Matematické algoritmy

Skriptum k předmětu 11MAG

Jan Přikryl, Petr Přikryl

27. září 2014

Ústav aplikované matematiky
ČVUT v Praze, Fakulta dopravní

Obsah

Obsah **1**

Seznam obrázků **3**

1 Úvod **5**

Literatura **6**

2 Algoritmy a algoritmizace **7**
2.1 Algoritmy a algoritmizace 7
2.2 Aplikace 9
2.3 Prerekvizity předmětu 11

Literatura **12**

3 Analýza algoritmů a jejich složitost **13**
3.1 Úvod 13
3.2 Algoritmus 15
3.3 Porovnání variant 23
3.4 NP-úplné problémy 23

Literatura **25**

4 Prvočísla a dělitelnost. **27**
4.1 Prvočísla 27
4.2 Nejvyšší společný dělitel 36
4.3 Dělitelnost 37
4.4 Modulární aritmetika 39
4.5 Malá Fermatova věta 44
4.6 Příklady 45

Literatura **48**

5 Šifrování veřejným klíčem **49**
5.1 Čínská věta o zbytcích 49
5.2 Mocnění 51

5.3 Eulerova funkce . . . 53
5.4 Šifrování . . . 54
5.5 Příklady . . . 61

Literatura **62**

6 Opakování teorie grafů **63**
6.1 Úvod do teorie grafů . . . 63
6.2 Implementace grafů . . . 68
6.3 Základní grafové úlohy . . . 70

Literatura **77**

7 Jemný úvod do numerických metod **79**
7.1 Úvod . . . 79
7.2 Zobrazení čísel . . . 83
7.3 Chyby . . . 85
7.4 Typy úloh . . . 89

Literatura **90**

8 Kořeny nelineárních funkcí **91**
8.1 Nelineární rovnice . . . 91
8.2 Iterační metody . . . 98
8.3 Dodatky . . . 109

Literatura **111**

9 Numerická integrace **113**
9.1 Numerická integrace . . . 113
9.2 Newtonovy-Cotesovy vzorce . . . 119
9.3 Gaussovy kvadraturní vzorce . . . 124
9.4 Složené kvadraturní vzorce . . . 129
9.5 Dodatky . . . 130

Literatura **133**

10 Řešení soustav lineárních rovnic **135**
10.1 Soustavy . . . 135
10.2 Numerické metody . . . 145
10.3 Dodatky . . . 160

Literatura **162**

11 Aproximace a interpolace **163**
11.1 Úlohy . . . 163

11.2 Interpolace . . . 168
11.3 Dodatky . . . 180

Literatura **183**

12 Náhodná čísla a Monte Carlo **185**
12.1 Historie . . . 185
12.2 Stochastická simulace . . . 187
12.3 Generátory náhodných čísel . . . 189
12.4 Monte Carlo integrace . . . 194
12.5 Kvazináhodná čísla . . . 197

Literatura **198**

Rejstřík **201**

Seznam obrázků

2.1 Formální pohled na algoritmus ze systémového hlediska . . . 7
2.2 Fraktály (vlevo tzv. Juliina množina, vpravo). . . . 9
2.3 Metoda konečných prvků. . . . 10
2.4 I tento jev, způsobený prouděním okolo křídla letounu, lze popsat Navierovými-Stoeksovými rovnicemi. . . . 10
2.5 Rychlá Fourierova transformace – analýza EEG signálu . . . 12

6.1 Graf popisující možné cesty MHD mezi jednotlivými budovami ČVUT FD: **F** označuje budovu Na Florenci, **H** Horskou a **K** Konvikt. Ohodnocení hran je nejrychlejší cesta v kombinaci *pěšky* a *MHD* v minutách. . . . 64
6.2 Oba grafy na obrázku popisují to samé – jsou izomorfní. . . . 66
6.3 Kružnice a trojúhelník . . . 67
6.4 Úplný graf, jenž vznikl doplněním hrad do kružnice na Obrázku 6.3 67
6.5 Sedm mostů v Královci. Upravená dobová rytina. Převzato z Wikipedia Commons [?] (vlevo). Odpovídající graf (vpravo) . . . 70

7.1 Pozice numerické úlohy a numerických metod v kontextu matematického modelování. . . . 81
7.2 Typy chyb v matematickém modelování . . . 86
7.3 Taxonomie matematických úloh . . . 90

8.1 Příklad nelineární funkce s kořeny x_1 a x_2 na intervalu $[a, b]$. Tento interval je pro danou funkci $f(x)$ uzávěrou kořene. 93
8.2 Podmíněnost kořenů nelineární rovnice $f(x) = 0$. Vlevo: dobře podmíněná úloha, vpravo: špatně podmíněná úloha. 95
8.3 Pevný bod $(2, 2)$ nelineárních funkcí. 102
8.4 Metoda postupných aproximací pro první a druhou funkci g. . . . 102
8.5 Metoda postupných aproximací pro třetí a čtvrtou funkci g. 103
8.6 Newtonova metoda pro řešení nelineární rovnice. 105
8.7 Metoda sečen pro řešení nelineární rovnice. 107

9.1 Integrace funkce $f(x) = \mathrm{e}^{-x^2}$ Newtonovými-Cotesovými kvadraturními pravidly. 121
9.2 Kancelace chyb u obdélníkového (vlevo) a Simpsonova (vpravo) pravidla. 123
9.3 Typické rozložení uzlů u adaptivní kvadratury. 132

10.1 Dobře podmíněná (vlevo) a špatně podmíněná (vpravo) soustava dvou lineárních rovnic o dvou neznámých. 143

11.1 Původní funkce $f(x) = \sqrt{x}$ a její nejlepší Čebyševova a L_2-aproximace 166
11.2 Průběh polynomu ω_3 pro uzly $x_i = i$. 179
11.3 Interpolace Rungovy funkce - ekvidistantní uzly. 181
11.4 Interpolace Rungovy funkce – Čebyševovy uzly. 182

12.1 100 002 hodnot vygenerovaných generátorem pseudonáhodných čísel RANDU a použitých po trojicích jako souřadnice bodů zobrazeno jako trojrozměrný graf. Převzato z Wikipedie 192
12.2 441 hodnot vygenerovaných jako rovnoměrná mřížka 21×21 bodů, psudonáhodným generátore a Haltonovou kvazináhodnou sekvencí 198

KAPITOLA 1

Úvod

Toto skriptum si klade za cíl podat studentům ČVUT FD přehled různých tříd užitečných a v praxi důležitých matematických algoritmů (tedy metod pro řešení různých tříd matemtickcých problémů na počítači). Budeme studovat mnoho různých aplikačních oblastí a vzhledem k omezenému rozsahu předmětu 11MA se vždy zaměříme pouze na ty zcela základní, fundamentální algoritmy, o nichž by studenti magisterského studia na technické vysoké škole měli mít povědomí. Vzhledem k šíři záběru předmětu nebudeme mít prostor k tomu, abychom se zabývali velkým množstvím algoritmů a jejich detailními charakteristikami – tento úkol s radostí můžeme přenechat studentům výpočetní techniky a informatiky. Budeme se ale snažit nad vybranými algoritmy strávit dostatek času na to, aby čtenáři pochopili základní principy dané metody a její případná specifika.

