public cloud computing*) – někdy je označován jako klasický model cloud computingu. Jedná se o model, kdy je poskytnuta a nabídnuta široké veřejnosti výpočetní služba;
- ***soukromý*** (*private cloud computing*) – cloud je v tomto případě provozován pouze pro organizaci, a to buď organizací samotnou, nebo třetí stranou;
- ***hybridní*** (*hybrid cloud computing*) – hybridní cloudy kombinují jak veřejné, tak soukromé cloudy. Navenek vystupují jako jeden cloud, ale jsou propojeny v podstatě analogickým způsobem, který platí pro gridovou infrastrukturu;
- ***komunitní*** (*community cloud computing*) – jde o model, kdy je infrastruktura cloudu sdílena mezi několika organizacemi skupinou lidí, kteří ji využívají. Tyto organizace může spojovat bezpečnostní politika, stejný obor zájmu. Opět se jedná o řadu prvků společných s gridovou infrastrukturou.

Cloud computing představuje posun paradigmatu, kdy se odděluje fyzická infrastruktura (a následně i další nad ní provozované služby) od toho, co skutečně uživatel potřebuje. Ten zpravidla nepotřebuje provozovat fyzické počítače (a ani jejich operační systémy – na druhou stranu nabídka dnes již obsahuje i operační systémy provozovatelné přímo v prohlížeči, např. eyeOS, Cloud či iCloud [5]). Uživatele zajímá jen konkrétní aplikace (kancelářský balík, databáze typu SAP, webové služby, ale třeba i aplikační software pro modelování či simulaci). Uživatelé se tak mohou soustředit na to, co skutečně potřebují a také jen za to platit. Ostatní vrstvy mohou být zcela nebo částečně skryté. Na rozdíl od gridové infrastruktury základní model cloud computingu nepočítá s tím, že by uživatel vlastnil a do cloud zapojoval nějaké své zdroje (ovšem viz hybridní cloud výše). Role uživatele a poskytovatele je tak jasně oddělena (a umožňuje mimo jiné i jasně definovat finanční toky).

Cloud computing přirozeně vzbudil zájem i v kontextu vědeckých výpočtů, kde je rovněž často těžké – zejména ve fázi vývoje – správně odhadnout potřebný výpočetní výkon a elasticitu cloud computingu by bylo možné velmi dobře využít. Navíc virtualizace již dnes prakticky nesnižuje výpočetní výkon. Ten je na fyzické vrstvě stejné jako ve virtuálním počítači, který na tom fyzickém běží.

Ukazuje se však, že jedním z omezení je přístup k datům. Jednak virtualizovaná vrstva má stále nemalé výkonnostní problémy (přístupy na lokální disk čí přes síť zpřístupněné diskové pole mohou být výrazně pomalejší než v případě přímého využití fyzického hardware), jednak poskytovatelé cloudu zpravidla finančně penalizují příliš rozsáhlé operace s daty (mimo jiné proto, že zatěžují jejich infrastrukturu mnohem více než prosté výpočty či jednoduché aplikace typu webových serverů či balíků kancelářských programů).

[69] http://www.google.com/apps/intl/cs/business/index.html
[70] http://www.cargopass.cz/

Výzkumná komunita má proto stále větší zájem o vytváření vlastních akademických cloudů, kde spojením na národní či dokonce nadnárodní úrovni lze dosáhnout situace, kdy poskytovatel takového cloudu má již k dispozici dostatek prostředků na uspokojení prakticky jakéhokoliv požadavku (jako je tomu např. v případě Amazonu či Google). Přitom finanční a další podmínky využití lze nastavit s ohledem na požadavky výzkumné a akademické komunity.

Zájem roste rovněž o menší privátní cloud, např. na úrovni univerzity, kde důsledné využití cloud technologií může přinést nemalé úspory (zmenší se počet fyzických serverů a míst, kde jsou umístěny, zvýší se jejich využití). Přitom je možné dynamičtěji reagovat na měnící se potřeby.

Značný komerční úspěch cloud computingu naznačuje, že je s ním potřeba počítat v budoucnosti i ve vědeckých výpočtech. Lze rovněž očekávat, že postupně budou překonány současné výkonnostní problémy a cloud platforma se stane zcela přirozenou součástí výpočetní infrastruktury, která bude pro vědecké výpočty k dispozici. Cloud computing bude stále více integrován s gridovými systémy. Již se s tím můžete setkat v souvislosti s mobilními systémy, o nichž pojednává následující kapitola.

4.1.3 Mobilní platformy

V průběhu posledních let dochází k masivnímu rozvoji v oblasti vývoje mobilních platforem, které tvoří malé přenosné elektronické bezdrátové přístroje s vlastním napájením a s různými aplikacemi. Často jsou vybaveny dotykovým displejem a/nebo miniaturní klávesnicí. Tato přenosná zařízení, ať už tzv. chytré telefony (např. iPhone), tablety (např. iPad) nebo osobní digitální pomocník PDA (*Personal Digital Assistant*), se zdokonalují, zvyšuje se jejich výkon, výpočetní síla i rozsah funkcionality. V důsledku konkurenčního boje mezi jejich výrobci se snižuje jejich cena, a tím roste jejich dostupnost pro veřejnost. Současné mobilní platformy zahrnují v sobě operační systém, middleware, uživatelské rozhraní a aplikace. Mezi nejvíce zastoupené operační systémy na trhu patří operační systém iOS firmy Apple a Android, jehož vývoj vede společnost Google.

Rozvoj mobilních platforem přináší nové možnosti i do oblasti vědeckých výpočtů. Výhodou mobilních zařízení je jejich přenosnost a flexibilita, a s tím související možnosti použití při výpočtech a zpracování dat v terénu. Zařízení navíc obsahují i prvky hardware, které mohou rozšiřovat výpočetní potenciál stolních počítačů. Jedná se zejména o nástroje, jako jsou zabudovaná kamera, GPS, akcelerometr[71], gyroskop nebo kompas. Takové nástroje umožňují sbírat data, která jsou dále použitelná například jako parametry přímo ve výpočtu nebo sekundárně jako doplňující informace o kontextu výpočetního problému.

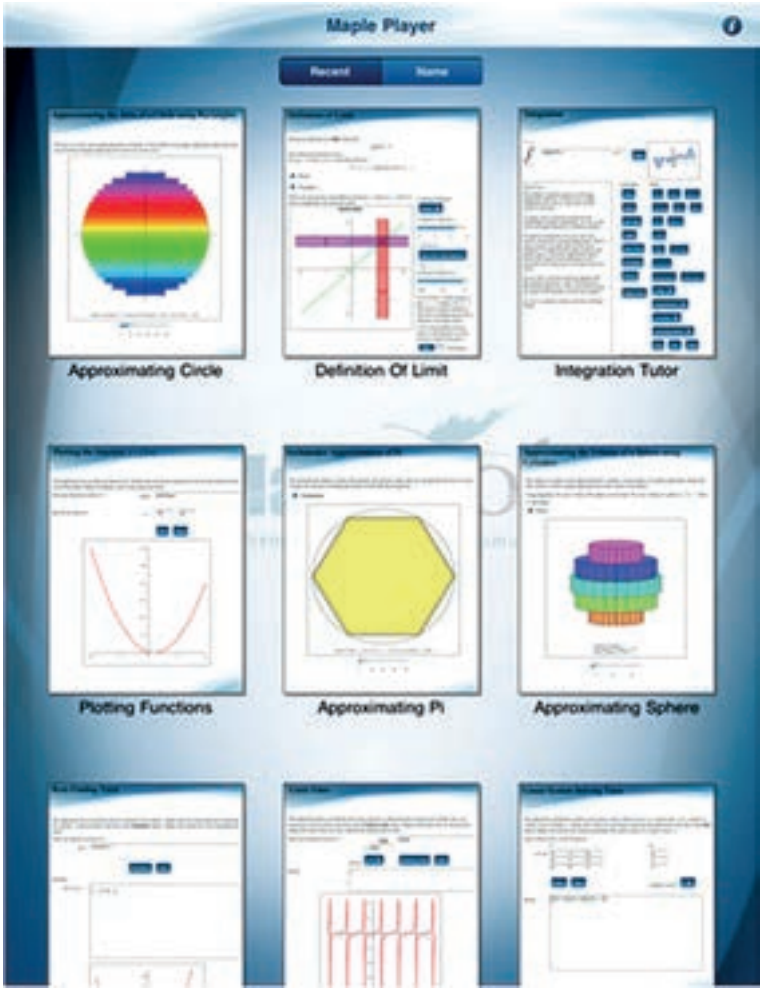
Velký potenciál nabízejí tyto nástroje i ve spojení se systémy rozšířené reality. Aplikace rozšířené reality na mobilních zařízeních fungují tak, že uživatel snímá zabudovanou kamerou okolní prostředí a senzory v zařízení zaznamenávají polohu uživatele a zařízení. Na základě získaných údajů nebo po rozpoznání určitých objektů je pak snímaný obraz rozšířen o vrstvy virtuálních informací o prostředí. To umožňuje zasazovat počítačem generované modely a simulace do kontextu reálného světa. Tyto skutečnosti mohou být užitečné především v oblastech jako architektura a stavebnictví, ale i v medicíně, environmentalistice a vzdělávání.

[71] Akcelerometr je přístroj, který měří vibrace nebo zrychlení při pohybu. Síla způsobující vibrace nebo změnu pohybu (akceleraci) působí na hmotu snímače, která pak stlačuje piezoelektrický prvek generující elektrický náboj úměrný stlačení. Protože je elektrický náboj úměrný síle a hmota snímače je konstantní, je tedy elektrický náboj také úměrný zrychlení – akceleraci, viz http://carreaum.blog.mobilmania.cz/2009/04/co-je-akcelerometr/

V současnosti se pohybuje využití mobilních platforem při řešení vědeckých výpočtů převážně na experimentální úrovni. Existuje pro to několik důvodů. Náročnost řešených úloh například ve velké míře zatěžuje procesor a negativně ovlivňuje výdrž baterie. Mnohé aplikace také vyžadují pro své fungování připojení k internetu a disponibilitu pro přenos velkého objemu dat. Následkem toho v případě použití mobilních datových sítí je evidentní zpomalení výpočtu a prodloužení doby odezvy. Mobilní zařízení proto zatím nejsou ideální média pro řešení úloh citlivých na čas a bezprostřední odezvu.

Komplikace způsobuje i fragmentace vývoje aplikací pro jednotlivé platformy. Nástroje určené k řešení vědeckých výpočtů se obvykle vyznačují velkým množstvím funkcí a vstupních parametrů. Samotné výpočty mohou vyžadovat komunikaci se speciálními senzory v zařízení. Uživatelské rozhraní musí být navíc funkčně i ergonomicky přizpůsobené dotykovému ovládání. Kontrola aplikací v mnoha případech žádá nestandardní interakce. Do grafických klávesnic, které slouží k zadávání vstupů, bývá nutné zakomponovat speciální znaky.

Za těchto podmínek není možné ovládat aplikace přes internetový prohlížeč prostřednictvím webového rozhraní jednotného pro různé typy zařízení. Obvykle je třeba každou aplikaci vyvíjet jako nativní program (tj. aplikaci spouštěnou v operačním systému) určený pro konkrétní platformu.



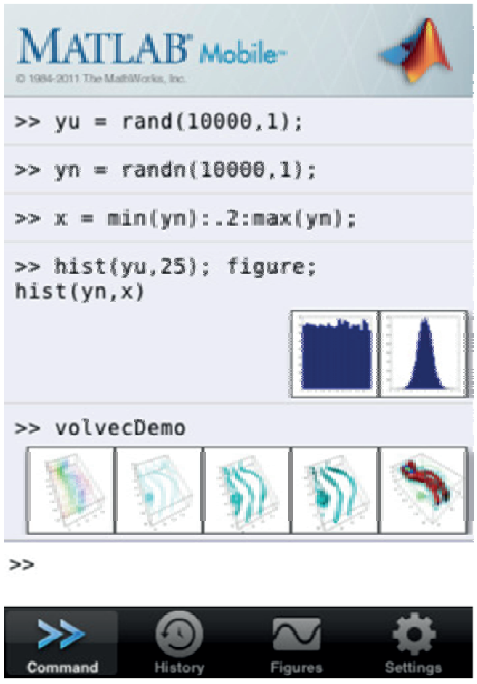
Obr. 4.1 *Rozhraní aplikace Maple Player pro iPad.*

Nejrozšířenějším modelem řešení náročných výpočetních úloh na mobilních zařízeních v dnešní době je architektura typu klient-server. Uživatel prostřednictvím mobilního zařízení

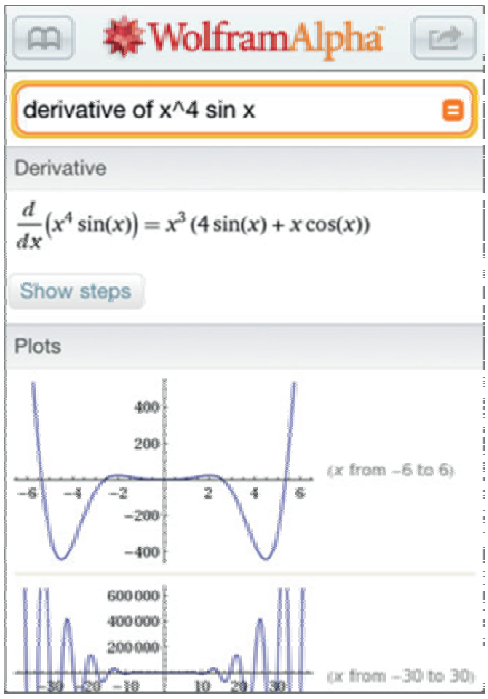
sbírá nebo zadává data, která odesílá na server. Údaje se na vzdáleném počítači zpracují a výsledek přichází ve formě odpovědi zpět k uživateli. Mobilní zařízení tak funguje výhradně jako rozhraní pro práci Samotný výpočet probíhá na výkonných procesorech vzdálených strojů. Za skrytou hardwarovou vrstvou se přitom může podle potřeby nacházet server poskytovatele, ale i rozsáhlá infrastruktura sdíleného výpočetního prostředí.

Příkladem nástroje pro počítání náročných výpočtů na mobilních platformách je aplikace Maple Player[72] pro iPad, obrázek 4.1. Maple Player poskytuje sadu interaktivních dokumentů připravených v systému Maple, které umožňují například počítat integrály, derivace, limity, kořeny rovnic a mnohé jiné. Uživatel může měnit koeficienty, zobrazovat výsledky a manipulovat s vizualizacemi grafů, dvojrozměrných i trojrozměrných modelů. V dalších fázích vývoje aplikace by měla být rozšířena kolekce základních dokumentů a přibýt i možnost vkládat vlastní dokumenty.

Dalším příkladem je aplikace MATLAB Mobile[73] určená pro zařízení s operačním systémem iOS. Mobilní aplikace poskytuje vzdálený přístup k prostředí MATLAB běžícímu na počítači s operačním systémem Windows, Mac OS nebo Linux. Uživatel přistupuje k výpočetnímu prostředí prostřednictvím instrukcí zadávaných do příkazového řádku na mobilním zařízení. Samotné výpočty pak probíhají na počítači uživatele nebo v prostředí MathWorks Computing Cloud.



Obr. 4.2 *Rozhraní aplikace MATLAB Mobile.*



Obr. 4.3 *Rozhraní aplikace Wolfram Alpha pro iPhone.*

Na principu klient-server funguje i webová služba WolframAlpha[74]. Jde o jednu z uživatelsky nejrozšířenějších aplikací pracujících se symbolickými výpočty.

Kromě těchto aplikací existuje na trhu celá řada dalších matematických nástrojů určených k řešení obvyklých výpočtů. Patří mezi ně například různé typy vědeckých kalkulátorů (Real-

[72] http://itunes.apple.com/ca/app/maple-player/id469981983
[73] http://itunes.apple.com/us/app/matlab-mobile/id370976661
[74] http://www.wolframalpha.com/

Calc Scientific Calculator[75] nebo Andie Graph[76] pro Android, Scientific Calculator[77] pro iOS). Jde o nástroje určené pro výpočty, zobrazování grafů a trojrozměrných modelů (Grapher[78], 3D graph OpenGL[79] pro Android, Graphing Calculator[80] pro iOS), ale i aplikace pro statistické zpracování dat (aplikace STA: Statistical Toolbox[81], StatCalc[82] pro Android).

4.2 Grafika ve vědeckých výpočtech

Grafické výstupy ve vědeckých výpočtech jsou v současné založeny na využití grafických procesorových jednotek. Standardní grafické jednotky vznikly v minulém století jako vysoce specializované řešení pro podporu grafických operací souvisejících s vizualizací informací na zobrazovacích zařízeních počítačů. V průběhu let se tyto grafické jednotky staly vysoce programovatelnými, což vyústilo v uvedení prvních tzv. ***grafických procesorů*** (*Graphic Processor Unit* - GPU)[83]. V letech 1999 – 2000 začali vědci a inženýři z oblasti informatiky, ale i z dalších aplikačních oblastí (zpracování obrazu v lékařském výzkumu, aplikace elektromagnetických a materiálových simulací) využívat grafické procesory pro realizaci náročných numerických výpočtů. Hlavní motivací bylo zjištění, že grafické procesory disponují extremním výpočetním výkonem, který lze efektivně využívat ve vědeckých výpočtech. Tato situace byla zrodem rozsáhlé iniciativy nazývané GPGPU[84] (*General Purpose computing on GPU*) – univerzální resp. obecné počítání s využitím grafických procesorů.

Hlavním problém v této době byla skutečnost, že GPU byl založen na tzv. grafických programovacích jazycích (OpenGL[85] nebo Cg[86]), což vyžadovalo od programátorů nahlížet na jejich algoritmy ve vědeckých výpočtech jako na grafické aplikace a transformovat problém do algoritmů, které spočívaly v „zobrazování trojúhelníků a polygonů“. Tato skutečnost byla hlavním limitujícím faktorem pro efektivní využití, jinak zcela unikátního, výpočetního výkonu GPU procesorů ve vědeckých výpočtech.

Na tuto situaci reagovala americká společnost NVIDIA[87] zásadním inovativním krokem, že modifikovala grafické procesory tak, aby se z nich staly plně programovatelné numerické akcelerátory pro obecné vědecké výpočty. Tato iniciativa společnosti NVIDIA následně vyústila v plnou podporu standardních programovacích jazyků C/C++ dnes označovanou jako architektura CUDA[88] pro grafické procesory NVIDIA.

Základní filosofie GPU výpočtů je založena na společném (hybridním) využití standardních (CPU) a grafických (GPU) procesorů současně. Sériová resp. sekvenční část aplikace využívá dominantně standardní procesor (CPU) a výpočetně náročná část je spuštěna na vysoce paralelním grafickém procesoru (GPU). Z pohledu uživatele celá aplikace běží výrazně rychleji, protože využívá extrémně vysokého výpočetního výkonu grafického procesoru.

---

[75] https://market.android.com/details?id=uk.co.nickfines.RealCalc
[76] https://market.android.com/details?id=net.supware.tipro
[77] http://itunes.apple.com/us/app/scientific-calculator/id287645182
[78] https://market.android.com/details?id=com.opticron.grapher
[79] https://market.android.com/details?id=si.simon.sander.graph3d
[80] http://itunes.apple.com/us/app/graphing-calculator-3d/id331462840
[81] https://market.android.com/details?id=dk.evolve.android.sta
[82] https://market.android.com/details?id=com.aeidesign.statscalc
[83] http://cs.wikipedia.org/wiki/GPU
[84] http://gpgpu.org/
[85] http://cs.wikipedia.org/wiki/OpenGL
[86] http://en.wikipedia.org/wiki/Cg_%28programming_language%29#The_standard_Cg_library
[87] http://www.nvidia.co.uk/page/home.html
[88] http://www.nvidia.com/object/cuda_home_new.html

Programátor však musí modifikovat aplikaci tak, že přenese realizaci výpočetně náročného jádra aplikace do grafického procesoru (GPU). Zbytek aplikace zůstává i nadále na CPU. Přenos vybrané funkce do GPU vyžaduje přepis jejího zdrojového kódu tak, aby bylo využito vysokého paralelismu a přidáním vhodných „direktiv“ resp. klíčových slov, které zajistí přenos dat mezi procesory CPU a GPU.

NVIDIA Tesla[89] je vývojová generace grafických procesorů využívající architekturu CUDA a je vysoce optimalizovaná pro výpočetně náročné vědecké výpočty a numerickou simulaci, což je dáno hlavně hardwarovou implementací výpočtů v tzv. dvojité přesnosti, přímým přístupem, sdílením celé paměti všemi výpočetními jádry grafických procesorů (GPU) a přítomností lokálních cache[90] pamětí u každého výpočetního jádra a řadou dalších opatření pro zvýšení výpočetního výkonu.

4.2.1 CUDA model

Paralelní hardwarová architektura CUDA je nyní doplněna plně paralelním programovacím modelem CUDA, který poskytuje rozsáhlou skupinu programovacích nástrojů (tzv. abstraktů) pro efektivní a přímočaré využití datového a úlohového paralelizmu. Programátoři mají nyní možnost využívat vysokého stupně paralelismu přímo v rámci celé řady programovacích jazyků (C/C++, Java, Fortran atd.) nebo využívat aplikační rozhraní pro další vývojová a výpočetní prostředí (OpenCL[91], DirectX-11 Compute, Maple, MATLAB, Python, Java, atd.)

Prvním programovacím jazykem plně podporovaným NVIDIA CUDA je jazyk C. Vývojové prostředí a nástroje CUDA SDK[92] umožňují programovat grafické procesory přímo s využitím jazyka C s minimálním počtem nových příkazů a rozšíření. Paralelní programovací model CUDA poskytuje programátorům detailní návod a nástroje, jak daný výpočetní algoritmus rozdělit na separátní sub-problémy, které jsou následně současně implementovány s využitím vysokého stupně paralelismu.

GPU architektura a paralelní programovací model CUDA je v současnosti široce využívám v celé řadě aplikací.

4.3 Chyby ve vědeckých výpočtech

Chyby mohou nastat při řešení problémů ve vědeckých výpočtech v jakékoliv fázi řešení, od počáteční fáze sběru dat a vytváření zjednodušujících předpokladů pro algoritmus až do pokročilejších fází řešení souvisejících s implementací výpočetního modelu na počítači. A právě chybami vznikajícími implementací výpočetního modelu na počítači se budeme zabývat v této kapitole. Doporučujeme čtenáři, aby se nejprve seznámil s první kapitolou knihy [3], jejíž terminologii budeme dodržovat.

4.3.1 Přesnost a preciznost

Ve výpočetní vědě rozlišujeme dva pojmy definované v metrologii[93] − přesnost a preciznost − které v přirozeném jazyce často splývají. Přesnost (angl. *accuracy*) vyjadřuje,

[89] http://www.nvidia.com/object/tesla_computing_solutions.html
[90] http://cs.wikipedia.org/wiki/Cache
[91] http://cs.wikipedia.org/wiki/OpenCL
[92] http://www.root.cz/clanky/uvod-do-technologie-cuda/
[93] Metrologie (nebo také metronomie) je obor zabývající se mírami pro stanovení velikosti různých technických a fyzikálních veličin a jejich měřením [31].

jak blízko je určitá hodnota (aproximace skutečné hodnoty) ke skutečné (správné) hodnotě. Preciznost (angl. *precision*) určuje, kolik cifer (číslic) je použito k reprezentaci určité hodnoty [15].

Příklad 4.1: Je dáno číslo $k = 10{,}0605$ a jeho aproximace vyjádřené čísly

$$a = 10;\ b = 10{,}0;\ c = 10{,}06;\ d = 10{,}258.$$

Číslo *a* má ze všech aproximací čísla *k* nejnižší preciznost (2 cifry). Jeho přesnost lze vyjádřit pomocí absolutní nebo relativní chyby, viz [3]. Číslo *b* má vyšší preciznost než číslo *a* (3 cifry), ovšem jeho přesnost je stejná, jakou má číslo *a*. Číslo *c* má vyšší přesnost i preciznost, než mají čísla *a* a *b*. Číslo *d* má ze všech aproximací nejvyšší preciznost (5 cifer), avšak jeho přesnost je nejnižší.

4.3.2 Reprezentace čísel v počítači a ve výpočetním prostředí

Je zřejmé, že při zvoleném typu zobrazení a předepsané délce paměťového místa je možno v počítači zobrazit pouze konečný počet čísel. Proto často hovoříme o ***konečné aritmetice*** či aritmetice s končenou přesností. Množina reálných čísel R je v počítači reprezentována svojí konečnou podmnožinou F, $F \subset Q \subset R$, kterou nazýváme ***soustavou čísel s pohyblivou řádovou čárkou*** (*floating point number system*), kde Q značí množinu racionálních čísel. Prvky podmnožiny F lze zapsat ve tvaru

$$y = \pm m \times \beta e^{-t}$$

kde celé číslo β je nazýváno základem, celé číslo t určuje přesnost, celé číslo m pohybující se v rozsahu $0 \leq m < \beta t - 1$ je nazýváno mantisou a celočíselný parametr e exponentem. Množina F je plně určena parametry β, t a horní, resp. dolní mezí celočíselného exponentu $e_{min} \leq e \leq e_{max}$.

Čísla se v počítači ukládají zpravidla v binární (dvojkové) soustavě ($\beta = 2$) uvažující dva základní symboly 0 a 1, případně v dalších soustavách, jejichž počet symbolů je založený na mocninách čísla 2, viz [3]. Jde zejména o oktalovou (osmičkovou) soustavu (vychází z 2^3 základních symbolů) a hexadecimální (šestnáctkovou) soustavu (vychází z 2^4 základních symbolů).

Ve výpočetním prostředí máme obecně různé datové typy, které dále určují, v jakém tvaru a s jakou precizností bude celé nebo reálné číslo uloženo v paměti počítače.

Pro uložení celého čísla se vyhradí n bitů (typicky jde o mocninu čísla 2), přičemž první bit je většinou vyhrazen pro znaménko čísla. Množina čísel, která takto můžeme uložit (se znaménkem), je tedy $\{-2^{n-1},\ldots,2^{n-1}-1\}$ a výpočty s těmito čísly jsou absolutně přesné v rámci uvedené množiny [3]. Pokud dojde k tzv. přetečení[94], jsou výpočty chybné.

Pro uložení desetinného čísla se nejčastěji používá norma IEEE 754, což je standard pro aritmetiku v pohyblivé řádové čárce. Tento standard definuje pět základních formátů, z nichž nejrozšířenější jsou tzv. jednoduchá přesnost (single precision) o 32 bitech a dvojitá přesnost (double precision) o 64 bitech. V případě jednoduché přesnosti je vyhrazeno 24 bitů pro mantisu a 8 bitů pro exponent. U dvojité přesnosti tvoří 53 bitů mantisu a 11 bitů exponent. Počet bitů mantisy určuje preciznost, s níž jsme schopni uložit číslo do paměti. Pro jednoduchou přesnost je to přibližně 7 cifer v desítkové soustavě, pro dvojitou přesnost přibližně 16 cifer v desítkové soustavě. Uložení desetinného čísla do formátu dvojité přesnosti prakticky znamená jeho zaokrouhlení na 16 platných míst v desítkové soustavě.

[94] Přetečení je jev, který nastane, pokud je výsledek aritmetické operace tak vysoký (v absolutní hodnotě), že jej nelze vyjádřit v daném číselném formátu [32].

Zmíněné zaokrouhlení ovšem ovlivňuje také většinu desetinných čísel, která mají v desítkové soustavě konečný desetinný rozvoj méně než 16 platnými místy. Jejich reprezentace v binární soustavě je totiž nekonečná.

Příklad 4.2: Uvažujme číslo 0,2 v desítkové soustavě. Jeho binární reprezentací (tj. reprezentací ve dvojkové soustavě, zapsané ciframi 0 a 1) je: $0,\overline{0011}$. Ve formátu dvojité přesnosti bude mít mantisa s prvním znaménkovým bitem (celkem 53 bitů) tvar:

0110011001100110011001100110011001100110011001100110011001100

a v exponentu bude uloženo[95] číslo 2^{-3}. Z nekonečného (periodického) dvojkového rozvoje tedy do paměti uložíme pouze prvních 55 cifer (včetně prvních tří nul, které se „schovají“ v exponentu), a číslo tak bude uloženo nepřesně.

Největší kladné číslo, které můžeme ve formátu dvojité přesnosti uložit, je v desítkové soustavě řadově rovné hodnotě 10^{308}. Nejmenší kladné číslo téhož formátu je řádově rovné hodnotě 10^{-308}. U formátů definovaných normou IEEE 754 může docházet jak k přetečení, tak k podtečení[96].

4.3.3 Příklady chyb vzniklých konečnou reprezentací čísel v počítači

Na několika příkladech ilustrujme chyby, k nimž dochází v důsledku konečné reprezentace čísel v počítači. Využijeme k tomu výpočetní prostředí aplikačních software Maple a MATLAB.

Výpočetní prostředí software Maple má implementovanou desítkovou soustavu s nastavitelnou precizností (standardně je používaná preciznost na deset platných míst, kterou lze měnit pomocí systémové proměnné `Digits`).

Software MATLAB poskytuje datový typ ***double***, který představuje výše uvedený formát ***dvojité přesnosti***, což představuje výpočetní prostředí s precizností na šestnáct platných míst.

Tuto skutečnost je nutné mít na zřeteli při výpočtech v těchto softwarech a analýze chyb, které vznikají při jejich používání.

Příklad 4.3: Určeme hodnotu rozdílu 5,1 – 4,4.

Řešení:

Maple

```
5.1-4.4;
  0.7
```

MATLAB

```
>> 5.1-4.4;

ans =
  0.7000

>> format long
>> 5.1-4.4;

ans =
  0.699999999999999
```

[95] Norma IEEE 754 předepisuje exponent ve tvaru s „posunutou nulou“ (viz např. [33]), což pro jednoduchost neuvažujeme.

[96] Podtečení je jev, který nastane, pokud je nenulový výsledek aritmetické operace tak malý (v absolutní hodnotě), že jej nelze vyjádřit v daném číselném formátu [34].

V Maple získáme očekávaný výsledek 0,7. V MATLABu také. MATLAB ovšem standardně vypisuje čísla ve formátu ***short***, který číslo zaokrouhlí na první čtyři cifry za desetinným oddělovačem (tečkou). Jestliže před provedením rozdílu změníme nastavení formátu příkazem `format long`, zjistíme, že ve skutečnosti je výsledek v MATLABu roven číslu `0,699999999999999`.

Příklad 4.4: Pro $x = 1{,}2 \cdot 10^{-5}$ určeme hodnotu výrazu

$$\frac{1-\cos(x)}{x^2}.$$

Řešení: V Maple realizací příkazu

```
(1-cos(1.2e-5))/((1.2e-5)^2);
```

získáme řešení rovné 1. Pokud příkaz obměníme na tvar

```
eval((1-cos(x))/(x^2),x=1.2e-5);
```

který lze rovněž použít k vyhodnocení daného výrazu, však obdržíme řešení rovné číslu `0,6944444444`.

MATLAB po provedení příkazu

```
(1-cos(1.2e-5))/(1.2e-5)^2
```

vypíše (ve formátu `long`) hodnotu `0,499999732974901`.

Správné řešení je přitom rovné číslu 0,5.

Maple

```
(1-cos(1.2e-5))/(1.2e-5)^2;
                    1.
eval((1-cos(x))/(x^2),x=1.2e-5);
             0.6944444444

Digits:=12:
eval((1-cos(x))/(x^2),x=1.2e-5);
           0.500000000000
```

MATLAB

```
>> (1-cos(1.2e-5))/(1.2e-5)^2

ans =

          0.499999732974901
```

Nepřesnost výpočtu v Maple je způsobena standardně nastavenou precizností na deset platných míst. Různost obdržených výsledků je způsobena tím, že v prvním případě Maple zaokrouhlí (na standardních deset platných míst) zvlášť čitatel, zvlášť jmenovatel a až poté zaokrouhlená čísla vydělí. V druhém případě se nejprve zaokrouhlí pouze čitatel a následně se čísla vydělí. V obou případech se však ztrácí přesnost. Zvýšením preciznosti na dvanáct platných míst v Maple získáme správné řešení 0,5.

Příklad 4.5: V bodě $x = 1{,}0000072337$ určeme hodnotu polynomu

$$x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1.$$

Řešení:

Maple

```
eval(x^8-8*x^7+28*x^6-56*x^5+70*x^4-56*x^3+28*x^2-8*x+1,x=1.000007237);
  −4.6 10^−8

Digits:=16:
eval(x^8-8*x^7+28*x^6-56*x^5+70*x^4-56*x^3+28*x^2-8*x+1,x=1.000007237);
  4. 10^−14

Digits:=44:
eval(x^8-8*x^7+28*x^6-56*x^5+70*x^4-56*x^3+28*x^2-8*x+1,x=1.000007237);
  4. 10^−42

restart;
eval((x-1)^8,x=1.000007237);
  7.524342934 10^−42
```

MATLAB

```
>> format long
>> x=1.000007237;
>> x^8-8*x^7+28*x^6-56*x^5+70*x^4-56*x^3+28*x^2-8*x+1

ans =
  -1.243449787580175e-014

>> (x-1)^8

ans =
  7.524342933677922e-042
```

V Maple (pro standardní nastavení `Digits = 10`) získáme řešení o hodnotě $-4{,}6 \cdot 10^{-8}$.

V MATLABu obdržíme řešení o hodnotě $-1{,}243449787580175 \cdot 10^{-14}$.

Správné řešení je však rovno přibližně $7{,}5 \cdot 10^{-42}$.

MATLAB vyšší preciznost výpočtu než na šestnáct platných míst neposkytuje (a přesnější řešení tedy v něm tímto způsobem nezískáme).

Jestliže v Maple nastavíme preciznost na šestnáct platných míst, dostaneme řešení stejného řádu jako v MATLABu. Pro získání alespoň řádové přesnosti vypočteného řešení je třeba v Maple nastavit preciznost na čtyřicet čtyři platných míst.

Důvodem, proč je získané řešení o tolik odlišné od řešení správného je, že u sčítání jednotlivých členů polynomu dochází ke ztrátě přesnosti v důsledku rozdílných řádů exponentů sčítanců u aritmetických operací, při nichž je třeba zaokrouhlovat. Zadaný polynom je možné přepsat na tvar $(x-1)^8$, jehož výpočet je mnohem přesnější.

Příklad 4.6: Určeme hodnotu derivace funkce sin(x) v bodě $x = 1$, tj. sin´(1) a porovnejme ji s numerickou derivací.

Řešení: Derivací funkce sin(x) je funkce cos(x). Maple ji určí analyticky jako cos(x). Hodnota výrazu cos(1) v Maple je vyjádřena jako 0,5403023059 při standardním nastavení počtu cifer mantisy (`Digits` = 10).

V MATLABu je hodnota výrazu cos(1) rovna 0,5403023058681397.

Derivaci funkce lze však také vyjádřit aproximací numerické derivace, viz [3]. Derivaci funkce sin(*x*) v bodě *x* = 1 tedy nyní budeme aproximovat výrazem

$$\frac{\sin(1+h)-\sin(1)}{h},$$

kde *h* je reálné číslo „blízké“ nule. Zvolme $h = 10^{-15}$ a vypočítejme aproximaci derivace.

Maple hledanou aproximaci vypočítá jako číslo 0, neboť při standardním nastavení počtu cifer mantisy (`Digits=10`) se při operaci sčítaní $1 + h$ „ztratí“ druhý sčítanec a součet $1 + 10^{-15}$ bude roven 1. Pro získání správného výsledku součtu v argumentu funkce sinus je třeba nastavit preciznost v Maple alespoň na šestnáct platných míst (tj. `Digits` = 16). Teprve pak získáme aproximaci jako číslo `0,5000000000000000`.

MATLAB vyhodnotí hledanou aproximaci jako číslo `0,555111512312578`. Upozorněme, že pro $h \leq 10^{-16}$ při operaci sčítaní $1 + h$ se i v MATLABu „ztratí“ druhý sčítanec a součet $1 + h$ bude roven 1 (stejně jako při standardním nastavení v Maple).

Maple

```
cos(1.);
  0.5403023059

eval((sin(1+h)-sin(1.))/h, h = 1e-
15);
  0.

Digits:=16:
eval((sin(1+h)-sin(1.))/h, h = 1e-
15);
  0.50000000000000000
```

MATLAB

```
>> format long
>> cos(1

ans =
  0.540302305868140

>> (sin(1+1e-15)-sin(1))/1e-15

ans =
  0.555111512312578
```

Z předchozích příkladů vidíme, že uložení čísel v počítači má nezanedbatelný vliv na přesnost výpočtů. Může se zdát, že se vše vyřeší použitím systému, např. Maple, případně jiného systému počítačové algebry, a možností nastavení potřebné preciznosti výpočtů. Musíme mít však na zřeteli, že vyšší preciznost výpočtů je časově i paměťově náročnější a u složitějších problémů může být z tohoto důvodu prakticky nepoužitelná. V takových případech (a při použití systémů s omezenou precizností výpočtů, např. MATLAB) je nutné provádět výpočty tzv. numericky stabilními metodami [3].

Čtenáře se zájmem o to, jaké závažné důsledky mělo v minulosti zaokrouhlování čísel v počítači, odkazujeme na webovou stránku profesora K. Vuika [16].

4.3.4 Numerické řešení

Ve vědeckých výpočtech často využíváme algoritmy hledající přibližné numerické řešení daného problému – numerické metody [3]. Algoritmy buď vytváříme (programujeme) sami, anebo používáme metody implementované ve výpočetním prostředí zvoleného, již existujícího aplikačního software (např. Maple nebo MATLAB). Použití takových metod vede zpravidla k hledání řešení (ne)lineárních, příp. diferenciálních rovnic, příp. jejich systémů, dále k optimalizaci zadané funkce, numerické vyhodnocení derivací, příp. integrálů funkcí apod.