Naučit se a dobře pochopit činnost studovaného algoritmiu lze nejlépe při pokusech o jeho implementaci. Tento postupu při studiu látky, obsažené v tomto skriptu, důrazně doporučujeme. Není naším cílem z čtenáře vychovat špičkového programátora, k pochopení vykládané látky je ale vhodné algoritmy implementovat a na reálnách problémech zkoušet jejich různé varianty. Ve skriptu se nechceme vázat na jeden konkrétní programovací jazyk, algoritmy zapisujeme proto v pseoudokódu, jehož dikci lze snadno převést do všech běžných vyšších programovacích jazyků.

Předpokládáme, že čtenáři tohoto skripta mají za sebou základní kursy programování v rozahu, vyučovaném v bakalářské části studia na ČVUT FD – zejména budeme předpokládat, že znají základy programování ve vyšších programovacích jazycích jako je Basic, Delphi, Java, Pascal, Python či Matlab. Budeme také předpokládat jistou znalost elementárních algoritmů a základních datových struktur jako je pole a seznam, případně zásobník, fronta či strom. Některá témata, jež budeme studovat, předpokládají znalost základ-

ního matematického kalkulu, lineární algebry a teorie grafů, vše opět v rozsahu, v jakém se studují v bakalářské části studia na ČVUT FD.

O čem si budeme povídat: algoritmy diskrétní matematiky, slasti a strasti výpočtů v plovoucí řádové čárce, numerická matematika.

O čem budou cvičení: praktické hrátky s algoritmy, Matlab/Python/C/C++/Java ...

Co když neumím programovat? To, že jste postoupili až do prvního magisterského ročníku garantuje, že programovat umíte. V případě nouze se urychleně doučíte. Učebnic základů algoritmizace a programování existuje celá řada, můžete zkusit třeba [4, 1] či informace na stránkách Katedry počítačů ČVUT FEL [2], například ty o předmětu Algoritmizace [3].

Podle čeho se to učí: Literatura je téměř výhradně anglicky, kompletní seznam i s případnými odkazy naleznete na webových stránkách předmětu a také na konci tohoto textu.

Literatura

[1] Bohuslav Hudec. *Programovací techniky.* Skripta. Česká technika – nakladatelství ČVUT, 2001.

[2] Katedra počítačů ČVUT FEL.

[3] Jiří Zdeněk Miroslav Balík, Božena Mannová. Algoritmizace (x36alg) – podklady k přednáškám.

[4] Miroslav Virius. *Základy algoritmizace.* Skripta. Česká technika – nakladatelství ČVUT, 2008.

KAPITOLA 2

Algoritmy a algoritmizace

xxxx

2.1 Algoritmy a algoritmizace

Algoritmus je

- přesný návod či postup, kterým lze vyřešit daný typ úlohy.
- efektivní postup pro výpočet hodnoty nějaké funkce vyjádřený konečným počtem instrukcí.

Definice 2.1. *Algoritmem rozumíme postup, podle kterého se z dat vstupních* $x[n]$ *vygenerují data výstupní* $y[n]$.

- Typy algoritmů
- Co potřebujete znát ?
- Kam až můžeme dojít ?

Obrázek 2.1: Formální pohled na algoritmus ze systémového hlediska

Vlastnosti algoritmů

Každý algoritmus musí mít následující vlastnosti:

1. **Konečnost:** výpočet se ukončí v „rozumně“ konečném čase.
2. **Hromadnost:** není sestrojen pouze na jediné $u[n]$, ale na celou řadu možných vstupů.
3. **Jednoznačnost:** přechod do následujícího stavu algoritmu je jednoznačně určen výsledkem stavu předchozího.

Komentář k vlastnostem algoritmů

1. **Konečnost:** předpověď počasí na zítra dosažená výpočtem o den později nemá význam.
2. **Hromadnost:** program pro výpočet odmocniny pracuje nad množinou čísel, není konstruován pro každé číslo zvlášť.
3. **Jednoznačnost:** každý algoritmus je složen z kroků, které na sebe vzájemně navazují. Každý krok je charakterizován jako přechod z jednoho stavu do jiného. Každý stav algoritmu je určen zpracovávanými daty a na tom, jak data v jednotlivých stavech vypadají. Je tedy pevně určeno, který krok bude následovat.

Příklad 2.1. *Numerický výpočet odmocniny Další variantou je numerický výpočet druhé odmocniny čísla x pomocí nelineární diferenční rovnice*

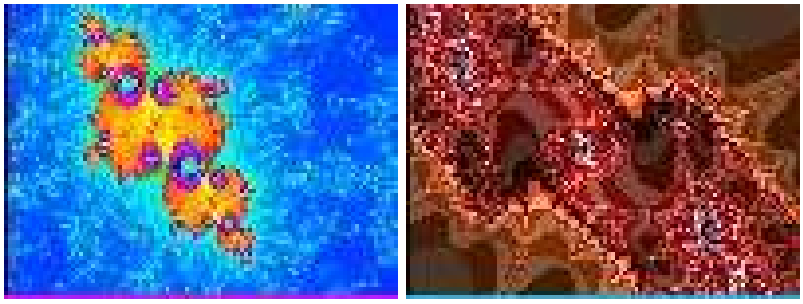
$$y[n+1] = \frac{1}{2}\left(y[n] + \frac{u[n]}{y[n]}\right),$$

kde vstupní posloupnost $u[n] = x \cdot \mathbf{1}[n]$ a výstupní posloupnost $y[n]$ postupně konverguje k hodnotě odmocniny. Počáteční podmínku můžeme volit v podstatě libovolnou, například $y[0] = x$ nebo $y[0] = 1$.

Odmocnina z čísla 10 je s přesností na 10 desetinných míst rovna $\sqrt{10} = 3{,}16227766017$.

Pro $u[n] = 10$ dostáváme postupně

$$\begin{array}{ll} y[0] = 3 & y[0]^2 = 9 \\ y[1] = 3{,}165 & y[1]^2 = 10{,}017225 \\ y[2] = 3{,}162278 & y[2]^2 = 10{,}0000021493 \\ y[3] = 3{,}1622776601 & y[3]^2 = 9{,}9999999996 \\ \vdots & \end{array}$$



Obrázek 2.2: Fraktály (vlevo tzv. Juliina množina, vpravo).

2.2 Aplikace algoritmů

Internet umožňuje lidem na celém světě rychle vyhledávat a přistupovat k obrovskému množství informací. Aby to fungovalo, musí poskytovatelé internetu a poskytovatelé internetových služeb používat chytré algoritmy, umožňující zpracovávat a spravovat tak velké množství dat. Příklady úloh, na které v této oblasti narazíme, jsou například *hledání vhodných cest* pro datové pakety, cestující mezi jednotlivými uzly sítě, či *vyhledávání* stránek s určitým obsahem.

Velký objem obchodů je v dnešní době uzavírán elektronicky, a mnoho služeb funguje i na elektronické bázi. Pro celý obor e-komerce je zcela klíčová schopnost uchovat důvěrné údaje (čísla kreditních karet, hesla, či bankovní informace) opravdu v tajnosti. Mezi úlohy z této oblasti, na které narazíme, patří *šifrování veřejným klíčem* či *digitální podpis*.