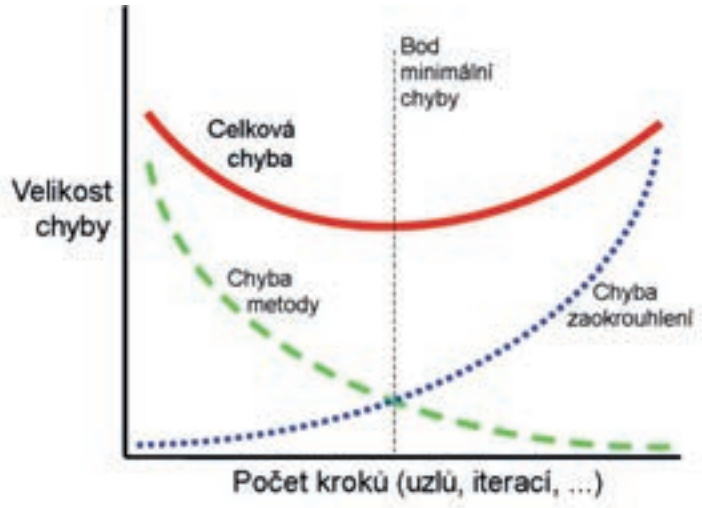
Určujeme vesměs přibližné řešení, tedy takové, které není přesné, a tudíž obsahuje chybu. Základní vlastnosti, jež po numerické metodě (potažmo jejím algoritmu) požadujeme, jsou ***stabilita*** a ***konvergence*** [3].

Stabilita zajistí, že vypočtené přibližné řešení je přesným řešením téhož problému s „blízkými“ vstupními daty.

Konvergence vyjadřuje, že touto metodou v jistém smyslu lze získat řešení daného problému s libovolnou přesností. Obvykle se tak děje zvyšováním počtu kroků metody (tj. snižováním délky jednotlivých kroků), zvyšováním počtu uzlů, iterací apod.

Při vědeckých výpočtech s využitím numerické metody na počítači nesmíme zapomínat na reprezentaci čísel. Zvyšováním počtu kroků (uzlů, iterací,…) v numerické metodě sice zvyšujeme přesnost aproximace, současně však zvyšujeme chybu způsobenou zaokrouhlováním, což lze ilustrovat obrázkem 4.4. Cílem je minimalizovat celkovou chybu.

Výpočetní systémy nabízejí pro řešení daného problému zpravidla několik numerických metod se standardním nastavením parametrů ovlivňujících přesnost metody. Mezi parametry, které je možné měnit/volit, bývají tzv. tolerance absolutní a relativní chyby (tj. hodnoty, které odhad příslušné chyby výpočtu metody nesmí přesáhnout), dále (minimální, maximální) velikost kroku metody a (maximální) počet vyhodnocení zadané funkce.



Obr. 4.4 *Výpočetní chyby.*

Příklad 4.7: Uvažujme Cauchyho počáteční problém

$$y'(t) = -y(t), \qquad y(0) = 10^{-8}.$$

Určeme graficky řešení tohoto problému na intervalu <0; 5>, tj. vykresleme graf jeho numerického řešení $y(t)$ pro $t \in$ <0; 5>.

Řešení: V Maple lze ukázat, že analytickým řešením počátečního problému je funkce $y(t) = 10^{-8} \cdot e^{-t}$, která má stejný průběh jako je na grafu na obrázku 4.5 (vpravo).

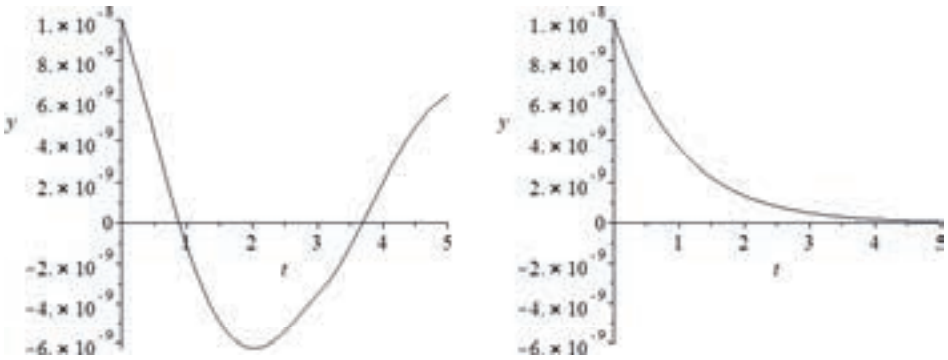
Jestliže řešíme tento problém v Maple numericky, tj. užitím příkazu `dsolve`, při standardním nastavení jeho parametrů ve tvaru

```
res := dsolve({diff(y(t),t)=-y(t),y(0)=1e-8},numeric):
```

kde je použita Fehlbergova modifikace metod Runge-Kutta řádů 4 a 5 viz [17], a příkazu `plot` tvaru

```
plot[odeplot](res,0..5);,
```

získáme graf numerického řešení na obrázku 4.5 (vlevo), který se podstatně liší od analytického řešení.



Obr. 4.5 *Grafy numerického řešení rovnice z příkladu 4.7.*

Nastavením tolerance diskretizační absolutní chyby (`abserr`) numerického řešení na hodnotu 10^{-10} nebo změnou užité numerické metody na přesnější spojitou metodu Runge-Kutta řádů 7 a 8 (`dverk78`), obdržíme graf tohoto numerického řešení v daném intervalu na obrázku 4.5 (vpravo), který je shodný s grafem analytického řešení. Zmíněným nastavením parametrů odpovídají příkazy

```
res2:=dsolve({diff(y(t),t)=-y(t),y(0)=1e-8},numeric,abserr=1e-10):
res3:=dsolve({diff(y(t),t)=-y(t),y(0)=1e-8},numeric,method=dverk78):
plot[odeplot](res2,0..5); plot[odeplot](res3,0..5);
```

Příklad 4.8: Uvažujme počáteční problém

$$y'(t) = y(t) - 2 \cdot e^{-t}, \qquad y(0) = 1.$$

Určeme v MATLABu graficky numerické řešení tohoto počátečního problému na intervalu <0; 12>, tj. vykresleme graf numerického řešení $y(t)$ pro $t \in$ <0; 12>.

Řešení: Lze opět ukázat, že analytickým řešením problému je funkce $y(t) = e^{-t}$. Jestliže řešíme problém numericky v MATLABu pomocí modifikace metody Runge-Kutta řádů 4 a 5 a při standardně nastavené toleranci diskretizační chyby, tj. pomocí příkazů:

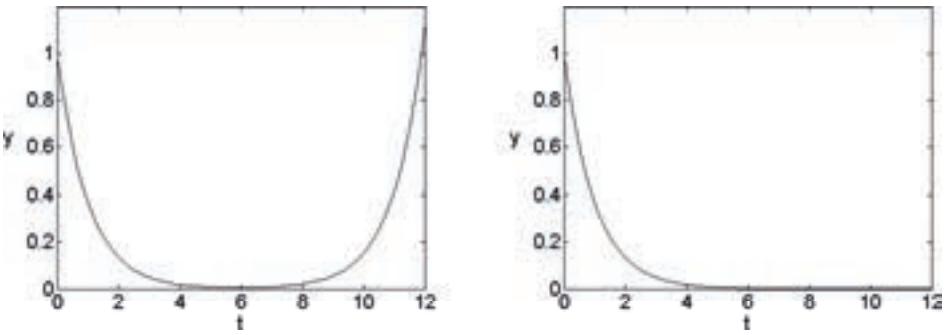
```
f=@(t,y)y-2*exp(-t);
[T,Y]=ode45(f,[0,12],1);,
```

získáme následným vykreslením graf numerického řešení daného problému, který je uveden na obrázku 4.6 (vlevo) a který se liší podstatným způsobem od analytického řešení tohoto počátečního problému.

Přesnější numerické řešení obdržíme buď zvýšením tolerancí diskretizační absolutní a relativní chyby, anebo změnou numerické metody (viz příklad 4.7). Výpočet numerického řešení s nastavením tolerancí diskretizační absolutní chyby na hodnotu 10^{-15} a relativní chyby na hodnotu 10^{-8} provedeme v MATLABu příkazem:

```
[T,Y]=ode45(f,[0,12],1,odeset('AbsTol',1e-15,'RelTol',1e-8));
```

Výsledný graf je uveden na obrázku 4.6 (vpravo).



(a) Graf řešení při standardním nastavení *(b) Graf řešení při změně přesnosti*

***Obr. 4.6** Grafy numerického řešení rovnice z příkladu 4.8.*

4.3.5 Úlohy k zamyšlení (k možnosti procvičení prezentované problematiky)

Příklad 4.9: Navrhněte numericky stabilní metodu výpočtu pro výraz v příkladu 4.4. Aplikujte ji v systémech Maple i MATLAB, výsledky porovnejte.

Příklad 4.10: Navrhněte numericky stabilní metodu výpočtu pro výraz v příkladu 4.5. Aplikujte ji v systémech Maple i MATLAB, výsledky porovnejte.

Příklad 4.11: Zjistěte, jaký typ diferenciálních rovnic je označován názvem stiff a jaké numerické metody k jejich řešení poskytují systémy Maple a MATLAB. Ilustrujte na příkladu.

4.4 Výpočetní náročnost

Pokud při řešení určitého problému používáme počítač, zpravidla nás také zajímá, jak dlouho bude vlastní řešení trvat. Doba výpočtu je přirozeně závislá na počtu instrukcí, které konkrétní počítač musí při řešení dané úlohy vykonat. Primitivním vyjádřením výpočetní náročnosti je tedy počet instrukcí, které je třeba k vykonání konkrétního programu a konkrétního zadání provést. Počet instrukcí závisí na konkrétním zadání a rovněž na tom, jak dobře je původní problém formulován a základní algoritmus implementován, jak je program optimalizován atd. Abychom dokázali abstrahovat od těchto příliš mnoha závislosti, využijme pojmy z oblasti, která se zabývá výpočetní složitostí.

V oblasti vědeckých výpočtů výpočetní náročnost programu zpravidla určují algoritmy, které byly pro řešení problému vyvinuty. Odhad výpočetní náročnosti programu (dobře implementovaného a dobře optimalizovaného) můžeme pak založit na odhadech výpočetní náročnosti jednotlivých algoritmů.

Základním pojmem v této oblasti je ***asymptotická výpočetní složitost***, která vyjadřuje, jak roste počet operací, které je třeba na provedení algoritmu udělat, s rostoucím množstvím vstupních dat. Zapisuje se pomocí „Omikron notace“ („velké O notace“) jako $O(f(N))$. Množina Omikron funkce f je množina všech funkcí, které mají „stejný“ nebo „nižší“ řád[97] růstu jako funkce f. Prohlásíme-li, že složitost algoritmu leží v množině Omikron funkce f, znamená to, že náročnost algoritmu bude růst vždy srovnatelně nebo méně rychle než funkce f. Vidíme, že v případě výpočetní složitosti uvažujeme o relativním růstu počtu operací v závislosti na velikosti vstupních dat. Například časová složitost $O(N)$ je lineární, tj. doba výpočtu algoritmu

[97] Řád růstu funkce f není řádem ve smyslu nejvyššího exponentu polynomiální funkce, nýbrž takové „jednoduché“ funkce g, pro kterou platí, že f je asymptoticky ohraničena g z obou stran (až na konstantu).

se zvýší přibližně tolikrát, kolikrát se zvýší velikost vstupu, obvykle tedy počet vstupních údajů. Na druhou stranu u složitosti $O(N^2)$ se doba trvání výpočtu zvyšuje kvadraticky. U časové složitosti $O(1)$ na počtu vstupních dat nezáleží a doba výpočtu je stále stejná. Řekneme-li, že výpočet má složitost $O(N^2)$, chceme tím říci, že při ztrojnásobení objemu vstupních dat očekáváme nejvýše devětkrát vyšší počet operací.

Zápis $O(f(N))$ představuje ve skutečnosti lineární funkci $A + B \cdot f(N)$, kde A a B jsou konstanty. Při Omikron zápisu tyto konstanty zanedbáváme, protože nás zajímá asymptotické chování, tedy situace při nekonečně velkých objemech vstupních dat. Může se tak stát, že asymptotická složitost neodráží skutečně pozorované chování algoritmů – pokud je např. konstanta A velmi vysoká a naopak konstanta B nízká, může v širokém rozsahu velikosti zadání převažovat hodnota konstanty A, a pozorovaný čas výpočtu algoritmu se tak může jevit jako konstantní. Notaci Omikron tedy lze chápat jako vyjádření „nejhoršího případu“ pro nekonečně velká vstupní data.

U převážné většiny algoritmů je situace dále komplikována tím, že pro dva vstupy stejné velikosti může výpočet trvat různě dlouho. Uvažujme algoritmus uspořádání prvků vektoru, který je postaven na jednoduchém principu výměny dvou prvků v případě, že leží v opačném pořadí. Zadáme-li takovému algoritmu na vstupu vektor, který je již uspořádán, algoritmus po jednom „průchodu“ všech prvků zjistí, že nemusel pořadí žádných dvou prvků zaměnit, a skončí. Pokud by však prvky vstupního algoritmu byly uspořádány v opačném než požadovaném pořadí, musí algoritmus procházet vektor tolikrát, jako je počet prvků v něm obsažených. V prvním případě tedy utřídění vektoru trvalo $O(N)$ (jednou musíme projít všechny prvky). Ve druhém případě $O(N^2)$ (musíme projít celý vektor N-krát). Asymptotická složitost tohoto algoritmu bude $O(N^2)$, protože počítá s nejobjemnějším a nejhorším případem. Reálně pozorovaná časová složitost algoritmu se bude pohybovat „někde mezi“ lineární a kvadratickou.

Obecně mluvíme o operační náročnosti algoritmu:

- operační náročnost v nejlepším případě (pro algoritmus nejpříznivější případ);
- operační náročnost v nejhorším případě (označovaná také amortizovaná složitost);
- operační náročnost ve středním případě (průměrná hodnota).

Důsledkem toho je skutečnost, že pokud chceme pochopit chování konkrétního algoritmu, nevystačíme s asymptotickou složitostí. Musíme však provést analýzu až na úroveň operační náročnosti algoritmu a studovat skutečnou závislost jak na velikosti, tak i na charakteru vstupních dat.

Přes tyto komplikace je pojem složitosti velmi důležitý a i „pouhá“ znalost asymptotické složitosti použitého algoritmu je velmi důležitá. Udává totiž odhad nejhoršího případu chování algoritmu. Algoritmy se však mohou chovat výrazně jinak, než odpovídá jejich asymptotické složitosti v případě malých rozsahů vstupních dat – tam bez obav může platit fakt, že algoritmus asymptotické složitosti $O(N)$ vyžaduje pro své vykonání menší počet operací (a je tedy rychlejší) než algoritmus složitosti $O(N^2)$.

Pokud máme zvolit ze dvou variant algoritmů pro řešení stejného případu, je dobré si uvědomit (nebo alespoň experimentálně vyzkoušet), jak „daleko“ od asymptotického chování se nacházejí předpokládaná vstupní data. V každém případě bychom však měli alespoň tušit, jakou asymptotickou složitost má námi použitý algoritmus a proč používáme právě jej.

Respektive je dobré mít na zřeteli, jaká rizika hrozí v případě, že překročíme původně plánovaný rozsah (například: původně jsme počítali s tím, že budeme hledat uspořádání vektoru do rozsahu rovnému nejvýše stu složek – tam může být výhodné využití i méně složitých algoritmů s vyšší asymptotickou složitostí – poté stejný program použijeme pro vektor s milionem prvků – uspořádání pak může trvat mnohem déle než naivně očekávané nejvýše desetitisící násobné prodloužení).

Na závěr této kapitoly ještě zmiňme rozdělení algoritmů do tříd složitosti podle typu funkce $f(N)$. Pokud je tato funkce polynomem, hovoříme o ***polynomiálně omezených algoritmech***. Takové problémy, které řeší nějaký polynomiální algoritmus, patří do tzv. třídy *P*. Speciálním případem této třídy jsou samozřejmě algoritmy s konstantní složitostí $O(1)$.

Pokud pro řešení zadaného problému neexistuje polynomiálně omezení algoritmus (např. $f(N)$ je exponenciální či ještě rychleji rostoucí funkcí), pak hovoříme o ***nepolynomiálních algoritmech***, které patří do třídy *NP*[98]. Typickým příkladem algoritmů, které patří do této třídy, jsou algoritmy exhaustivního (úplného) prohledávání, různé kombinatorické algoritmy apod. Pro řadu problémů neznáme řešení založené na algoritmech patřících do třídy *P*. V těchto případech pak musíme používat speciální přístupy (např. heuristiky, přibližné výpočty aj.), pokud musíme tyto problémy řešit (typickým příkladem je problém obchodního cestujícího, který musí projet předem danou konečnou množinou měst tak, aby ujel co nejméně kilometrů – tento problém nemá řešení v *P*). Otázka, zda *P=NP* (tj. zda pro každý algoritmus, pro nějž aktuálně známe jen řešení patřící do třídy *NP*, existuje algoritmus polynomiální) je jednou z velkých otevřených otázek informatiky.

[98] Teorie je samozřejmě mnohem složitější, zájemce se s ní může seznámit např. v monografii [35].

5 Neurčitost a citlivost ve vědeckých výpočtech

V předchozí kapitole jsme se věnovali mimo jiné chybám ve vědeckých výpočtech vznikajícím užitím numerických metod v počítačovém prostředí. Zmínili jsme přitom, že chyby mohou nastat v jakékoli fázi řešení problému. V této kapitole se budeme zabývat chybami ve vědeckých výpočtech obecněji.

5.1 Analýza neurčitosti

Chyba vzniká ve chvíli, kdy se výpočet odchyluje od správné hodnoty hledané veličiny. K tomu, abychom o hodnotě nějaké veličiny mohli tvrdit, že je zatížena chybou, musíme znát její správnou hodnotu. Naše možnosti (znalosti) nám to však většinou neumožňují. Z tohoto důvodu zavádíme a používáme termín ***neurčitost*** vyjadřující nedostatek znalosti o dané veličině ve zkoumaném objektu, jevu či souvislostech. Lze říci, že jakékoliv řešení skutečného problému z praxe v sobě zahrnuje neurčitosti a není možné je zcela odstranit. S tím se musíme smířit, avšak je třeba se pokusit neurčitosti určitým způsobem popsat a minimalizovat je.

Ve [2] jsou neurčitosti rozděleny do tří hlavních skupin, a to na ***neurčitosti v matematickém popisu, neurčitosti v datech*** a ***neurčitosti v aplikaci modelu***. V literatuře je možné nalézt i jiná dělení neurčitostí, vždy se však týkají tří zmíněných oblastí. Každý popis reality je nepřesný, podobně nepřesná jsou data (závislá jednak na přesnosti měření, ale také na jejich reprezentativnosti) a zejména v případě vědeckých výpočtů se neobejdeme bez použití ICT (a případných dalších nepřesných metod pro hledání řešení problému).

Pro názornost uveďme tři jednoduché příklady, kde v prvních dvou příkladech vystupuje neurčitost v matematickém popisu a ve třetím příkladu se projeví neučitosti v aplikaci modelu na počítači.

Příklad 5.1: Na vymezeném území žije populace určitých živých organismů. Určeme, jak se bude vyvíjet velikost populace $N(t)$ v následujícím časovém období (dnech, měsících,…), jestliže počáteční velikost populace v čase $t = 0$ je $N(0) = 100$ a vnitřní koeficient růstu populace je $r = 0{,}1$.

Řešení: V [2] je uvedeno několik matematických modelů růstu populace živých organismů. Předpokládejme, že naše populace splňuje kvantovací a vzorkovací podmínku (viz [3]). Jedním z možných popisů jejího růstu je rovnice

$$\frac{d}{dt}N(t) = r \cdot N(t), \qquad (5.1)$$

kde $N(t)$ je velikost populace v čase t. Řešení rovnice (5.1) má tvar:

$$N(t) = N(0) \cdot e^{r \cdot t}.$$

Pokud nemáme k dispozici žádnou další informaci o tom, na čem závisí vnitřní koeficient růstu populace r (a jak), odpovědí na zadaný problém může být to, že se velikost populace bude v čase t zvětšovat podle vztahu (po dosazení $r = 0{,}1$)

$$N(t) = 100 \cdot e^{0{,}1 \cdot t}.$$

Pokud je ovšem na vymezeném území omezená kapacita prostředí hodnoty K, je třeba pro predikci růstu populace použít jiný model, např. dle [2]

$$\frac{d}{dt}N(t) = r \cdot N(t) \cdot \left(1 - \frac{N(t)}{K}\right). \qquad (5.2)$$

Řešení rovnice 5.2 má tedy tvar:

$$N(t) = \frac{K}{1 + e^{-rt} \cdot N(0)}.$$

Velikost populace se tedy „po určitém čase“ ustálí na hodnotě *K*. K této hodnotě se bude velikost populace přibližovat v závislosti na vztahu hodnoty $N(0)$ k hodnotě *K*. Velikost populace v čase klesá, pokud $N(0) > K$, roste, pokud $N(0) < K$, případně zůstává konstantní, podrobněji viz [2], str. 26.

Další změnu přístupu k modelování růstu populace může přinést doplňující informace, například o výskytu nějakého predátora na vymezeném území či jiných vlivů ovlivňujících počet jedinců sledované populace. To vše je třeba zohlednit při vymezení neurčitosti v matematickém popisu modelu vývoje velikosti populace, viz [2]

Příklad 5.2: Uvažujme opět, že na vymezeném území žije populace určitých živých organismů. V daný den byla změřena její velikost, která činila $N(0) = 100$ jedinců. Dále je známo, že prostředí, v němž populace žije, uživí (dlouhodobě) maximálně 500 jedinců (tj. kapacita prostředí $K = 500$) a že pro vnitřní koeficient růstu populace platí: $r \in \langle -0{,}1;\ 0{,}1\rangle$. Určeme vývoj velikosti populace v následujícím období (dnech, měsících,…).

Řešení: Předpokládejme nyní, že růst sledované populace odpovídá rovnici (5.2). V tomto případě se však objevuje neurčitost v hodnotě vnitřního koeficientu růstu populace *r*, která způsobuje, že odpověď na zadanou otázku pro řešení problému je opět velmi neurčitá. Velikost populace se bude postupně zvyšovat až k hodnotě $K = 500$ (pro $r > 0$), anebo naopak klesat až do úplného vymření populace (pro $r < 0$), popř. může zůstat konstantní (pro $r = 0$).

Příklad 5.3: Nalezněme pevný bod funkce $f(x) = 11 \cdot x - 2$.

Řešení: Nejprve si připomeňme, že pevným bodem *x* funkce $f : M \to M$ je takový prvek $x \in M$, pro nějž platí: $x = f(x)$. Je zřejmé, že nejjednodušším způsobem pro jeho nalezení je vyřešení rovnice

$$x = 11 \cdot x - 2, \qquad (5.3)$$

kdy okamžitě získáme výsledek $x = 0{,}2$.

Pokusme se však hledat řešení metodou prosté iterace $x_i = f(x_{i-1})$, $i = 2, 3, \ldots$ (viz [3]) s počáteční iterací rovnou správnému řešení[99], tj. $x_1 = 0{,}2$. V aplikačním software Maple a MATLAB vypočtěme několik prvních iterací. Bez ohledu na jejich počet bychom očekávali, že každá z iterací bude mít hodnotu 0,2.

V Maple lze pro výpočet a výpis prvních patnácti iterací použít kód:

```
x[1]:=0.2:
for i from 2 to 16 do x[i] := 11*x[i-1]-2 end do;
```

V MATLABu pro totéž využijeme kód:

[99] Počáteční iteraci značíme jako $\boldsymbol{x}_1$, neboť v Matlabu je pole indexováno od čísla 1.

```
x(1)=0.2;
for i=2:16 x(i) = 11*x(i-1)-2; end
x
```

Zatímco v Maple obdržíme všechny iterace rovné správné hodnotě 0,2, v MATLABu (při standardním zobrazení na čtyři desetinná místa) se již dvanáctá iterace liší od správné hodnoty. Tento rozdíl se v následujících iteracích stále zvyšuje. Patnáctá iterace má v MATLABu hodnotu 0,2674 a v pořadí dvacátá iterace (pokud bychom ji vypočetli) již má hodnotu řádově rovnu 10^4. Když nastavíme příkazem `format long` zobrazení čísel s vyšší precizností, zjistíme, že již třetí počítaná iterace v MATLABu není přesně rovna hodnotě 0,2, ale 0,200000000000002 atd.

S obdobnými případy jsme se setkali již v předcházející kapitole. Na vině (výsledku zatíženého chybou) je jednak samo zobrazení čísel v počítači a jednak fakt, že pevný bod funkce ***f*** je tzv. ***odpuzujícím pevným bodem*** (viz [3]).

Neurčitosti můžeme analyzovat postupně v rámci skupin, do nichž jsme je výše rozdělili. ***Neurčitost v matematickém popisu*** souvisí s nedostatkem informací (resp. s uvažovanými předpoklady) získaných při sestavování matematického modelu. Analýza neurčitostí pak sestává z porovnávání různých struktur matematického modelu s měřenými daty. Jedná se o nejsložitější část celkové analýzy neurčitosti [2].

Datová neurčitost zahrnuje nepřesnost měření parametrů vystupujících v modelu (způsobenou jak chybou měřicího přístroje, tak vstupem lidského faktoru) a nepřesnost vzniklou omezeným množstvím vzorků sledovaného jevu/skutečnosti. V závislosti na informacích o hodnotách parametrů modelu reprezentujeme datovou neurčitost nejčastěji pomocí intervalové aritmetiky, fuzzy teorie, či pravděpodobnostní analýzy jak je to ukázáno v [2].

Poslední typ neurčitosti je předmětem numerické analýzy. Zkoumá se, jaký vliv na hodnotu řešení má použití ICT (uložení čísel v počítači/softwarovém systému) a numerické metody [3] hledající přibližné řešení.

5.1.1 Pravděpodobnostní analýza neurčitosti

Parametry matematického modelu obsahují neurčitosti, které je sice většinou možné popsat intervaly, typicky má však každý parametr svou střední hodnotu, jež je v jistém smyslu „nejpravděpodobnější“. Snahou je tak chápat parametr jako náhodnou veličinu s příslušným rozdělením pravděpodobnosti. Tzv. propagace (šíření) neurčitostí modelem (viz [2]) je pak zpravidla[100] prováděna pomocí metody Monte Carlo. Její užití zajistí vygenerování „dostatečného počtu“ realizací náhodných veličin reprezentujících parametry modelu, které poslouží k jeho různým vyhodnocením. Získáme tak pravděpodobnostní rozdělení hodnot řešení.

Použití metody Monte Carlo „přináší“ další neurčitosti, neboť se jedná o přibližnou metodu. Se vzrůstajícím počtem generovaných realizací náhodných veličin a vyhodnocení modelu konverguje získané rozdělení hodnot ke skutečnému rozdělení hodnot řešení vyhodnocovaného modelu. Přesnost získané aproximace pravděpodobnostního rozdělení testujeme porovnáním základních statistických charakteristik (střední hodnota, rozptyl) u dvou různých skupin vygenerovaných řešení nebo použitím odhadu chyby metody Monte Carlo.

Uvažujme model tvaru

$$Y = f(X_1,\ldots, X_m),$$

kde Y je hledané řešení, pro $m \in \mathbb{N}$ jsou $X_1,\ldots,X_m$ parametry modelu reprezentované náhodnými veličinami po řadě s pravděpodobnostními rozděleními $p_1(x_1),\ldots,p_m(x_m)$, $\mathbb{N}$ množina přirozených čísel.

[100] U (velmi) jednoduchých modelů nebo naopak u výpočetně náročných tomu tak být nemusí.

Označme $\boldsymbol{X} = (X_1,\ldots, X_m)$, $\boldsymbol{x} = (x_1,..., x_m)$ a $p(\boldsymbol{x})$ sdruženou hustotu pravděpodobnosti náhodného vektoru $\boldsymbol{X}$. Nechť dále M je počet vygenerovaných realizací náhodného vektoru $\boldsymbol{X}$, kde $\boldsymbol{X}_{(k)}$ je jeho k-tá realizace. Pak pro odhad střední hodnoty funkce $f(\boldsymbol{X})$ využijeme vzorce [18]:

$$\widehat{E}\left(f(\boldsymbol{X})\right) = \frac{1}{M} \cdot \sum_{k=1}^{M} f\left(\boldsymbol{X}_{(k)}\right). \tag{5.4}$$

Tento odhad konverguje (podle silného zákona velkých čísel) ke skutečné střední hodnotě pro $M \to \infty$, jestliže:

$$\int |f(\boldsymbol{x})| \cdot p(\boldsymbol{x})\, d\boldsymbol{x} < \infty.$$

Rozptyl v_M takto určených odhadů středních hodnot pro dané M určíme ze vzorce (viz [18]):

$$v_M = \mathrm{V}\left(\frac{1}{M} \cdot \sum_{k=1}^{M} f\left(\boldsymbol{X}_{(k)}\right)\right) = \frac{1}{M} \cdot \mathrm{V}\left(f(\boldsymbol{X})\right).$$

kde jsme využili vlastnosti rozptylu pro nezávislé náhodné veličiny $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2$ a konstanty a, b:

$$V(a \cdot Z_1 + b \cdot Z_2) = a^2 \cdot V(Z_1) + b^2 \cdot V(Z_2),$$

a současně předpokladu

$$\int \left(f(\boldsymbol{x})\right)^2 \cdot p(\boldsymbol{x})\, d\boldsymbol{x} < \infty.$$

Hodnotu $\sqrt{v_M}$ nazýváme chybou metody Monte Carlo (viz [18]). Aplikací nevychýleného odhadu pro rozptyl $V(f(\boldsymbol{X}))$ můžeme v_M odhadnout vztahem:

$$\widehat{v}_M = \frac{1}{M \cdot (M-1)} \cdot \sum_{k=1}^{M} \left(f\left(\boldsymbol{X}_{(k)}\right) - \widehat{E}(f(\boldsymbol{X}))\right)^2, \tag{5.5}$$

případně:

$$\widehat{v}_M = \frac{1}{M \cdot (M-1)} \cdot \left(\sum_{k=1}^{M} f^2\left(\boldsymbol{X}_{(k)}\right) - M \cdot \left(\widehat{E}(f(\boldsymbol{X}))\right)^2\right). \tag{5.6}$$

5.2 Zobrazovací metody a lokální analýza citlivosti

Analýzu neurčitosti obvykle doplňuje ***analýza citlivosti***. To poskytuje možnost dozvědět se, jak které parametry ovlivňují řešení modelu. Metody analýzy citlivosti lze rozdělit do tří skupin, a to na metody ***zobrazovací***, ***lokální*** a ***globální***.

Účelem zobrazovacích metod je identifikace nejvýznamnějších parametrů, které nejvíce ovlivňují řešení modelu. Zobrazovací metody bývají tzv. „***one-at-a-time***“. To znamená, že v jednom kroku zkoumají vždy vliv jednoho parametru na řešení modelu, zatímco ostatní parametry nabývají svých středních hodnot (a jsou pro tento krok konstantní). Používají se především při analýze citlivosti modelů s velkým počtem parametrů (až se stovkami). Získaný výsledek má kvalitativní charakter. Zjistíme, které parametry jsou citlivější (tj. které ovlivňují více řešení modelu) než jiné. Nedozvíme se však, jakou měrou (tj. o kolik). Výhodou je nižší výpočetní náročnost (oproti ostatním metodám analýzy citlivosti), nevýhodou pak zmíněná „nedostatečná“ informace o citlivosti parametrů. Více o zobrazovacích metodách analýzy citlivosti lze najít např. v [19].

Lokální analýza citlivosti byla již stručně představena v [2]. Pro detailnější informace týkající se metod lokální analýzy citlivosti odkážeme čtenáře např. na [19].

5.3 Globální analýza citlivosti

Analýza lokální citlivosti zkoumá lokální efekt změny parametrů (kolem jednoho bodu v *m*-rozměrném prostoru parametrů) na řešení modelu. Pokud však parametry nabývají širšího spektra hodnot, využívají se spíše metody globální analýzy citlivosti.

Uvažujme nadále model popsaný rovnicí:

$$Y = f(X_1,\ldots, X_m), \tag{5.7}$$

kde Y je hledané řešení a $X_1,\ldots, X_m$ ($m \in \mathbb{N}$) parametry modelu reprezentované náhodnými veličinami s pravděpodobnostními rozděleními $p_1(x_1),\ldots, p_m(x_m)$. Metodou Monte Carlo můžeme získat odhad pravděpodobnostního rozdělení $p_Y(y)$ pro řešení Y a s ním i odhady střední hodnoty a rozptylu. V této části nás bude zajímat, jak se na zmíněném rozptylu podílejí parametry $\boldsymbol{X} = (X_1,\ldots,X_m)$.

5.3.1 Indexy globální citlivosti

Podobně jako se v analýze lokální citlivosti definují tzv. ***lokální citlivosti*** (resp. ***indexy lokální citlivosti***) parametrů modelu, zavádíme v globální analýze citlivosti tzv. ***indexy globální citlivosti*** příslušné parametrům modelu. Princip spočívá v rozložení rozptylu řešení $V(Y)$ způsobeného neurčitostmi v parametrech $X_1,\ldots,X_m$, a to na rozptyly tohoto řešení způsobené jak jednotlivými parametry, tak jejich interakcemi.

Vlivem prvního řádu parametru X_i na řešení Y rozumíme rozptyl řešení Y způsobený parametrem X_i, tj. výraz:

$$V_i = V_{X_i}(E_{X_{-i}}(Y|X_i)), \tag{5.8}$$

kde zápis X_{-i} vyjadřuje všechny parametry X_j, $j = 1,\ldots,m$ vyjma parametru X_i. Je třeba upozornit, že nelze vliv prvního řádu definovat výrazem $V_{X_{-i}}(Y|X_i)$, neboť bychom jednak určovali rozptyl řešení při zafixovaném parametru X_i na jedné hodnotě (ten však může nabývat kterékoli hodnoty z jistého intervalu hodnot) a jednak by se mohlo stát, že $V_{X_{-i}}(Y|X_i) > V(Y)$. Naproti tomu pro výše definovaný vliv parametru X_i výrazem (5.8) platí (viz [19]) požadovaná nerovnost $V_i \leq V(Y)$.

Relativní hodnotu vlivu prvního řádu (v porovnání s celkovým rozptylem řešení) pak nazveme ***indexem citlivosti prvního řádu***, označíme S_i a definujeme vztahem:

$$S_i = \frac{V_i}{V(Y)}. \tag{5.9}$$

Vliv druhého řádu parametrů X_i a X_j (pro $i \neq j$) na řešení Y definujeme vztahem (viz [20]):

$$V_{ij} = V_{X_{i,j}}(E_{X_{-(i,j)}}(Y|X_i, X_j)) - V_i - V_j, \tag{5.10}$$

kde podobně $X_{-(i,j)}$ vyjadřuje všechny parametry X_k, $k = 1,\ldots,m$ vyjma parametrů X_i a X_j. Od celkového rozptylu způsobeného parametry X_i a X_j odečítáme na pravé straně (5.10) vlivy prvního řádu, abychom dostali pouze vliv způsobený interakcí mezi X_i a X_j.

Následně pak lze označit S_{ij} ***index citlivosti druhého řádu*** a definovat jej vztahem:

$$S_{ij} = \frac{V_{ij}}{V(Y)}. \tag{5.11}$$

Tímto způsobem bychom mohli pokračovat až po zavedení vlivu (indexu citlivosti) *m*-tého řádu. Komplexně jsme tak popsali vlivy parametrů $X_1,\ldots,X_m$ na rozptyl řešení *Y*, přičemž pro rozložení rozptylu řešení *V*(*Y*) na jednotlivé vlivy platí (viz např. [21]):

$$V(Y) = \sum_{i=1}^{m} V_i + \sum_{i=1}^{m-1} \sum_{\substack{j=2 \\ i<j}}^{m} V_{ij} + \ldots + V_{12\ldots m} \tag{5.12}$$

a tedy lze odvodit, že platí i:

$$1 = \sum_{i=1}^{m} S_i + \sum_{i=1}^{m-1} \sum_{\substack{j=2 \\ i<j}}^{m} S_{ij} + \ldots + S_{12\ldots m}. \tag{5.13}$$

V praxi většinou nepočítáme všechny indexy citlivosti (všech řádů), neboť je to výpočetně náročné (počet indexů citlivosti je $2^m - 1$) a ani to není nutné. Zpravidla nás zajímá, jak ovlivňuje výstup modelu každý parametr samostatně (tj. index citlivosti prvního řádu) a které parametry jsou nevýznamné (a je případně možné je považovat pro model jako exogenní, a tedy v daném/ve sledovaném okamžiku je nahradit konstantou).

Nízký index citlivosti prvního řádu ještě nemusí znamenat, že je příslušný parametr nevýznamný. Jeho vliv se totiž může projevit především v interakcích s jinými parametry. Z toho důvodu definujeme celkový index citlivosti S_{T_i} vztahem (viz [20]):

$$S_{T_i} = 1 - \frac{V_{X_{-i}}(E_{X_i}(Y|X_{-i}))}{V(Y)} = \frac{E_{X_{-i}}(V_{X_i}(Y|X_{-i}))}{V(Y)}. \tag{5.14}$$

Příklad 5.4: Pro konkrétní představu výše popsaných indexů citlivosti uvažujme, že $m = 3$. Model je tedy zadán vztahem:

$$Y = f(X_1,X_2,X_3).$$

Řešení: Indexy citlivosti prvního řádu jsou S_1, S_2 a S_3. Indexy citlivosti druhého řádu jsou S_{12}, S_{13} a S_{23}. Index citlivosti třetího řádu je jediný, a to S_{123} . Celkové indexy citlivosti jsou $S_{T_1} = S_1 + S_{12} + S_{13} + S_{123}$, $S_{T_2} = S_2 + S_{12} + S_{23} + S_{123}$, $S_{T_3} = S_3 + S_{13} + S_{23} + S_{123}$.

5.3.2 Výpočet indexů globální citlivosti

Výše definované indexy citlivosti většinou nejsme schopni podle definovaných vztahů určit. Jednou z metod jejich výpočtu je tzv. ***Soboľova metoda***.

Hlavní myšlenkou této metody je dekompozice (též známá jako ANOVA dekompozice, viz [22]) funkce $f(X_1,\ldots,X_m)$ z rovnice (5.7) na ortogonální[101] sčítance:

$$f(X_1, \ldots, X_m) = f_0 + \sum_{i=1}^{m} f_i(X_i) + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} f_{ij}(X_i, X_j) + \cdots + f_{1,2,\ldots,m}(X_1, \ldots, X_m), \tag{5.15}$$

[101] Dva prvky jsou ortogonální, jestliže je jejich skalární součin roven 0.

kde

$$f_0 = E(f(\boldsymbol{X})), \quad f_i(X_i) = E_{\boldsymbol{X}_{-i}}(f(\boldsymbol{X})|X_i) - f_0,$$
$$f_{ij}(X_i, X_j) = E_{\boldsymbol{X}_{-(i,j)}}(f(\boldsymbol{X})|X_i, X_j) - f_i(X_i) - f_j(X_j) - f_0, \quad \ldots$$

Jelikož jsou členy pravé strany rovnice (5.15) ortogonální, dostáváme po aplikaci rozptylu na levou i pravou stranu rovnici (5.12), viz např. [19], [23].

Hledané rozptyly $V(Y), V_i, V_{ij}, \ldots$ pak můžeme vypočítat následovně:

$$V(Y) = \int_{\mathbb{R}^m} f^2(\boldsymbol{x}) \cdot p(\boldsymbol{x})\, d\boldsymbol{x} - f_0^2,$$
$$V_{i_1,\ldots,i_s} = \int_{\mathbb{R}^s} f^2_{i_1,\ldots,i_s}(x_{i_1}, \ldots, x_{i_s}) \cdot p_{i_1,\ldots,i_s}(x_{i_1}, \ldots, x_{i_s})\, dx_{i_1} \ldots dx_{i_s}.$$

kde $i_1, \ldots, i_s \in \{1, 2, \ldots, m\}, i_1 < \ldots < i_s$ a $p_{i_1,\ldots,i_s}(x_{i_1}, \ldots, x_{i_s})$ je sdružená hustota pravděpodobnosti náhodných veličin $X_{i_1}, \ldots, X_{i_s}$.

Předpokládejme, že jsme provedli simulaci metodou Monte Carlo, a máme tak k dispozici M realizací pro každou z náhodných veličin $X_1,\ldots,X_m$ a Y označené $x_{11},\ldots,x_{1M},\ldots,x_{m1},\ldots,x_{mM}$ a $y_1,\ldots,y_M$. K určení výše definovaných vlivů (rozptylů) lze využít následujících numerických odhadů[102] (viz [19]):

$$\widehat{f_0} = \frac{1}{M} \cdot \sum_{k=1}^{M} f(x_{1k}, \ldots, x_{mk}), \tag{5.16}$$
$$\widehat{V}(Y) = \frac{1}{M} \cdot \sum_{k=1}^{M} f^2(x_{1k}, \ldots, x_{mk}) - \left(\widehat{f_0}\right)^2, \tag{5.17}$$
$$\widehat{V_i} = \frac{1}{M} \cdot \sum_{k=1}^{M} f\left(x_{-ik}^{(1)}, x_{ik}^{(1)}\right) \cdot f\left(x_{-ik}^{(2)}, x_{ik}^{(1)}\right) - \left(\widehat{f_0}\right)^2, \tag{5.18}$$
$$\widehat{V_{-i}} = \frac{1}{M} \cdot \sum_{k=1}^{M} f\left(x_{-ik}^{(1)}, x_{ik}^{(1)}\right) \cdot f\left(x_{-ik}^{(1)}, x_{ik}^{(2)}\right) - \left(\widehat{f_0}\right)^2, \tag{5.19}$$

kde $x_{-ik} = x_{1k}, \ldots, x_{i-1,k}, x_{i+1,k}, \ldots, x_{mk}$.

Horní indexy $^{(1)}$ a $^{(2)}$ v rovnicích (5.18) a (5.19) vyjadřují, že pro získání odhadů $\widehat{V_i}$ a $\widehat{V_{-i}}$ potřebujeme ve skutečnosti $2 \cdot M$ realizací náhodných veličin $X_1,\ldots,X_m,Y$ rozdělených do dvou skupin po M hodnotách (pro každou z veličin). Index $^{(1)}$ značí, že příslušné realizace vybíráme z první skupiny, index $^{(2)}$ představuje, že příslušné realizace vybíráme ze druhé skupiny. Analogicky bychom mohli odvodit odhady pro zbylé rozptyly ($V_{ij}, V_{ijk}, \ldots$).

Přesnost odhadů (5.16) – (5.19) je možné určit podle chyby v metodě Monte Carlo, které jsme definovali dříve (viz (5.5), resp. (5.6)). Na členy v rovnicích (5.16) až (5.19) se totiž můžeme dívat jako na odhady příslušných středních hodnot, tj.:

[102] Opět vycházíme z toho, že máme k dispozici M realizací náhodného vektoru X a náhodné veličiny Y ze simulace metody Monte Carlo simulace.

$$\begin{aligned}
\widehat{f_0} &= \widehat{E}(f(\boldsymbol{X})), \\
\widehat{V}(Y) + \left(\widehat{f_0}\right)^2 &= \widehat{E}(f^2(\boldsymbol{X})), \\
\widehat{V}_i + \left(\widehat{f_0}\right)^2 &= \widehat{E}\left(f\left(X_{-i}^{(1)}, X_i^{(1)}\right) \cdot f\left(X_{-i}^{(2)}, X_i^{(1)}\right)\right), \\
\widehat{V}_{-i} + \left(\widehat{f_0}\right)^2 &= \widehat{E}\left(f\left(X_{-i}^{(1)}, X_i^{(1)}\right) \cdot f\left(X_{-i}^{(1)}, X_i^{(2)}\right)\right).
\end{aligned}$$

Odhad chyby celkového rozptylu lze například vyjádřit jako:

$$\widehat{v}_M\left(\widehat{V}(Y)\right) = \frac{1}{M \cdot (M-1)} \cdot \left(\sum_{k=1}^{M} f^4\left(\boldsymbol{X}_{(k)}\right) - M \cdot \left(\widehat{E}(f^2(\boldsymbol{X}))\right)^2\right) + \widehat{v}_M\left(\left(\widehat{f_0}\right)^2\right). \quad (5.20)$$

Přestože se zvyšujícím se M jsou odhady (5.16) až (5.19) stále blíže skutečným hodnotám, M bude vždy konečné a odhady nepřesné.

Uvažujme, že proměnná X_i vůbec neovlivňuje řešení $Y = f(X)$. Její příslušný index citlivosti tak má být $S_i = 0$.

Podle (5.18) platí vztah:

$$\widehat{V}_i = \frac{1}{M} \cdot \sum_{k=1}^{M} f\left(x_{-ik}^{(1)}, x_{ik}^{(1)}\right) \cdot f\left(x_{-ik}^{(2)}, x_{ik}^{(1)}\right) - \left(\widehat{f_0}\right)^2$$

Absolutní hodnota výše uvedeného výrazu však nikdy nebude nižší, než je hodnota výrazu:

$$\left|\frac{1}{M} \cdot \sum_{k=1}^{M} f\left(\boldsymbol{x}_{(k)}^{(1)}\right) \cdot f\left(\boldsymbol{x}_{(k)}^{(2)}\right) - \left(\widehat{f_0}\right)^2\right|. \quad (5.21)$$

Homma a Saltelli (viz [24]) proto doporučují korekční člen (5.21) aplikovat bez absolutní hodnoty[103] na jednotlivé rozptyly $V_{i_1,\ldots,i_s}$ (pro $i_1,\ldots,i_s \in \{1,2,\ldots,m\}$, $i_1 < \ldots < i_s$) tak, že od rozptylů prvního řádu (V_i) se korekční člen odečte a k rozptylům druhého řádu (V_{ij}) se přičte. Od rozptylů třetího řádu (V_{ijk}) se odečte atd. Vlivy všech řádů jsou tímto nedostatkem přesnosti ovlivněny stejně, takže pokud odečteme korekční člen od V_i i od V_j, odečítáme jej od V_{ij} již dvakrát (viz definice V_{ij}), a proto v tomto případě jednou korekční člen přičteme. Z tohoto důvodu se znaménka střídají (při aplikaci korekčního členu na již vypočtené rozptyly $V_{i_1,\ldots,i_s}$).

Příklad 5.5: Uvažujme model tvaru:

$$Y = \sin(X_1) + 7 \cdot \sin^2(X_2) + \frac{1}{10} \cdot X_3^4 \cdot \sin(X_1)$$

kde X_1, X_2, X_3 jsou všechny náhodné veličiny s rovnoměrným rozdělením pravděpodobnosti na intervalu $(-\pi, \pi)$. Proveďme globální analýzu citlivosti s využitím aplikačního software Maple i MATLAB.

[103] Absolutní hodnota byla přidána, aby příslušná věta měla správný matematický význam. Znaménko členu v absolutní hodnotě je však podstatné!

***Řešení*:** Nejprve určíme střední hodnotu a rozptyl řešení *Y*. Výpočítáme je podle známých vztahů [37] nebo určíme jejich hodnoty pomocí balíku `Statistics` v Maple:

```
with(Statistics):
X[1] := RandomVariable(Uniform(-Pi, Pi)):
X[2] := RandomVariable(Uniform(-Pi, Pi)):
X[3] := RandomVariable(Uniform(-Pi, Pi)):
Y := sin(X[1])+7*sin(X[2])^2+(1/10)*X[3]^4*sin(X[1]):
Mean(Y);
Variance(Y);
```

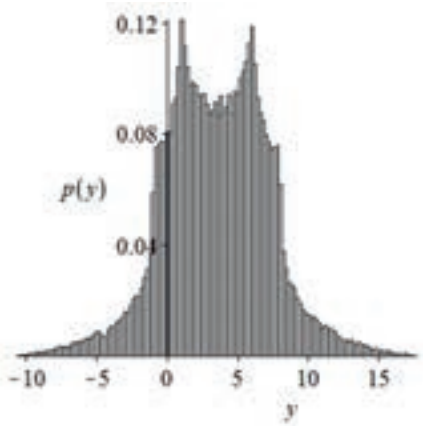
Obdržíme tak:

$$E(Y) = \frac{7}{2}, \quad V(Y) = \frac{1}{1800} \cdot \pi^8 + \frac{1}{50} \cdot \pi^4 + \frac{53}{8}.$$

Pravděpodobnostní rozdělení veličiny *Y* můžeme odhadnout z výše vygenerovaných náhodných veličin `X[1], X[2], X[3]`, následným vyhodnocením `Y` a jeho zobrazením v histogramu pro $M = 10^5$ příkazem

```
Histogram(Sample(Y, 100000), averageshifted = 3);
```

který je uveden na obrázku 5.1.



Obr. 5.1 *Histogram vygenerovaných řešení systémem Maple.*

Podle vztahu (5.8) vypočítáme pomocí programu Maple, který je uveden v Příloze, hodnoty vlivů V_i prvního řádu jednotlivých parametrů X_i na řešení *Y* (pro $i = 1, 2, 3$):

$$V_1 = V_{X_1}(E(Y|X_1)) = V\left(\sin(X_1) + \frac{\pi^4}{50} \cdot \sin(X_1)\right) = \frac{1}{5000} \cdot \pi^8 + \frac{1}{50} \cdot \pi^4 + \frac{1}{2},$$

$$V_2 = V_{X_2}(E(Y|X_2)) = V\left(-7 \cdot \cos^2(X_2)\right) = \frac{49}{8},$$

$$V_3 = V_{X_3}(E(Y|X_3)) = V\left(\frac{7}{2}\right) = 0.$$

Když vlivy V_i (pro $i = 1, 2, 3$) zaokrouhlíme na čtyři desetinná místa, dostaneme:

$$V_1 = 4{,}3458 \qquad V_2 = 6{,}125 \qquad V_3 = 0.$$

Následně vypočítáme programem Maple v Příloze podle vztahu (5.9) indexy globální citlivosti S_1, S_2, S_3 pro parametry X_1, X_2, X_3, které zaokrouhlíme na čtyři desetiná místa:

$$S_1 = \frac{V_1}{V(Y)} \doteq 0,3139; \quad S_2 = \frac{V_2}{V(Y)} \doteq 0,4424; \quad S_3 = \frac{V_3}{V(Y)} = 0$$

V tomto jednoduchém příkladu bylo možné získat indexy globální citlivosti pomocí Maple analyticky zavedenými vztahy (5.8) - (5.12). Z této analýzy vyplývá, že největší vliv na řešení modelu Y má druhý parametr X_2.

Pro názornost vypočítejme nyní indexy globální citlivosti pomocí Soboľovy metody a výsledky porovnáme s předchozími výpočty. Programy opět vytvoříme v Maple a MATLABu a uvedeme je v Příloze.

Nejprve musíme vygenerovat dvě skupiny realizací parametrů X_1, X_2, X_3. Nechť každá skupina obsahuje $M = 10^5$ realizací pro každý z parametrů. Zvolíme jednu ze skupin, spočteme M vyhodnocení modelu Y (pro jednotlivé realizace parametrů X_1, X_2, X_3) a určíme jejich průměr (tj. střední hodnotu) a rozptyl a zaokrouhlíme je na čtyři desetiná místa. Z programu Maple i Matlab v Příloze obdržíme[104]:

$$\widehat{E}(Y) \doteq 3,4958; \quad \widehat{V}(Y) \doteq 13,9476; \quad (V(Y) \doteq 13,8446)$$

Nyní přistoupíme k výpočtu odhadů vlivů jednotlivých parametrů definovaných vztahem (5.18). Pro výpočet vlivu V_1 parametru X_1 vypočteme dalších M vyhodnocení modelu Y, a to tak, že budeme brát realizace parametru X_1 z první skupiny (té dříve zvolené) a realizace parametrů X_2, X_3 z druhé skupiny na začátku vygenerovaných realizací. Zcela analogicky vypočteme jiných M vyhodnocení modelu Y jak pro určení vlivu V_2 parametru X_2, tak pro určení vlivu V_3 parametru X_3. Získáme níže uvedené hodnoty:

$$\widehat{V}_1 \doteq 4,3680; \quad \widehat{V}_2 \doteq 6,1619; \quad \widehat{V}_3 \doteq -0,0561$$

Po výpočtu vlivů V_1, V_2, V_3 jednotlivých parametrů X_1, X_2, X_3 můžeme přistoupit k výpočtu indexů globální citlivosti prvního řádu podle vztahu (5.9):

$$\widehat{S}_1 \doteq 0,3131; \quad \widehat{S}_2 \doteq 0,4418; \quad \widehat{S}_3 \doteq -0,0040$$

Jestliže aplikujeme korekční člen (5.21), jehož hodnota je přibližně −0,0571, pak obdržíme:

$$\widehat{V}_1 \doteq 4,4250; \quad \widehat{V}_2 \doteq 6,2190; \quad \widehat{V}_3 \doteq 0,0010;$$
$$\widehat{S}_1 \doteq 0,3173; \quad \widehat{S}_2 \doteq 0,4459; \quad \widehat{S}_3 \doteq 0,0001$$

Porovnáním s předchozími výpočty vidíme, že Soboľovova metoda dává výsledky, které se liší až na třetím desetinném místě.

Zdrojové kódy pro výpočet výše uvedených hodnot v příkladu 5.5 v Maple i MATLABu jsou uvedeny v Příloze.

[104] V závorce je uvedená pro porovnání zaokrouhlená hodnota přesné hodnoty $V(Y)$.

6 Aplikační software pro vědecké výpočty

V této kapitole se stručně seznámíme s aplikačním softwarem Maple, MATLAB a SPSS, který je používán na Masarykově univerzitě ve výuce matematiky a statistiky, matematickém a stochastickém modelování, symbolických, numerických a statistických výpočtech a počítačové simulaci.

6.1 Vědecké výpočty s využitím Maple

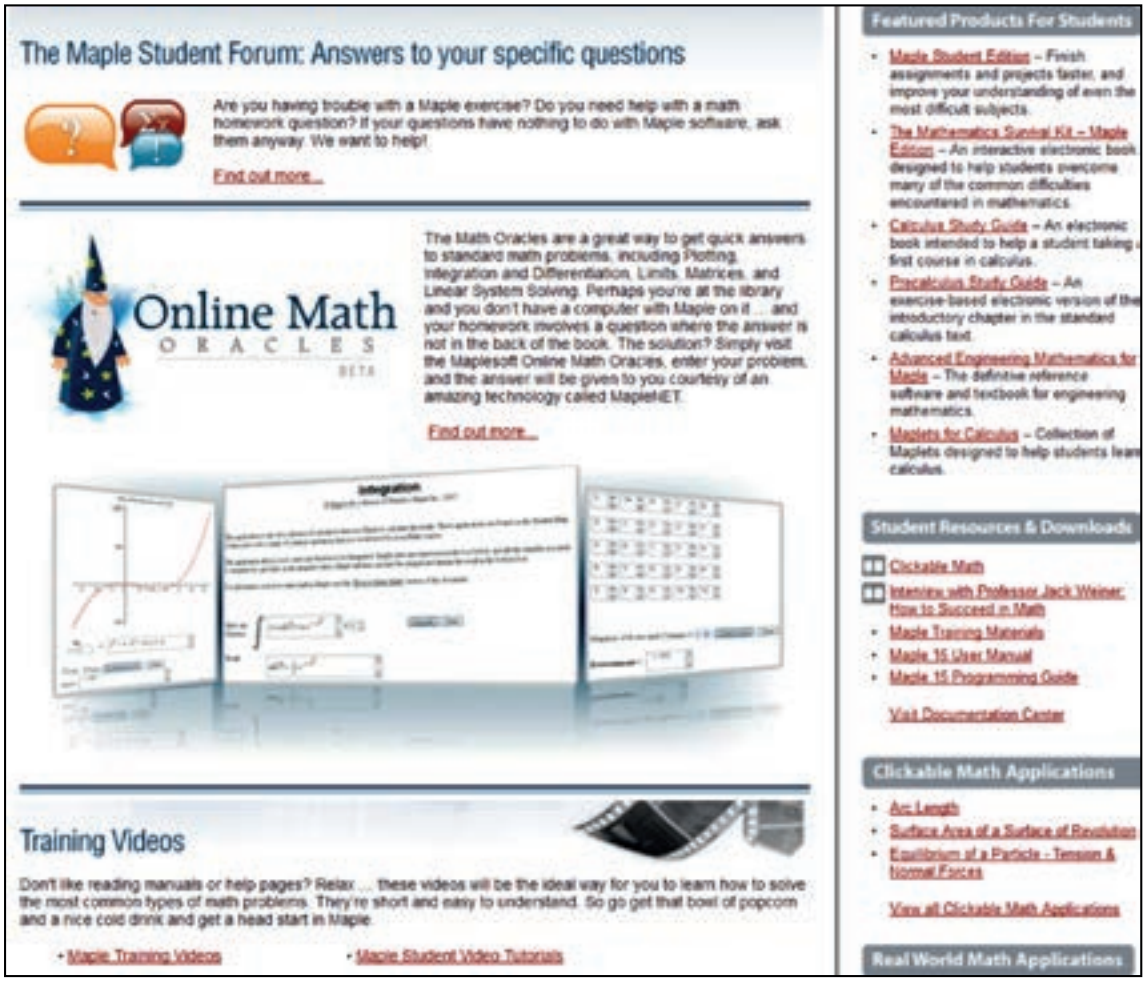
Aplikační software Maple patří do skupiny systémů počítačové algebry. Umožňuje symbolické i numerické výpočty a slouží především k tvorbě speciálních dokumentů, prezentací, interaktivních výpočetních modulů v prostředí Maple. Tento systém též obsahuje komponenty podporující výuku matematiky.

Maple je přibližně třicet let vyvíjený kanadskou společností Maplesoft Inc., jejíž webové stránky[105] poskytují rozsáhlé informace o tomto software dalších s ním příbuzných softwarech, jako jsou MapleSim, MapleNet, Maple T.A. a mnoha dalších ICT nástrojích a programech, viz obrázek 6.1.

Obr. 6.1 *Web společnosti Maplesoft*[105].

[105] http:www.maplesoft.com

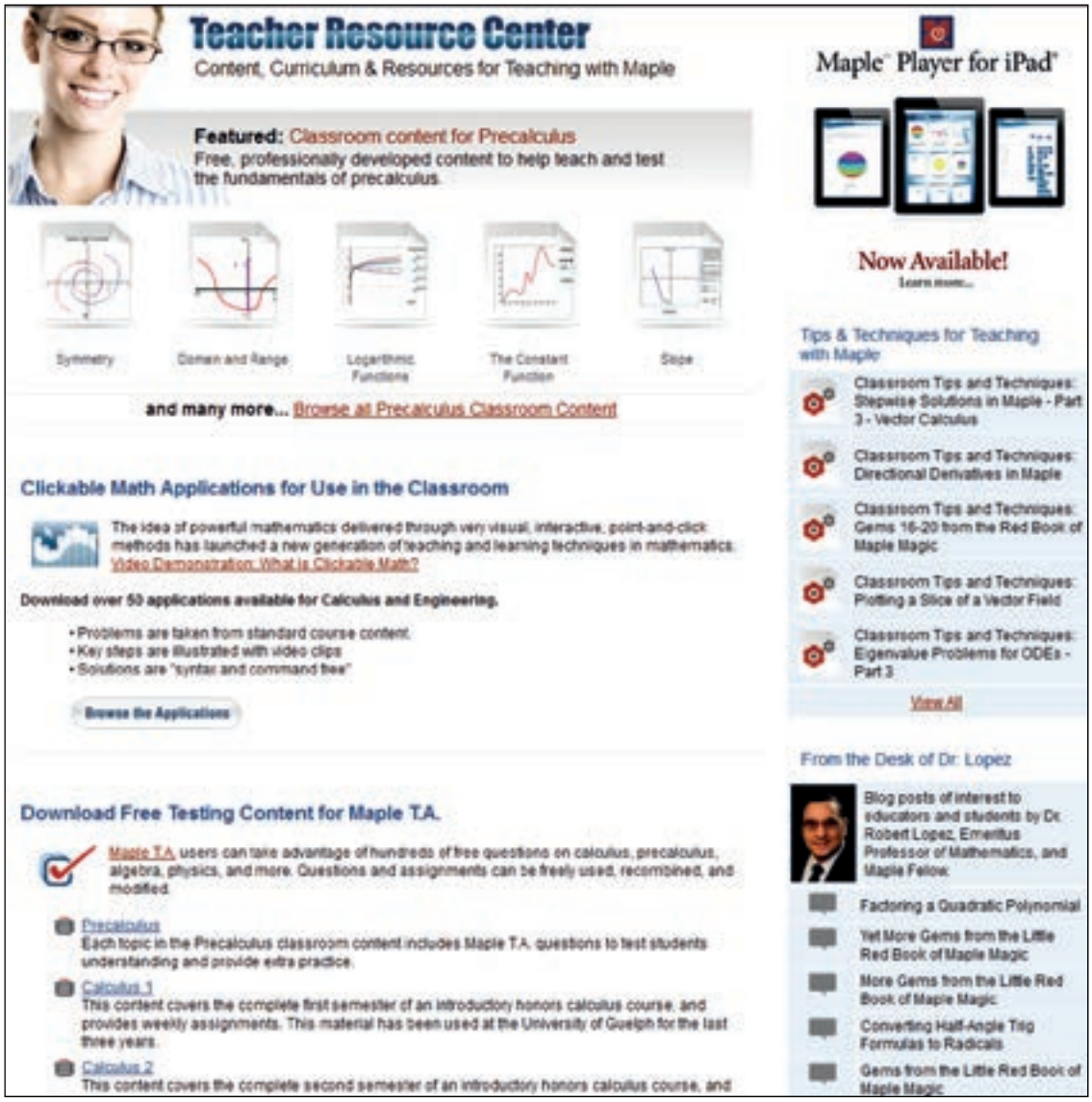
Na těchto webových stránkách je možno se zaregistrovat, a získat tak autorizovaný přístup k informacím pro využívání Maple v ***technických vědách***[106] (*Engineering*), ve ***výuce***[107] (*Education*) a v různých oblastech ***aplikovaného výzkumu***[108] (*Applied Research*). Registrace poskytuje i možnost zajistit si odběr elektronického časopisu Maple Reporter[109] a příjem zpráv o pořádaných webinářích[110].
Webové stránky mimo jiné obsahují i ***Aplikační centrum***[111] (*Application Center*), z nějž si může každý zaregistrovaný uživatel při dodržení platných podmínek stáhnout ukázkové programy demonstrující použití systému Maple při řešení mnoha stovek různých matematických i technických problémů. Stránky dále obsahují ***Maple T.A. centrum***[112] (*Maple T.A. Content Center*), které nabízí pro celou řadu vědních oborů stovky dokumentů a výpočetních zápisníků (worksheets), výukových kurzů a on-line ověřovacích testů.



Obr. 6.2 *Student Help Centrum*[113].

[106] http://www.maplesoft.com/solutions/engineering/
[107] http://www.maplesoft.com/solutions/education/
[108] http://www.maplesoft.com/solutions/applied_research/
[109] http://www.maplesoft.com/company/reporter/index.aspx
[110] http://www.maplesoft.com/company/webinars/index.aspx
[111] http://www.maplesoft.com/Applications/index.aspx
[112] http://www.maplesoft.com/tacontent/index.aspx

Webové stránky Maplesoft poskytují dále přístup i do ***Studentského centra***[113] (*Student Help Center*), viz obrázek 6.2 a ***Učitelského centra*** (*Teachers Help Center*), obrázek 6.3, kde si zaregistrovaný student nebo pedagog může stáhnout mnoho studijních a výukových materiálů.



Obr. 6.3 *Teacher Help Centrum*[114].

Dalšími významnými zdroji informací o Maple je diskuzní fórum uživatelů Maple[115] (*Maple Primes*), které umožňuje vzájemnou výměnu informací a názorů jeho uživatelů týkajících se využití Maple a dalších příbuzných softwarů, jako jsou MapleSim, Maple T.A. další.

Důležitým informačním pramenem pro české a slovenské uživatele je web distributora systému Maple pro Českou a Slovenskou republiku[116], kde lze nalézt převážnou většinu dokumentů a návodů pro používání Maple v českém jazyce[117].

[113] http://www.maplesoft.com/studentcenter/index.aspx
[114] http://www.maplesoft.com/TeacherResource/
[115] http://www.mapleprimes.com/
[116] http://www.maplesoft.cz

6.1.1 Výpočetní prostředí Maple

V následujícím textu se budeme zabývat dosud poslední verzí software Maple 15.

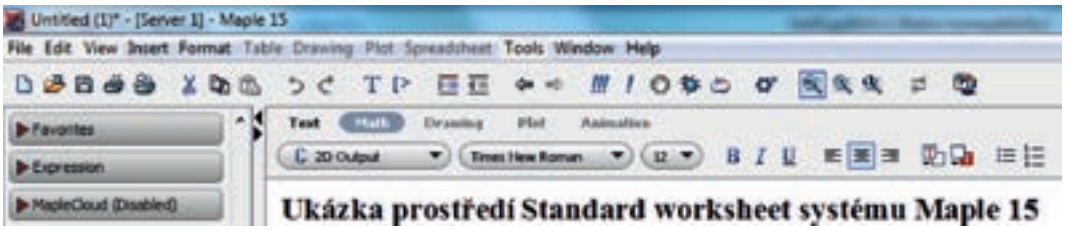
Se softwarem Maple je možné pracovat několika různými způsoby, které volíme při spuštění programu ze startovacího menu počítače nebo kliknutím na příslušnou ikonu na ploše.

Výpočetní prostředí a ovládání Maple jsou dostatečně prezentovány v [2], a proto jejich popis nebudeme v této kapitole podrobně upřesňovat. Budeme přepokládat, že čtenář je s publikací [2] seznámen. Zaměříme se však na důležité aspekty využití Maple v oblasti vědeckých výpočtů.

Grafické uživatelské rozhraní Maple zvané ***Standard Worksheet*** (dále zápisník) se spustí ze startovacího menu počítače výběrem položky (Programy > Maple 15) nebo kliknutím na ikonu Maple 15 na ploše obrazovky.

Toto prostředí poskytuje veškeré možnosti využití systému Maple pro vytváření elektronických dokumentů (zápisníků) zobrazujících matematické výpočty, matematické texty a komentáře spolu s propracovanou počítačovou grafikou. Některé výpočty je možné v zápisníku „schovat", a tedy nechat „odkryté" jen takové pasáže, které uživatel považuje za nejdůležitější pro to, aby dokument poskytoval potřebné informace například o vedení výpočtu, formalizaci a posloupnosti myšlenek apod. Jelikož vytvořené dokumenty jsou interaktivní, tj. v jistém smyslu „živé", může si uživatel sám operativně upravovat předdefinované hodnoty parametrů, vyhodnocovat příkazy, modifikovat grafické výstupy, a tak aktualizovat stávající výsledky nebo získávat výsledky nové.

Menu zápisníku v Maple má tři vodorovné lišty: ***hlavní menu*** (zcela nahoře), ***nástrojovou lištu*** (pod hlavním menu) a ***kontextovou lištu*** (pod nástrojovou lištou). Dále obsahuje ***palety nástrojů***, které se automaticky zobrazují v levém svislém bloku, viz obrázek 6.4. Uživatelské rozhraní zápisníku nabízí blok pro palety i po pravé straně obrazovky (automaticky zavřený), kam je možné některé z nich (ale i všechny) přesunout. Vlastní pracovní pole zápisníku – ***dokument***, slouží uživateli k zadávání příkazů systému, textů a provádění veškerých výpočetních a grafických akcí. Pracovní pole je možné zobrazit na celou obrazovku skrytím palet, nástrojové lišty a kontextové lišty (kliknutím na příslušné položky v záložce View hlavního menu), viz obrázek 6.4.



Obr. 6.4 *Menu zápisníku Maple.*

Jednou z klíčových výhod Maple je jeho stabilita při provádění symbolických matematických výpočtů. Při práci s Maple si můžeme dovolit ponechat přesné vyčíslení hodnoty zpracovaného a upravovaného výrazu až do „poslední chvíle". Maple je totiž schopen pracovat s nedefinovanými, či spíš s obecně definovanými, ale zatím konkrétně číselně nezadanými prvky nebo s obecně zadanými proměnnými a výrazy jako se symboly, kterým může být později přiřazena konkrétní hodnota.

Maple provádí přesně numerické výpočty s racionálními (zejména s celými) a některými iracionálními čísly. Každý zadaný matematický výraz se snaží co nejvíce zjednodušit (např.

[117] http://www.maplesoft.cz/navody-publikace

zlomek zkrátit a převést na základní tvar, upravit algebraický výraz apod.), avšak nikoli za cenu ztráty přesnosti. To znamená, že například racionální čísla (zlomky) udržuje stále v jejich základním tvaru. S konstantami π, *e* a dalšími, s odmocninami a jinými výrazy pracuje jako se symboly. Tímto je v sytému Maple zaručena absolutní přesnost numerických výpočtů s racionálními (zejména s celými) a některými iracionálními čísly na rozdíl od provádění obdobných výpočtů v aplikačních softwarech MATLAB a SPSS.

Maple rozeznává přesná čísla (mezi něž patří i zmíněné symboly π a ***e***, zlomky atp.) a čísla typu ***floating-point***, tj. čísla v pohyblivé řádové čárce [3]. S reálnými čísly pracuje v pohyblivé řádové čárce. Jestliže systému Maple zadáme výraz, v němž některý z jeho podvýrazů bude typu ***floating-point***, pak Maple na celý výraz pohlíží jako na výraz tohoto typu a výsledky výpočtů zaokrouhlí. To nejlépe uvidíme na dalších příkladech na obrázku 6.5.

$$\frac{2}{3}+\frac{3}{2}=\frac{13}{6}, \quad \textit{ale} \quad \frac{2}{3}+1.5=2.166666667$$

$$\sqrt{2}+\sqrt{3}=\sqrt{2}+\sqrt{3}, \quad \textit{ale} \quad \sqrt{2.0}+\sqrt{3}=1.414213562+\sqrt{3}$$

Obr. 6.5 *Přesná čísla a čísla typu floating-point.* Zdroj [2].

V případě, kdy potřebujeme znát přibližnou hodnotu reálného nebo racionálního čísla v pohyblivé řádové čárce, pak počet platných cifer mantisy lze v Maple stanovit systémovou proměnnou `Digits`, která je standardně nastavena na hodnotu 10. Všechny výpočty, při nichž je nutné čísla zaokrouhlovat, provádí proto Maple s přesností na deset platných míst. Přitom proměnnou `Digits` můžeme nastavit na velmi vysokou číselnou hodnotu (přirozené číslo). Omezení, jak vysoké takové číslo může být, zjistíme v Maple příkazem `kernelopts(maxdigits)`. Pro představu uveďme, že ve stávající verzi Maple 15 je toto číslo rovné 268 435 448. Tedy jde o možnost pracovat s čísly s více než dvě stě šedesáti osmi miliony platných cifer, se kterými se systém dokáže vypořádat.

V současné době počítače disponují větším počtem jader v CPU. Maple proto umožňuje paralelní režim pro své matematické výpočetní jádro (*math engine*). V paralelním režimu tak zpracovává i více výrazů najednou, viz sedmá kapitola. V počítači s více jádry v CPU mu to dovoluje řešit problémy rychleji.

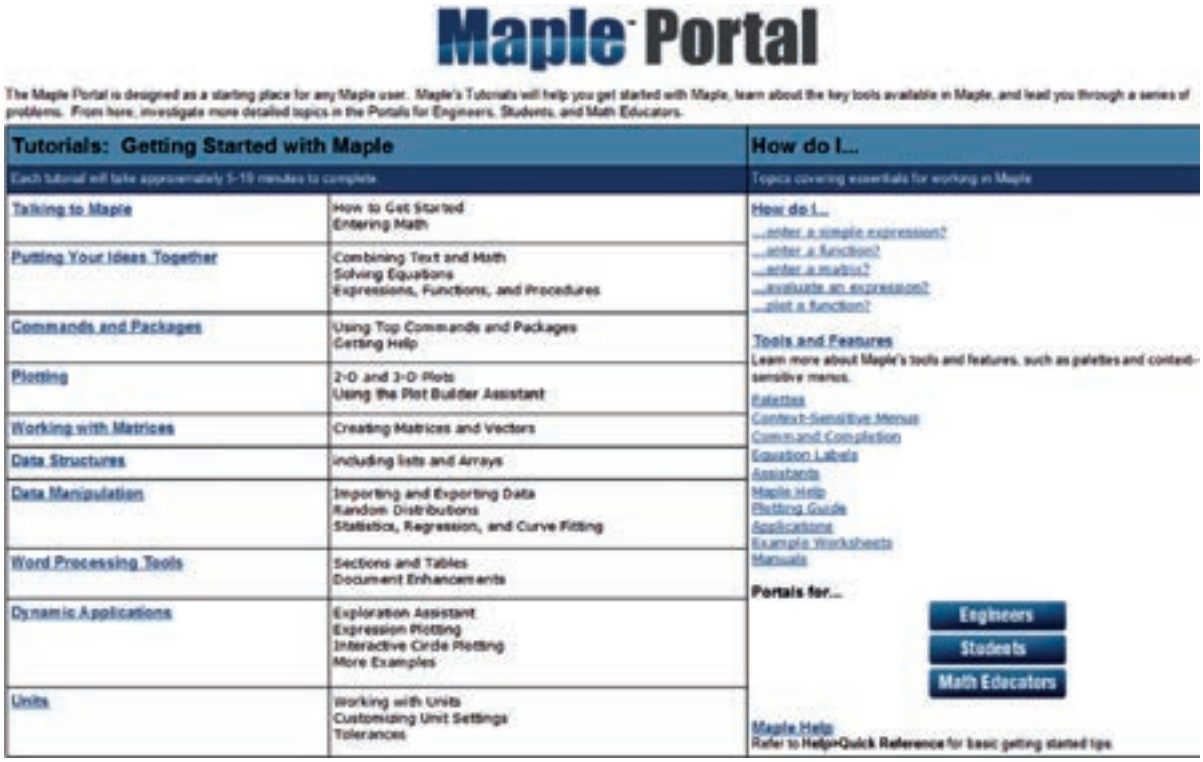
6.1.2 Nápověda v Maple

Nápověda a pracovní manuály k obslužnosti v současnosti hrají významnou úlohu každého systému. Maple má rozsáhlou nápovědu, které lze využít při vyhledávání již vytvořených algoritmů pro řešení nejrůznějších vědeckých problémů.

Novinkou systému je ***Maple Portal***. Lze jej spustit aktivací samostatné ikony (Maple Portal má vlastní ikonu na ploše obrazovky počítače) nebo cestou nápovědy v hlavním menu Maple (Help > Manuals, Resources, and more > Maple Portal), viz obrázek 6.6.

Maple Portal slouží jako pomocník všem novým (a nejen těm) uživatelům systému Maple. Je v něm možné rychle najít detailní popis práce se systémem od řešení nejjednodušších problémů až po řešení velmi složitých úloh.

Z nápovědy Maple 15 je možno vyvolat i anglické manuály: ***User manual; Programming Guide*** a ***Getting Started with Maple Toolboxes,*** podrobně popisující možnosti systému Maple 15. Vyvoláme je opět cestou nápovědy v hlavním menu Maple (Help > Manuals, Resources, and more > Manuals) a pak kliknutím na zvolený manuál.