Ve oblastech výroby a přepravy řeší firmy často problém optimální alokace zdrojů tak, aby byly na jednu stranu minimalizovány výrobní či režijní náklady, na druhou stranu aby byl co možná nejvyšší užitek. Letecký dopravce se bude snažit přiřazovat posádky na jednotlivé lety způsobem, jenž generuje co nejmenší dodatečné náklady – zároveň je jeho manévrovací prostor ovšem omezen nařízeními z oblasti bezpečnosti provozu, všeobecnými právními předpisy a podobně. Poskytovatel internetu při investicích do infrastruktury potřebuje investovat své zdroje cíleně tak, aby výsledek co nejefektivněji sloužil zákazníkům. Obě tyto úlohy řeší algoritmy matematické optimalizační techniky zvané *lineární programování*.

Proč to zkoumat?

Numerické řešení algebraických rovnic, diferenciálních rovnic a speciálních funkcí:

Metoda konečných prvků – řešení složitých parciálních diferenciálních rovnic

Obrázek 2.3: Metoda konečných prvků.

Obrázek 2.4: I tento jev, způsobený prouděním okolo křídla letounu, lze popsat Navierovými-Stoeksovými rovnicemi.

s praktickými aplikacemi:

Jinak těžko řešitelné úlohy: nelineární parciální diferenciální rovnice, například Navierovy-Stokesovy rovnice. Tyto rovnice popisují proudění a počítáme je například při studiu obtékání vzduchu okolo křídel, viz. Obrázek 2.4.

Navierovy-Stokesovy rovnice:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\nabla)\mathbf{u} = \mathbf{f} - \nabla p + \nu \triangle \mathbf{u}$$

kde $\mathbf{u}$ a $\mathbf{f}$ jsou vektorové funkce rychlosti a síly, p je tlak a ν je úměrná viskozitě kapaliny.

$$\frac{\partial u_x}{\partial t} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_y}{\partial y} + u_z \frac{\partial u_z}{\partial z} =$$

$$= f_x(x, y, z, t) - \frac{\partial p}{\partial x} + \nu \left[\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \right]$$

Výše uvedené seznamy nejsou zdaleka vyčerpávající, můžeme z nich ale odvodit dvě základní charakterisitky, společné mnoha zajímavým algoritmům:

1. Pro daný problém existuje většinou velké množství možných řešení, z nichž většina z různých důvodů není to, co potřebujeme.
2. Algoritmy mají praktický užitek. Metoda hledání nejkratší cesty v grafu umožní přepravní společnosti snížit přepravní náklady, neboť sníží náklady na palivo a na práci personálu. Ten samý algoritmus může ve směrovači počítačové sítě hledat způsob, jak co nejrychleji doručit vaši zprávu adresátovi.

Algoritmy jsou technologie

Výpočetní čas je omezený zdroj, stejně tak, jak je omezená velikost operační paměti počítače. Oba tyto zdroje je třeba využívat rozumně, – algoritmy, jež jsou efektivní z hlediska doby výpočtu či nároků na operační paměť, jsou nástrojem k takovému rozumnému využití.

Algoritmy, stejně jako počítačový hardware, lze v dnešní době považovat za technologii. Celkový výkon systémů závisí na volbě efektivních algoritmů do stejné míry, jako závisí na volbě dostatečně výkonného hardware. A s tím, jak se vyvíjí počítačové technologie, se ruku v ruce vyvíjí i algoritmy.

2.3 Prerekvizity předmětu

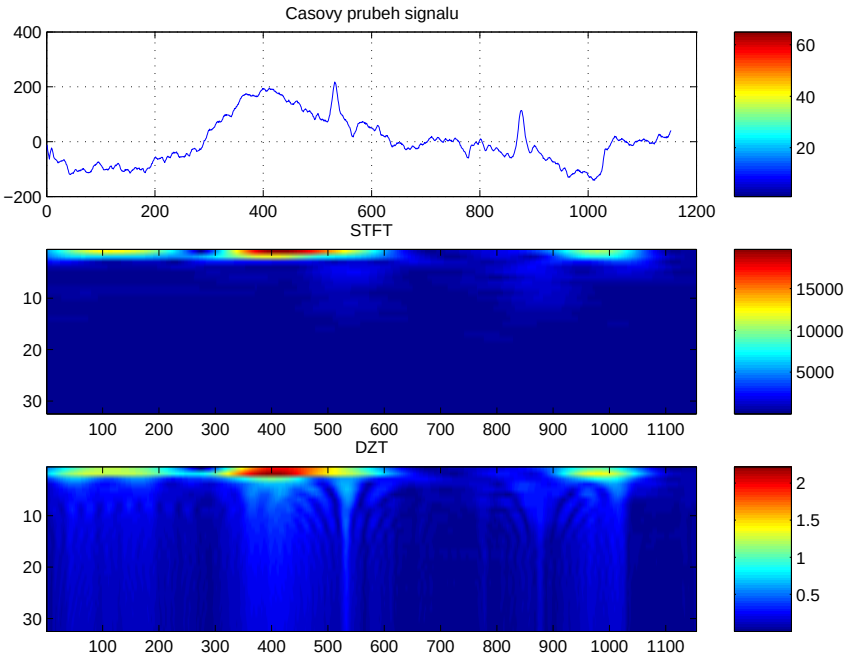
Předpokládáme:

- základy algebry a matematické analýzy
- základy numerické matematiky
- diferenční rovnice a jejich řešení
- základy strukturovaného programování
- aktivní znalost alespoň jednoho programovacího jazyka (C, C++, Python, Java, Basic) nebo alespoň prostředí MATLAB

Kam až můžeme dojít?

- Objevit krásu některých algoritmů.

Rychlá Fourierova transformace – analýza EEG signálu



Obrázek 2.5: Rychlá Fourierova transformace – analýza EEG signálu

- Pochopit třeba numerické základy kryptologie.
- Nebát se inženýrských úloh, které vyžadují algoritmizaci.
- Pochopit rychlé algoritmy s aplikacemi v reálném světě

Kurs pokrývá standardní algoritmy, jež nabízí pro daný problém a pro daná vstupní data optimální výkon.

Dvě nejčastější chyby při výběru algoritmu pro danou úlohu:

- **ignorujeme výkon algoritmu** – rychlejší algoritmy jsou současně složitější na implementaci
- **příliš zkoumáme výkon algoritmu** – nepatrně rychlejší algoritmus může být výrazně složitější na implementaci

Literatura

KAPITOLA 3

Analýza algoritmů a jejich složitost

přidat následující text jako abstrakt

Tento text je ručním přepisem poslední verze přednášky, upravené těsně před Štedrým dnem roku 2010. Na Štěpána mi pak z auta nějaký dobrák ukradl notebook a disk se všemi zálohami, včetně zdrojových souborů pro LaTeX. Snažím se postupně doplňovat, co jsem tehdy již doplnil, omluvte prosím občasné nekonzistence textu a prezentace (oboje se generuje ze stejného zdroje).

3.1 Úvod

Analýza algoritmů

Algoritmus:

- myšlenka řešení nějakého problému
- konečný počet kroků řešení
- vyjadřujeme nejčastěji slovně nebo pseudokódem

Program:

- implementace algoritmu ve zvoleném programovacím jazyce

Bez algoritmu nelze napsat program.

Každý problém lze v matematice řešit mnoha různými postupy. Existuje tedy i mnoho různých algoritmů, které vedou ke stejnému cíli. Pokud programátor píše program, který bude použit jenom jednou, zvolí patrně řešení, které lze snadno a rychle implementovat. Jaký algoritmus bychom ale měli preferovat v případě, kdy bude zvolený kus kódu používán opakovaně? V takovém případě je třeba posuzovat různá subjektivní hlediska (srozumitelnost kódu, rozšiřitelnost algoritmu na různé typy vstupních dat, přenositelnost na jiné architektury počítačů a efektivitu výpočtu).