Obr. 6.6 *Maple Portal.*

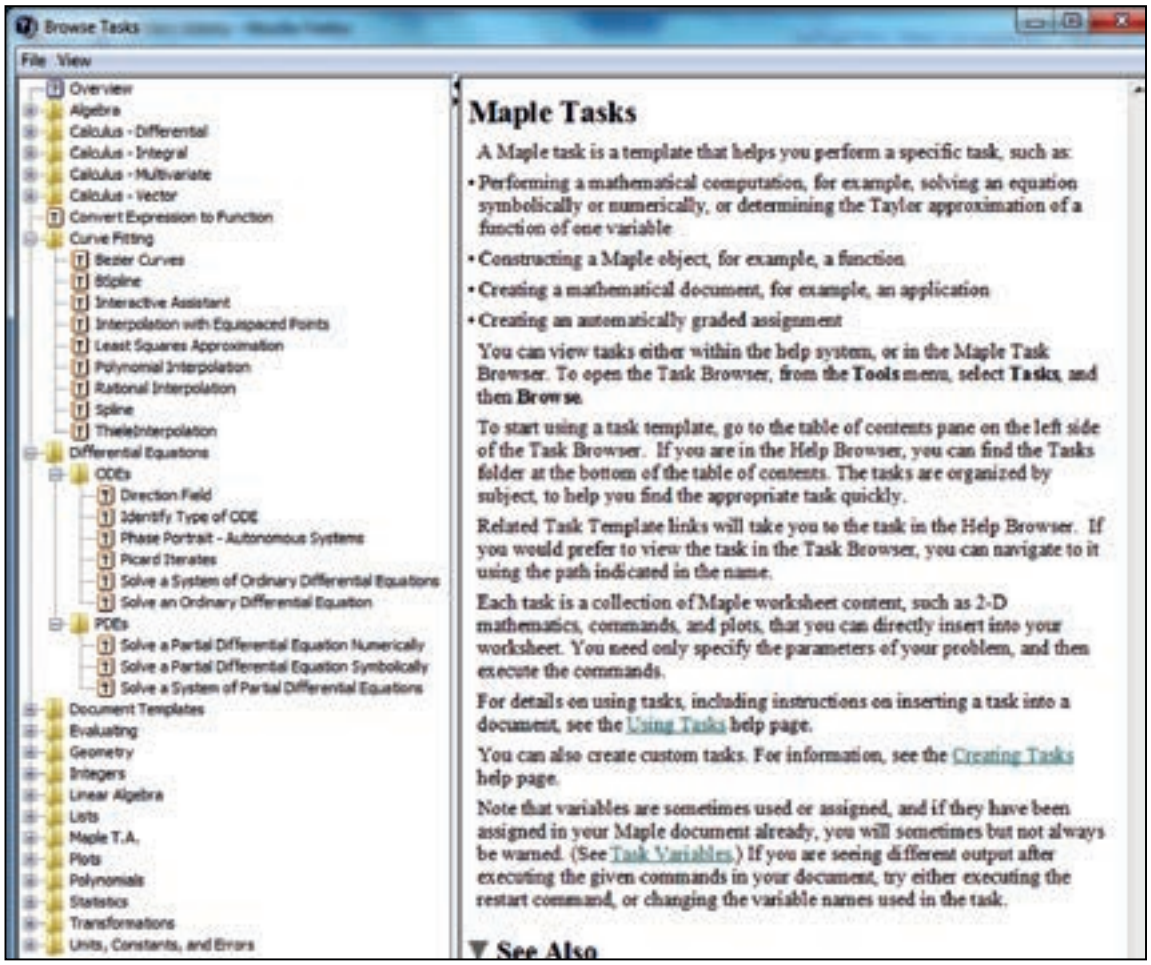
Maple je koncipován tak, aby kromě pohodlné interaktivní práce umožňoval i programování aplikací. Programovací jazyk obsahuje všechny nezbytné příkazy pro psaní programů, jako jsou podmíněné příkazy, větvicí příkazy, cykly a podobně. Přitom přes jednoduchost, srozumitelnost a snadnou uživatelskou zvladatelnost je jazyk Maple úplným programovacím jazykem čtvrté generace, ve kterém je možné vyvíjet i velmi složité aplikace. Jeho manuál ***Programming Guide*** je možno vyvolat z nápovědy v hlavním menu Maple (Help > Manuals, Resources, and more > Manuals > Programming Guide) a klinutím na něj. Obsahuje podrobný popis jazyka Maple, programování v něm, včetně paralelního programovaní, ladění a testování programů a měření jeho výpočetní náročnosti a výkonnosti.

Z nápovědy je možno vyvolat i již vytvořené programy (řešené příklady) demonstrující výpočetní možnosti systému Maple. Lze je vyvolat (pomocí nápovědy) v hlavním menu Maple (Help > Manuals, Resources, and more > Applications and Examples) a následným kliknutím na zvolený příklad. Verze Maple 15 poskytuje celkem dvanáct takových řešených příkladů

Maple poskytuje také již připravené „pomocné nástroje" pro řešení úloh ve vědeckých výpočtech. Jsou to tzv. ***Pomocníci*** (*Assistants*), ***Instruktoři*** (*Tutors*) a ***Úlohy*** (*Tasks*), které vyvoláme z hlavního menu Maple (Tools > Assistants nebo Tools > Tutors anebo Tools > Tasks).

Úlohy (*Tasks*) zobrazují na příkladech, jak řešit různé úlohy. Zobrazí se vyvoláním z hlavního menu Maple (Tools > Tasks > Browse), viz obrázek 6.7.

Pomocníci (*Assistants*) mají např. nástroje pro hledání funkční závislosti v datech, optimalizaci funkcí, řešení diferenciálních rovnic a další. Pro daný typ úlohy mají implementováno několik často používaných algoritmů. Po vyvolání provedou uživatele nastavením a specifikací parametrů úlohy a zvolenou metodou úlohu vyřeší.



Obr. 6.7 *Přehled řešených úloh v Maple 15.*

Instruktoři (*Tutors*) provedou uživatele řešenou problematikou pomocí jednoduchých názorných příkladů. Ukažme to na příkladu numerického řešení Cauchyova počátečního problému.

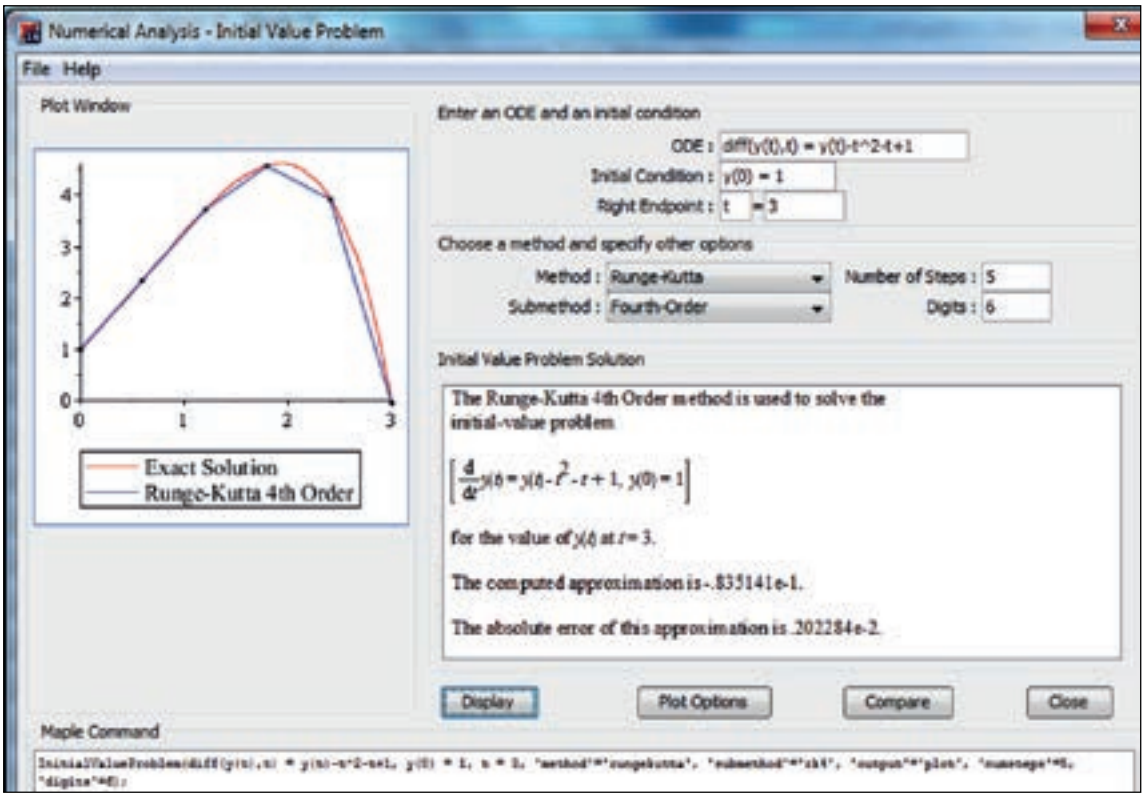
Příklad 6.1: Nalezněme numerické řešení Cauchyova počátečního problému

$$y'(t) = y(t) - t^2 - \mathrm{t} + 1,\ y(0) = 1 \qquad (6.1)$$

na intervalu <0; 3> pomocí metody Runge-Kutta čtvrtého řádu, viz [17], v bodě $t = 3$ v pěti krocích s výstupem na šest cifer mantisy.

Řešení: Z hlavního menu Maple vyvoláme (Tools > Tutors > Numerical Analysis > Initial Value Problem) instruktora pro numerickou analýzu počátečního problému, která spustí Maplet, viz obrázek 6.8. V tomto Mapletu zapíšeme v notaci příkazů Maple diferenciální rovnici (6.1) v poli ODE: `diff(y(t),t) = y(t)-t^2-t+1`, v poli počáteční podmínku (*Initial condition*): `y(0) = 1` a koncový bod (*Right Endpoint*): `t = 3`.

Dále vybereme z nabídky numerických metod metodu Runge-Kutta čtvrtého řádu [17] Method: `Runge-Kutta` a Submethod: `Fourth-Order`, počet kroků metody (*Number of steps*): 5 a počet cifer mantisy (*Digits*): 6.



Obr. 6.8 *Instruktorský Maplet pro numerické řešení počátečního problému (6.1) v Maple 15.*

Po kliknutí na tlačítko Display obdržíme výsledek, který je prezentován výše na obrázku 6.8. Tj. hodnota numerického řešení počátečního problému (6.1) v bodě $t = 3$ je rovna hodnotě $-0{,}835141 \cdot 10^{-1}$ a absolutní chyba této aproximace hodnotě $0{,}202284 \cdot 10^{-2}$, tedy může zatížit již druhé místo mantisy.

Téhož výsledku bychom dosáhli v Maple pomocí příkazu:

```
InitialValueProblem(diff(y(t),t)=y(t)-t^2-t+1,y(0)=1,t=3,
'method'='rungekutta','submethod'='rk4','output'='plot',
'numsteps'=5,'digits'=6);
```

6.1.3 Knihovny (balíky) v Maple

Současná verze Maple 15 obsahuje sto devatenáct nejrůznějších knihoven (balíků), které pokrývají nejrůznější oblasti výpočtů. Uvedeme zde jen několik vybraných knihoven, které jsou důležité pro vědecké výpočty:

- Balík ***CUDA*** umožňuje Maple využít grafiky v NVIDIA(R) Compute Unified Device Architecture (CUDA)[118] hardware, aby zrychlil určité moduly v balíku Linear Algebra.
- Balík ***CurveFitting*** tvoří sada příkazů, které podporují prokládání křivek. S jejich pomocí lze vytvořit funkci, která vede v blízkosti nebo prochází množinou datových bodů. Většina příkazů v CurveFitting poskytuje interpolační funkce.
- Balík ***Database*** využívá dotazovací jazyk SQL pro komunikaci s databází. Využívá k tomu aplikaci ***Query Builder Maplet*** umožňující dotazy na databázi, aniž zná SQL.

[118] http://www.nvidia.com/object/cuda_home.html

Data jsou automaticky převáděna z příslušné databáze do formátu Maple, který umožňuje s nimi manipulovat v každém prostředí. Balík Database využívá Java Database Connectivity[119] (JDBC). Pro připojení k příslušné databázi má Maple zabezpečen přístup pomocí ovladače JDBC.

- Balík ***DEtools*** obsahuje příkazy, které podporují řešení diferenciálních rovnic.
- Balík ***ExcelTools*** umožňuje spolupráci se soubory aplikace MS Excel, především načítání hodnot z tabulek a jejich zápis.
- Balík ***Grid*** tvoří sada programovacích nástrojů pro paralelní výpočty. Zejména, Grid balík poskytuje podporu pro multiprocesorové paralelní výpočty. To se vztahuje i na distribuované multiprocesorové paralelní výpočty na clusteru nebo v síti.
- Balík ***LinearAlgebra*** nabízí procedury pro konstrukci a manipulaci s maticemi a vektory, výpočtu standardních maticových operací, výsledky dotazů a řešení problémů z oblasti lineární algebry.
- Balík ***LREtools*** obsahuje příkazy pro manipulaci a nalezení určitých typů řešení lineárních rekurentních rovnic.
- Balík ***Matlab*** obsahuje příkazy, které komunikují se systémem MATLAB a příkazy, které překládají kód MATLABu do Maple.
- Balík ***Optimization*** tvoří sada příkazů pro numerické řešení optimalizačních úloh, které se týkají hledání minima nebo maxima účelové funkce více proměnných jak bez omezení, tak s omezeními. Balík využívá algoritmy poskytované společností Numerical Algorithms Group[120] (NAG). Řeší též lineární, kvadratické a nelineární úlohy a úlohy vedoucí k využití metody nejmenších čtverců jak pro lineární, tak nelineární případ. Obecně se předpokládá, že účelové funkce jsou spojité a lokální řešení (minimum nebo maximum) jsou hledány pro funkce, které nejsou konvexní. Nicméně, příkaz `LPSolve` akceptuje celočíselné funkce a příkaz `NLPSolve` umožňuje globální optimalizační algoritmus pro funkce s omezeními.
- Balík ***PDEtools*** tvoří soubor příkazů a postupů pro hledání analytických řešení parciálních diferenciálních rovnic (PDE). Je založený na knize ES Cheb-Terraba a K. von Bülow [25] a na pokračování této publikace stejných autorů z roku 2004.
- Balíku ***Plottools*** tvoří příkazy pro vytváření základních grafických objektů a umožňuje změnit stávající struktury grafů. Grafické objekty jsou např. geometrické tvary, šipky, a body, a mohou být zobrazeny buď ve stávajících grafech nebo na oddělených osách grafu. Transformace zahrnuje rotaci, posunutí a úpravu všech typů grafů.
- Balík ***ScientificErrorAnalysis*** poskytuje reprezentaci a konstrukce číselných veličin, které jsou zastoupeny střední hodnotou a přidruženou neurčitostí nebo zatížením chybou s určitou mírou/stupněm přesnosti, jejíž hodnota je předem známa.
- Balík ***Sockets*** tvoří sada nástrojů pro síťovou komunikaci v Maple. Příkazy v tomto balíku umožňují připojit se k procesům na vzdálených počítačích v síti (například v rámci intranetu nebo internetu) a vyměňovat si data s těmito procesy. Zejména umožňuje, aby dva nezávislé Maple procesy běžící na různých počítačích v síti komunikovaly spolu navzájem.
- Balík ***Threads*** poskytuje na uživatelské úrovni příkazy pro psaní paralelních kódů v Maple, tzv. Multithreading[121]. Poskytuje tak více cest pro současné provádění sou-

[119] http://cs.wikipedia.org/wiki/Java_Database_Connectivity

[120] http://www.nag.co.uk/about_nag.asp

[121] Multithreading je obecné označení pro schopnost programu sám sebe větvit - program se větví na tzv. vlákna (threads), která mohou běžet současně. Možnost paralelního provádění několika větví výpočtu v rámci jednoho procesu nad jedním adresovým prostorem (vícevláknové aplikace). Všechna vlákna (jedné aplikace) sdílejí systémové zdroje, např. paměť (na rozdíl od multitaskingu, kde

běžných výpočtů. Maple pak zároveň provádí větší počet výpočtů, běží-li na vícejádrovém stroji (počítači).

- Balík ***Tolerance*** poskytuje prostředí pro provádění výpočtů v Maple s veličinami zahrnujícími tolerance.

6.1.4 Shrnutí

Maple je výkonné, interaktivní prostředí pro vědecké výpočty a jejich vizualizaci. Maple integruje symbolické výpočty prakticky ve všech oblastech matematiky (integrální a diferenciální počet, lineární algebra, nelineární, obyčejné, parciální diferenciální a integrální rovnice, numerická analýza) s numerickými výpočty a zpracováním grafů do uživatelsky příjemného prostředí, ve kterém se problémy a řešení zapisují stejně jako v matematice bez nutnosti jejich programování.

V Maple je možné vykreslit různé druhy grafů: dvourozměrné pro funkce jedné proměnné, třírozměrné pro funkce dvou proměnných, dále vizualizace numerického řešení obyčejných, parciálních a algebraicko-diferenciálních rovnic Všem grafům je možné interaktivně měnit vzhled, a to jak při jejich vytváření, tak po jejich nakreslení. Maple umožňuje stínovat třírozměrné grafy s určením zdroje dopadajícího světla, animovat dvou a třírozměrné grafy a mnohem více. Maple dovoluje vkládat do grafů ovládací prvky (tlačítka apod.), a vytvářet tak aktivní graficky ovládané uživatelské rozhraní. Existuje mnoho voleb parametrů, které mohou být použity v příkazech `plot` a `plots`. Patří mezi ně možnosti zvolení rastru (mřížky, os, titulky a další), možnosti generování grafů (adaptivní mřížka a detekce nespojitosti) a možnosti zobrazení (barvy, styly osvitu a výplně, průhlednost, atd.).

Maple je interaktivní systém počítačové algebry, který umožňuje řešit mnoho matematických problémů podstatně rychleji než při použití klasických numerických metod. Maple má také export kódu do programovacích jazyků C, C++, Fortran, Java, Visual Basic a MATLAB, konektivitu s MS Excel, MATLAB, C, CAD systémy, Java, Fortran i webovými portály. Základní vlastností Maple je jeho otevřenost a snadná rozšiřitelnost, která umožňuje doplňovat příkazy Maple o napsané funkce i celé aplikace. K Maple je možné navíc si stáhnout z Application Centre celou řadu specificky zaměřených aplikací, které jsou určeny pro řešení konkrétních typů problémů. Jednou z nejoceňovanějších vlastností Maple pro jeho použití ve vědeckých výpočtech je jeho velmi rozsáhlá nápověda, která byla v této kapitole informativně zmíněna.

6.2 Vědecké výpočty s využitím MATLABu

Aplikační software MATLAB je integrované programovací prostředí pro vývoj algoritmů (programovací jazyk) pro vědecké výpočty, zejména matematické modelování, numerické výpočty a počítačovou simulaci, analýzu dat a vizualizaci. Dále paralelní výpočty, zpracování signálů, návrhy řídicích a komunikačních systémů. Umožňuje jak interaktivní práci, tak vývoj rozsáhlých aplikací.

---

jsou jednotlivé procesy zcela oddělené). Čas vláknům přiděluje operační systém (podle jejich priorit).

MATLAB vyvíjí více než třicet let americká společnosti Mathworks Inc. Na její webové stránce[122] (obrázek 6.9) lze nalézt další informace o MATLABu týkající se např. vývoje algoritmů a aplikací[123], analýzy a zpracování dat[124], vizualizace dat[125], demoverzí aplikací a webinářů[126].

Obr. 6.9 *Webová stránka společnosti Mathworks popisující MATLAB*[112].

Distributorem MATLABu v ČR je společnost HUMUSOFT s.r.o, která poskytuje na svých webových stránkách velké množství nejnovějších informací o MATLABu v češtině[127], viz obrázek 6.10.

Je zde např. uveden seznam všech aplikačních knihoven (toolboxů)[128], nebo seznam aplikačních oblastí[129], kde je jedna z aplikačních oblastí výpočetní biologie[130].

[122] http://www.mathworks.com/products/matlab/
[123] http://www.mathworks.com/products/matlab/description2.html
[124] http://www.mathworks.com/products/matlab/description3.html
[125] http://www.mathworks.com/products/matlab/description4.html
[126] http://www.mathworks.com/products/matlab/demos.html
[127] http://www.humusoft.cz/produkty/matlab/matlab/
[128] http://www.humusoft.cz/produkty/matlab/aknihovny/
[129] http://www.humusoft.cz/produkty/matlab/#aplikacni-oblasti

Obr. 6.10 *Webová stránka společnosti HUMUSOFT popisující MATLAB.*

6.2.1 Výpočetní prostředí systému MATLAB

MATLAB ukládá všechna čísla v tzv. dvojité přesnosti (double precison) a umožňuje zvolit i úspornější formu, jak bylo zmíněné ve čtvrté kapitole.

Kromě jednodušších datových typů podporuje MATLAB také typy složitější. Například jde o vícerozměrná pole reálných nebo komplexních čísel, dále o tzv. pole buněk, tedy struktury podobné maticím, ve kterých ovšem každý prvek může být jiného typu.

Podobně lze tvořit datové struktury připomínají struktury známé z programovacích jazyků Pascal, Fortran, C++ apod. Skládáním těchto datových typů je pak možné vytvořit libovolně složité datové struktury.

Výpočetní jádro je v MATLABu implementováno s využitím základních matematických knihoven s podporou více jader podobně jako v Maple. K využití této vlastnosti systému stačí pouze základní verze MATLABu. Přitom implementace žádné dodatečné aplikační knihovny (toolboxu), ani psaní paralelních algoritmů není třeba. Není ani nutné nijak upravovat dříve vyvinuté programy, zrychlení výpočtu se projeví automaticky, podrobněji je to uvedeno v sedmé kapitole.

Nejpodstatnější součástí numerického jádra MATLABu jsou algoritmy pro operace s maticemi reálných a komplexních čísel. MATLAB umožňuje provádět všechny běžné operace, jako jsou násobení matic, určení inverzní matice k dané matici, výpočet determinantu atd. V nejjednodušší podobě je možno jej použít jako maticový kalkulátor, protože všechny příslušné operace se do MATLABu zapisují téměř stejně, jako bychom je zapsali ručně v papírové podobě.

Vektor reálných čísel může v MATLABu představovat i koeficienty polynomu a operace s polynomy jsou v něm rovněž obsaženy. Vektory v MATLABu mohou také reprezentovat časové řady nebo signály a MATLAB obsahuje funkce pro jejich analýzu, jako jsou výpočet

[130] http://www.humusoft.cz/produkty/matlab/oblasti-vyuziti/biologie/

střední hodnoty, hledání extrémů, výpočet směrodatné odchylky, korelačních koeficientů, rychlé Fourierovy transformace aj..

Grafika v MATLABu umožňuje snadné zobrazení a prezentaci výsledků získaných výpočtem. Je možné vykreslit různé druhy grafů: dvourozměrné (2D) pro funkce jedné proměnné, třírozměrné (3D) pro funkce dvou proměnných, histogramy, výsečové grafy a další. Všem grafickým objektům je možné libovolně měnit vzhled, a to jak při jejich vytváření, tak po jejich nakreslení. Je možné stínovat třírozměrné grafy s určením zdroje dopadajícího světla, provádět animaci 2D a 2D grafů, zobrazovat kontury a transparentní objekty, používat pseudo-barevné zobrazení, a mnoho dalšího. Většiny těchto efektů je možné docílit jedním nebo několika příkazy.

Obrázky v grafických oknech MATLABu jsou dynamické. Každému nakreslenému objektu je přiřazen identifikátor, jehož prostřednictvím je možné měnit vlastnosti objektu, a tím i jeho vzhled. Vzhled grafických objektů je také možno měnit interaktivně pomocí lišty nástrojů umístěné pod záhlavím obrázku. Grafický systém v MATLABu, nazvaný ***Handle Graphics,*** dovoluje vkládat do obrazů ovládací prvky (tlačítka apod.), a tak vytvářet aktivní graficky ovládané uživatelské rozhraní.

Všechny moduly MATLABu jsou popsány v rozsáhlé dokumentaci ve formátu pdf nebo v hypertextové on-line dokumentaci, která uživatelům usnadňuje orientaci v spektru možnosti pro užívání funkcí MATLABu. Otevřená architektura MATLABu inspirovala mnoho firem k vývoji a distribuci vlastních produktů, které buď rozšiřují výpočetní prostředí systému MATLAB o další knihovny a nástroje, anebo zajišťují propojení MATLABu s jinými specializovanými programy. Rodina MATLABu tak obsahuje kromě více než devadesáti modulů vyvinutých společností MathWorks ještě dalších více než tři sta produktů of jiných firem a velké množství volně přístupných akademických aplikací na webu.

6.2.2 Základní práce se systémem MATLAB

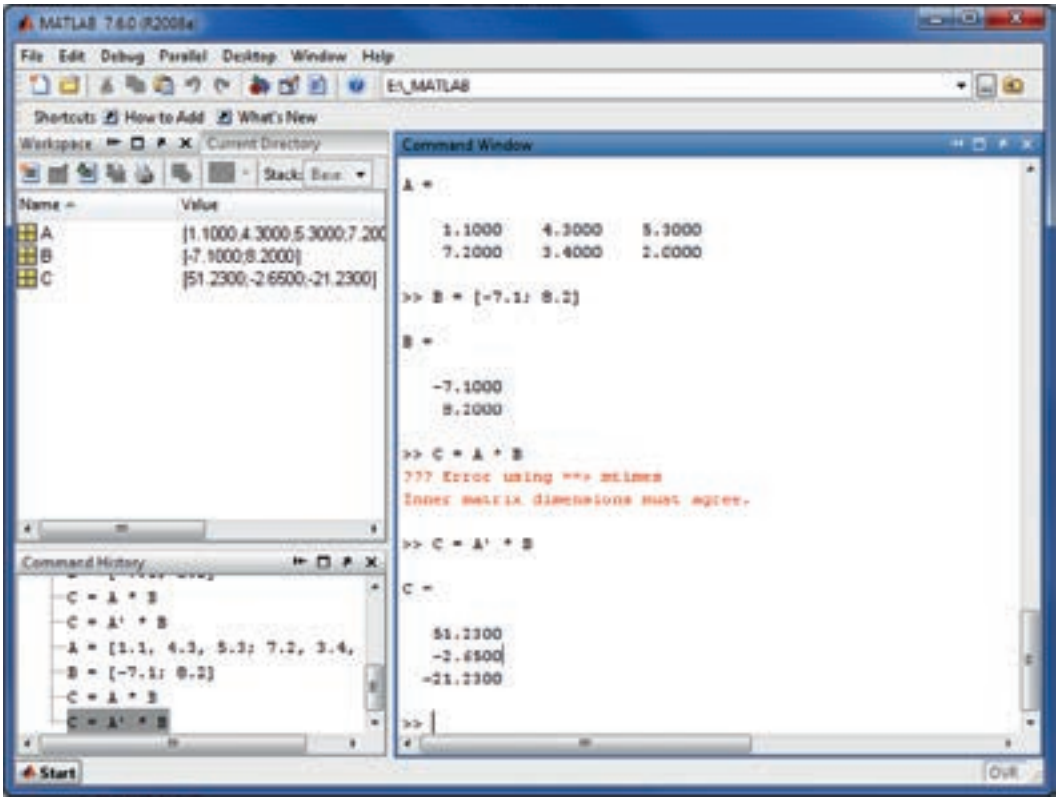
Hlavní okno MATLABu je na obrázku 6.11. Pod hlavičkou hlavního okna je umístěno standardní menu, které sestává ze sedmi submenu (MATLAB verze 7.6.0 - R2008a).

Submenu File obsahuje příkazy pro standardní práci s tzv. *m*-soubory (*m*-soubory obsahují posloupnosti volání funkcí MATLABu). Příkaz New/M-file slouží k vytvoření prázdného nového *m*-souboru. Pomocí příkazu Open otevřeme existující *m*-soubor. Z dalších položek zmiňme příkaz Set Path, který slouží k nastavení cesty do pracovního adresáře, a položku Preferences sloužící k nastavení uživatelského prostředí MATLABu.

Submenu Edit obsahuje standardní editační příkazy jako kopírování vybraného textu z okna do schránky (Copy), vložení obsahu schránky do okna (Paste), mazání vybraného textu v okně (Delete) apod.

Další submenu Debug obsahuje příkazy pro krokování spuštěných skriptů a nastavení tzv. ***break-pointů*** (míst, kde se má spuštěný program zastavit). Submenu Parallel pak obsahuje nástroje pro nastavení konfigurací pro paralelní spouštění skriptů.

Dále zmiňme submenu Desktop, které umožňuje přizpůsobit uživatelské rozhraní. Lze aktivovat ty části uživatelského prostředí, které je třeba mít zobrazeny a deaktivovat části, které chceme skrýt. Lze provádět i další nastavení uživatelského prostředí. Submenu Windows obsahuje seznam otevřených pracovních oken a otevřených *m*-souborů. A konečně submenu Help umožňuje přístup k nápovědě MATLABu.



Obrázek 6.11 *MATLAB uživatelské rozhraní.*

Pod řádkem s menu je umístěna ***nástrojová lišta*** s tlačítky, která odpovídají nejčastěji používaným položkám menu. Pod nástrojovou lištou je již plocha hlavního okna MATLABu s pracovními okny. Nejdůležitější okno je umístěno zcela vpravo a nese název Command Window (příkazové okno). Do příkazového okna vepisujeme volání funkcí MATLABu. MATLAB na každé volání bezprostředně reaguje. Tedy pracuje jako interpret.

Podívejme se na obsah příkazového okna na obrázku 6.11. Na prvém řádku nás symbol >> vyzývá k zadání příkazu do MATLABu. Vložením textu:

```
A = [1.1, 4.3, 5.3; 7.2, 3.4, 2.0]
```

vytvoříme matici *A* o dvou řádcích a třech sloupcích. Prvky matice vepisujeme do lomených závorek po řádcích. Čísla v řádku oddělujeme čárkou (případně mezerou). Řádky matice od sebe oddělujeme středníkem. Pokud po zapsání uvedeného textu stiskneme klávesu ***Enter***, zadaná matice se v příkazovém okně vypíše ve tvaru, na který jsme u matic zvyklí. Před výpisem chybí symbol >>, což indikuje skutečnost, že se jedná o výpis. Chceme-li zabránit tomu, aby se zadaná matice v příkazovém okně vypsala, musíme ukončit řádek středníkem.

Druhým příkazem, který je zapsaný do příkazového okna MATLABu, vytvoříme sloupcový vektor *B*:

```
B = [-7.1; 8.2].
```

Ve třetím příkazu chceme vypočítat matici *C*, která bude součinem matice *A* a vektoru *B*, ale zapomněli jsme matici *A* transponovat, a proto MATLAB ohlásil chybu, viz obrázek 6.11. Ve čtvrtém příkazu jsme matici *A* již transponovali (operátor apostrof), takže výpočet proběhl správně.

Z uvedeného příkladu je vidět, že MATLAB provádí maticové operace. Nemusíme tedy vzájemně násobit jednotlivé prvky matic a vektorů a součiny sčítat. Stačí korektně zapsat ma-

ticovou operaci a o zbytek se postará výpočetní jádro MATLABu, které je pro maticové operace optimalizováno.

Vlevo nahoře od příkazového okna na obrázku 6.11 je okno se záložkami Workspace (pracovní prostor), kde jsou informace o proměnných alokovaných v rámci vykonávaného programu (v našem případě matice *A*, *B* a *C*) a Current directory (současný adresář), který zobrazuje přesnou cestu do adresáře a seznam souborů, které se v tomto adresáři nacházejí.

Vlevo dole od příkazového okna je okno se záložkami Command History (historie příkazů) obsahující časovou posloupnost příkazů, které jsme postupně zadávali do příkazového okna. Příkazy zobrazené v okně s historií příkazů lze z tohoto okna přímo spouštět. Lze je také kopírovat do příkazového okna či vytvářeného *m*-souboru, atd.

6.2.3 Aplikační knihovny

Obr. 6.12 *Aplikační knihovny pro MATLAB.*

Otevřená architektura MATLABu vedla ke vzniku aplikačních knihoven funkcí, nazývanými toolboxy, které rozšiřují jeho použití v příslušných vědních a technických oborech (obrázek 6.12). Tyto knihovny nabízejí předzpracované specializované funkce, které je možno rozšiřovat, modifikovat, anebo jen čerpat informace z přehledně dokumentovaných algoritmů.

Oblasti jejich využití jsou:

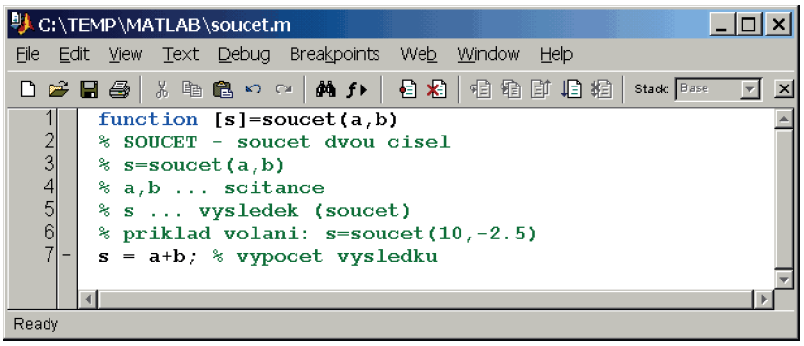
- aplikovaná matematika – matematické výpočty, analýza, vizualizace a vývoj algoritmů;
- automatické řízení a regulace – model-based design v návrhu řídicích systémů, zahrnující simulace, rapid prototyping a automatické generování kódu pro embedded systémy;
- zpracování signálu a komunikace – model-based design v návrhu komunikačních systémů a systémů pro zpracování signálu zahrnující simulaci, automatické generování kódu a verifikaci;
- zpracování obrazu – sběr obrazových dat, zpracování obrazu a videa, analýza, vizualizace a vývoj vlastních algoritmů;
- měření a testování – propojení s měřícími kartami a přístroji, analýza dat, vizualizace a nástroje pro aplikace v oblastech měření a testování;
- matematická biologie – analýza, vizualizace a simulace biologických dat a systémů;
- finanční modelování a analýza – finanční analýza, modelování a tvorba aplikací;
- modelování fyzikálních soustav – intuitivní tvorba modelů fyzikálních soustav metodou fyzikálního modelování.

6.2.4 *M*-soubory

Práci v MATLABu značně usnadňuje vytváření vlastních programů – skriptů. Tato skutečnost dovoluje uživateli při řešení rozsáhlejších problémů zapisovat potřebné příkazy do skriptů, které lze uložit na disk počítače a později kdykoliv vyvolat.

Uživatelem vytvořené skripty se ukládají do souborů s příponou *m* a z tohoto důvodu jsou i často označovány jako *m*-soubory (*m*-files). Jinými slovy skript je textový *m*-soubor obsahující seznam příkazů MATLABu, který po zavolání v příkazovém řádku jednotlivé příkazy postupně zpracovává. Napíšeme-li pak v příkazovém okně za symbol >> jméno tohoto *m*-souboru (bez přípony *.m*), MATLAB začne brát řádek po řádku jeho obsah a postupně vykonávat v něm uvedené instrukce.

Uživatel skripty vytváří ve speciálním ***M-editoru***, viz obrázek 6.13. Jelikož se však jedná o textové soubory, je možné je prohlížet nebo i editovat běžným textovým editorem. V MATLABu jsou obdobným způsobem, tedy pomocí skriptů, vytvářeny také veškeré funkce či objektové třídy.



Obrázek 6.13 *Okno M-editoru.*

6.2.5 Shrnutí

MATLAB je výkonné interaktivní prostředí pro vědecké a inženýrské výpočty a vizualizaci dat. MATLAB integruje numerickou analýzu, maticové výpočty, zpracování signálů a grafiku do uživatelsky příjemného prostředí, ve kterém se problémy a řešení zapisují

stejně jako v matematice bez tradičního programování. MATLAB je interaktivní systém, jehož základním datovým prvkem je matice, u které se nezadává typ. To umožňuje řešit mnoho numerických problémů podstatně rychleji než při použití klasických programovacích jazyků (Fortran, Basic nebo C). Snad nejoceňovanější vlastností MATLABu je jeho snadná rozšiřitelnost, která umožňuje doplňovat systém o individuálně napsané funkce (*m*-soubory) i celé aplikace. K MATLABu lze navíc pořídit celou řadu specificky zaměřených nadstaveb, aplikačních knihoven (toolboxů), což je kolekce *m*-souborů, která je určena pro řešení konkrétních typů problémů.

6.3 Vědecké výpočty s využitím SPSS

Aplikační software SPSS (*Statistical and Presentational System Software*, dříve *Statistical Package for Social Scientists*) byl původně vytvořen pro zpracování dat z oblasti sociologického a marketingového výzkumu. Během svého vývoje se však rozrostl na rozsáhlý aplikační software použitelný pro analýzu dat z nejrůznějších oblastí, včetně technických oborů, biologie a dalších přírodních věd. Vzhledem k jednoduchému a intuitivnímu ovládání systému, které nevyžaduje dlouhé studium, je v některých zemích (hlavně západní a severní Evropy) preferován pro výuku zpracování a analýzy dat před jinými podobnými software v této oblasti, jako jsou např. SAS[131] system, Statgraphics[132], Statistica[133], BMDP[134] apod.

V SPSS jsou implementovány všechny běžně používané jednoduché i složitější statistické analýzy. SPSS do značné míry uspokojí i náročné uživatele, kteří potřebují využívat klasifikační techniky, regresní modely, zpracovávat data z oblasti analýzy přežití a jiné pokročilé analytické nástroje. Dále obsahuje základní funkce pro manipulaci s daty a možnost uložení a exportu výsledků.

První verze aplikačního software SPSS, kterou vyvinuli N. H. Nie a C. H. Hull na Standfordské univerzitě, byla uvolněna na trh již v roce 1968. V počáteční fázi SPSS obsahoval základní statistické nástroje a funkce umožňující management dat (např. výběr dle kritérií – *case selection*). Manuál k software SPSS [22] vydaný v roce 1970 byl označen jako jedna z knih, které nejvíce ovlivnily tehdejší sociologii.

Postupem vývoje se SPSS rozrostl o nástroje pro analýzu a statistické testování dvou a více proměnných (ANOVA, *t*-test, korelace), jejich neparametrické varianty, lineární a nelineární regresi (predikci), clusterování, shlukování, diskriminační analýzu a další funkce. V návaznosti na tento vývoj jsou rozšiřovány možnosti nastavení parametrů analýz.

V roce 2006 byl grafický uživatelský interface přepsán do jazyka Java (verze 16.0) a SPSS se stal plně kompatibilní s operačními systémy Windows, Mac OS X a Unix (Linux). Aktuální verze SPSS 20.0 byla uvolněna na trh v listopadu 2011. Další verze je očekávána v dubnu 2012.

Dřívějšího hlavního distributora PASW (Predictive Analytics SoftWare) nahradila odkoupením SPSS Inc. v listopadu 2009 společnost IBM, která vydává a distribuuje nové verze IBM SPSS Statistics pravidelně každých dvanáct až osmnáct měsíců.