Efektivita algoritmu přímo závisí na tom, kolik zvolený algoritmus spotřebovává výpočetních zdrojů, tedy například jak dlouho bude trvat jeho vykonávání pro určitou množinu vstupních dat, kolik bude potřebovat operační paměti, do jaké míry bude využívat diskový subsystém či síťové rozhraní.

Primárním hlediskem posuzování efektivity algoritmu je ve většině případů *čas*.[1] Uvědomme si ale, že v mnoha případech musí při implementaci docházet k určitým kompromisům – může nastat situace, kdy lze stávající algoritmus zrychlit pouze za cenu neúměrného zvýšení spotřeby operační paměti, spotřeby energie či dalších (ve většině případů přeci jenom omezených) zdrojů. Mnohdy také platí, že implementace velmi efektivních algoritmů jsou mnohem hůře pochopitelné a jejich „údržba“ je proto časově náročnější.

Ideální kombinace: *efektivní algoritmus + efektivní implementace*

Analýza algoritmu:

- poskytne **předpověď výkonnosti** algoritmu
- je **vhodnější než experimenty**
- umožní výběr **vhodné varianty** řešení

Asymptotická složitost **paměťová** × složitost **časová**

Rychlost vykonávání programu, jenž je implementací zvoleného algoritmu, závisí na zvoleném programovacím jazyce, výpočetním výkonu počítače, jeho momentální zátěži či na zvoleném kompilátoru. Přesto bychom rádi znali předem orientační odhad toho, jak efektivní je algoritmus, jenž jsme zvolili k implementaci.

Existují dva základní způsoby, jak si udělat představu o tom, jak se algoritmus chová:

- experimentální ověření chování implementace

[1] Pouze ve velmi ojedinělých případech se setkáváme s nutností omezit paměťový otisk výsledné aplikaci či její energetickou náročnost.

- analýza na základě pseudokódu

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.

Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: *„nenalezli jsme způsob, jak prokázat, že algoritmus je špatně“*.

Záruky funkčnosti poskytuje jedině **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač × pracovní stanice.*

3.2 Vývojová stádia algoritmu

Poprvé popsána italským matematikem Leonardem z Pisy, známým také jako Fibonacci (1202).

Růst populace králíků za poněkud idealizovaných podmínek.

Číslo $F[n]$ popisuje velikost populace po n měsících, předpokládáme-li, že

- první měsíc se narodí jediný pár,
- nově narozené páry jsou produktivní od druhého měsíce svého života,
- každý měsíc zplodí každý produktivní pár jeden další pár,
- králíci nikdy neumírají, nemají predátory.

Fibonacciho číslo	Stav
$F[1] = 1$	začínáme s jedním párem
$F[2] = 1$	ještě jsou příliš mladí
$F[3] = 2$	tento měsíc již zplodí první potomky
$F[4] = 3$	druhý pár potomků
$F[5] = 5$	první potomci třetí generace

Obecně

$$F[n+2] = F[n+1] + F[n]$$

Příklad 3.1 (Výpočet Fibonacciho posloupnosti)**.** *Jak efektivně zjistit $F[n]$ pro zvolené n?*

Algoritmus pomocí explicitního řešení

Explicitní nerekurzivní vztah pro n-tý člen Fibonacciho posloupnosti je

$$F[n] = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989\cdots \approx 1{,}618.$$

Algoritmus 3.1 Přímá varianta výpočtu $F[n]$

Require: n
Ensure: $F[n]$
$F[n] \leftarrow \dfrac{\phi^n - (1-\phi)^n}{\sqrt{5}}$

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F[n] = \frac{1{,}61803^n - 0{,}61803^n}{2{,}23606}.$$

Jaké budou výsledky?

$F[2] = 1{,}00000$ je v pořádku, $F[3] = 1{,}78884$ zaokrouhlíme na 2, $F[20] = 6764{,}69$ ještě zaokrouhlíme na 6765, $F[21] = 10945{,}4$ už zaokrouhlíme na 10945 místo 10946, $F[25] = 75020{,}6$ by mělo být 75025.

Existuje přesnější varianta výpočtu?

Algoritmus 3.2 Rekurzivní varianta výpočtu $F[n]$

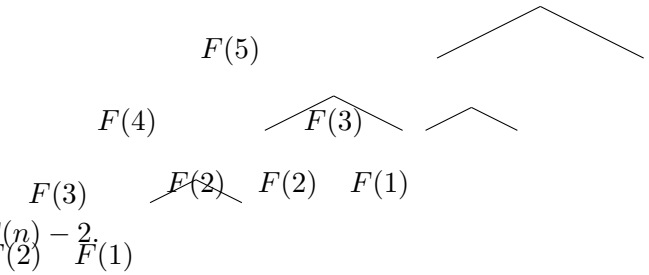
Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $y \Leftarrow 1$
5: **else**
6: $y \Leftarrow F[n-1] + F[n-2]$
7: **end if**

Rekurzivní algoritmus

Hodnoty $F[0]$ až $F[2]$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Jak efektivní je daný algoritmus?

Posloupnost výpočtu $F(5)$:



Počet kroků pro $F(n)$ je $\tau(n) = 3 \cdot F(n) - 2$.

Dynamické programování

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$
Ensure: $y = F[n]$
1: Alokuj $[f_1, f_2, \ldots, f_n]$
2: $f[2] \leftarrow f[1] \leftarrow 1$
3: **for** $i = 3$ **to** n **do**

4: $f_i \leftarrow f_{i-1} + f_{i-2}$
5: **end for**
6: $y \leftarrow f_n$

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.

Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$
Ensure: $y = F[n]$
1: **if** $n = 0$ **then**
2: $y \leftarrow 0$
3: **else**
4: $a \leftarrow 1, b \leftarrow 1$
5: **for** $i = 3$ **to** n **do**
6: $c \leftarrow a + b$
7: $a \leftarrow b, b \leftarrow c$
8: **end for**
9: $y \leftarrow b$
10: **end if**

Maticová varianta pomocí opakovaného mocnění

Pro členy Fibonacciho posloupnosti platí také

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Použitím opakovaného mocnění snížíme počet kroků na $\mathcal{O}(\log n)$.

Důkaz: Indukcí pro $n \to n + 1$.

Začneme pro $n = 1$, kde platí

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix}$$

a vztah tedy pro $n = 1$ dává korektní výsledek. Za předpokladu, že vztah platí

pro n, dostaneme pro $n+1$

$$\begin{aligned}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1}+F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix}\end{aligned}$$

Odvození celého maticového předpisu vychází z teorie dvouprvkových rekurencí a na žádost některých z vás zde uvádím stručný výtah. K dispozici máte volně stažitelnou plnou verzi článku [3], z níž jsem čerpal.

Věta 3.1. *Množina $\mathcal{R}(a,b)$ Mějme dvojici reálných čísel a a $b \neq 0$ a posloupnost A_n definovanou rekurentním vztahem*

$$A_{n+2} = a \cdot A_{n+1} + b \cdot A_n.$$

Potom pro pevné hodnoty parametrů a a b označíme $\mathcal{R}(a,b)$ množinu všech takto parametrizovaných posloupností.

Jedním z význačných prvků $\mathcal{R}(1,1)$ je Fibonacciho posloupnost F s počátečními členy $F_0 = 0$ a $F_1 = 1$.

Definice 3.2. *Posloupnost A Všechny prvky posloupnosti $A_0, A_1, A_2, \ldots$ tvořící vektor v $\mathbb{R}^\infty$ označíme jako* posloupnost A.

Věta 3.3. *Operátor* posunu doleva*, označovaný $\lhd$, odstraní z posloupnosti A její nejlevější prvek.*

Z původní posloupnosti $A = A_0, A_1, A_2, A_3, \ldots$ vznikne posunem doleva o jednu pozici posloupnost $\lhd A = A_1, A_2, A_3, A_4, \ldots$.

Studiem vlastností dvouprvkových rekurencí snadno dospějeme k závěru, že vlastnosti rekurencí $A \in \mathcal{R}(a,b)$ jsou zcela určeny volbou A_0 a A_1. Prostor $\mathcal{R}(a,b)$ je proto dvourozměrný a každé $A \in \mathcal{R}(a,b)$ musíme být schopni vyjádřit jako lineární kombinaci dvou bázových posloupností X a Y s prvky $X_0 = 1, X_1 = 0$ a $Y_0 = 0, Y_1 = 1$,

$$\begin{aligned} X &= 1, 0, b, ab, a^2b + b^2, \ldots, \\ Y &= 0, 1, a, a^2 + b, a^3 + 2ab, \ldots. \end{aligned}$$

Pro všechna $A \in \mathcal{R}(a,b)$ je tedy

$$A = A_0 X + A_1 Y.$$

Všimněte si, že pro $a = 1$ a $b = 1$ je $Y = F$ (půjde o Fibonacciho posloupnost) a také, že

$$\lhd X = b \cdot Y,$$

takže můžeme jakoukoliv posloupnost $A \in \mathcal{R}(a, b)$ psát jako posloupnost členů

$$A_n = A_0 b Y_{n-1} + A_1 Y_n = A_0 b F_{n-1} + A_1 F_n.$$

Operátor posunu doleva lze v $\mathcal{R}(a, b)$ s bázovými prvky X a Y vyjádřit maticí $\mathbf{M}$ jako

$$\lhd \begin{bmatrix} X \\ Y \end{bmatrix} = \mathbf{M}^{\mathsf{T}} \begin{bmatrix} X \\ Y \end{bmatrix}.$$

Prvky matice $\mathbf{M}$ jsou dány vztahy

$$\begin{aligned} \lhd X &= b \cdot Y, \\ \lhd Y &= X + a \cdot Y, \end{aligned}$$

a proto

$$\mathbf{M} = \begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}.$$

Posloupnost $A \in \mathcal{R}(a, b)$ je v bázi $[X, Y]$ možno zapsat jako

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

přičemž pro n-krát posunutou posloupnost bude platit

$$\overset{n}{\lhd} A = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Vzhledem k definici transformační matice $\mathbf{M}$ je ale také $\lhd^n A = \mathbf{M}^n A$, a proto

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Pro n-násobné posuny bázové posloupnosti X a Y můžeme psát

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \end{bmatrix} = \begin{bmatrix} X_n \\ X_{n+1} \end{bmatrix},$$

respektive

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \end{bmatrix} = \begin{bmatrix} Y_n \\ Y_{n+1} \end{bmatrix},$$

z čehož vyplývá

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} X_n & Y_n \\ X_{n+1} & Y_{n+1} \end{bmatrix}.$$

Již dříve jsme si všimli, že $Y = F$ (a tedy $Y_n = F_n$) a že $\Lambda X = b \cdot F$ (a tedy $X_n = b \cdot F_{n-1}$). V $\mathcal{R}(a, b)$ bude proto platit

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n = \begin{bmatrix} b \cdot F_{n-1} & F_n \\ b \cdot F_n & F_{n+1} \end{bmatrix}$$

a v $\mathcal{R}(1, 1)$ ekvivalentně

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}.$$

Algoritmus 5

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $y \Leftarrow 0$
3: **else**
4: $\mathbf{M} \Leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
5: $\mathbf{M} \Leftarrow \text{matpow}(\mathbf{M}, n-1)$
6: $y \Leftarrow M[0, 0]$
7: **end if**

Algoritmus 5a

Require: $n \geq 0, \mathbf{A}[2 \times 2]$
Ensure: $\mathbf{B} = \text{matpow}(\mathbf{A}, n)$
1: **if** $n > 1$ **then**
2: $\mathbf{B} \Leftarrow \text{matpow}(\mathbf{A}, n/2)$
3: $\mathbf{B} \Leftarrow \mathbf{BA}$
4: **end if**
5: **if** n je liché **then**
6: $\mathbf{B} \Leftarrow \mathbf{A} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
7: **end if**

Jakým způsobem se bude chování algoritmu měnit v závislosti na velikosti (počtu, objemu) vstupních dat?

Dva základní typy:

- **časová složitost** – vliv na dobu výpočtu
- **paměťová složitost** – nároky na operační paměť

Značíme:

- $\mathcal{O}(N)$ – lineární složitost,
- $\mathcal{O}(N^2)$ – kvadratická složitost,
- $\mathcal{O}(\log N)$ – logaritmická složitost.

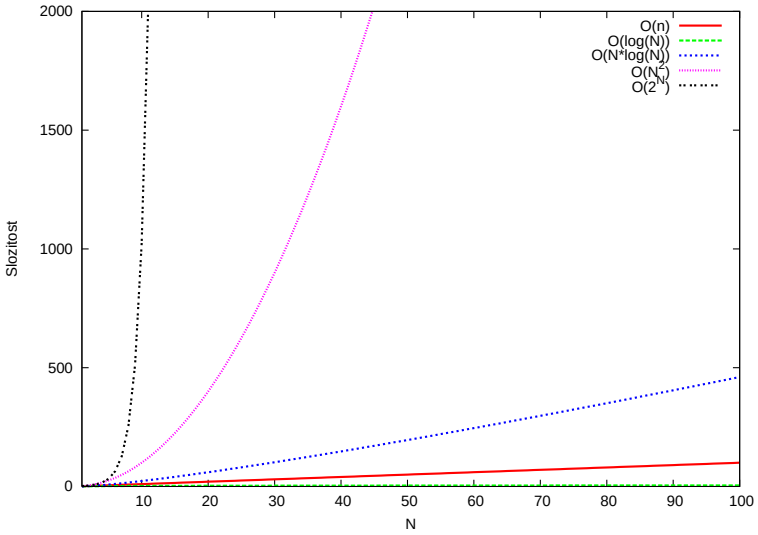
Složitost $\mathcal{O}(N)$ znamená lineárně rostoucí nároky, $\mathcal{O}(N) \sim k \cdot N + q$ pro nějaká $k, n \in \mathbb{N}$, pro $\mathcal{O}(1)$ jsou nároky konstantní, $\mathcal{O}(1) \sim q$.

Vliv asymptotické časové složitosti

Pro $\mathcal{O}(N^2)$ má zdvojnásobení objemu vstupních dat za následek čtyřnásobnou dobu vykonávání algoritmu. Pro $\mathcal{O}(\log N)$ může mít čtyřnásobný počet dat na vstupu za následek dvojnásobnou dobu vykonávání algoritmu. Pro $\mathcal{O}(1)$ je doba vykonávání algoritmu nezávislá na velikosti vstupu.