Na webovém portálu IBM věnované software IBM SPSS Statistics[135], která je na obrázku 6.14, společnost IBM informuje uživatele komplexně současné verzi tohoto software. V IBM SPSS Statistics 20 pokračuje zpřístupňování pokročilých statistických metod díky vylepše-

[131] http://www.sas.com/
[132] http://www.statlets.com
[133] http://www.statsoft.cz
[134] http://www.statistical-solutions-software.com/
[135] http://www-01.ibm.com/software/analytics/spss/

ným nástrojům, výstupům a snadno použitelným funkcím. Tato verze se zaměřuje na rozšiřování analytických možností pomocí:

- zobrazování v mapách – přidání geografického rozměru do analýz a reportů;
- zdokonalení existujících procedur;
- další vylepšení vedoucí k větší analytické produktivitě.

Obr. 6.14 *Webový portál IBM věnovaný IBM SPSS Statistics.*

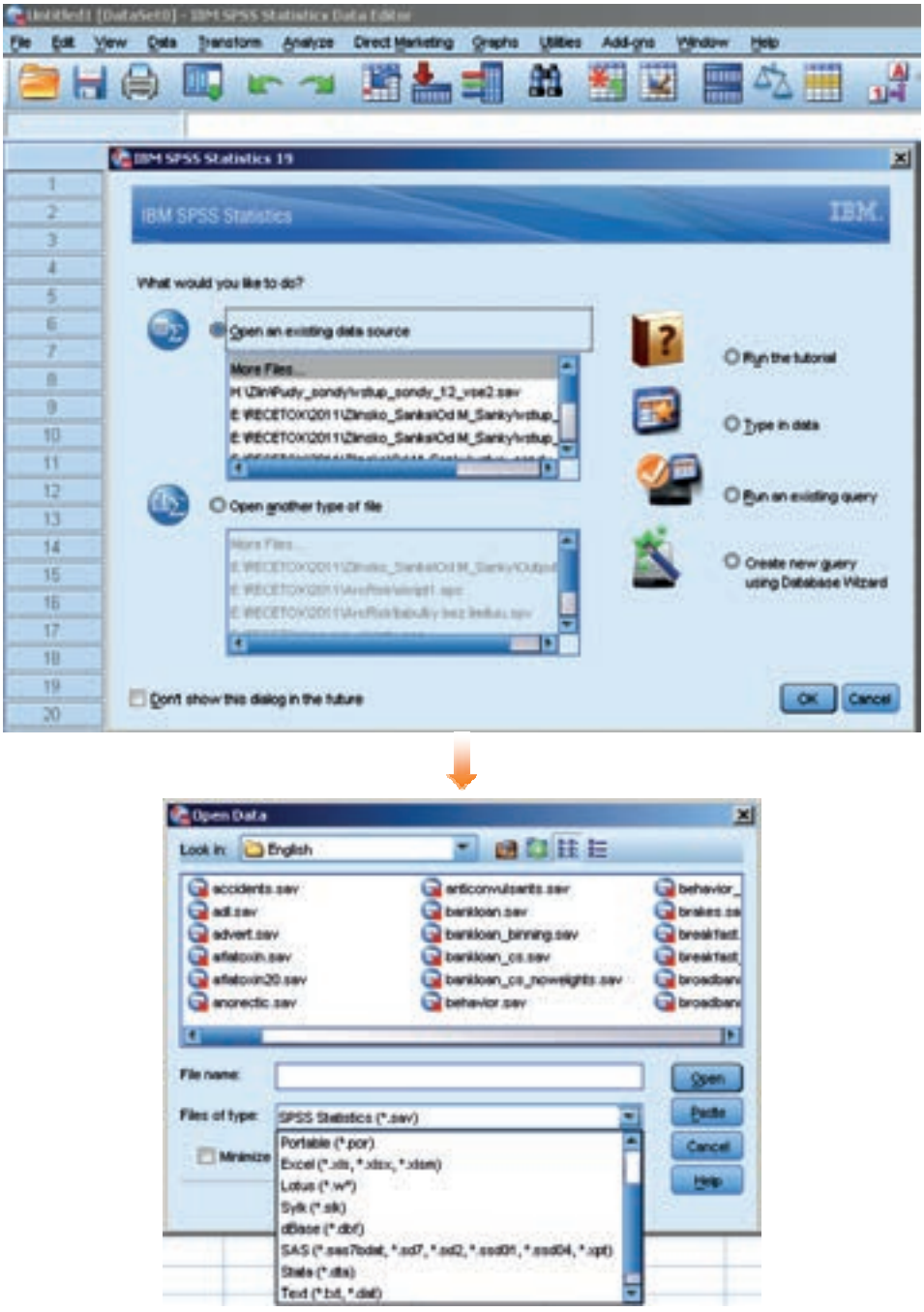
Výhradním partnerem společnosti IBM pro distribuci softwaru IBM SPSS Statistics (dále zkráceně SPSS) v České a Slovenské republice a poskytovatelem analytických, statistických a data miningových služeb je společnost ACREA CR[136], která byla založena v roce 1995 a nabízí široké portfolio řešení zaměřených na sběr dat, statistické analýzy, data mining, prezentaci a využití výsledků.

[136] http://www.acrea.cz/

6.3.1 Výpočetní prostředí SPSS

SPSS se spustí ze startovacího menu počítače výběrem položky (Programy > IBM SPSS Statistic) nebo kliknutím na ikonu IBM SPSS Statistics na ploše obrazovky.

Po spuštění se otevřou dvě okna. Aktivní okno má název IBM SPSS Statistics 20. Prostřednictvím tohoto okna lze otevřít dříve používané a otevřené soubory (z historie), spustit tutoriál, případně získat data z databáze. Pokud není zvolena žádná z těchto možností, pak se po kliknutí na tlačítko OK objeví menu pro výběr dat – Open Data**,** viz obrázek 6.15.



Obr. 6.15 *Vstupní obrazovka SPSS.*

Celkový koncept uspořádání voleb a ikon je navržen podobně jako u konkurenčního statistického software Statistica[137] nebo programů MS Office.

Hlavní okno, ve kterém se uživatelé nejčastěji pohybují, je ***Data Editor***, které se používá pro vstup, manipulaci s daty a výběr analytických funkcí. Toto okno se spouští automaticky při startu SPSS.

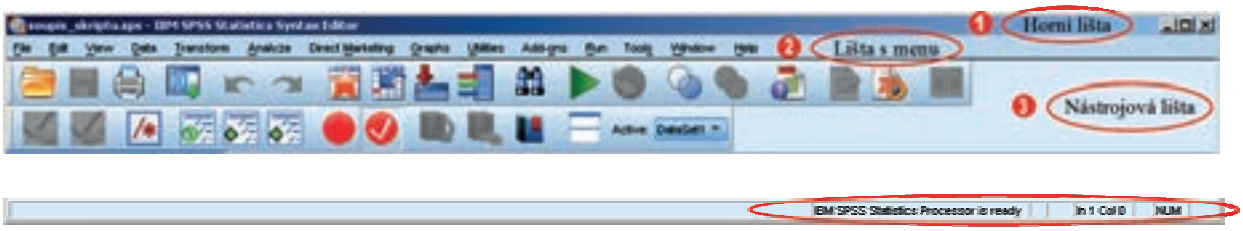
Dále se v SPSS zavedou zvláštní okna pro zobrazení výsledků z použitých statistických a grafických nástrojů: ***Output Viewer*** jako okno výstupů a ***Syntax Editor*** pro práci se skripty (syntaxí). Přepínání mezi okny zabezpečuje tabulátor umístěný v dolní liště obrazovky.

Okno Output Viewer se aktivuje automaticky vždy po provedení statistického nebo grafického zpracování dat. Výstupy zobrazené v okně Output Viewer lze kopírovat a přenášet do jiných programů k dalšímu zpracování, typicky do MS Excelu.

Okno Syntax Editor lze aktivovat pouze na základě uživatelova požadavku prostřednictvím menu File.

Jednotlivá okna se otevírají nezávisle a jejich obsah můžeme uložit v samostatných souborech s vlastní příponou. Oknu Data Editor je určena přípona ***.sav***, pro Output Viewer přípona ***.spv*** a pro Syntax Editor přípona ***.sps***.

Každé z výše uvedených oken se skládá z pěti částí, viz obrázek 6.16:



Obr. 6.16 *Horní lišta, lišta s menu a nástrojová lišta v Syntax Editoru. Dole pak stavová lišta.*

1. ***Horní lišta*** (*Title Bar*): klasická lišta operačního systému Windows. Název souboru, se kterým se právě pracuje, je umístěn vlevo. Vpravo se nacházejí tlačítka minimalizace, maximalizace a zavření okna.
2. ***Lišta s menu*** (*Menu Bar*): Nejdůležitější lišta obsahující nabídky pro zpřístupnění většiny funkcionalit systému SPSS. Jednotlivá okna v SPSS mají své menu relevantní k jejich účelu, avšak nejdůležitější nabídky jako ***Analyze*** (dříve ***Statistics***) a ***Graphs*** jsou zachovány ve všech oknech. Tím je usnadněno generování nových výstupů bez nutnosti přepínat mezi okny SPSS.
3. ***Nástrojová lišta*** (*Tool Bar*): Obsahuje ikony, které umožňují rychlý a snadný přístup k nejpoužívanějším funkcím. Nástrojová lišta se v jednotlivých oknech SPSS mírně liší (stejně jako lišta s menu), Některá okna mají více než jednu nástrojovou lištu.
4. ***Vlastní aktivní oblast***: V okně Data Editor se zobrazují vstupní data. V okně Output Viewer výstupy z analýz (typicky tabulky a grafy) a v okně Syntax Editor jednotlivé skripty. Aktivní oblast je ve všech oknech umístěna pod nástrojovou lištou.
5. ***Stavová lišta*** (*Status Bar*): Je ve spodní části každého okna a udává informace o stavu daného okna. Např. o průběhu výpočtu, zapnutí filtrů, pozici v syntaxi aj.

Jednotlivé funkce v lištách lze vyvolat kliknutím myši, nebo pomocí klávesových zkratek. Zatímco menu nástrojů je pro přehlednost koncipováno jako rolovací (drop down) nabídka, funkce v ostatních lištách se spustí jedním kliknutím.

[137] http://www.statsoft.cz/

6.3.2 Načtení dat do SPSS a práce s nimi

Při spouštění SPSS lze vybrat přímo z paměti počítače již vytvořený soubor k jeho otevření (viz předchozí kapitola) a práci s ním. Pokud tak neučiníme a i v okně Open Data klikneme na tlačítko Cancel, pak se otevře okno Data Editor, které je prázdné. Toto okno vypadá podobně jako prázdný soubor v programu MS Excel.

Na pracovní oblast Data Editoru lze nahlížet dvěma způsoby: Data View a Variable View. Přepínají se ve spodní části okna (obrázek 6. 17) nebo poklepáním na název proměnné v záhlaví aktivní oblasti Data Editoru.

Obr. 6.17 *Přepínání mezi Data View a Variable View.*

Zvolíme-li Data View (obrázek 6.18), jsou ve sloupcích jednotlivé parametry (proměnné – variables) a v řádcích jednotlivá měření (případy – cases). V buňkách jsou zobrazeny hodnoty proměnných jednotlivých měření. Tyto hodnoty lze editovat, kopírovat, přidávat atd.

	ID	Oznaceni	LOKALITA	Frakce	Frakce_2	Křemen	netekave_8_PAH	DDT	Naftalen	PCBs	HCHs
4	4	Cem_D	1	d	4	19,108129267	,02866	175,12728	3199,02575	563,02209	4,63290
5	5	Cem_E	1	e	5	19,662188381	,09075	573,64287	1644,16175	406,71856	,74230
6	6	Cem_F	1	f	6	16,857618107	,26875	726,76232	364,66100	1021,25363	329,83754
7	8	Lom_A	2	a	1	16,011081190	,05788	69,32828	6728,10920	47,10288	4,47390
8	9	Lom_B	2	b	2	22,646783898	,00836	19,06972	2827,49500	14,80748	1,27304
9	10	Lom_C	2	c	3	13,753015775	,00922	33,46996	5309,26320	17,59614	3,32691
10	11	Lom_D	2	d	4	18,416582875	,03028	40,30338	5795,77840	21,51268	5,24737

Obr. 6.18 *Hlavička pro Data View.*

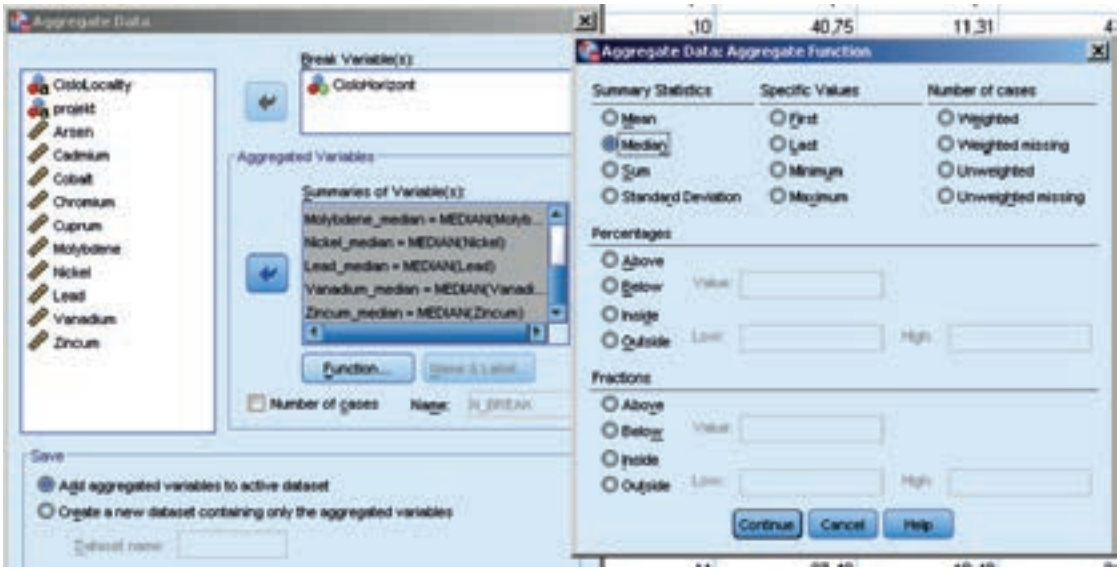
Pokud chceme pracovat s proměnnou, pak je nutné použít druhý způsob Variable View. Zde je možné přidávat/odebírat proměnné, přejmenovat je, definovat jejich datový typ, popisky aj. (obrázek 6.19). Jméno proměnné musí začínat písmenem a nesmí obsahovat mezery.

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
4	Frakce	String	1	0	Mereni	None	None	10	Center	Nominal	Input
5	Frakce_2	Numeric	11	0	Mereni_rec	None	None	10	Center	Nominal	Input
6	Křemen	Numeric	13	12	Kremen	None	None	10	Right	Scale	Input
7	netekave_8_...	Numeric	11	5	8_netekavePAH	None	None	10	Right	Scale	Input
8	DDT	Numeric	11	5	DDT	None	None	10	Right	Scale	Input
9	Naftalen	Numeric	11	5	Nap	None	None	10	Right	Scale	Input
10	PCBs	Numeric	11	5	PCBs	None	None	10	Right	Scale	Input
11	HCHs	Numeric	11	5	HCHs	None	None	10	Right	Scale	Input

Obr. 6.19 *Hlavička pro Variable View.*

Načtení dalších datových souborů se provede stejně jako např. v programech MS Office prostřednictvím nabídky (File > Open > Data) nebo pomocí tlačítka Open Data Document nacházejícího se pod nabídkou File. Načítané soubory mohou být vytvořené v SPSS (s příponou .sav), nebo musejí být uloženy v některém z podporovaných formátů (viz obrázek 6.15). V nabídce File jsou uvedeny také možnosti uložení, exportu, tisku dat a další související funkce.

Pro práci s daty je velmi důležitá nabídka v liště menu Data. Zde lze nalézt všechny významné funkce pro manipulaci s daty (ne s hodnotami!). Důležitá je možnost data validovat (hledání duplikací), řadit, restrukturalizovat (transpozice, přeskládání, vytvoření ortogonality) a vytvářet výběr z dat dle zvolených kritérií. Velmi užitečná je v SPSS funkce agregace, pomocí níž lze seskupit na základě zvoleného pravidla (např. průměrováním) jednotlivá měření – cases – dle určité proměnné (např. dle data – roku, viz obrázek 6.20).



Obr. 6.20 *Funkce agreggate.*

Hodnoty v parametrech lze také „rekódovat“ (převést na jiná vyjádření), přepočítávat na nové hodnoty (např. logaritmickou transformací), vytvářet z dat časové řady aj. Tyto funkce jsou součástí nabídky v menu Transform.

6.3.3 Analytické možnosti SPSS

SPSS provádí komplexní statistickou analýzu s danými daty. Analytické nástroje SPSS lze použít bez nutnosti znát syntaxi SPSS z nabídky Analyze. V SPSS jsou implementovány jak nástroje pro základní popis a charakterizaci dat, tak i poměrně velmi pokročilé prostředky (např. vyhledávání vzorů v datech).

Základní funkce pro přehledové hodnocení dat jsou: Reports, Descriptive Statistics. Pomocí těchto funkcí získáme základní popisnou statistiku (minimum, maximum, medián a mnoho dalších).

Tzv. ***křížový dotaz***, tj. hodnocení dat agregovaných jiným parametrem (např. průměrná koncentrace v jednotlivých letech), se vytváří pomocí funkce Custom Tables, kterou vyvoláme (Tables > Custom Tables), viz obrázek 6.21.

Funkce Compare Means umožňuje parametrické i neparametrické testování hypotéz pro jeden, dva či více parametrů (ANOVA). Dále obsahuje funkce pro regresní analýzu. Dostupná je jak jedno, tak víceparametrická regrese (techniky obecných a zobecněných lineárních modelů).

Nabídka Regression obsahuje mj. metody nelineárního modelování (probit regrese, ordinální regrese, váhované odhady parametrů a další).

Vzájemný vztah mezi parametry sledují korelační koeficienty. Nástroje pro parametrickou, neparametrickou i parciální korelaci jsou uvedeny v nabídce Correlation.

Obr. 6.21 *Custom tables – nastavení agregace dle parametrů „projekt“ a „CisloHorizont“.*

Ze zbývajících položek menu jmenujme Analyze, která obsahuje pokročilejší nástroje pro analýzu dat a vyhledávání v datech (data mining). Jsou mezi nimi již dříve zmíněné klasifikační techniky a analýza přežití, dále modely neuronových sítí, techniky redukce dimenzí (faktorová analýza, korespondenční analýza) a multidimenzionálního škálování a vykreslení ROC křivek[138]. Dalším velmi užitečným nástrojem je položka Forecasting. Umožňuje zpracovávat časové řady (spektrální analýza, sezónní dekompozice, (parciální) autokorelace aj.).

U většiny z výše uvedených funkcí lze upravovat metodiku výpočtu a ovlivnit velikost výstupu pomocí změny nastavení parametrů přímo v zadávacím menu dané funkce.

6.3.4 Grafické výstupy v SPSS

Výstupy z analýz v SPSS jsou jak tabulky hodnot počítaných veličin, tak jejich grafy, případně jen pouze jejich hodnoty. Základní grafy v SPSS lze vytvořit pomocí menu Graphs. Navíc lze na výstupu některých složitějších funkcí z Analyze nastavit zobrazení grafů relevantních k dané funkci.

Grafické výstupy se v SPSS tvoří nejčastěji užitím menu Chart Builder nebo pomocí jednoduchých průvodců *Legacy Dialogs*.

V menu Chart Builder si nejprve zvolíme typ grafu, který chceme vykreslit. Poté přidělíme osám příslušné proměnné a můžeme změnit některá nastavení před vykreslením grafu. Graf se vykreslí do okna Output Viewer, kde je možné poklepáním na oblast grafu jej dále upravovat.

Bohužel editace grafů není v SPSS dostatečně intuitivní (značně se liší od systémů Maple, MATLAB i MS Office) a vyžaduje relativně dlouhou dobu na „naučení se“ postupů pro rychlou úpravu grafu do podoby, která uživateli nejlépe vyhovuje.

6.3.5 Výstupy z SPSS: Output viewer

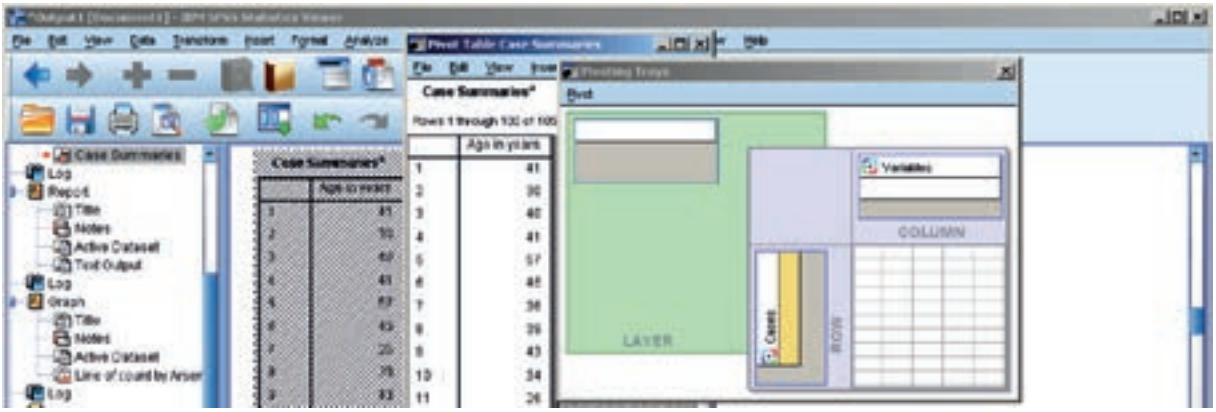
Všechny výstupy z analýz nebo grafické výstupy se v SPSS zobrazují v samostatném okně ***Output viewer.*** Každý výsledek vypsaný v okně Output viewer je charakterizován svým syntaxovým zápisem (log), názvem použité funkce, označením aktivního datasetu (zdrojový soubor dat pro použitou funkci a výsledkem – graf/tabulka).

Aktivní část okna výstupů (viz kapitola 6.3.1) je podélně rozdělena na dvě oblasti. V levé, užší, se postupně vytváří „strom“ výstupů (postupné popisy výstupů), zatímco v pravé se nacházejí samotné výstupy. Strom výstupů umožňuje snadnou a rychlou orientaci v historii

[138] http://en.wikipedia.org/wiki/ROC_curve

vypsaných/vykreslených výstupů. Jednotlivé položky ve stromové struktuře lze přejmenovat dvojím kliknutím na jejich název. Také lze jednotlivé „větve“ stromu mazat a hromadně označovat (pomocí tlačítka Ctrl).

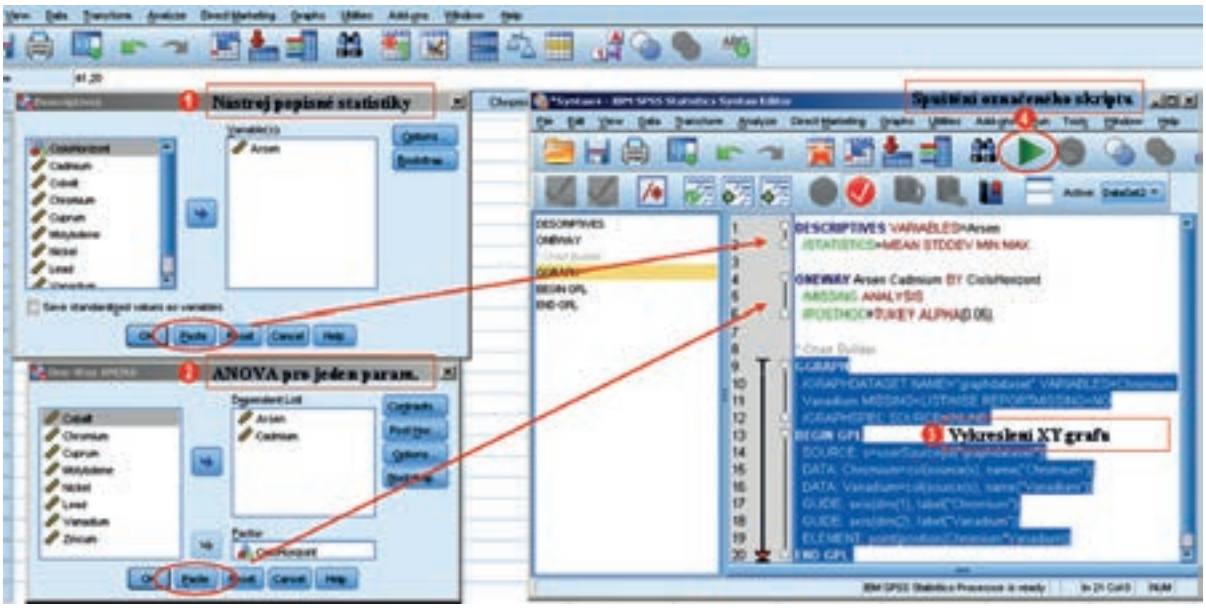
Samotné výstupy v pravé části okna lze pojmenovat (označení více objektů pomocí tlačítka Ctrl) a kopírovat do dalšího software (např. MS Office). Pokud poklepeme dvakrát na graf, vyvolá se Chart Editor, ve kterém můžeme grafy upravovat. V případě, že poklepeme dvakrát na tabulku obsahující výstup z funkce, zpřístupní se úprava/kopírování jejich hodnot. Po kliknutí pravým tlačítkem je možné spustit funkci Pivot Table Case Summaries, potažmo Pivot Trays pro úpravu vizualizace tabulky (způsob vykreslení), viz obrázek 6.22.



Obr. 6.22 *Výstupní tabulka s aktivovanými* menu *Pivot Table a Pivoting Trays.*

6.3.6 Syntax Editor

Možnost editovat syntaxi funkcí v SPSS, a tak „naprogramovat“ vlastní nastavení funkcí, je jednou ze silných stránek SPSS. Okno ***Syntax Editoru*** lze vyvolat ad hoc pomocí nabídky (File > New > Syntax) nebo použitím některého z nástrojů analýzy a/nebo grafických výstupů.



Obr. 6.23 *Vyvolání syntaxe pomocí tlačítka* Paste. *Ukázka dvou analytických nástrojů a grafu v syntaxi.*

Pokud použijeme k výpočtu jakoukoliv funkci dostupnou z menu Analyze nebo Graphs, nenachází se ve spodní části každého okna nastavení analýzy/grafu tlačítko Paste. Pomocí něj se otevře okno ***Syntax Editoru*** a vloží se do něj automaticky syntaxe dané funkce (včetně případných nastavení), viz obrázek 6.22.

V pravé části okna ***Syntax Editor*** jsou zobrazena použitá vyhrazená slova. Kliknutím na vyhrazené slovo jsme automaticky navigováni na příslušnou pozici v syntaxi, kde je toto slovo použité. V levé části je samotná syntaxe, kterou lze editovat a kopírovat. Syntaxi spouštíme označením myší a zelenou šipkou v nástrojové liště (nebo pomocí kláves Ctrl + r).

Z obrázku 6.22 je vidět, jak se lze k dříve proběhlým syntaxím (skriptům) vracet, upravovat jejich parametry, případně celé přepisovat s novými proměnnými a adjustovat je na nové datové soubory. Pomocí vhodně „naprogramované“ sady syntaxí rychle a jednoduše zvládneme širokou škálu rutinních, často se opakujících podobných úloh. Jednoduchou úpravou syntaxe dosáhneme požadovaného výstupu rychleji než pomocí dialogového okna s funkcemi. Editovatelnost syntaxí otevírá zcela nové možnosti analýz, které nejsou z nastavení funkcí přístupné. Editováním lze měnit jak nastavení výpočtu, tak požadovat nové výstupy. Přípravou syntaxe jsme schopni vytvořit i velmi komplexní aplikace, které nejsou cestou rolovacích nabídek a pomocí nastavení funkcí přístupné.

6.3.7 Shrnutí

Program SPSS patří mezi komerčně distribuovaný statistický software. Jedná se o uživatelsky přívětivý a snadno ovladatelný program využívaný mj. pro výuku statistického zpracování dat v kurzu Matematické biologie. Nabízené funkce SPSS lze nastavit kliknutím myší a pro běžné použití nevyžaduje zvláštní studium programovacích jazyků nebo syntaxe. SPSS je přizpůsoben pro zpracování malých i velmi objemných souborů (velikost souborů je omezena pouze pamětí počítače) z různých vědních oborů pomocí parametrické a neparametrické statistiky a široké palety modelovacích technik, včetně základního postupu pro data mining. Mezi jeho přednosti patří kombinovaný náhled na data ***Variable View/Data View***, možnosti široké úpravy vstupních dat (např. agregace na základě podmínek/parametrů), uložení historie použitých funkcí i s jejich nastavením v nástroji ***Recall Recently Used Dialogs***.

Samostatnou kapitolou je rozšíření o okno ***Syntax Editor***, pomocí kterého lze zjednodušit rutinní úkoly, zpřístupnit funkcionality v běžném menu nedostupné a vytvářet komplexní úlohy.

Mezi slabé stránky SPSS patří práce s grafy, která je značně neintuitivní a vyžaduje delší čas pro osvojení nabízených grafických funkcí. Navíc některé (v jiných software běžné) úpravy nejsou dostupné. Implementace analýzy časových řad je pro běžnou statistickou analýzu sice dostačující, ale neumožňuje jejich detailní zpracování. Metody z vícedimensionální analýzy a hodnocení vztahů chybí úplně.

7 Paralelní výpočty

7.1 Úvod

Ve vědeckých výpočtech se setkáváme s algoritmy a numerickými metodami, které obecně vykonávají velké množství aritmetických operací. Jsou výpočetně velmi náročné, jejich běh, i na nejmodernějších počítačích, trvá velmi dlouhou dobu. Proto je snaha takové vědecké výpočty urychlit tím, že se namísto běhu úlohy na jednom procesoru úloha rozloží do několika běhů na více procesorech. Takový přechod však není jednoduchý a obvykle vyžaduje úpravu algoritmu pro zpracování úlohy. Tento proces nazýváme *paralelizací*.

Pro paralelizaci úloh byla vyvinuta celá řada postupů a doprovodných ICT nástrojů, jako je např. *multithreading*. K běžně používaným nástrojům pro zajištění komunikace mezi dílčími úlohami slouží například procedury z knihovny PVM [26] aj. V této kapitole ukážeme paralelizaci v aplikačním software Maple a MATLAB.

7.2 Paralelní výpočty v Maple

Maple 15 automaticky určuje počet procesorů, které budou při výpočtu nasazeny. K nastavení tohoto počtu procesorů je využita systémová proměnná `numcpus`, kde se ukládá počet procesorů, který je v Maple automaticky optimalizován. Při zahájení výpočtu půjde o skutečný počet procesorů, kterými počítač disponuje. Procesory s vysokým počtem Intel technologií se považují za jeden procesor na fyzickém jádru počítače.

Maple 15 nabízí automatickou paralelizaci úloh, která přináší výrazné zrychlení. Využívá plný výkon procesoru počítače tak, že automaticky detekuje možnost využití všech dostupných procesorových jader k provedení mnoha výpočtů současně. Není k tomu třeba žádného speciálního programování, ani změna jakéhokoli nastavení jeho parametrů nebo dokonce znalost, kolika jádry počítač disponuje. V Maple 15 může být provedeno současně mnoho základních operací s využitím knihovny (balíku) `CodeTools`, která obsahuje subknihovny a příkazy, které pomáhají zkvalitnit účinnost a vlastnosti kódu Maple 15. Výsledky jsou pak k dispozici mnohem rychleji a je možné řešit rozsáhlé úlohy.

Příklad 7.1: Ukažme, jak s využitím knihovny `CodeTools` lze zlepšit účinnost a kvalitu kódu Maple 15, např. pro operaci násobení polynomů.

Pomocí příkazů `seq` a `randpoly` vygenerujme polynomy $f(x)$ a $g(x)$, oba stupně 10^4. Využijme příkazu `Usage` z knihovny `CodeTools`, abychom změřili čas a využití paměti počítače při provádění operací s polynomy. Nejprve vypočítejme jejich součin $p(x) = f(x) \cdot g(x)$ pomocí příkazu `expand`:

```
f,g := seq(randpoly(x,degree=10^4,dense),i=1..2):
p := CodeTools[Usage](expand(f*g)): # Výpočet součinu polynomů
```

který dá výsledek:

```
memory used=0.96MiB, alloc change=0.94MiB, cpu time=156.00ms, real
time=149.00ms
```

Jestliže stejný výpočet provedeme v Maple 14 (tedy v předchozí verzi Maple) dostaneme výsledek:

```
memory used=1.26MiB, alloc change=1.37MiB, cpu time=2.79s, real time=1.83s
```

Proveďme v Maple 15 roznásobení výrazu $(5 \cdot x - 3 \cdot y)^{10000}$ opět pomocí příkazu `expand`:

```
p := CodeTools[Usage](expand((5*x-3*y)^10000)):
```

který dá výsledek:

```
memory used=33.42MiB, alloc change=33.62MiB, cpu time=515.00ms, real time=528.00ms
```

Opět pro srovnání konfrontujme tento výsledek s výstupy v Maple 14, kde dostaneme:

```
memory used=137.81MiB, alloc change=59.68MiB, cpu time=3.51s, real time=3.71s
```

Z tohoto srovnání vidíme, že Maple 15 je mnohem efektivnější než Maple 14.

Řada výpočtů v Maple zahrnuje zpracování polynomů, včetně mnoha zabudovaných řešičů a integračních algoritmů. Tyto algoritmy se také automaticky paralelizují. Pozitivem těchto změn je především kratší výpočetní doba.

Další možností pro aktivaci automatického paralelismu v Maple 15 je oblast numerické lineární algebry (balík Linear Algebra). Součástí Maple 15 je nejnovější verze knihovny Intel Math Kernel Library[139] (MKL). Maple ji automaticky nastaví, aby využil všech dostupných výpočetních jader procesoru počítače, na kterém pracuje. Výsledkem toho je skutečnost, že mnoho výpočtů v numerické lineární algebře (tj. v příkazech balíku Linear Algebra) může probíhat automaticky současně, pokud počítač disponuje více jádry.

Maple 15 tak umožňuje spustit více výpočetních procesů přímo z uživatelské úrovně bez nutnosti jakékoliv předchozího nastavení a správy. Vícenásobná „back-end“ jádra sdílejí stejné uživatelské rozhraní, ale každé jádro je zcela nezávislé a bezpečné. Tento režim paralelismu nabízí ochranu sdílení dat.

V současnosti s běžně užívanými čtyřjádrovými stroji a stále více populárními osmi až dvanáctijádrovými stroji nová lokální gridová funkčnost umožňuje paralelní programování, a tím získání okamžitého zrychlení výpočtu, ať chceme nebo nechceme, v rozsahu přesahujícím lokální počítač.

Rozhraní pro programování aplikací (*Application programming interface* - API) je v Maple stejné, jako se používá pro rozsáhlé gridové počítání v clusteru nebo superpočítači. To umožňuje snadné prototypování a testování distribuovaného kódu na daném počítači a následného nasazení stejného kódu do velké gridové sítě.

V Maple 15 byla integrována podpora lokálních výpočtů na gridu s využitím maximálního možného výkonu počítače s bezpečným rozdělením a sdílením informací o jednotlivých výpočetních vláknech.

Maple je jediný technický počítačový systém, který umožňuje využít vícevláknového programování. Jeho programovací jazyk nabízí přímý přístup na spouštění a ovládání vláken a možnost pozastavení výpočtu u některého vlákna na daný čas. Kromě toho, Maple nabízí tzv. *Task Programming Model*, který zjednodušuje správu vláken a je podrobně popsán v nápovědě. Psaní paralelních algoritmů pomocí Task Programming Model snižuje a odstraňuje mnohé obtíže spojené se standardním vláknovým programováním. Touto paralelizací je možné výrazně zredukovat čas potřebný k výpočtu úloh. Vláknový výkon je evidentně lepší v Maple 15, který má důležité doplňky v API a který umožňuje psát paralelní programy efektivněji s využitím všech jader v počítači.

[139] http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/

Maple 15 může snadno nastavit víceprocesorové výpočty na lokálním počítači s využitím knihovny Grid. Aplikační knihovna *Grid Computing Toolbox* rozšiřuje tuto možnost vykonávat paralelní výpočty v gridu nebo na clusteru. Tyto dvě knihovny jsou plně kompatibilní. Algoritmus tak může být vytvořen a plně testován na lokálním počítači a pak nasazen v plném clusteru, a to bez změny algoritmu. Pro Maple 15 aplikační knihovna Grid Computing Toolbox je zjednodušena, takže je snadné vytvářet a testovat paralelně distribuované aplikace. Mezi nové funkce knihovny patří zjednodušené nastavení lokálního gridu, pružné předávání parametrů a rozšířené API.

Je možné využívat výpočty přímo na grafickém jádru Maple 15 pomocí technologie *CUDA* a tím výrazně zkrátit výpočetní čas.

7.2.1 Grid knihovna

Nová knihovna (balík) Grid v Maple 15 obsahuje řadu programovacích modulů (nástrojů) a příkazů pro paralelní výpočty. Podporuje především paralelní výpočty na více procesorech (dále uzlech).

To zvýhodňuje vícejádrovou a víceprocesorovou počítačovou architekturu. Knihovna Grid rozšiřuje tuto možnost také na distribuované paralelní výpočty na uzlech clusteru nebo na gridu, kde s využitím knihovny Grid Computing Toolbox je možno škálovat algoritmy pro běh na externích strojích s využitím stejného kódu.

Knihovny Grid a Grid Computing Toolbox obsahují příkazy: `Barrier`, `Interrupt`, `Launch`, `Map`, `MyNode`, `NumNodes`, `Receive`, `Send`, `Seq`, `Server`, `Setup` a `Status`. Používají společné rozhraní pro všechny režimy paralelních výpočtů. To umožňuje snadno vytvářet prototypy algoritmů v jednom režimu a používat je v jiném. Z výše uvedených příkazů těchto knihoven popíšeme jen ty nejdůležitější.