Vliv asymptotické paměťové složitosti

Pro $\mathcal{O}(N)$ má zdvojnásobení velikosti vstupu za následek dvojnásob vysoké nároky na operační paměť. Pro $\mathcal{O}(2^N)$ čtyřnásobná velikost vstupu zosmináso bí paměťové nároky.



3.3 Porovnání variant

Algoritmus	Paměťové nároky	Časová složitost
1	$O(1)$	$O(\log N)$
2	$O(N)$	$O(F(N))$
3	$O(N)$	$O(N)$
4	$O(1)$	$O(N)$
5	$O(\log N)$	$O(\log N)$

3.4 NP-úplné problémy

Jistě jste už někde slyšeli či četli, že nějaký matematický problém je NP-úplný (angl. *NP-complete*). Podívejme se na závěr této přednášky krátce na to, co tento termín znamená a proč je z hlediska algoritmů tak důležitý. Detailnější pohled na celý problém lze nalézt například v poznámkách prof. Demlové na FEL [2] či v oblíbené učebnici pánů Cormena, Leisersona, Rivesta a Steina [1].

Jako **rozhodovací úlohu** označujeme úlohu, jejímž řešením jsou výroky „ANO“ respektive „NE“. Jako výstup v případě počítače můžeme uvažovat hodnoty true a false nebo 1 a 0.

Běžné úlohy v matematice lze snadno převést na rozhodovací úlohy. Například hledání nejkratší cesty v ohodnoceném grafu lze převést na rozhodovací úlohu „Lze v grafu najít cestu, jejíž délka je menší, než nějaké c?“

Definice 3.4 (Třída P)**.** *Rozhodovací úloha L náleží do třídy P, pokud existuje* deterministický *Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.*

Minimální kostra grafu – existuje kostra s ohodnocením menším, než c?

Nejkratší cesta v grafu – existuje cesta mezi dvěma uzly s ohodnocením menším, než c?

Lineární programování – existuje $\arg\max_{\mathbf{x}} \mathbf{w}^{\mathsf{T}}\mathbf{x} > c$ za daných omezujících podmínek?

Komprese dat (LZW) – přidá komprese řetězce s do slovníku slovo t?

Definice 3.5. *Rozhodovací úloha L náleží do třídy NP, pokud existuje* nedeterministický *Turingův stroj, který tuto úlohu rozhodne v polynomiálním čase.*

Nederministický Turingův stroj: vstupům může odpovídat více, než jedna jediná akce (sekvence instrukcí se převede na neustále se větvící strom instrukcí). Jeho chování si lze představit tak, že v každém přechodu může dojít k naklonování několika kopií stroje se stejnou historií, ale jiným cílovým stavem daného přechodu. Výsledkem provádění důkazu rozhodovací úlohy je potom neustále se rozšiřující strom možností, z nichž v určitém čase jedna povede k cíli.

Platí P$\subseteq$NP. Pokud použijeme nedeterministický Turingův stroj v takové konfiguraci, že v každém přechodu nedojde k naklonování žádné kopie, stroj bude vlastně deterministický a jsme v třídě P.

Všechny úlohy třídy P.

Izomorfismus grafu – lze dané dva grafy nakreslit stejně?

Faktorizace čísel – pro dané n a k, existuje $f : 1 < f < k, f \mid n$?

Všechny NP-úplné úlohy.

Třída NP-úplných problémů je třídou rozhodovacích úloh, pro něž platí následující definice:

Definice 3.6. *Rozhodovací úloha L je NP-úplná, pokud náleží do třídy NP a zároveň jde o úlohu NP- těžkou.*

Co to znamená:

- jakékoliv řešení L lze ověřit v polynomiálním čase,
- Jakýkoliv problém z třídy NP lze převést na L transformací vstupů opět v polynomiálním čase.

Tyto typy úloh umíme řešit pouze *přibližně*!

Základním úskalím NP-úplných úloh je to, že je sice možné rychle (tedy v polynomiálním čase) ověřit, zda zadané řešení úlohy platí, není ale známý

žádný efektivní způsob, jak toto řešení pro dané vstupy najít. Tento paradox je jednou z nejvýznamnějších vlastností této třídy úloh: Pro NP-úplnou úlohu není znám algoritmus, jenž by dokázal dostatečně rychle najít její řešení. Čas, potřebný k vyřešení takové úlohy jakýmkoliv v současné době známým algoritmem roste velmi rychle spolu s tím, jak roste „velikost“ řešeného problému (například počet cifer faktorizovaného čísla). Doba řešení takových úloh v současné době známými algoritmy dosahuje i pro rozumně velké rozsahy vstupních data klidně miliónů let bez ohledu na to, jak roste výpočetní výkon současných počítačů.

Problém batohu – lze zabalit batoh tak, aby jeho hmotnost nepřesáhla m a cena věcí byla alespoň c?Vlastní problém lze možná lépe popsat takto: pokud mám množinu $\mathcal{A}$ obsahující N předmětů o hmotnosti μ a ceně γ, existuje $\mathcal{B} \subseteq \mathcal{A}$ taková, že

$$\sum_{j \in \mathcal{B}} \mu_j \leq m \text{ a zároveň } \sum_{j \in \mathcal{B}} \gamma_j \geq c?$$

Problém obchodního cestujícího – existuje v grafu hamiltonovská kružnice o délce nejvýše c? *Hamiltonovská kružnice* v grafu prochází právě jednou všemi jeho vrcholy, stejně tak jako si přejeme, aby náš obchodní cestující navštívil každé město právě jednou.

Obarvení grafu – lze uzly daného grafu obarvit nejvýše c barvami tak, aby sousedící uzly neměly stejnou barvu?

Problém čínského listonoše (pouze na smíšeném grafu) – existuje v grafu eulerovská kružnice o délce nejvýše c? *Eulerovská kružnice* v grafu prochází právě jednou všechny hrany, stejně tak, jako pošťák musí projet všechny ulice, z nichž některé jsou jednosměrné.

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, Cambridge, 2 edition, 2001.

[2] Marie Demlová. A4m01tal – teorie algoritmů, 2013.

[3] Dan Kalman and Robert Mena. The fibonacci numbers–exposed. *Mathematics Magazine*, 76(3):xx–xx, 2003.

KAPITOLA 4

Prvočísla a dělitelnost.

4.1 Prvočísla

Známe dvě skupiny přirozených čísel $n \in \mathbb{N}$.

Definice 4.1. *Prvočíslo<2-> **Prvočíslem** nazýváme takové přirozené číslo $n \in \mathbb{N}$, které je* beze zbytku dělitelné právě dvěma různými přirozenými čísly *a to jedničkou a samo sebou.*

Číslo 1 tedy není prvočíslo.

Definice 4.2. *Číslo složené<3-> Celé číslo různé od jedné, jež není prvočíslem, nazýváme **složené číslo**.*

Vlastnosti prvočísel:

- Pro prvočíslo p platí $p \mid a \cdot b \Rightarrow (p \mid a) \vee (p \mid b)$.
- Každé složené číslo lze jednoznačně vyjádřit jako součin prvočísel.

  Příklad 4.1. *<2->[Vzorový rozklad] Například* $42 = 2 \cdot 21 = 2 \cdot 3 \cdot 7$.

- Pokud p je prvočíslo a $a \in \mathbb{Z} : 0 < a < p$, pak $p \mid (a^p - a)$.
- Ke všem celým kladným číslům $a \in \mathbb{Z} : a > 0$ lze nalézt prvočíslo $p : a < p \leq 2a$.

  Příklad 4.2. *<4-> Nechť* $a = 42$. *Nerovnici* $p : 42 < p \leq 84$ *splňují prvočísla 43, 47, 53, 59, 61, 67, 71, 73, 79, a 83.*

1	*2*	*3*	4	5	6	7	8	9	10
11	12	*13*	14	15	16	*17*	18	*19*	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	*103*	104	105	106	*107*	108	*109*	110
111	112	113	114	115	116	117	118	119	120

Tabulka 4.1: Seznam prvočísel (vyznačena červeně) od 2 do 120.

Uměli byste pokračovat dál?

Označme $\pi(N)$ počet prvočísel $\leq N$.

Počítejme zkusmo hustotu prvočísel ϱ_N v intervalu $\langle 1, N\rangle$:

- v desítce čísel je $\pi(10) = 4$ prvočísla, tedy
$$\varrho_{10} = \frac{\pi(10)}{10} = \frac{4}{10} = 0{,}4$$
- ve stovce čísel je $\pi(100) = 25$ prvočísel, tedy
$$\varrho_{100} = \frac{\pi(100)}{100} = \frac{25}{100} = 0{,}25$$
- v tisícovce čísel je $\pi(1000) = 168$ prvočísel, tedy
$$\varrho_{1000} = \frac{\pi(1000)}{1000} = \frac{168}{1000} = 0{,}168$$

V roce 1792 si mladý C. F. Gauss všiml, že $\pi(N)$ je přibližně rovna podílu $N/\ln N$.

N	10	10^2	10^3	10^4	10^5	10^6
ϱ_N	0,400	0,250	0,168	0,123	0,096	0,078
$1/\ln N$	0,434	0,217	0,145	0,108	0,086	0,072
$N/\ln N$	4,3429	21,715	144,76	1085,7	8685,9	72382
$\pi(N)$	4	25	168	1229	9592	78498

Matematici někdy píší, že
$$\pi(N) \sim \frac{N}{\ln N}$$
a říkají, že $\pi(N)$ se **asymptoticky blíží** k $N/\ln N$.

Návrh 4.3. *Nejedná se o náhodný jev, při dostatečně velkém N je hustota prvočísel v intervalu $\langle 1, N\rangle$ rovna*

$$\lim_{N\to\infty} \varrho_N = \frac{1}{\ln N}$$

Gaussovi bylo tehdy patnáct let. Důkaz tohoto tvrzení přišel až o 100 let později.

Definice 4.4 (Prvočíselná věta)**.**

$$\lim_{N\to\infty} \frac{\pi(N)}{\frac{N}{\ln(N)}} = 1$$

Vlastnosti prvočísel

■ Eukleidův důkaz ■

Prvočísel je nekonečně mnoho:

- Předpokládejme, že existuje největší prvočíslo a označme jej p_M
- Sestrojíme součin všech prvočísel až do p_M:

$$N = 2 \cdot 3 \cdot 7 \cdots p_M = \prod_{i=1}^{M} p_i$$

- Číslo $N + 1$ nemůže být dělitelné ani jedním z prvočísel p_i, jež dělí N.
- To znamená, že $N + 1$ je buď *prvočíslo*, nebo *číslo složené*, jež má ve svém rozkladu jiné prvočíslo $p_N > p_M$.
- Spolu s Eukleidem jsme dospěli ke sporu!
- Musí tedy platit, že *prvočísel je nekonečně mnoho.*

Eukleidův důkaz je klasický existenční důkaz: Řeší pouze otázku existence nekonečné množiny prvočísel, neřeší otázku jak nalézt všechna prvočísla.

Goldbachova hypotéza říká, že každé sudé číslo větší než 2 lze vyjádřit jako součet dvou prvočísel, například

$$\begin{aligned} 8 &= 3+5 \\ 10 &= 3+7 \\ 12 &= 5+7 \\ 14 &= 3+11 \\ 16 &= 5+11 \\ 18 &= 7+11 \end{aligned}$$

Experimentálně prověřeno do hodnot 2×10^{17}

Zajímavosti

Párová prvočísla: jejich rozdíl je 2 (například 17 a 19), největší dosud známé prvočíselné páry jsou

$$\begin{aligned} 16\,869\,987\,339\,975 &\cdot 2^{171960} \pm 1 \\ 100\,314\,512\,544\,015 &\cdot 2^{171960} \pm 1 \end{aligned}$$

Odhalení chyby matematického koprocesoru originálního Intel Pentium P5 (*The Intel FDIV Bug*).

Thomas Nicely, Lynchburg College, Virginia (1994): Numerický výpočet součtu *harmonické řady s párovými prvočísly.*

O jaké řady jde:

- harmonická řada
$$\sum_{n=1}^{\infty} \frac{1}{n} \to \infty$$
- prvočíselná harmonická řada
$$\sum_{\forall p}^{\infty} \frac{1}{p} \to \infty$$

Obě tyto řady divergují.

Oproti tomu

$$\begin{aligned} \sum_{\forall p_2}^{\infty} \frac{1}{p_2} &= \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \cdots + \frac{1}{29} + \frac{1}{31} + \ldots \\ &= 1{,}902160583104 \end{aligned}$$

konverguje.

V červnu 1994 Thomas Nicely obdržel po povýšení starého počítače na P5 hodnotu

$$1{,}9021605778$$

lišící se od původních výpočtů na i486 – a v říjnu oznámil chybu v FPU Pentia.

Tim Coe, Vitesse Semiconductor, Southern California

$$\begin{aligned} c = \frac{4195835}{3145727} &= \frac{5 \cdot 7 \cdot (2^3 \cdot 3^4 \cdot 5 \cdot 37 + 1)}{3 \cdot 2^{20} - 1} = \\ &= 1{,}33382044\ldots \end{aligned}$$

FPU v Pentiu P5 však dávala hodnotu

$$c = \frac{4195835}{3145727} = \frac{5 \times 7 \times 119881}{13 \times 241979} = 1{,}33373906\ldots$$

Chyba nastává při reprezentaci čísel typu $M_n = 2^n - 1$, což jsou tak zvaná *Mersennova čísla*.

Mersennova čísla

Marin Mersenne (1588-1648) uveřejnil ve své knize Cogitata Physica-Mathematica (1644) tvrzení, že čísla tvaru

$$2^n - 1$$

jsou prvočísly pro $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$ a jsou čísla složená pro ostatní přirozená čísla $n \leq 257$.

Definice 4.5 (Mersennovo prvočíslo)**.** *Jestliže $2^n - 1$ je prvočíslo, pak se nazývá **Mersennovo prvočíslo**.*

Lze dokázat, že pokud je $2^n - 1$ prvočíslo, je i n prvočíslem.

Prvočíselný charakter Mersennových čísel nebylo snadné dokázat:

- Euler (1750): $2^{31} - 1$ je prvočíslo.
- Lucas (1876): $2^{127} - 1$ je prvočíslo.

- Pervouchine (1883): Mersenne zapomněl na $2^{61} - 1$.
- Powers (?) ukázal, že existují další čísla, která Mersenne neuvedl: $2^{89} - 1$ a $2^{107} - 1$.