Příkaz `Setup` umožňuje zvolit režim provozu pro paralelní výpočty. Musí být proveden před interaktivním dotazováním na stav gridu nebo spuštěn v paralelní úloze v gridu. Vyvolání příkazu `Setup` způsobí, že je inicializován určitý režim výpočtu. V dané výpočetní relaci lze zvolit pouze jeden režim:

- `"local"` - režim poskytující podporu paralelních výpočtů ve více uzlech (procesorech) na jednom počítači. Jde o výchozí nastavení knihovny Grid a je k dispozici jako součást instalace Maple 15;
- `"hpc"` - režim poskytující podporu paralelních výpočtů v distribuovaných uzlech v heterogenní síti gridu. Plánování a řízení výpočtu je monitorováno „daemon“ procesy spuštěnými v každém uzlu v gridu. Tento režim může být aktivován pouze v případě, že byla nainstalována knihovna Grid Computing Toolbox;
- `"mpi"` - režim poskytující podporu paralelních výpočtů v uzlech clustru pomocí standardních protokolů MPI (*Message Passing Interface*)[140]. Plánování úloh a spouštění dávek je řízeno operačním systémem. Tento režim může být aktivován pouze v případě, že byla nainstalována knihovna Grid Computing Toolbox.

Při volbě režimu `"hpc"` nebo `"mpi"`, příkaz `Setup` musí být vyvolán před použitím jakéhokoliv jiného příkazu knihovny Grid. Jinak bude inicializován režim `"local"`. To způsobí, že se další režimy stanou nedostupnými. Interaktivní zápisník Maple15 poskytuje speciální grafické rozhraní pro konfiguraci `"hpc"` režimu v gridu a zahájení práce.

Příkaz `NumNodes` vrací celkový počet N uzlů, ve kterých se provádějí paralelní výpočty. Pro daný paralelní výpočet bude mít každý uzel číslo identifikátoru mezi 0 a $N-1$ po celou dobu trvání tohoto výpočtu. To je nezbytné neopomenout při programování. Toto číslo vrací

[140] http://cs.wikipedia.org/wiki/Message_Passing_Interface

příkaz `MyNode` (tj. hodnotu mezi 0 a $N-1$). Příkaz `Recieve` identifikuje uzel, ze kterého přichází zpráva (výsledek).

Příkazem `Launch` se spustí paralelně výpočet (úloha), který určuje první parametr příkazu. Může to být buď textový řetězec obsahující příkaz Maple nebo název samostatné procedury v Maple. Samostatnou procedurou chápejme proceduru, která se neodkazuje na žádné jiné procedury nebo data, která nejsou v paměti vzdáleného uzlu. Je-li parametr uveden jako procedura, další argumenty v příkazu `Launch` budou mít význam jako argumenty této procedury. Ukážeme to v následujícím příkladu, kde porovnáme dobu výpočtů s paralelizací a bez ní.

Příklad 7.2: Pro počítač, který má čtyři procesory (uzly) vytvořme v Maple 15 program, který vygeneruje čtyři náhodná celá kladná čísla z intervalu <1; 10^{60}> a dvě procedury `primeChecker` a `primeCheckerParalel`, které s využitím procedury `primetest` zjistí pomocí příkazu `isprime`, zda tato čísla jsou prvočísly, a provedou jejich rozklad na prvočinitele pomocí příkazu `ifactor`. Procedura `primeChecker` provede výpočet bez paralelizace, zatímco procedura `primeCheckerParalel` provede výpočet s paralelizací ve třech uzlech. Současně určíme i dobu výpočtu obou procedur.

Řešení: Nejprve vytvoříme booleovskou proceduru `primetest`, jejíž odezvou je hodnota `true` nebo `false` podle toho, zda číslo v jejím parametru je nebo není prvočíslem:

```
primetest := proc(qq1) isprime(qq1) end:
```

Dále vytvoříme proceduru `primeChecker`, která provede postupně rozklad čísel v poli `userData` na součin prvočinitelů pomocí příkazu `ifactor` a zkontroluje příkazem `isprime`, zda jde o prvočíslo. Výsledek uloží do pole `rslt` s výpisem:

```
primeCheckerParalel := proc()
  local i,rslt;
  global userData,primetest;
# Rozklad na prvočinitele a test na prvočíslo pro číslo v userData[1]
# a uložení do pole rslt
rslt:=[ifactor(userData[1]),primetest(userData[1])];
# Rozklad na prvočinitele a test na prvočíslo pro číslo v userData[i]
# a uložení do pole rslt
   for i from 2 to nops(userData) do;
     rslt:=rslt,[ifactor(userData[i]),primetest(userData[i])];
   end do;
  return [rslt];
end proc:
```

Dále vytvoříme proceduru `primeCheckerParalel`, která paralelně ve třech uzlech s využitím knihovny Grid provede rozklad čísel v poli `userData` na součin prvočinitelů pomocí příkazu `ifactor` a zkontroluje příkazem `isprime`, zda jde o prvočíslo. Výsledek uloží do pole `rslt` s výpisem:

```
primeCheckerParalel := proc()
  uses Grid;
  local nn,thisNode,i,myVal,rslt,rply,b1;
  global userData,primetest;
# Určení počtu procesorů (uzlů) v počítači
  nn:=NumNodes();
# Určení uzlu MyNode, kde bude probíhat řízení výpočtu
  thisNode:=MyNode();# v thisNode je uloženo číslo uzlu
# Uložení čísla z pole userData
  myVal:= userData[thisNode+1];
  b1:=primetest(myVal);
```

```
  rslt:=[ifactor(myVal),b1];
  if thisNode <> 0 then
    Send(0, rslt);
  else
    rply := rslt;
    for i from 1 to (nn-1) do
      rslt:=Receive(i);
      rply := rply, rslt;
    end do;
    return [rply];
  end if;
end proc:
```

Nyní vygenerujeme čtyři náhodná celá kladná čísla z intervalu <1; 10^{60}>:

```
userData := [seq((rand(1 .. 10^60))(), i = 1 .. 4)]
```

```
[85734699556216385660363789189617426501041607254030835173326 8,
84826925116982698807786580565374413952802115416831573964768 9,
75755840814277205310447516509151038877682574357249085854455 7,
19217453903686029572991360433497295148705637888674320419067 7]
```

Vyvoláme proceduru `primeChecker` a změříme dobu jejího výpočtu:

```
times:=time[real]():
result:=primeChecker();
printf("time=%f s.\n", time[real]()-times);
```

```
[[[(2)^2 (44378928019882801) (33627364201) (366259600721741) (8774273)
(44691696769), false],
[(11) (1543) (15935633527867) (237939531553) (6005924634994543141153)
(94559) (23209), false],
[(2711071256918207909203) (279431389422025557139243184714527729919),
false],
[(859845439) (27365355113971287116793028193158138841 9) (94597)
(8633701), false]]
time=25.986000 s
```

Vyvoláme proceduru prime`CheckerParalel` a rovněž změříme dobu jejího výpočtu:

```
times := time[real]();
result:=Grid:-
Launch(primeCheckerParalel,numnodes=4,imports=['primetest','userData']);
printf("time=%f s\n", time[real]()-times)
```

```
[[[(2)^2 (44378928019882801) (33627364201) (366259600721741) (8774273)
(44691696769), false],
[(11) (1543) (15935633527867) (237939531553) (6005924634994543141153)
(94559) (23209), false],
[(2711071256918207909203) (279431389422025557139243184714527729919),
false],
[(859845439) (27365355113971287116793028193158138841 9) (94597)
(8633701), false]]
time= 11.879000 s
```

Vidíme, že využitím knihovny Grid a paralelním výpočtem ve třech uzlech snížíme výpočetní dobu aplikace o více než polovinu.

7.2.2 Aplikační knihovna: Grid Computing Toolbox

Aplikační knihovna Grid Computing Toolbox pro provádění paralelních výpočtů v Maple využívá stejné příkazy jako knihovna Grid. To umožňuje distribuci výpočtů jak na počítači

s více procesory, tak v síti pracovních stanic a superpočítačů. Obsahuje i Personal Grid Server[141], který umožňuje simulovat a testovat paralelní aplikace na počítači s více procesory před jejich spuštěním na skutečné gridové síti.

Aplikační knihovna Grid Computing Toolbox má následující vlastnosti:

- samostatné spouštění gridu v lokálních sítích se snadno použitelným interaktivním rozhraním pro spuštění paralelních úloh;
- integraci se systémy plánování úloh, jako je systém dávkového spouštění úloh PBS[142] (*Portable Batch System*);
- vysokou úroveň paralelizace operací, například příkazy `map` a `seq`, stejně jako obecný, „paralel divide-and-conquer“ algoritmus;
- automatickou detekci deadlocku a zotavení.

Před instalací knihovny Grid Computing Toolbox je nutno mít zabudován Maple 15. Po instalaci knihovny se zobrazí na pracovní ploše ikona GridComputing ve Windows, odkud ji lze spustit kliknutím.

Nejprve je však nutné nainstalovat server a síť. Pro `"mpi"` režim je nutná implementace knihovny MPI pro předávání zpráv. Více informací o těchto knihovnách lze nalézt na webu[143]. Aktuální MPI implementace v Grid Computing Toolbox podporuje Microsoft® High-Performance Computing (HPC).

Grid Computing Toolbox využívá další příkazy z Grid knihovny pro distribuované paralelní výpočty v Maple, jako je např. příkaz `Server`, který spouští/zastavuje činnost gridového serveru pomocí příkazů `StartGrid` a `StopGrid`.

Spuštění Grid Computing Toolbox

Je-li třeba začít efektivně řešit danou úlohu, je vhodné spustit lokální server na počítači a vyvinout a otestovat paralelní aplikaci ještě před jejím spuštěním na skutečném gridu. V Maple zápisníku Personal Grid Server spustíme lokální server. Tímto způsobem je možné simulovat grid s libovolným počtem uzlů (procesorů) v daném lokálním počítači.

Aby bylo možné nastavit grid pro provoz na clusteru v `"hpc"` režimu, musí se nastartovat gridový server na každém počítači, kde se chce spustit paralelní výpočet. Jakmile je server nastartován, může kdokoli přístupem k serverům, spustit libovolný počet paralelních úloh až do vypnutí serverů. V režimech `"mpi"` a `"local"` režimy, žádný externí Grid Server proces se nemusí startovat.

Nyní možno spustit paralelní úlohy, ale musí se nejprve vybrat způsob zpracování pomocí příkazu Grid[Setup] a to buď `"mpi"` nebo `"hpc"` režimem pak provést příkaz `Grid[Launch]()`. Tím se otevře tento interaktivní zápisník, který umožní snadné spouštění úloh.

Existuje celá řada jiných způsobů, jak spustit paralelní úlohy v Maple. Může se použít příkaz `Grid[Launch]` příkaz s uvedením počtu uzlů a odkazem na kód, který se chce spustit, jak je to v Příkladu 7.2 nebo se může použít příkazový řádek pro spuštění paralelní dávkové úlohy.

---

141 http://www.maplesoft.com/support/help/Maple/view.aspx?path=Grid%2Fwks%2FPersonalGridServer

142 http://en.wikipedia.org/wiki/Portable_Batch_System

143 http://www.mcs.anl.gov/research/projects/mpi/

7.3 Paralelní výpočty v MATLABu

Výpočetní jádro MATLABu je implementováno s využitím základních matematických knihoven s podporou více jader. K využití této vlastnosti stačí základní software MATLAB. Žádná dodatečná aplikační knihovna (toolbox) ani psaní paralelních algoritmů není potřeba. Není ani nutné nijak upravovat starší programy. Zrychlení se projeví automaticky. Pro paralelní výpočty je možné využít aplikační knihovnu Parallel Computing Toolbox, která umožňuje běh až dvanácti paralelních procesů na jednom počítači (nikoliv v síti počítačů). Společné využití nástrojů Parallel Computing Toolbox a MATLAB Distributed Computing Server přináší možnost běhu paralelních procesů nejen na jednom počítači s více procesory, ale také v síti počítačů. Úlohy vytvořené v rámci Parallel Computing Toolbox jsou beze změny přenosné do prostředí MATLAB Distributed Computing Server. Tedy samotný Parallel Computing Toolbox lze použít také pro otestování správného nastavení paralelizace úlohy před finálním spuštěním na výpočetním clusteru.

7.3.1 Aplikační knihovna: Parallel Computing Toolbox

Aplikační knihovna Parallel Computing Toolbox je nástavba MATLABu, která umožňuje rozložit zátěž při zpracování výpočetně a datově náročných problémů mezi více procesory. Jedná se o nástavbu, která rozšiřuje jazyk MATLAB o konstrukce, jako jsou paralelní cykly, distribuovaná pole, paralelní numerické algoritmy, funkce na posílání zpráv a další, které umožní uživatelům plně využít paralelismus v uživatelových aplikacích.

Pro vývoj aplikací určených ke zpracování „velkých“ dat knihovna Parallel Computing Toolbox poskytuje distribuovaná pole a paralelní funkce s nimi. Tato sada nástrojů nabízí více než sto padesát paralelních funkcí pro práci s distribuovanými poli. Použití distribuovaných polí a funkcí ulehčuje konverzi sériové aplikace na paralelní zejména proto, že tyto funkce řeší složitou komunikaci a koordinaci mezi různými segmenty dané aplikace. Takto lze sériový program konvertovat na paralelní pouze použitím několika anotací.

Po instalaci Parallel Computing Toolbox stačí přejít do paralelního režimu příkazem `pmode start` a je možné začít vyvíjet paralelní algoritmy. Okno *Parallel Command Window* umožňuje snadno sledovat průběh výpočtu všech paralelních procesů současně. Pro větší kontrolu uživatelských aplikací funkcí z knihovny Parallel Computing Toolbox poskytuje MATLAB přístup k procedurám pro posílání zpráv založených na standartu MPI.

Knihovna Parallel Computing Toolbox rozšiřuje interaktivní prostředí MATLABu. Umožňuje používat své vlastní prostředí pro vývoj a prototypování paralelních aplikací. Příkaz `matlabpool` nastavuje paralelní prostředí v MATLABu (paralelní příkazový řádek), ve kterém lze např. příkazem `parfor` spouštět interaktivně paralelní cykly, provádět paralelní zpracování dat, ve kterém je možné používat distribuovaná pole a funkce pro manipulace s nimi. Příkazy spuštěné v paralelním příkazovém řádku jsou vykonávány současně ve všech zpracovávaných procesech. Jejich výsledky jsou dostupné okamžitě. To umožňuje lépe sledovat chování aplikace v každém kroku prostřednictvím víceprocesorového systému. Příkazy mohou být seskupeny do procedury a poté volány v režimu *off-line*. Knihovna poskytuje i nástroje na profilování, které umožní optimalizovat výkon paralelní aplikace.

7.3.2 MATLAB Distributed Computing Server

Knihovna Parallel Computing Toolbox poskytuje možnost spustit paralelně až čtyři lokální zpracovatelské procesy, a současně poskytuje uživateli nástroje k prototypování a testování paralelních aplikací. Plného výkonu však lze dosáhnout ve spojení s MATLAB Distributed Computing Server, který umožňuje uživatelům MATLABu spouštět aplikace v clusteru.

Poté, když je aplikace otestována na lokálním počítači, může být nahrána na MATLAB Distributed Computing Server a s jeho pomocí může být využit plný výpočetní výkon clusteru.

Tento server obsahuje základní plánovač úloh *MathWorks job manager* a přímo podporuje platformy LSF[144], Microsoft Windows Compute Cluster Server[145], PBS Pro a TORQUE plánovače[146]. Lze však použít i jiných plánovačů, které mohou být do tohoto prostředí integrovány pomocí obecného rozhraní s tímto serverem dodávaným.

7.3.3 Způsoby paralelního programování v MATLABu

Dávkové úlohy

Parallel Computing Toolbox vytváří prostředí pro spouštění a zpracování úloh (dávek) v MATLABu v režimu *off-line*. Spuštění úloh v tomto režimu se provádí pomocí funkce `batch`, která také zajistí mechanizmus pro přesnost dat mezi klientem a zpracovatelskými procesy. Tato funkce hraje dvě role, jednak poskytovatele mechanizmů pro běh paralelní aplikace, jednak současně spouští sériové programy na jednotlivých zpracovatelských procesorech.

Parallel Computing Toolbox také vytváří úlohy jako objekty, které mají obecný mechanizmus pro spouštění paralelních aplikací stejný, jako je pro spouštění sériových programů v dávkovém režimu.

Distribuovaná pole

Distribuovaná pole jsou speciální pole, která ukládají data do segmentů, se kterými pracují zpracovatelské procesy podílející se společně na paralelních výpočtech. K těmto strukturám jsou k dispozici operace, jako jsou transformace a dekompozice a jiné, jako by to byla obyčejná pole.

Distribuovaná pole ukládají data do více segmentů, což dovoluje zpracovávat mnohem větší množství dat než v jediné „*client session*“[147]. Vytvářet distribuovaná pole lze různými způsoby, a to přímo použitím konstruktorů jako `rand, ones, zeros` spojováním variantních polí (pole se stejným jménem, obsahující různá data na každém segmentu); distribucí replikovaných polí (pole se stejným jménem a daty na každém segmentu). Navíc jsou k dispozici funkce, pomocí kterých můžeme kontrolovat, jak budou data distribuována.

Paralelní mód (p-mode)

p-mode umožňuje interaktivní práci s paralelními úlohami (job) běžícími paralelně na více počítačích. Příkazy psané do paralelního příkazového řádku jsou vykonávány paralelně na všech strojích. K synchronizaci dochází vždy po vykonání příkazu.

7.3.4 Ukázky paralelních úloh

Paralelizovat aplikaci lze rozdělením problému na nezávislé úlohy a ty pak zpracovávat současně. K tomu slouží paralelní cykly `pafor`, které dávají uživateli možnost rozdělit a distribuovat tyto úlohy do více souběžně je zpracovávajících procesů. To lze ovšem realizovat za předpokladu, že dané úlohy jsou na sobě nezávislé. Paralelní cykly zajišťují přesun dat mezi řídící „session“ a zpracovávajícími procesy, a zároveň automaticky detekují přítomnost vol-

[144] http://www.platform.com/workload-management/high-performance-computing
[145] http://www.microsoft.com/cze/windowsserver2003/ccs/
[146] http://www.open-mpi.org/faq/?category=tm
[147] "MATLABovská session", odkud je spuštěna aplikace. Využívá DCT k tvorbě úkolu (job)

ných procesů pro zpracování. V případě nedostatku procesů se prostředí vrací do normálního sériového zpracování. Paralelní úlohy lze explicitně použít jako MATLAB funkce nebo MATLAB skripty. Ty pak lze spouštět interaktivně nebo jako dávky.

Interaktivní spuštění paralelního cyklu

Paralelní cyklus `pafor` umožňuje distribuovat nezávislé úkoly do více procesů. Syntaxe je velmi podobná klasickému příkazu `for` pro cyklus. Jedinými požadavky na použití příkazu `parfor` jsou podmínky, že žádná iterace není závislá na jiné a že při zpracování cyklu neprobíhá komunikace mezi procesy.

Tato konstrukce je dostupná po použití příkazu `matlabpool`. Hlavní vlastností příkazu `matlabpool` je umožnění interakce paralelního příkazového okna se zpracovatelskými procesy. To má za následek možnost spojení sériového a paralelního kódu, aniž by se musela spouštět dávka na clusteru. Příkaz `parfor` zjistí přítomnost zpracovatelských procesů pomocí příkazu `matlabpool` a provede přenos dat a výpočty.

Příklad 7.3: Nejprve navrhněme kód, který pomocí klasického cyklu spočítá hodnoty funkce sin(*x*) a vykreslí sinusoidu. Totéž pak proveďme pomocí paralelního cyklu.

Řešení: V MATLABu se klasický cyklus pro výpočet hodnot funkce sin(*x*) a její vykreslení provedou takto:

```
for i=1:1024
  A(i) = sin(i*2*pi/1024);
end
plot(A)
```

Interaktivní spuštění paralelního řádku se vyvolá příkazem `matlabpool`, který aktivuje ***worker,*** který lze chápat jako „MATLABovskou session“ vykonávající jednotlivé úlohy (task[148]). Na nich pak systém může spouštět paralelní cyklus. Přitom příkaz `matlabpool` lze spustit jak na lokálním počítači, tak i případně na vzdáleném výpočetním clusteru, a to takto:

```
>> matlabpool open
```

Pak lze modifikovat původní kód tak, aby byl použit paralelní cyklus `parfor`:

```
parfor i=1:1024
    A(i) = sin(i*2*pi/1024);
end
plot(A)
```

Jediným rozdílem je to, že se namísto klíčového slova `for` použije jeho paralelní ekvivalent `parfor`.

Protože je výpočet dílčích iterací prováděn na jednotlivých workerech, musí být každá iterace zcela nezávislá na ostatních iteracích. Například worker, který počítá hodnotu pro proměnnou *A*(100), může být jiný, než ten který počítá hodnotu pro proměnnou *A*(500). Navíc číselné hodnoty v závorce ani negarantují pořadí, ve kterém se budou jednotlivé výpočty provádět. Např. proměnná *A*(900) tak může být vypočtena dříve než proměnná *A*(400). Jediné místo, kde jsou všechny složky pole *A* přístupné je `klient`, kterému jsou z jednotlivých workerů po ukončení výpočtu cyklu tyto hodnoty předány.

Nakonec je potřeba uzavřít `matlabpool` a uvolnit alokované workery příkazem:

[148] Úloha, která je rozdělena na jednotlivé úkoly.

```
>> matlabpool close
```

Cyklus `parfor` distribuuje výpočet do více procesů rozdělením iterací mezi různé procesory. Distribuce výpočtu se provádí dynamicky. Namísto toho, aby byl alokován fixní rozsah procesorů, je na začátku každé iterace přiřazen volný procesor pro její zpracování. To zajišťuje rovnoměrné rozložení zátěže na všechny procesory. Navíc lze použít i rozšiřující syntaxi příkazů `matlabpool` a `parfor`, což zajistí lepší kontrolu nad zpracováním paralelního cyklu.

Spuštění dávkové úlohy

Pro předání zpracování úlohy z aktivní session (worker) na jiný worker slouží příkaz `batch`.

Příklad 7.4: V tomto příkladu ukažme, jak lze na vzdáleném počítači nechat provést úlohu (task) a jak získat zpět její výsledek.

Nejprve pomocí následujícího příkazu spustíme editor a v něm vytvoříme nový soubor *mywave.m*:

```
>>edit mywave
```

V tomto editoru zapíšeme program, který chceme nechat provést. Uložíme jej a editor zavřeme:

```
for i=1:1024
  A(i) = sin(i*2*pi/1024);
end
```

Dále použijeme příkaz `batch` na příkazovém řádku, čímž spustíme tento MATLABovský skript na odděleném workeru:

```
>>job = batch('mywave')
```

Spuštěný skript nyní neblokuje aktuálně spuštěný worker. Je však třeba vyčkat, dokud se skript neprovede. Teprve potom lze získat výsledek výpočtu a data vykreslit:

```
>>wait(job)
>>load(job, 'A')
>>plot(A)
```

Až je požadovaný MATLABovský skript proveden, uvolníme alokovanou paměť příkazem `destroy`:

```
>>destroy(job)
```

Spuštění dávkové úlohy s paralelním cyklem parfor

Nyní ukážeme, jak zkombinovat dávkové spuštění úlohy a její paralelní zpracování.

Podobně jako v Příkladu 7.4 nejprve otevřeme editor s novým souborem *mywave.m* a vložíme do něj programový kód, který obsahuje paralelní cyklus `parfor`:

```
>>edit mywave
parfor i=1:1024
  A(i) = sin(i*2*pi/1024);
end
```

Nyní opět použijeme příkaz `batch` na příkazovém řádku. Pomocí další dvojice parametrů předáme informaci o tom, že systém má použít příkaz `matlabpool` pro zpracování paralelního `parfor` a jaký je počet workerů, které se mají pro tento výpočet použít:.

```
>> job = batch('mywave','matlabpool',3)
```

Podobně jako v příkladu 7.4 získáme výsledek výpočtu. Data vykreslíme a paměť s daty uvolníme:

```
>>wait(job)
>>load(job, 'A')
>>plot(A)
>>destroy(job)
```

Použití distribuovaných polí, spmd a kompozice

Vzhledem k tomu, že jednotlivé workery v `matlabpool` dokáží mezi sebou komunikovat, můžeme pracovní pole mezi tyto workery rozdělit. Každý worker bude obsahovat jednu část tohoto pole a všechny workery budou mít k dispozici informaci o tom, která část pole patří kterému workeru. Nejprve otevřeme `matlabpool`:

```
>>matlabpool open  % Use default parallel configuration
```

Pak použijeme funkci `distributed` pro rozdělení pole jednotlivým workerům:

```
>>M = magic(4) % a 4-by-4 magic square in the client workspace
>>MM = distributed(M)
```

Nyní je proměnné *MM* přiřazeno rozdělené pole, které je ekvivalentní s polem *M* a se kterým můžeme manipulovat, resp. přistupovat k jednotlivým jeho prvkům stejně, jako by šlo o standardní pole (k distribuovanému poli *MM* přistupujeme z pozice `klient`, ale jednotlivé jeho části jsou fyzicky uloženy v paměti na jednotlivých workerech), například pomocí příkazů:

```
>>M2 = 2*MM;  % M2 is also distributed, calculation performed on workers
>>x = M2(1,1) % x on the client is set to first element of M2
```

Jestliže již s distribuovaným polem *MM* nepotřebujeme pracovat, pak `matlabpool` uzavřeme:

```
>>matlabpool close
```

Single Program Multiple Data (spmd)

Konstruktor `spmd` umožňuje definovat blok kódu v MATLABu, který bude spuštěn paralelně na dostupných (všech nebo i jen některých) workerech v `matlabpool`. Následující příkazy MATLABu vytvoří matici *R* typu (4,4) s náhodnými generovanými čísly v každém workeru. K jednotlivým vygenerovaným maticím lze pak přistupovat pomocí indexu *R*`{1}`, *R*`{2}` atd.

```
>>matlabpool    % Use default parallel configuration
>>spmd          % By default uses all labs in the pool
    R=rand(4);
end
```

Kompozice

Po provedení `spmd` bloku jsou hodnoty tohoto bloku přístupné z klientské pozice, avšak fyzicky jsou uloženy na jednotlivých workerech. Z pohledu `klient` tyto proměnné nazýváme ***kompozitními objekty***. Každý prvek tohoto kompozitního objektu odkazuje na jednotlivé hodnoty (data), které jsou uloženy v `matlabpool` u jednotlivých workerů. Vhledem k tomu, že jednotlivé elementy nemusejí být definovány na všech workerech, kompozitní objekt tak může mít i některé nedefinované elementy. Jestliže dále navážeme úvahami na zadání před-

chozího příkladu, pak na výstupu `klient` bude mít kompozitní proměnná *R* jeden element pro každý worker:

```
>>X = R{3};  % Set X to the value of R from lab 3.
```

Předchozí kód získá data z workeru označeného číslem 3 a uloží je do proměnné *X*. Následující kód přičte k proměnné *X* číslo 2 a odešle její hodnotu (data) do workeru označeného číslem 3:

```
>>X = X + 2;
>>R{3} = X; % Send the value of X from the client to lab 3.
```

V případě, že `matlabpool` zůstává otevřený (neukončíme-li jej příkazem `matlabpool close`), pak v jednotlivých `spmd` blocích zůstávají workerům přiřazena táž data.

```
spmd
    R = R + labindex  % Use values of R from previous spmd.
end
```

O typické použití `spmd` bloku jde tehdy, kdy chceme spouštět týž kód na jednotlivých workerech a každý z nich přistupuje k různým částem dat. Jako příklad uveďme následující kód:

```
spmd
    INP = load(['somedatafile' num2str(labindex) '.mat']);
    RES = somefun(INP)
end
```

V tomto případě proměnná `RES` z jednotlivých workerů je dostupná z pozice `klient` jako `RES{1}` pro worker 1, `RES{2}` pro worker 2 atd. Jsou dvě možnosti jak indexovat kompozitní proměnnou:

- `AA{n}` ……vrátí hodnotu *AA* z workeru označeného číslem *n*;
- `AA(n)` ……vrátí buňky pole obsahu proměnná *AA* z workeru označeného číslem *n*.

Nakonec (pokud již nebudeme potřebovat data a skončili jsme práci se všemi `spmd` bloky) uzavřeme `matlabpool` příkazem:

```
>>matlabpool close
```

Ačkoliv data zůstávají v jednotlivých workerech, je potřeba si uvědomit, že i přechodem z jednoho `spmd` bloku na jiný po ukončení `matlabpool` tato data ztratíme.

Příklad 7.5:

Metodu Monte Carlo vypočtěme odhad hodnoty Ludolfova čísla π. Výpočet provedeme dvěma způsoby:

1. Použitím sekvenčního cyklu `for`.
2. Použitím paralelního cyklu `parfor`.

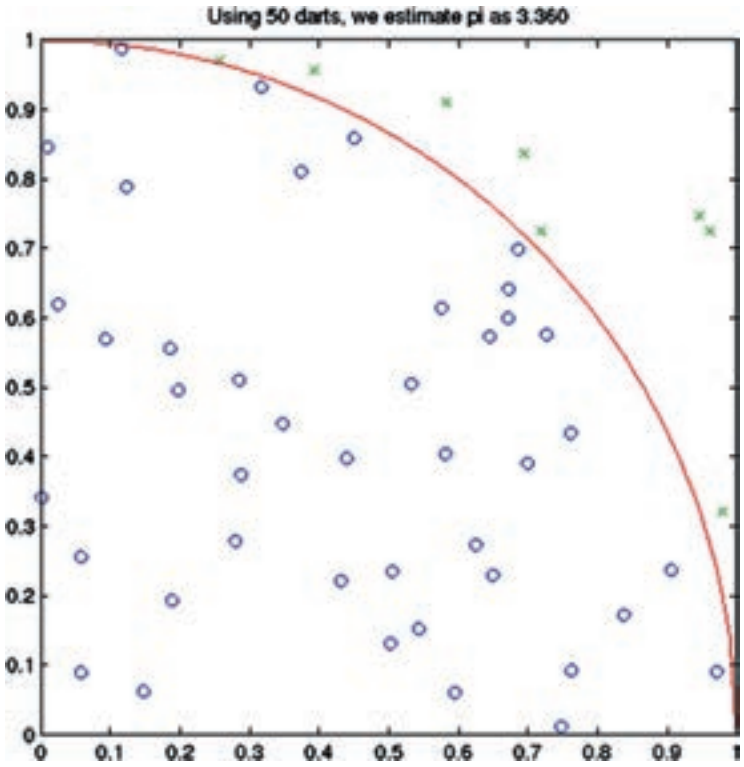
Řešení: Základem metody Monte Carlo pro odhad hodnoty π je výpočet obsahu *A* čtverce o straně délky 2*r* a obsahu *C* jemu vepsaného kruhu o poloměru *r*. Ludolfovo číslo π pak lze vypočítat z příslušného podílu obsahů těchto rovinných útvarů vynásobeného čtyřmi takto:

$$\pi = 4 \cdot \frac{C}{A}.$$

Metodu pro výpočet π založenou na tomto vztahu lze interpretovat tak, že zvolíme náhodně dostatečně velké množství bodů o souřadnicích [*x*; *y*] ve čtverci. Pak čtyřnásobek poměru počtu bodů, které náleží současně čtvrtkruhu i čtverci, k počtu všech zvolených bodů ve

čtverci se blíží k číslu π. (Získání náhodně zvolených bodů čtverce si lze představit například jako body získané hodem šipek do tohoto čtverce.)

Za účelem zjednodušení výpočtu zvolme $r = 1$ a omezíme se na čtvrtkruh, kde obě souřadnice x, y bodů jsou nezáporné, viz obrázek 7.1. Předpokládejme dále, že obě souřadnice x, y bodů čtverce jsou rovnoměrně distribuovány v intervalu <0; 1>.



Obr. 7.1 *Grafické znázornění výpočtu π pomocí metody Monte Carlo při padesáti náhodných bodech.*

Ad 1) Výpočet π pomocí sekvenčního cyklu `for`:

Nejprve vytvoříme algoritmus (funkci) `sequentialMonteCarloPI(darts,R)`, která má dva parametry: počet hodů šipek `darts` a poloměr kruhu `R`. Výstupem funkce je počet hodů šipek do čtverce, které zasáhly současně čtvrtkruh:

```
  function myPI = sequentialMonteCarloPI(darts,R)
  count = 0; % Number of darts that fell inside of the circle.
    for i = 1:darts
% Compute the X and Y coordinates of where the dart hit the
% square using uniform distribution.
      x = R*rand(1);
      y = R*rand(1);
      if x^2 + y^2 <= R^2
% Increment the count of darts that fell inside of the circle.
        count = count + 1;
       end
    end
% Compute PI using the ratio of darts that fell inside of
% the circle to the total number of darts thrown.
    myPI = 4*count/darts;
  end
```

Zvolíme počet hodů šipek `darts` 10^8, poloměr kružnice `R` položíme roven 1 a vyvoláme funkci `sequentialMonteCarloPI`. Vypočteme dobu trvání výpočtu. Zobrazíme výsledek výpočtu a dobu jeho trvání:

```
>> darts = 1e8;
>> R = 1;
>> tic
>> myPI = sequentialMonteCarloPI(darts, R);
>> seqElapsedTime = toc;
>> fprintf('The computed value of pi is %8.7f.\n',myPI);

The computed value of pi is 3.1419218.

>> fprintf('The sequential-loop Monte Carlo method executed in
%8.2f',...
seqElapsedTime);
>> fprintf(' seconds\n and computed %4.2e darts per second.\n',...
darts/seqElapsedTime);

The sequential-loop Monte Carlo method executed in    12.36 seconds
 and computed 8.09e+06 darts per second.
```

Ad 2) Výpočet π pomocí `for` paralelního cyklu `pafor`

Ve výše uvedeném algoritmu `sequentialMonteCarloPI` jednotlivé průchody cyklem `for` jsou na sobě nezávislé a že také nezáleží na pořadí, ve kterém budeme jednotlivé průchody počítat (operace sčítání je asociativní). Můžeme tak lokálně na jednotlivých workerech spočítat výskyty bodů v čtvrtkruhu. Podmínka pro nezávislost jednotlivých průchodů cyklem je klíčová pro to, aby se dal algoritmus zparalelizovat. Protože jednotlivé výpočty jsou nezávislé, vytvoříme paralelní algoritmus (funkci) `parallelMonteCarloPI` velmi podobných způsobem jako `sequentialMonteCarloPI` tak, že namísto cyklu `for` použijeme cyklus `pafor`. Jednotlivé výskyty bodů v čtvrtkruhu na konci cyklu `parfor` poté sečteme a dle výše uvedeného návodu vypočtěme odhad Ludolfova čísla π.:

```
  function myPI = parallelMonteCarloPI(darts,R)
  count = 0; % Number of darts that fell inside of the circle.
    parfor i = 1:darts
% Compute the X and Y coordinates of where the dart hit the
% square using uniform distribution.
      x = R*rand(1);
      y = R*rand(1);
        if x^2 + y^2 <= R^2
% Increment the count of darts that fell inside of the circle.
           count = count + 1;
         end
    end
% Compute PI
  myPI = 4*count/darts;
  end
```

Před výpočtem s funkcí `parallelMonteCarloPI` otestujeme, zdali je spuštěn `matlabpool`. Do proměnné `poolSize` uložíme počet workerů a vytiskneme jejich počet. V opačném případě vytiskneme chybovou zprávu:

```
>> poolSize = matlabpool('size');
>> if poolSize == 0
>>   error('distcomp:demo:poolClosed','This demo needs an open MATLAB
pool to run.');
```

```
>> end
>> fprintf('This demo is running on %d MATLABPOOL wor-
kers.\n',matlabpool('size'));

This demo is running on 8 MATLABPOOL workers.
```

Poté vynásobíme počet šipek `darts` počtem workerů `poolSize` a vyvoláme funkci `parallelMonteCarloPI`, vypočteme dobu trvání výpočtu. Vytiskneme výsledek výpočtu a dobu jeho trvání:

```
>> parforDarts = darts * poolSize;
>> R = 1;
>> tic
>> myPI = parallelMonteCarloPI(parforDarts,R);
>> parElapsedTime = toc;
>> fprintf('The computed value of Pi is %8.7f.\n',myPI);

The computed value of Pi is 3.1415835.

>> fprintf('The parallel-loop Monte Carlo method executed in %8.2f',...
parElapsedTime);
>> fprintf(' seconds\n and computed %4.2e darts per second.\n',...
parforDarts/parElapsedTime);

The parallel-loop Monte Carlo method executed in 14.72 seconds
 and computed 5.44e+07 darts per second.
```

Porovnáme-li dobu trvání výpočtu sekvenční a paralelní verze metody Monte Carlo vidíme, že i když jsou časy jednotlivých implementací srovnatelné, tak v cyklu `parfor` se provede `poolSize` krát více iterací. Spočítá se tak přesnější aproximace hodnoty Ludolfova čísla π. Změna kódu tohoto významného navýšení výkonu spočívá pouze ve změně klíčového slova `parfor` v cyklu programu.

8 Případové studie

8.1 Modelování šíření znečištění ve vodě

Znečištění vod je celosvětový problém, který je jednou z hlavních příčin úmrtí a onemocnění živých organismů. Voda může být znečištěna jak nebezpečnými toxickými látkami, tak látkami, které jsou v přirozeném množství prospěšné (trofizace). Cílem této studie bude vytvořit model určující koncentraci polutantu (znečištění) v jezeře v závislosti na čase, resp. na čase a poloze s využitím aplikačních softwarů Maple a MATLAB. Pro vytvoření (nejen) takového modelu jsou zásadní předpoklady, při jejichž platnosti model budeme sestavovat. Uvedeme dva možné přístupy: modelování s využitím obyčejné diferenciální rovnice a modelování s využitím parciální diferenciální rovnice.

8.1.1 Koncentrace znečištění jako funkce času

V případě modelování s využitím ***obyčejné diferenciální rovnice*** předpokládáme, že koncentrace znečištění $C(t)$ v jezeře je závislá pouze na čase t. Přitom pro zjednodušení zavedeme ještě několik dalších omezujících předpokladů:

- jezero je homogenní těleso, v němž je znečištění dokonale promíchané (a jeho koncentrace $C(t)$ tedy v každém místě stejná),
- znečištění podléhá pouze jedinému transportnímu procesu (pohybu), advekci (tj. unášení proudem vody), a ta je v každém bodě jezera stejná,
- jezero má jediný přítok Q_{in} a jediný odtok Q,
- znečištění se do jezera dostává pouze přítokem,
- objem jezera $V(t)$, jeho přítok $Q_{in}(t)$ i odtok $Q(t)$, (tj. průtok v místě, kde voda přitéká do jezera (tj. na přítoku) a průtok v místě, kde voda odtéká z jezera (tj. na odtoku)) jsou závislé na čase t.

Aplikací zákona zachování hmoty získáme rovnici:

$$\frac{d}{dt}\left(C(t)\cdot V(t)\right) = Q_{in}(t)\cdot C_{in}(t) - Q(t)\cdot C(t), \qquad (8.1)$$

kde jsou:

$V(t)$……. objem jezera v čase t [m^3],

$C(t)$ .....koncentrace znečištění v jezeře (resp. na jeho odtoku) v čase t [gm^{-3}],

$C_{in}(t)$ .....koncentrace znečištění na přítoku do jezera v čase t [gm^{-3}],

$Q_{in}(t)$ .....přítok (tj. objem vody na přítoku do jezera, který proteče daným místem za jednotku času t) [m^3s^{-1}],

$Q(t)$ .....odtok (tj. objem vody na odtoku z jezera, který proteče daným místem za jednotku času t) [m^3s^{-1}].

K rovnici (8.1) zbývá ještě doplnit ***počáteční podmínku***. Nechť tedy:

$$C(0) = C_0.$$

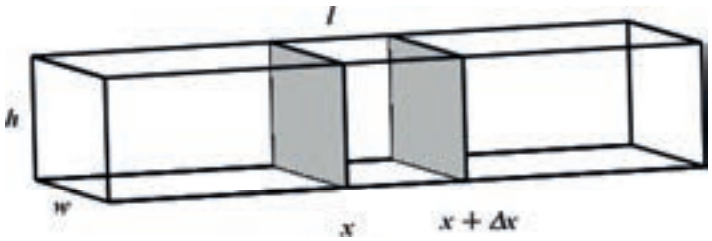
8.1.2 Koncentrace znečištění jako funkce času i polohy

V případě modelování s využitím ***parciální diferenciální rovnice*** chápeme koncentraci znečištění jako funkci času t i polohy x. K tomuto modelu přijmeme ještě několik předpokladů:

- jezero je homogenní těleso tvaru kvádru s jediným přítokem Q_{in} a jediným odtokem Q,
- znečištění se do jezera dostává pouze přítokem,
- znečištění v jezeře podléhá pouze jedinému transportnímu procesu, advekci,
- rychlost advekce je v každém bodě jezera stejná a přitom rovná rychlosti odtoku vody Q z jezera,
- objem jezera $V(t)$, přítok $Q_{in}(t)$ i odtok $Q(t)$ jsou závislé na čase t, (délku nádrže uvažujme konstantní).

Nechť má jezero délku l, výšku (resp. hloubku) h a šířku w. Délka l vodního tělesa je konstantní, zbylé rozměry (h a w) se však v čase mohou měnit, z toho vyplývá, že plocha kolmého průřezu vodním tělesem v čase t je $A(t) = h(t) \cdot w(t)$.

Podobně jako v předchozím modelu (8.1) vyjádříme pomocí zákona zachování hmotnosti časovou změnu množství znečištění v jezeře v oblasti ohraničené kolmými plochami v bodech x a $x + \Delta x$ (viz obrázek 8.1).



Obr. 8.1 *Nákres tvaru jezera.*

Obdržíme tak rovnici:

$$\frac{\partial}{\partial t}\left(C(x,t)\cdot A(t)\cdot \Delta x\right) = Q(t)\cdot C(x,t) - Q(t)\cdot C(x+\Delta x,t), \qquad (8.2)$$

kde jsou:
$A(t)$.....plocha kolmého průřezu vodním tělesem v čase t [m^2],
$C(x,t)$.....koncentrace znečištění v jezeře (resp. na jeho odtoku) v místě x v čase t [gm^{-3}],
$Q(t)$ .....odtok z jezera v čase t [m^3s^{-1}].

Zavedením obvyklého limitního přechodu pro $\Delta x \to 0$ získáme rovnici modelu:

$$\frac{\partial}{\partial t}\left(C(x,t)\cdot A(t)\right) = -Q(t)\cdot \frac{\partial}{\partial x}C(x,t). \qquad (8.3)$$

K rovnici (8.3) dále přiřadíme *počáteční a okrajovou podmínku*.

Počáteční podmínka $C_0(x)$ definuje koncentraci znečistění $C(x,t)$ v každém místě x v čase $t = 0$:

$$C(x,0) = C_0(x).$$

Okrajová podmínka $C_{in}(t)$ určuje koncentraci znečištění $C(x,t)$ na „okraji“ jezera (chápejme tím místo přítoku vody do jezera) v libovolném čase t:

$$C(0,t)= C_{in}(t).$$

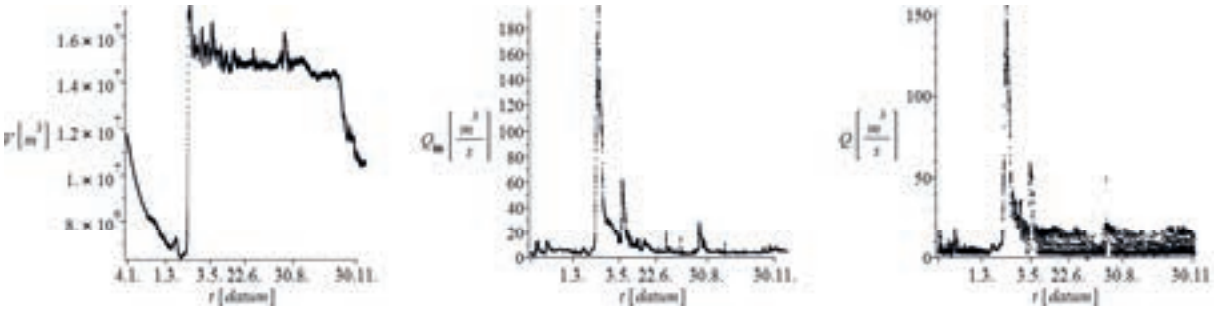
8.1.3 Reálná data z Brněnské přehrady

Brněnská přehrada se již řadu let potýká s problémem přebytku živin, které výrazně ovlivňují kvalitu vody. V rámci projektu Čistá Svratka [149] je sledována jakost vody a stanovována mj. koncentrace fosforu a dusíku na přítoku i odtoku vody do/z přehrady. Povodí Moravy[150] poskytlo data z roku 2006.

Tabulka 8.1: Měřené hodnoty koncentrace dusíku a fosforu na přítoku a odtoku [mg/l].

Datum	Dusík		Fosfor	
	Přítok	Odtok	Přítok	Odtok
4. 1. 2006	4,64	3,85	0,148	0,103
1. 2. 2006	5,70	5,01	0,116	0,105
1. 3. 2006	6,78	5,73	0,120	0,081
19. 4. 2006	7,69	8,15	0,092	0,101
3. 5. 2006	8,71	7,20	0,161	0,079
31. 5. 2006	6,74	6,25	0,153	0,041
22. 6. 2006	5,23	5,46	0,198	0,044
2. 8. 2006	4,24	3,85	0,280	0,033
30. 8. 2006	4,76	3,71	0,163	0,075
4. 10. 2006	3,78	3,35	0,130	0,070
31. 10. 2006	3,64	3,27	0,149	0,061
30. 11. 2006	4,71	4,47	0,142	0,089

Zatímco koncentrace dusíku a fosforu jsou měřeny přibližně jedenkrát za měsíc, hodnoty objemu nádrže a průtoku vody na přítoku a odtoku jsou zaznamenávány každou hodinu. Vývoj těchto hodnot z období od 4. ledna do 30. listopadu 2006 zachycují časové řady, které jsou vizualizovány pomocí Maple na obrázku 8.2.



(a) Objem nádrže $V(t)$ *(b)* Přítok vody $Q_{in}(t)$ *(c)* Odtok vody $Q(t)$

Obr. 8.2 *Časové řady objemu nádrže a průtoku vody na přítoku a odtoku.*

8.1.4 Modelování koncentrace dusíku a fosforu

Nyní využijeme obou modelů (8.1) a (8.3) k simulaci vývoje koncentrace dusíku (příp. fosforu) na odtoku vody z přehrady v období od 4. ledna do 30. listopadu 2006. Predikované hodnoty znečištění následně srovnáme s měřenými hodnotami (viz tabulka 8.1).

[149] http://www.cistasvratka.cz/
[150] http://www.pmo.cz

Modely mají následující parametry: $C_{in}(t)$, $V(t)$ (resp. $A(t)$), $Q_{in}(t)$ a $Q(t)$, které jsou všechny závislé na čase t.

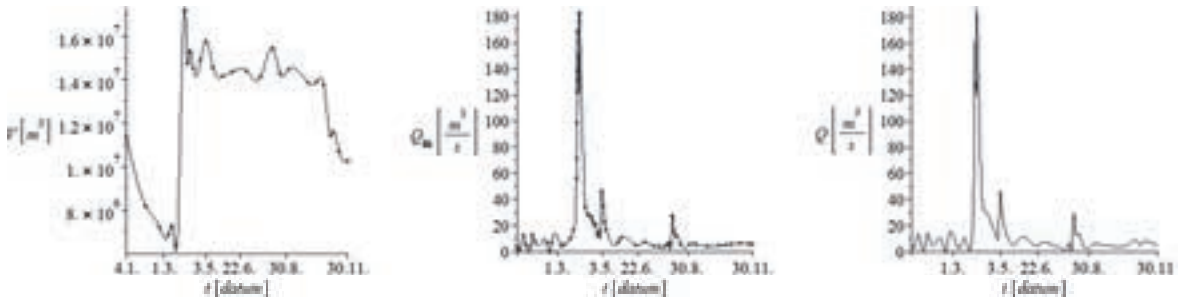
Hodnoty koncentrace znečištění na přítoku $C_{in}(t)$ známe pouze přibližně v měsíčních intervalech a nevíme, jakých hodnot koncentrace znečištění dosahuje mezi dvěma měřeními (ani je nedokážeme ohraničit). Přesto z hodnot v tabulce 8.1 lze předpokládat, že se koncentrace znečištění v průběhu roku „výrazně" neměnila. Budeme proto uvažovat, že funkce $C_{in}(t)$ je spojitá a mezi po sobě jdoucími měřeními je monotónní (např. lineární).

Hodnoty zbylých parametrů $V(t)$ (resp. $A(t)$), $Q_{in}(t)$ a $Q(t)$ známe v hodinových intervalech. Zatímco hodnota objemu nádrže $V(t)$ se v průběhu jedné hodiny změní „minimálně" (zpravidla méně než o 1 %), v případě přítoku $Q_{in}(t)$ a odtoku $Q(t)$ tomu tak být nemusí (navíc se může měnit znaménko rozdílu těchto hodnot, což je pro model (8.1) zásadní). V obou modelech přitom předpokládáme, že platí:

$$\frac{d}{dt}V(t) = Q_{in}(t) - Q(t)$$

Z tohoto důvodu zvolíme jeden z průtoků, např. odtok $Q(t)$, který „dopočítáme" ze znalosti objemu vody $V(t)$ v jezeře a přítoku $Q_{in}(t)$. Funkci $V(t)$ pro reprezentaci objemu nádrže v čase t a funkci $Q_{in}(t)$ pro reprezentaci přítoku (tedy jako modely závislostí objemu nádrže a přítoku na čase t) získáme proložením naměřených dat vhodnými spojitými (a hladkými) funkcemi. Vzhledem k charakteru dat se může stát, že „dopočítaná" funkce $Q(t)$ bude pro některá t záporná. V takových případech je třeba funkce $Q_{in}(t)$ a $Q(t)$ upravit tak, aby obě byly nezáporné.

K proložení naměřených dat funkcemi $V(t)$ a $Q_{in}(t)$ zvolme kubické splajny[151], které zajistí spojitost derivace v uzlových bodech. Uzlové body vybereme tak, aby funkce $V(t)$ a $Q_{in}(t)$ zachovaly „významné" lokální extrémy a přibližně odpovídaly naměřeným datům. Z dvaceti sedmi uzlových bodů pro $V(t)$ a šedesáti uzlových bodů pro $Q_{in}(t)$ získáme funkce vykreslené pomocí Maple na obrázku 8.3.

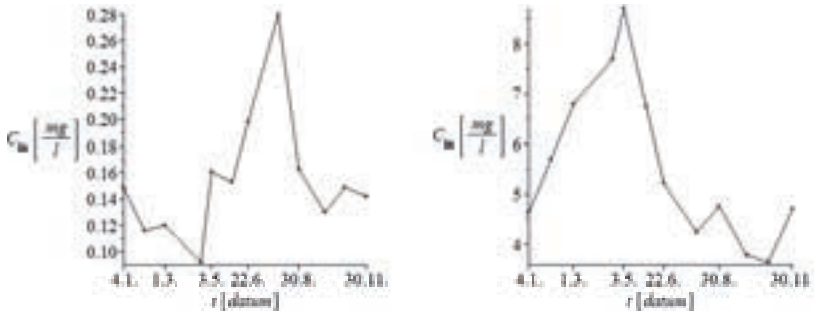


(a) Objem nádrže V(t) *(b) Přítok vody $Q_{in}(t)$* *(c) Odtok vody Q(t)*

Obr. 8.3 *Časové řady objemu nádrže a průtoku vody na přítoku a odtoku.*

Jak již bylo zmíněno, funkce $C_{in}(t)$ budeme uvažovat po částech lineární a s uzly shodnými s dny měření. Bude se tedy jednat o lineární splajny. Pro stanovené koncentrace dusíku a fosforu na přítoku dostáváme funkce vizualizované pomocí Maple na obrázku 8.4.

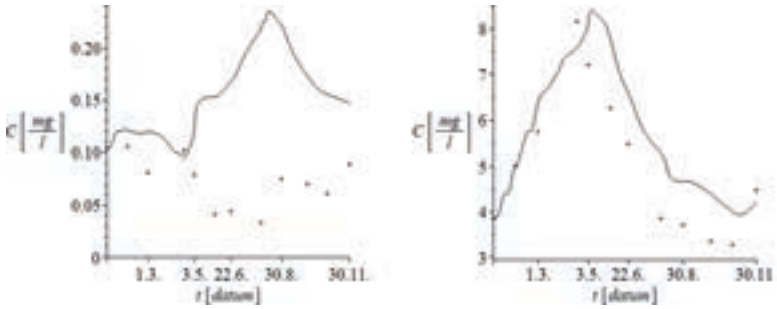
[151]Splajny jsou aproximující, po částech polynomiální, funkce. Přívlastek „kubický" značí, že příslušný polynom je nejvýše třetího stupně, viz [3].



(a) Fosfor *(b) Dusík*

Obr. 8.4 *Aproximace časových řad koncentrace dusíku a fosforu na přítoku do nádrže.*

Řešení modelu (8.1) v Maple pro výše popsané parametry je zobrazeno spolu s naměřenými hodnotami na odtoku (jednotlivé body v grafu) na obrázku 8.5.



(a) Fosfor *(b) Dusík*

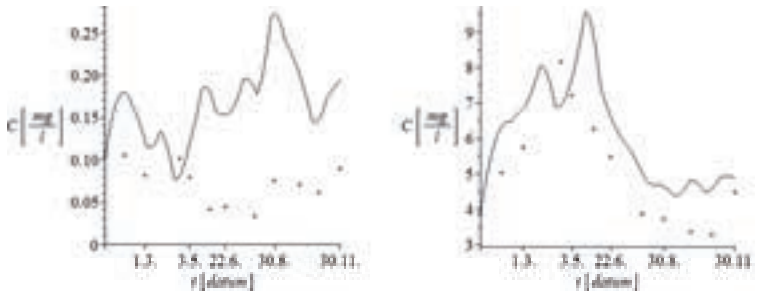
Obr. 8.5 *Modelované a naměřené koncentrace dusíku a fosforu na odtoku z nádrže.*

Pro model (8.3) je třeba navíc specifikovat funkce $A(t)$ a $C_0(x)$.

Předpokládáme, že plocha průřezu nádrží $A(t)$ je v každém místě nádrže stejná, a tedy přímo úměrná objemu nádrže $V(t)$. Konstantou přímé úměrnosti je převrácená hodnota délky l nádrže, přičemž $l = 9\,300\ m$, viz [27].

Dále platí, že $C_0(0)$ je rovno koncentraci znečištění na přítoku a $C_0(9300)$ je koncentrace znečištění na odtoku vody z přehrady. Pro jednoduchost předpokládejme, že koncentrace znečištění v čase $t = 0$ závisí na poloze (tím míníme na vzdálenosti od místa přítoku) lineárně.

Řešení modelu (8.3) v Maple pro výše uvedené parametry je zobrazeno spolu s naměřenými hodnotami na odtoku (jednotlivé body v grafu) na obrázku 8.6.



(a) Fosfor *(b) Dusík*

Obr. 8.6 *Modelované a naměřené koncentrace dusíku a fosforu na odtoku z nádrže.*

Z obrázků 8.5 a 8.6 je vidět, že modely (8.1) a (8.3) nezachycují dostatečně přesně naměřené koncentrace dusíku a fosforu na odtoku z nádrže.

Zatímco při modelování koncentrace dusíku na obou obrázcích 8.5 i 8.6 modelové (teoretické) hodnoty přibližně kopírují trend měřených dat, u fosforu tomu tak není. Na obou obrázcích 8.5 a 8.6 se trend modelu koncentrace fosforu výrazně liší od trendu měřených hodnot. V obou případech pro modelování koncentrací daných znečišťujících látek předpokládáme, že znečištění podléhá jedinému transportnímu procesu – advekci, což je zejména v případě fosforu nesprávný předpoklad (více v části 8.1.6), a proto modelováním dochází ke značnému zkreslení.

8.1.5 Výpočet řešení v Maple a v MATLABu

Měřená data z Brněnské přehrady jsou uložená v souboru ***D:/data prehrada.xlsx*** v listu pojmenovaném ***data***. Hodnoty příslušné danému parametru se nacházejí v samostatném sloupci.

Postup výpočtu numerického řešení modelů (8.1) a (8.3) užitím systému Maple nebo systému MATLAB spočívá v načtení dat ze souboru ***data_prehrada.xlsx*** do zvoleného systému, vytvoření funkcí $C_{in}(t)$, $V(t)$, $A(t)$, $Q_{in}(t)$ pomocí splajnů, „dopočítáním“ $Q(t)$ a přiřazením lineární funkce $C_0(x)$. Nakonec pak numericky vyřešíme příslušnou (obyčejnou/parciální) diferenciální rovnici.

Při výpočtech zahrnujících derivaci objemu (tj. při počítání odtoku $Q(t)$ a řešení $C(t)$, resp. $C(x,t)$) získané hodnoty vydělíme číslem 3 600, aby byla změna objemu vztažena na sekundu (nikoli na hodinu). Při aproximaci přítoku $Q_{in}(t)$ splajny jsou přidány interpolační uzly v časech t = 1 800 h a t = 1 900 h tak, aby $Q(t)$ i $Q_{in}(t)$ byly nezáporné funkce.

Systém MATLAB poskytuje pro řešení parciálních diferenciálních rovnic prvního řádu pouze příkaz `pdepe`, který však „umí“ řešit jen diferenciální rovnice parabolického a eliptického typu. Modelová rovnice (8.3) je přitom hyperbolického typu. Z webové stránky profesora Shampineho [28] lze pro hyperbolické parciální diferenciální rovnice stáhnout implementované numerické metody s názvem `hpde`, které využijeme.

V Příloze této publikace jsou uvedeny zdrojové kódy pro výpočet obou modelů v Maple i v MATLABu pro koncentraci dusíku.

8.1.6 Analýza neurčitosti

Neurčitosti v modelu

Nejprve se budeme zabývat neurčitostmi v matematickém modelu. Při sestavování matematického popisu zadaného problému jsme využili dvou různých postupů modelování. Jednak pomocí obyčejné diferenciální rovnice (8.1) a jednak pomocí parciální diferenciální rovnice (8.3), kde jsme aproximovali parametry modelů $V(t)$ (resp. $A(t)$), $Q_{in}(t)$ a $Q(t)$ pomocí kubických splajnů z naměřených dat.

Na první pohled (viz obrázky 8.5 a 8.6) dávají oba modely podobné výsledky. Avšak bližším vizuálním vyhodnocením lze konstatovat, že pro data, která máme k dispozici, se jako vhodnější jeví model (8.1).

Dále z grafů na obrázcích 8.5 a 8.6 vidíme, že za adekvátní lze považovat oba modely v případě modelování šíření koncentrace dusíku, ale pro fosfor jsou oba modely nevyhovující.

Brněnská přehrada se nejen v roce 2006 potýkala s přemnožením sinic. Nádrž byla eutrofní až hypertrofní (viz klasifikace vod podle úživnosti [29]). Vodní mikroorganismy a rostliny přitom fosfor přijímají a zabudovávají jej do své biomasy. Určitá část fosforu

v nádrži navíc tvoří nerozpustné sloučeniny s kovy (jako je vápník, železo, hliník či hořčík), které se stávají součástí sedimentu na dně nádrže[152].

Vzhledem k výše zmíněnému je pro modelování šíření fosforu v Brněnské přehradě nutné oba modely (8.1), (8.3) doplnit o popsané procesy, aby daly adekvátnější výsledky.

Datová neurčitost

Datovou neurčitostí jsou ovlivněny interpolační polynomy (resp. splajny), jimiž modelujeme funkce $V(t)$, $Q(t)$, $Q_{in}(t)$, $C_{in}(t)$, $C_0(x)$, příp. $A(t)$.

Objem nádrže $V(t)$, resp. plocha průřezu nádrží $A(t)$, se během hodiny výrazně nemění. Můžeme proto neurčitost v hodnotách těchto dvou parametrů zanedbat a zaměřit se na neurčitosti zbylých parametrů.

Vliv parametrů na řešení modelu vyjádříme globální analýzou citlivosti pro koncentraci dusíku na odtoku.

Zvolíme časový interval mezi prvními dvěma měřeními koncentrace znečištění $C_{in}(t)$ (jde o dobu mezi daty 4. ledna až 1. února 2006), na němž aproximujme objem nádrže $V(t)$ lineární funkcí. Objem nádrže v tomto období klesá, takže odtok $Q(t)$ musí být (v průměru) větší, než je přítok. Přítok $Q_{in}(t)$ budeme reprezentovat náhodnou veličinou s trojúhelníkovým rozdělením pravděpodobnosti na intervalu $\langle \min(Q_{in}(t)); \max(Q_{in}(t)) \rangle$ pro t z uvedeného období[153] a odtok $Q(t)$ dopočítáme.

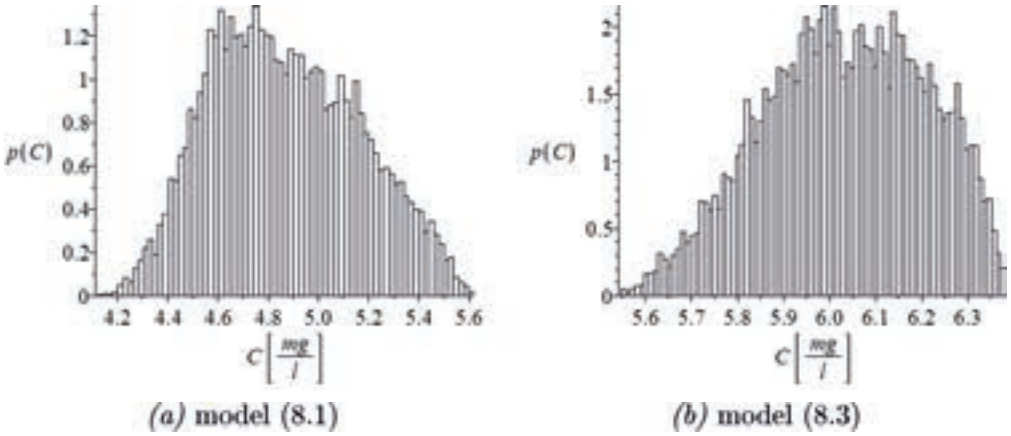
Podobně koncentraci znečištění na přítoku $C_{in}(t)$ přiřadíme rovnoměrné rozdělení pravděpodobnosti na intervalu $\langle C_{in}(4.1.); C_{in}(1.2.) \rangle$.

Vzhledem k uvedeným omezením a jednoduchosti modelu zkoumáme vliv pouze těchto dvou nezávislých parametrů $Q_{in}(t)$ a $C_{in}(t)$ na řešení modelu. Analýzou globální citlivosti pro model (8.1) provedenou Soboľovou metodou (i s aplikací korekčního členu) získáváme:

$$S_{Q_{in}} = 0,32, \quad S_{C_{in}} = 0,61.$$

Jestliže totéž provedeme pro model (8.3), kde zkoumáme vliv parametrů $Q(t)$ a $C_{in}(t)$ na řešení modelu, obdržíme:

$$S_Q = 0,03, \quad S_{C_{in}} = 0,84.$$



Obr. 8.7 *Histogramy modelových hodnot koncentrace dusíku na odtoku 1. 2. 2006.*

[152] Více o znečištění povrchových vod živinami v [36].

[153] Trojúhelníkové rozdělení vyplývá z histogramu hodnot přítoku na daném intervalu.

Histogramy predikovaných hodnot koncentrací dusíku dne 1. února 2006 na odtoku poskytuje obrázek 8.7.

Celkově lze učinit závěr, že z pohledu datové neurčitosti ovlivňuje výslednou koncentraci dusíku na odtoku nejvíce hodnota koncentrace dusíku na přítoku. Hodnota koncentrace dusíku na přítoku (resp. odtoku) pak vyjadřuje, „jak rychle“ se hodnota koncentrace dusíku na odtoku bude přibližovat hodnotě koncentrace dusíku na přítoku.

Neurčitosti v aplikaci modelu

Neurčitost v aplikaci modelu vychází jednak z uložení čísel v počítači a jednak z použití numerických metod při řešení diferenciálních rovnic. Jelikož samy parametry ve svých hodnotách zahrnují neurčitosti mnohonásobně převyšující neurčitost (v tomto případě chybu) způsobenou zaokrouhlením čísla při uložení v počítači, je bezpředmětné řešit vliv těchto chyb na řešení modelu.

Při výpočtu řešení modelu (8.1) byla v Maple standardně použita (vložená) numerická metoda Runge-Kutta-Fehlberg řádů 4 a 5, viz [17]. V MATLABu byla zvolena obdobná metoda Dormand-Prince stejných řádů, viz [30]. Při výpočtu Fehlbergovy metody v Maple je volen krok výpočtu tak, aby pro odhad chyby každého z nich platilo:

$$odhad\_chyby \leq abs\_err + rel\_err \cdot |C(t)|,$$

V MATLABu podobně pro odhad chyby každého kroku metody Dormanda-Prince platí:

$$odhad\_chyby \leq \max(abs\_err, rel\_err \cdot |C(t)|),$$

přičemž *abs_err* a *rel_err* značí tolerance pro absolutní a relativní chybu, jejichž standardní hodnoty jsou 10^{-7} a 10^{-6} v Maple, 10^{-6} a 10^{-3} v MATLABu.

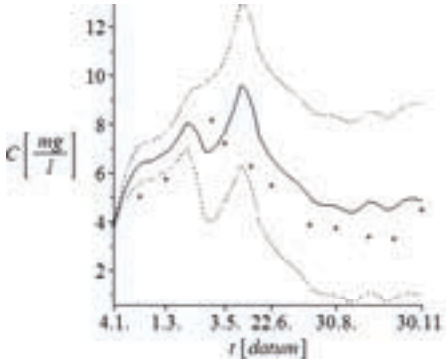
V Maple neprovede Fehlbergova metoda více kroků, než je počet vyhodnocení zadané derivace funkce. Maximální možný počet vyhodnocení derivace funkce přitom určuje parametr `maxfun`, jehož standardní hodnota je 30 000. Jelikož pro všechna t z roku 2006 platí $|C(t)| < 9$, můžeme jednoduše shora odhadnout chybu numerického řešení v Maple hodnotou[154] $30000 \cdot 10^{-5} = 0{,}3$.

V MATLABu získáme nastavením parametru `Stats` informaci o počtu kroků, který pro celé časové období 4. 1. – 30. 11.2006 je roven 28. Chybu je pak možné shora odhadnout hodnotou $28 \cdot 10^{-2} = 0{,}28$.

K výpočtu řešení modelu (8.3) poskytuje Maple také výpočet odhadu chyby numerického řešení. Ten pak můžeme použít například k vykreslení tzv. chybových mezí (resp. hranic), mezi nimiž se přesné řešení nachází. Pro standardní nastavení v našem příkladu tak situaci vizualizuje obrázek 8.8, v němž jsou chybové meze vyznačeny bodovými grafy.

Funkce `hpde` v MATLABu ohraničení pro chybu výpočtu neposkytuje.

[154] Lze ukázat, že počet vyhodnocení derivace funkce, který Maple při výpočtu provede, je roven 1605. To nabízí možnost odhad chyby numerického řešení dále snížit.



Obr. 8.8 *Řešení modelu (8.3) s chybovými mezemi.*

Pro lepší pochopení problematiky (nebo pro procvičení dříve zavedených pojmů a vztahů) doporučujeme čtenáři provést další analýzy, jako jsou např.: sledování vývoje hodnoty $C(t)$ modelu (8.1), jestliže hodnota $C_{in}(t)$ bude konstantní nebo bude nulová; dále pak zjištění doby, za jak dlouho podle modelu (8.1) klesne koncentrace znečištění v jezeře kupříkladu pod 5 % aktuální hodnoty v případě, že na přítoku již do jezera nebude vstupovat žádné znečištění a například $Q_{in}(t) = Q(t) = 5\ m^3 \cdot s^{-1}$ a $V(t) = 10^7\ m^3$; anebo určení lokální citlivosti parametrů u obou modelů, (8.1) a (8.3), jestliže jsou parametry konstantní, tj. $C_{in}(t) = C_{in}$, $V(t) = V$, $Q_{in}(t) = Q_{in}$, $Q(t) = Q$, $A(t) = A$.

8.2 Modelování znečištění v ovzduší

Na šesti lokalitách v okolí Brna byl monitorován prach automatickým vzorkovačem, ve kterém byl poté analyzován obsah chemických látek. Na každé z lokalit (cementárna, kamenolom, letiště, ulice v centru Brna, malá obec a velká obec) byla automatickým vzorkovačem naměřena data, která se pro podchycení variability měřila pětkrát v pravidelném intervalu. K jejich analýze se využil systém SPSS, jehož úkolem bylo zjistit základní statistiku všech měřených látek v ovzduší a poté u vybrané látky zjistit, jestli se od sebe významně liší její koncentrace v monitorovaných lokalitách. Využijme k tomu metodu ANOVA.

8.2.1 Příprava dat pro SPSS

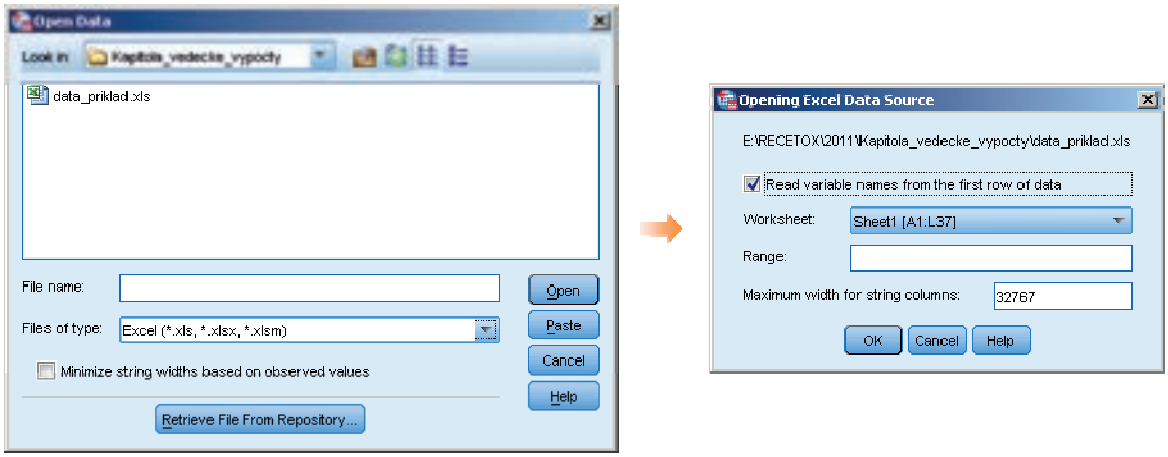
Vstupní data pro užití SPSS je vhodné upravit do standardní podoby, kdy sloupce tvoří jednotlivé parametry (např. název lokality, látka, popis aj.), první řádek obsahuje názvy parametrů a do dalších řádků zapisujeme naměřené hodnoty či konkrétní záznamy ze vzorkovače (nejčastěji text, datum nebo číslo; viz obrázek 8.9). Tento postup je vhodné dodržet nejen v tomto případě, ale u každé analýzy dat programem SPSS. Ušetříme následné možné problémy při zpracování dat.

	A	B	C	D	E	F	G	H	I
1	ID	Oznaceni	LOKALITA	Frakce	Frakce_2	Křemen	8_netekave PAH	DDT	Naftalen
2	1	Cem_A	1	a	1	21.13	0.01394	788.03186	190.944
3	2	Cem_B	1	b	2	22.63	0.01469	908.01625	768.724
4	3	Cem_C	1	c	3	20.69	0.01536	971.72527	3216.141
5	4	Cem_D	1	d	4	19.11	0.02886	175.12728	3199.026
6	5	Cem_E	1	e	5	19.66	0.09075	573.64287	1644.162
7	6	Cem_F	1	f	6	16.86	0.26875	726.76232	364.661
8	8	Lom_A	2	a	1	16.01	0.05788	69.32828	6728.109

Obr. 8.9 *Standardní úprava dat pro vstup do SPSS.*

8.2.2 Načtení dat do SPSS

Po spuštění SPSS načteme data pomocí vstupního dialogu (viz. kap. 6.2.3, obrázek 6.13). V dialogu musíme nastavit u pole Files of type hodnotu `Excel`, v opačném případě nebude soubor v dialogovém okně viditelný. Po stisknutí tlačítka Open se objeví další dialogové okno (obrázek 8.10), ve kterém zatrhneme hodnotu `Read variable names from first row of data`, jelikož pro tento formát načtení máme připravená data. V položce Worksheet se automaticky nastaví maximální rozsah dat z vybraného listu Excelu.

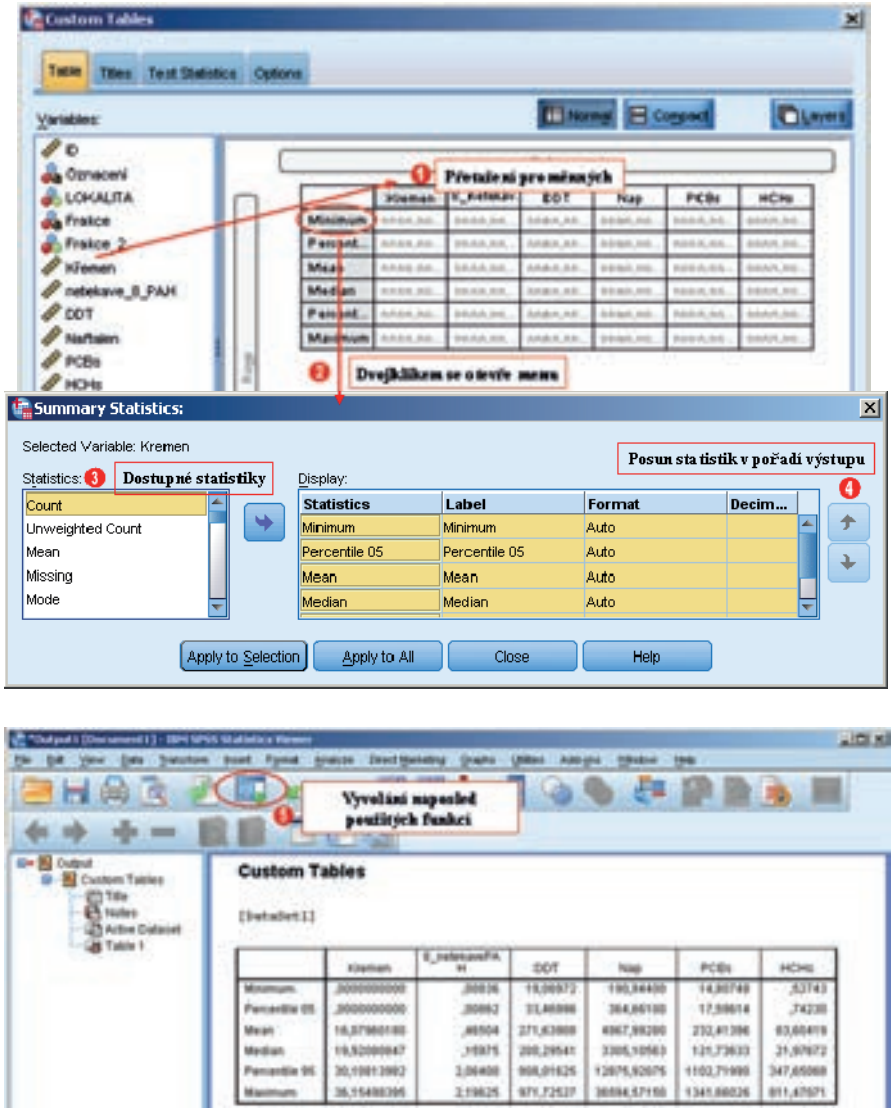


Obr. 8.10 *Import dat.*

Při importu se automaticky opraví nepřípustné názvy proměnných, mezery jsou nahrazeny podtržítkem a před čísla na začátku názvu je vložen znak @ (zavináč). Názvy lze v módu Variable View dodatečně editovat ručně. Také je vhodné doplnit do tabulky sloupec Label, popisky proměnných a zkontrolovat datový typ jednotlivých proměnných.

8.2.3 Analýza dat

Při analýze dat nejdříve spočítáme základní statistiku (minimum, 5 % a 95 % percentil, medián, průměr a maximum) pro všechny látky v prachu. K tomuto je nejvhodnější použít menu (Analyze > Tables > Custom Tables), kde do sloupců vložíme hodnocené proměnné a dvojitým kliknutím na položku Mean pod jejich jménem nastavíme požadované statistické ukazatele a jejich pořadí.

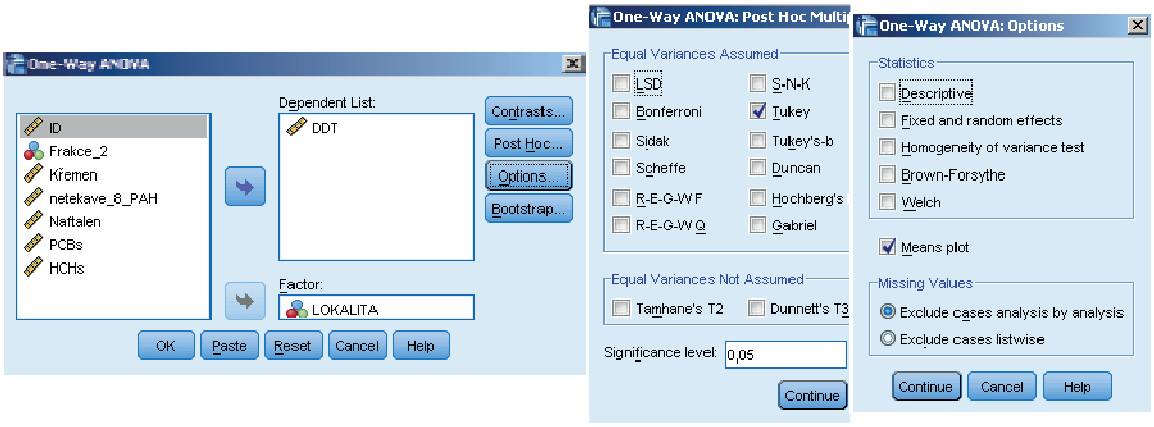


Obr. 8.11 *Nastavení analýzy a výstup.*

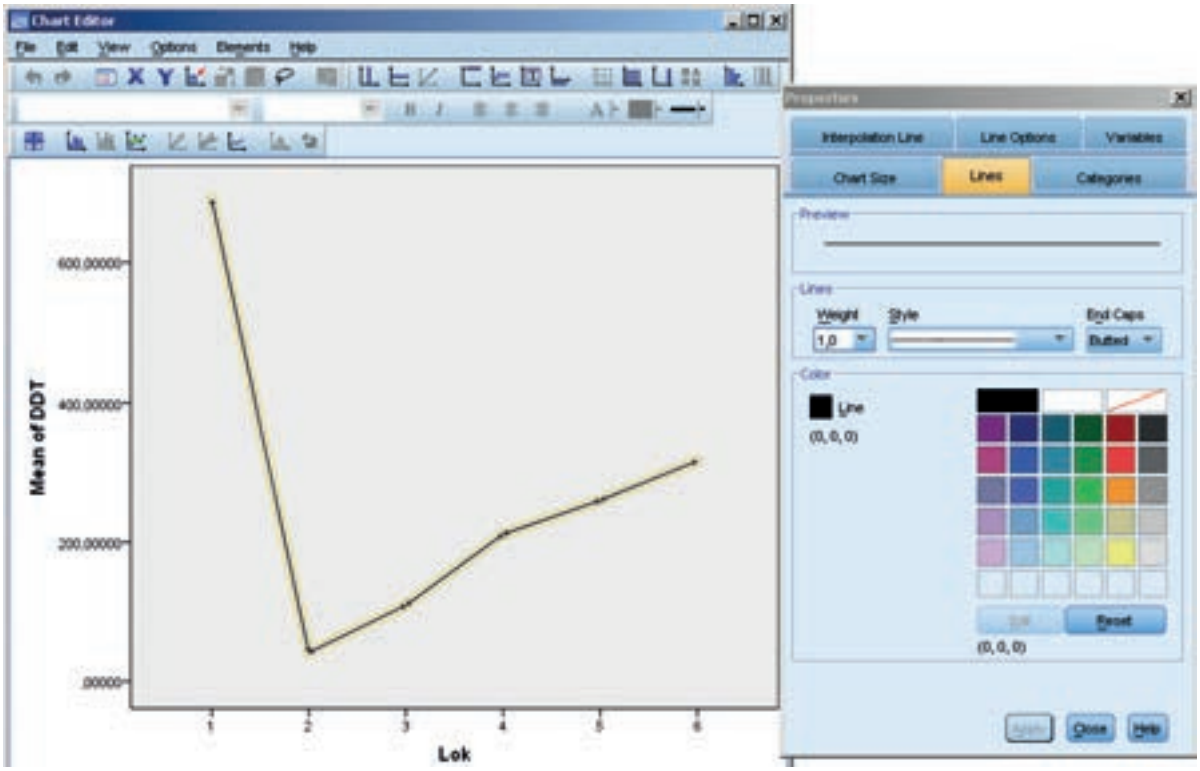
Z obrázků 8.11 je zřejmé, že všechny možnosti v menu lze ovládat pomocí myši, a tudíž není nutné pro běžné funkce používat syntaxi. Výstupy se ukládají do samostatného souboru (zde pojmenován Output 1), přičemž je zřejmá hierarchická struktura výstupů. Pokud chceme úlohu změnit, lze použít tlačítko pro vyvolání naposledy použitých funkcionalit (bod 5 na obrázku 8.11).

Další část úlohy provedeme obdobně. Jedna z metod pro porovnání více výběrů – lokalit – se nazývá ANOVA. Nachází se v menu (Analyze > Compare Means > One-Way ANOVA).

Nechť hodnocenou látkou je například DDT. Budeme ji hodnotit podle lokality (obrázek 8.12 vlevo). Jako podpůrné výstupy si nastavíme v nabídce Post Hoc Tukey test a v Options graf průměrů (obrázek 8.13).



Obr. 8.12 *Nastavení ANOVA.*

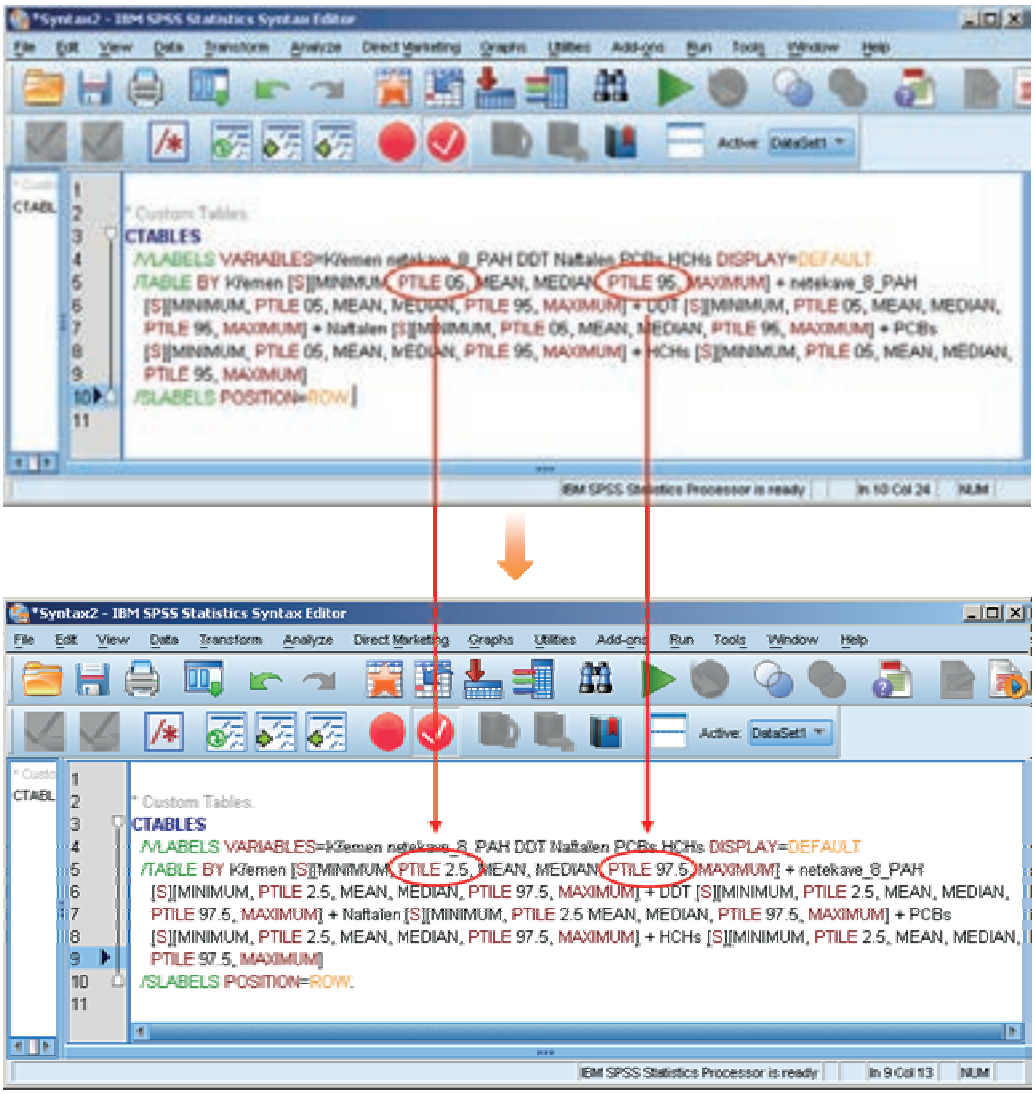


Obr. 8.13 *Graf v SPSS a jeho editační okno.*

8.2.4 Syntaxe

Sada všech zabudovaných funkcí v SPSS nabízí pouze omezenou (i když dostatečně širokou) škálu předdefinovaných nastavení pro analýzy. Pokud je však třeba spočítat určitou speciální hodnotu nějaké veličiny, nebo použít nestandardní nastavení, je třeba využít ***Syntax Editor.***

Jako příklad toho dobře poslouží výpočet percentilu jiného typu, než je přímo dostupný prostřednictvím menu Custom Tables. Nechť úkolem pro všechny měřené látky je vypočítat 2,5 % a 97,5 % percentil namísto předdefinovaných hladin percentilů. Základ pro úpravu získáme kliknutím na tlačítko Paste v menu Custom Tables (poznamenejme, že na obrázku 8.11 není toto tlačítko viditelné, protože je překryto tabulkou Summary statistics). Nevytvoří se výstup, nýbrž syntaxe v ***Syntax Editoru***, kterou můžeme snadno upravit (modifikovat) přepsáním jejich parametrů.



Obr. 8.14 *Syntax a příklad jeho úpravy.*

Podobně lze upravovat jakýkoliv parametr syntaxe. Z tohoto důvodu je výhodné vytvářet a ukládat během práce nejčastěji používané syntaxe a poté je pouze upravovat dle aktuálních potřeb analyzovaného souboru (např. měnit názvy proměnných).

Literatura

[1] Hřebíček, J., Žižka, J., Vědecké výpočty v biologii a biomedicíně, Masarykova universita, Brno [online]. (2007) [cit. 2011-10-31]. Dostupné na <http://portal.med.muni.cz/download.php?fid=502>

[2] Hřebíček, J., Pospíšil, Z., Urbánek, J. Úvod do matematického modelování s využitím Maple, Elportál, Masarykova univerzita, Brno, [online]. (2011) [cit. 2011-10-31]. Dostupné na <http://is.muni.cz/elportal/?id=930553>

[3] Horová I., Zelinka J., Numerické metody. 2. vyd., Masarykova univerzita, Brno (2004).

[4] Hřebíček, J., Kubásek, M., Environmentální informační systémy. CRM, Brno (2011)

[5] Wikipedia. Computational Science, [online]. (2011) [cit. 2011-10-31]. Dostupné na <http://en.wikipedia.org/wiki/Scientific_computing>

[6] Popper, K. R.: Logika vědeckého bádání. Oikoymenh. Praha (1997)

[7] Hřebíček, J., Kopeček, I., Kučera, J., Polcar, P. Fortran 77 a vědecké výpočty. Praha, Academia (1988)

[8] Peter, H. Python Scripting for Computational Science. Third edition. Heidelberg, Springer (2008)

[9] Press, W. H., Teukolsky, S. A., Vetterling, W. T. Numerical Recipes: The Art of Scientific Computing. Third edition. Cambridge, Cambridge University Press (2007)

[10] Wolfram, S. A. New Kind of Science. [online]. (2002) [cit. 2011-10-31]. Dostupné na <http://www.wolframscience.com/nksonline/toc.html>

[11] Gander, W., Hřebíček, J. Solving Scientific Problems Using Maple and MATLAB. 4th exp. and rev. ed. Springer, Heidelberg (2005)

[12] Golub, G., Ortega, J. M. Scientific Computing: An Introduction with Parallel Computing. New York, Academic Press (1993)

[13] Wang, C. Research Programs from BAA - Computing Sciences. 1.3 Information and Software Assurance (2010) [cit. 2011-10-31]. Dostupné na <http://www.arl.army.mil/www/default.cfm?page=518>

[14] Foster, I., Kesselman, C. eds. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann. San Francisco, California. (1999)

[15] University of Waterloo: Numerical Analysis for Engineering – Precision and Accuracy [online]. [cit 2011-10-29]. Dostupný z WWW: <https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/01Error/PrecisionAccuracy/>.

[16] Vuik, K.: Computer Arithmetic Tragedies [online]. [cit 2011-10-29]. Dostupný z WWW: <http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html>.

[17] Vitásek, E. Numerická matematika II: Numerické řešení diferenciálních rovnic SPN, Praha (1981)

[18] Komárek, A.: Bayesovské metody (druhá část) [online]. [cit 2011-09-01]. Dostupný z WWW: <http://www.karlin.mff.cuni.cz/~komarek/vyuka/2010_11/nstp021/ NSTP021-Bayes-Komarek-tisk.pdf>.

[19] Saltelli, A., Chan, K., Scott, E. M.: Sensitivity analysis. New York, Wiley (2000)

[20] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global sensitivity analysis. New York, Wiley (2008)

[21] Saltelli, A., Tarantola, S., Campolongo, F.: Sensitivity Analysis as an Ingredient of Modeling. Statistical Science (2000)

[22] Wellman, B.: "Doing It Ourselves: The SPSS Manual as Sociology's Most Influential Recent Book.", Pp. 71-78 in Required Reading: Sociology's Most Influential Books, edited by Dan Clawson. Amherst: University of Massachusetts Press (1998).

[23] Roustant, O.: Analytical computation of Sobol indices with a Gaussian process meta-model [online]. [cit 2011-09-09]. Dostupný z WWW: <http: //www.emse.fr/~roustant/Documents/Analytical%20computation%20of%20Sobol% 20indices_kriging%20mean.pdf>.
[24] Homma, T., Saltelli, A.: Importance measures in global sensitivity analysis of nonlinear models. Reliability Engineering and System Sa]ety (1996)
[25] Cheb-Terrab, E.S., and von Bulow, K. A Computational Approach for the Analytical Solving of Partial Differential Equations. Computer Physics Communications, Vol. 90, 102-116 (1995).
[26] Sunderam, V. S., PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice and Experience, Vol. 2, No. 4, pp 315-339, (1990)
[27] Brněnská přehrada [online]. [cit 2011-10-11]. Dostupný z WWW: <http://www. brnenskaprehrada.cz/p_cisla.html>.
[28] Shampine, L. F.: Some Current Work [online]. [cit 2011-10-18]. Dostupný z WWW: <http://faculty.smu.edu/shampine/current.html>
[29] Wikipedia: Trophic state index [online]. [cit 2011-10-24]. Dostupný z WWW: <http: //en.wikipedia.org/wiki/Trophic_state_index>
[30] Wikipedia: Dormand–Prince method [online]. [cit 2011-10-23]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Dormand-Prince>.
[31] Wikipedia: Metrologie [online]. [cit 2011-10-23]. Dostupný z WWW: <http://cs. wikipedia.org/wiki/Metrologie>.
[32] Wikipedia: Integer overflow [online]. [cit 2011-11-11]. Dostupný z WWW: <http: //en.wikipedia.org/wiki/Integer_overflow>.
[33] Wikipedia: Double precision floating-point format [online]. [cit 2011-11-11]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Double_precision_floating-point_ format>.
[34] Wikipedia: Arithmetic underflow [online]. [cit 2011-11-11]. Dostupný z WWW: <http: //en.wikipedia.org/wiki/Arithmetic_underflow>.
[35] Arora, S., Barak, B. Computational Complexity: A Modern Approach. Cambridge, Cambridge University Press. (2009) [online]. [cit. 2011-10-31]. Dostupné v rukopise na <http://***www.cs.duke.edu/~reif/courses/complectures/books/.../ABbook.pdf***>
[36] Maršálek, B., Müller, B.: Znečištění povrchových vod živinami: Příčiny, důsledky a možnosti řešení (eu)trofizace. Praha (2009)
[37] Budíková, M., Králová, M., Maroš, B. Průvodce základními statistickými metodami. Praha, Grada Publishing, a.s., (2010) 272 s. ISBN 978-80-247-3243-5.

Manuály a základy
http://www.its.qmul.ac.uk/training/manuals/SPSSIntro.pdf - úvod k programu SPSS
http://www.ssc.upenn.edu/scg/spss/verybasicSPSS.pdf
http://ittraining.iu.edu/ematerials/samples/SPSBAv8.0.0.TRUNC.pdf

Fóra a weby věnované SPSS
http://www-01.ibm.com/software/analytics/spss/ - stránky IBM, které vyvíjí SPSS
http://www.spssforum.com/ - volně přístupné fórum uživatelů SPSS
http://www.talkstats.com/forumdisplay.php/17-SPSS - další fórum

Příloha

Zdrojové kódy Maple a MATLABu pro výpočet hodnot vlivu jednotlivých parametrů X_1, X_2, X_3 a indexů globální citlivosti Soboľovou metodou v příkladu 5.5.

Výpočet indexů globální citlivosti pomocí systému Maple

```
# Vypocet hodnot vlivu V1, V2, V3 a citlivosti S1, S2, S3 parametru
       X[1], X[2], X[3] analyticky
restart:
with(Statistics):
X[1] := RandomVariable(Uniform(-Pi, Pi)):
X[2] := RandomVariable(Uniform(-Pi, Pi)):
X[3] := RandomVariable(Uniform(-Pi, Pi)):
Y : = sin(X[1])+7*sin(X[2])^2+(1/10)*X[3]^4*sin(X[1]):
E := Mean(Y): V := Variance(Y):

Mean(sin(x1)+7*sin(X[2])^2+(1/10)*X[3]^4*sin(x1));
V1 := Variance((1/50)*sin(X[1])*Pi^4+7/2+sin(X[1]));

Mean(sin(X[1])+7*sin(x2)^2+(1/10)*X[3]^4*sin(X[1]));
V2 := Variance(-7*cos(X[2])^2+7);

Mean(sin(X[1])+7*sin(X[2])^2+(1/10)*x3^4*sin(X[1]));
V3 := Variance(7/2);

S1 := evalf(V1/V, 4);
S2 := evalf(V2/V, 4);
S3 := evalf(V3/V, 4);

# Vypocet indexu citlivosti Soborovou metodou
restart:
M := 10^5:
with(Statistics):
X[1] := RandomVariable(Uniform(-Pi, Pi)):
X[2] := RandomVariable(Uniform(-Pi, Pi)):
X[3] := RandomVariable(Uniform(-Pi, Pi)):

X1s := Sample(X[1], M):  # M realizaci veliciny X[1]
X2s := Sample(X[2], M):  # M realizaci veliciny X[2]
X3s := Sample(X[3], M):  # M realizaci veliciny X[3]

Ys := sin~(X1s) + 7*(sin~(X2s))^~2 + ((1/10)*X3s^~4)*~(sin~(X1s)):

f[0] := Mean(Ys):      # odhad stredni hodnoty
V[0] := Variance(Ys):  # odhad rozptylu

X1s2 := Sample(X[1], M):  # "druha skupina" M realizaci veliciny X[1]
X2s2 := Sample(X[2], M):  # "druha skupina" M realizaci veliciny X[2]
X3s2 := Sample(X[3], M):  # "druha skupina" M realizaci veliciny X[3]
Ys2 := Vector(M):

# citlivost parametru X[1]
Ys2 := sin~(X1s) + 7*(sin~(X2s2))^~2 + ((1/10)*X3s2^~4)*~(sin~(X1s)):
V[1] := Mean(Ys*~Ys2)-f[0]^2;
S[1] = V[1]/V[0];
```

```
# citlivost parametru X[2]
Ys3 := sin~(X1s2) + 7*(sin~(X2s))^~2 + ((1/10)*X3s2^~4)*~(sin~(X1s2)):
V[2] := Mean(Ys*~Ys3)-f[0]^2;
S[2] = V[2]/V[0];

# citlivost parametru X[3]
Ys4 := sin~(X1s2) + 7*(sin~(X2s2))^~2 + ((1/10)*X3s^~4)*~(sin~(X1s2)):
V[3] := Mean(Ys*~Ys4)-f[0]^2;
S[3] = V[3]/V[0];

# korekcni clen
Ysk := sin~(X1s2) + 7*(sin~(X2s2))^~2 +
       ((1/10)*X3s2^~4)*~(sin~(X1s2)):
V[k] := Mean(Ys*~Ysk)-f[0]^2;
```

Výpočet indexů globální citlivosti pomocí systému MATLAB

```
M=100000;

% M realizaci jednotlivych velicin
X1s = unifrnd(-pi, pi,[M 1]);
X2s = unifrnd(-pi, pi,[M 1]);
X3s = unifrnd(-pi, pi,[M 1]);

Ys = sin(X1s) + 7*sin(X2s).^2 + (1/10)*X3s.^4.*sin(X1s);

mean_Ys = mean(Ys);  % odhad stredni hodnoty
var_Ys = var(Ys);    % odhad rozptylu

% "druha skupina" M realizaci jednotlivych velicin
X1s2 = unifrnd(-pi, pi,[M 1]);
X2s2 = unifrnd(-pi, pi,[M 1]);
X3s2 = unifrnd(-pi, pi,[M 1]);

% "druha reseni" pro jednotlive vlivy
Ys2 = sin(X1s) + 7*sin(X2s2).^2 + (1/10)*X3s2.^4.*sin(X1s);
Ys3 = sin(X1s2) + 7*sin(X2s).^2 + (1/10)*X3s2.^4.*sin(X1s2);
Ys4 = sin(X1s2) + 7*sin(X2s2).^2 + (1/10)*X3s.^4.*sin(X1s2);
Ysk = sin(X1s2) + 7*sin(X2s2).^2 + (1/10)*X3s2.^4.*sin(X1s2);

% citlivost parametru X[1]
V1 = mean(Ys.*Ys2) - mean_Ys^2;
S1 = V1/var_Ys

% citlivost parametru X[2]
V2 = mean(Ys.*Ys3) - mean_Ys^2;
S2 = V2/var_Ys

% citlivost  parametru X[3]
V3 = mean(Ys.*Ys4) - mean_Ys^2;
S3 = V3/var_Ys

% korekcni clen
Vk = mean(Ys.*Ysk) - mean_Ys^2;
S3 = Vk/var_Ys
```

Níže jsou uvedeny zdrojové kódy Maple i MATLABu pro výpočet řešení modelů v případové studii v kapitole 8.1. Kód řešení rovnice 8.3 přímo navazuje na kód řešení rovnice 8.1, který je nutné spustit přednostně.

Řešení modelové rovnice (8.1) pomocí systému Maple

```
# nacteni potrebnych baliku
with(ExcelTools):
with(Statistics):
with(plots):
with(CurveFitting):
with(LinearAlgebra):

# nacteni dat ze souboru
Data := Import("D:/data_prehrada.xlsx", "data", "D74:L7993"):
Data_matice := Matrix(Data):
Objem := Column(Data_matice, 3):
Pritok := Column(Data_matice, 1):
Odtok := Column(Data_matice, 2):
Fosfor_pritok := Column(Data_matice, 6)[1 .. 12]:
Fosfor_odtok := Column(Data_matice, 7)[1 .. 12]:
Dusik_pritok := Column(Data_matice, 8)[1 .. 12]:
Dusik_odtok := Column(Data_matice, 9)[1 .. 12]:

# casy mereni koncentrace polutantu (v hodinach ode dne 4.1.)

casy := <0, 673, 1345, 2521, 2857, 3529, 4057, 5041, 5713, 6553, 7201,
       7920>:

# aproximace koncentrace na pritoku pomoci linearnich splajnu

Cin(t):=Spline([seq([casy[i],Dusik_pritok[i]],i= 1..12)],t,degree=1):

# interpolacni uzly, aproximace objemu
objem_indexy := [1, 690, 1200, 1500, 1650, 1800, 2100, 2200, 2260,
       2400, 2650, 2850, 3100, 3450, 3800, 4300, 4800, 5250, 5500,
       5800, 6500, 6800, 7050, 7300, 7400, 7600, 7920]:
objem_body := seq([objem_indexy[i], Objem[objem_indexy[i]]],
       i = 1 .. nops(objem_indexy)):
V(t):=Spline([objem_body],t,degree=3):

# interpolacni uzly, aproximace pritoku
pritok_indexy := [1, 105, 215, 300, 450, 510, 600, 1000, 1120, 1160,
       1500, 1600, seq(2000+10*j , j = 0 .. 10), 2300, 2400,
       seq(2500+50*j, j=0 .. 10), 3400, 3500, 4000, 4300, 4600, 4800,
       5000 , 5100 , 5170, 5210, 5275, 5350, 5500, 5800, 6000, 6400,
       6800, 7000, 7200, 7500, 7800, 7920]:
pritok_body := seq([pritok_indexy[i], Pritok[pritok_indexy[i]]],
       i = 1 .. nops(pritok_indexy)), [1800, 10], [1900, 16]:
Qin(t):=Spline([pritok_body],t,degree=3):

# dopocitani odtoku
Q(t):=Qin(t)-(diff(V(t),t))/3600:

# vypocet reseni modelu
rovnice := diff ((1/3600)*C(t)*V(t), t)=Qin(t)*Cin(t)-Q(t)*C(t):
poc_podm := C(0) = Dusik_odtok[1]:
res := dsolve({rovnice, poc_podm}, C(t), numeric);

# vykresleni reseni modelu spolu s namerenymi hodnotami
display({odeplot(res, [t, C(t)], t = 0 .. 7920, color = black, labels
       = ["t [datum]",
       "C [mg/l]"], tickmarks = [[1 = "4.1.", 1345 = "1.3.", 2857
```

```
            = ”3.5.”, 4057 = ”22.6.”, 5713 = ”30.8.”, 7920 = ”30.11.”], de-
            fault]),
    pointplot([seq([casy[i], Dusik_odtok[i]], i = 1 .. 12)])});
```

Řešení modelové rovnice (8.1) v MATLABu

hlavní skriptový soubor

```
% Nacteni dat ze souboru
Data = xlsread('D:/data_prehrada.xlsx', 'data', 'D74:L7993');
% Prirazeni hodnot ve sloupcich k jednotlivym parametrum
Pritok = Data(:,1);
Odtok = Data(:,2);
Objem = Data(:,3);
Fosfor_pritok = Data(1:12,6);
Fosfor_odtok  = Data(1:12,7);
Dusik_pritok  = Data(1:12,8);
Dusik_odtok   = Data(1:12,9);

% numericka derivace objemu
der_Objem = diff(Objem);

% casy mereni koncentrace polutantu (v hodinach ode dne 4.1.)
casy = [0, 673, 1345, 2521, 2857, 3529, 4057, 5041, 5713, 6553, 7201, 7920];

% vypocet reseni modelu pro koncentraci dusiku
[T, C] = ode45(@(t,C_odtok) ode_model(t,C_odtok,Objem,der_Objem, ...
               Pritok,Dusik_pritok,casy),[0 7920],Dusik_odtok(1));

% vykresleni reseni modelu spolu s namerenymi hodnotami
plot(casy,Dusik_odtok,'o',T, C);
title('Koncentrace dusiku na odtoku v roce 2006 - model i merene hodnoty')
xlabel('t [datum]')
ylabel('C [mg/l]')
set(gca,'XTick',casy([1:4,6,8:10,12]))
set(gca,'XTickLabel',{'4.1.';'1.2';'1.3.';'19.4.';'31.5.';...
                      '2.8.';'30.8';'4.10.';'30.11.'})
```

soubor `ode_model.m`

```
function dC = ode_model(t,C,V,dV,Qin,Cin,casy)

% interpolacni uzly pro aproximaci objemu
objem_indexy = [1, 690, 1200, 1500, 1650, 1800, 2100, 2200, 2260, ...
             2400, 2650, 2850, 3100, 3450, 3800, 4300, 4800, 5250, ...
             5500, 5800, 6500, 6800, 7050, 7300, 7400, 7600, 7919];

% aproximace objemu pomoci kubickych splajnu
V = interp1(objem_indexy,V(objem_indexy),t,'spline');

% aproximace derivace objemu pomoci kubickych splajnu
dV = interp1(objem_indexy,dV(objem_indexy),t,'spline');

% interpolacni uzly pro aproximaci pritoku
pritok_indexy = [1, 105, 215, 300, 450, 510, 600, 1000, 1120, 1160, ...
              1500, 1600, 2000:10:2100, 2300, 2400, 2500:50:3000, ...
              3400, 3500, 4000, 4300, 4600, 4800, 5000, 5100, 5170, ...
              5210, 5275, 5350, 5500, 5800, 6000, 6400, 6800, 7000, ...
              7200, 7500, 7800, 7920];

% aproximace pritoku pomoci kubickych splajnu
```

```
Qin = interp1(horzcat(pritok_indexy,[1800,1900]), ...
              horzcat(Qin(pritok_indexy).',[10,16]),t,'spline');

% "dopocitani" odtoku, aby odpovidal zmenam objemu
Q = Qin - dV/3600;

% aproximace koncentrace na pritoku pomoci linearnich splajnu
Cin = interp1(casy,Cin,t);

% rovnice modelu (vyjadreno pro derivaci koncentrace)
dC = (3600./V).*(Qin.*Cin - Q.*C) - C.*(dV./V);
```

Řešení modelové rovnice (8.3) pomocí systému Maple

```
# aproximace obsahu průřezu nadrzi
A(t):=(V(t))/(9300):

# vykresleni reseni modelu spolu s namerenymi hodnotami
PDE := diff((1/3600)*C(x,t)*A(t), t)+Q(t) * (diff(C(x, t), x)) = 0:
IBC := {C(0, t) = Cin(t),
        C(x, 0) = ((Dusik_odtok[1]–Dusik_pritok[1])/9300)*x + Du-
sik_pritok[1]}:
pds := pdsolve(PDE, IBC, numeric, time = t, range = 0 .. 9300);

# vykresleni reseni modelu spolu s namerenymi hodnotami
display ({pointplot([seq([casy[i], Dusik_odtok[i]], i = 1 .. 12)]),
          pds[plot](t=0..7920, x=9300, numpoints=3000, color=black, labels=
                    [”t [datum]”, ”C [mg/l]” ], tickmarks=[[1=”4.1.”,
1345=”1.3.”,
                   2857=”3.5.”, 4057=”22.6.”, 5713=”30.8.”, 7920=”30.11.”],
default])});
```

Řešení modelové rovnice (8.3) v MATLABu

```
function studie1pde

global Pritok Odtok Objem Dusik_pritok Dusik_odtok At dA casy

% Nacteni dat ze souboru
Data = xlsread('D:/data_prehrada.xlsx', 'data', 'D74:L7993');

% Prirazeni hodnot ve sloupcich k jednotlivym parametrum
Pritok = Data(:,1);
Odtok = Data(:,2);
Objem = Data(:,3);
dA = Data(:,4);
Dusik_pritok = Data(1:12,8);
Dusik_odtok  = Data(1:12,9);

% plocha kolmeho prurezu nadrzi
At = Objem/9300;

% casy mereni koncentrace polutantu (v hodinach ode dne 4.1.)
casy = [0, 673, 1345, 2521, 2857, 3529, 4057, 5041, 5713, 6553, 7201, 7920];

% vyber metody vypoctu z "vyskakujiciho" menu
meth = menu('Zvolte metodu vypoctu','LxF','LxW','SLxW');
switch meth
    case 1
        method = 'LxF';
    case 2
        method = 'LxW';
```

```
    case 3
        method = 'SLxW';
end

t = 0;  Nx = 94;
x = linspace(0,9300,Nx); % body prostoru, v nichz bude pocitano reseni

% pocatecni podminka
u = Dusik_pritok(1)+x*(Dusik_odtok(1)-Dusik_pritok(1))/9300;

% nastaveni parametru pro prikaz "hpde"
sol = setup(1,@flux,t,x,u,method,[],@bcs);

dt = 1;  % casovy krok vypoctu metody
tout = [0:100:7900,7920]; % casove body, v nichz si uchovame reseni
Nt = length(tout);

% matice reseni
reseni = vertcat(u,zeros(Nt-1,Nx));

% cyklus vyhodnoceni modelu pro casove body spec. v prom. "tout"
for m = 2:Nt
    howfar = tout(m) - tout(m-1);
    sol = hpde(sol,howfar,dt);
    reseni(m,:) = sol.u(1,:);
end

% vykresleni reseni modelu spolu s namerenymi hodnotami
figure
plot(casy,Dusik_odtok,'o',tout,reseni(:,Nx))
title('Koncentrace dusiku na odtoku v roce 2006 - model i merene hodnoty')
xlabel('t [datum]')
ylabel('C [mg/l]')
set(gca,'XTick',casy([1:4,6,8:10,12]))
set(gca,'XTickLabel',{'4.1.';'1.2';'1.3.';'19.4.';'31.5.';...
                      '2.8.';'30.8';'4.10.';'30.11.'})

%=========================================================================

% Subfunkce

function F = flux(t,x,u,u_x)
      global Odtok At dA
         tt = int8(t)+1; % prevod "t" na celociselne (prirozene) "tt"
         % modelova rovnice vyjadrena pro du/dt
         F = -(dA(tt)/At(tt))*u - (3600*Odtok(tt)/At(tt))*u_x;


   % okrajova podminka
     function [uL,uR] = bcs(t,uL,uR)
      global Dusik_pritok casy

      % aproximace koncentrace na pritoku pomoci linearnich splajnu
        uL = interp1(casy,Dusik_pritok,t);
```

Obsah

Summary

The publication of “Scientific computing in mathematical biology” was founded in solving the ESF project No. CZ.1.07/2.2.00/07.0318 “MULTIDISCIPLINARY INNOVATION OF STUDY OF MATHEMATICAL BIOLOGY”, which aims to upgrade the content and consistency of objects from the root set of objects of study “Mathematical Biology” study program of “Biology”, Faculty of Science, Masaryk University (MU), profiling study field undertaken by the staff of the Institute of Biostatistics and Analyses MU.

It is oriented to current concepts of computer science and scientific computing, their history, used information technology, problems of computing environment and application software Maple, MATLAB and SPSS including their use so that it can be used by teachers and students of other fields of study program "Biology".

The aim of computational science or scientific computing is solving real-world problems using information and communication technologies (ICTs), or computing architecture (software, hardware and communications). Use of ICT is necessary because current scientific problems are often of theoretical or experimental solution too complex or time consuming. Because the current development of computational science and scientific computing is advancing very rapidly forward, created publication help improve teaching one of ordinary objects field "Mathematical Biology" with new and updated information. It will help readers in understanding the terminology needed for scientific computing, and about where to find and use sources of information on scientific computing, algorithms and their applications. This will serve all the chapters of this publication.

After an explanatory introduction, this publication gives in its second chapter the concept of computer science and related terminology in the field of scientific computing. It clarifies what is scientific computing or computational science, as currently conceived, and also mentioned some of their applications in biology and biomedicine.

The third chapter is devoted to the history of scientific computing since its beginning in the forties of last century to the present developments, including those in the Czech Republic and at MU. Today's computing development environment (algorithms, methods and ICT) offers new possibilities for computational science and scientific computing, but also some limitations. This issue is devoted to the fourth chapter with grid and cloud computing, the use of mobile platforms, and analysis of errors in calculations and application of numerical methods.

The fifth chapter is devoted to the impact of uncertainty in models, data and computer processing. There is also discussed the issue of sensitivity, particularly the global sensitivity of dealing with the applications on their parameters in this chapter. The sixth chapter briefly describes the most used application software in scientific computing (Maple, MATLAB, SPSS) in "Mathematical Biology" and in the seventh chapter is discussed the possibilities of its effective use in parallel computing.

Currently, there are many possible algorithms, and free (or open source) software / freeware that with the use of available licensed software allow MU students and teachers to focus on their own solutions of biological or biomedical problems without wasting time with programming algorithms for these problems. Therefore, in the eighth chapter are described two case studies in the use of application software Maple, MATLAB and SPSS.

This publication will help readers in understanding the terminology needed for scientific computing, and about where to find and use sources of information on scientific computing, algorithms and their applications. This will serve all the chapters of this publication.

Vědecké výpočty
prof. RNDr. Jiří Hřebíček, CSc., RNDr. Miroslav Kubásek, Ph.D., Mgr. Lukáš Kohút,
prof. RNDr. Luděk Matyska, CS, Mgr. Lucia Tokárová, Mgr. Jaroslav Urbánek

Recenzenti: doc. RNDr. Stanislav Bartoň, Ph.D., doc. RNDr. Tomáš Pitner, Ph.D.
Jazyková redakce: Bc. Kateřina Chvátalová
Obálka: Radim Šustr, DiS.
Vydalo: AKADEMICKÉ NAKLADATELSTVÍ CERM, s. r. o. Brno
Purkyňova 95a, 612 00 Brno
www.cerm.cz
Tisk: FINAL TISK s. r. o. Olomučany
Náklad: 200 ks
Vydání: první
Vyšlo v roce 2012

ISBN 978-80-7204-781-9