Mersennův interval $n \leq 257$ byl úplně prozkoumán v roce 1947 a bylo dokázáno, že správné tvrzení obsahuje 12 exponentů:

$$n = 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127.$$

K dnešnímu dni bylo nalezeno celkem 47 Mersennových prvočísel $M_{521}, M_{607}, M_{1\,279}, \ldots, M_{42\,643\,801}$ Čísla ovšem nejdou popořadě, k dnešnímu dni víme pouze, že známe prvních 41 Mersennových prvočísel.

Příklad 4.3 (**GIMPS** (The Great Internet Mersenne Prime Search))**.** *Paralelizované hledání jehly v kupce sena:*

- *Distribuovaný výpočet ve volných cyklech procesoru*
- *Zatím poslední nalezené Mersennovo prvočíslo má 12837064 cifer a bylo nalezeno 12. dubna 2009 ve tvaru*

$$2^{42\,643\,801} - 1.$$

- *Zatím největší bylo nalezeno 23. srpna 2008 ve tvaru*

$$2^{43\,112\,609} - 1.$$

- *http://www.mersenne.org/*

Fermatova čísla

Pro nezáporné $n \geq 0$ nazýváme n-tým **Fermatovým číslem** výraz

$$F_n = 2^{2^n} + 1.$$

Je známo, že F_n je

- prvočíslem pro $0 \leq n \leq 4$ a
- číslem složeným pro $5 \leq n \leq 23$.

Fermat se původně domníval, že F_n jsou obecně prvočísla.

Jak vlastně rozhodneme, na jaké součinitele rozložit složené číslo N?

Faktorizace prvočísel

Definice 4.6 (Základní věta artimetiky)**.** *Každé přirozené číslo větší než 1 lze jednoznačně rozložit na součin prvočísel.*

Nalezení rozkladu malých čísel na prvočísla je relativně jednoduché:

Zkouška dělením

Pro výpočet prvočíselných součinitelů čísla N stačí otestovat všechna prvočísla $p_i < \sqrt{N}$. Prvočinitele získáme například použitím Eratosthenova síta.

Náročnost faktorizace *výrazně roste s délkou* prvočísla.

Praktické důsledky:

- (+) kryptografie (šifrování veřejným klíčem, RSA),

Metody prosévání:

- Eratosthenovo síto
- (Generické|Speciální) prosévání číselného pole
- Pollardova ρ-metoda
- Rozklad na řetězové zlomky

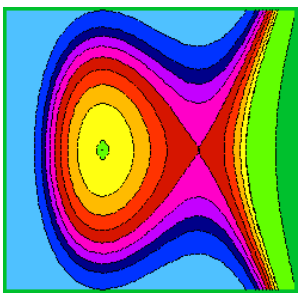
Pro rozklad velkých čísel na prvočíselné součinitele se používají celočíselné vlastnosti eliptických křivek

$$y^2 = x^3 + a\,x + b$$

Na stránkách

http://www.alpertron.com.ar/ECM.HTM

můžete ověřit účinnost těchto matematických metod.



- Euler (1732) nalezl rozklad

$$F_5 = 641 \cdot 6\,700\,417 = 4\,294\,967\,297.$$

- Žádné další prvočíslo tvaru $F_n = 2^{2^n} + 1$ není pro $n \geq 24$ známo.

Dějiny faktorizace

Rozklad Fermatových čísel se od doby Eulera stal velkou soutěží o vhodné algoritmy.

Samostatný mikrokosmos numerické matematiky: F_n roste v počtu cifer závratně rychle – algoritmus vhodný pro faktorizaci F_n nemusí být použitelný pro F_{n+1}.

V roce 1880 Landry zveřejnil součin

$$F_6 = 274\,177 \cdot p_{14}.$$

Algoritmus, kterým Landry k tomuto výsledku dospěl, nebyl nikdy publikován.

V roce 1970 Morrison a Brillhart nalezli pomocí metod řetězových zlomků součin

$$F_7 = 59\,649\,589\,127\,497\,217 \cdot p_{22}.$$

V letech 1877–1970 bylo objeveno několik nevelkých součinitelů Fermatových čísel ve tvaru $k \cdot 2^{n+2} + 1$ pro $n \geq 9$.

Western, 1903

Například již v roce 1903 Western nalezl

$$F_9 = 2\,424\,833 \times C_{148},$$

kde C_{148} je celé 148-ciferné číslo.

V roce 1980 Brent a Pollard nalezli součin

$$F_8 = 1238926361552897 \times p_{62}$$

Pollardovou ρ-metodou.

V roce 1990 skupina matematiků a počítačových odborníků kolem Pollarda použila více než 700 pracovních stanic rozmístěných po celém světě a odvodili metodou SNFS (Special Number Field Sieve) pro

$$F_9 = 2\,424\,833 \times p_{49} \times p_{99}.$$

V říjnu 2003 John Cosgrave se spolupracovníky na St. Patrick's College nalezli součinitele Fermatova číslu

$$F_{2478782} = (3 \times 2^{2478785} + 1) \cdot k.$$

Faktorizace:

- není kompletní pro všechna Fermatova čísla, o kterých víme, že jsou rozložitelná (F_1 až F_{32}),
- například pro F_{12} není znám součinitel C_{1187} o velikosti 1187 cifer,
- podobně pro $F_{13}, F_{15}, \ldots, F_{19}, F_{25}, \ldots, F_{32}$ chybí součinitele různých ciferných délek,
- pro $F_{20}, \ldots, F_{24}$ součinitele neznáme vůbec.

n	$F_n = 2^{2^n} + 1$
0	3
1	5
2	7
3	257
4	65 537
5	$641 \cdot 6\,700\,417$
6	$274\,177 \cdot 67\,280\,421\,310\,721$
7	$59\,649\,589\,127\,497\,217 \cdot 5\,704\,689\,200\,685\,129\,054\,721$
8	$1\,238\,927\,497\,217 \cdot p_{62}$
9	$2\,424\,833 \cdot p_{49} \cdot p_{99}$

V tabulce označuje p_k k-ciferné prvočíslo. Například $F_6 = 274\,177 \cdot 67\,280\,421\,310\,721 = 274\,177 \cdot p_{14}$.

Euklidova čísla

Čísla definovaná rekurencí

$$e_n = e_1\, e_2\, e_3 \ldots\, e_{n-1} + 1$$

nazýváme **Euklidova čísla**.

První čtyři Euklidova čísla

$$\begin{aligned} e_1 &= 1 + 1 = 2 \\ e_2 &= 2 + 1 = 3 \\ e_3 &= 2 \times 3 + 1 = 7 \\ e_4 &= 2 \times 3 \times 7 + 1 = 43 \end{aligned}$$

jsou *prvočísla.*

Další Euklidova čísla až na e_6

$$e_5 = 2 \cdot 3 \cdot 7 \cdot 43 + 1 = 1\,807 = 13 \cdot 139 \tag{4.1}$$
$$e_6 = 2 \cdot 3 \cdot 7 \cdot 43 \cdot 1\,807 + 1 = 3\,263\,443 \tag{4.2}$$
$$e_7 = 547 \cdot 607 \times 1\,033 \cdot 31\,051 \tag{4.3}$$
$$e_8 = 29\,881 \cdot 67\,003 \cdot 9\,119\,521 \cdot 6\,212\,157\,481 \tag{4.4}$$

jsou *složená čísla.* Pro všechna čísla $e_9 \ldots e_{17}$ je dokázáno, že jsou to složená čísla.

Fakt 4.7. *Euklidova čísla jsou* ***nesoudělná čísla****, protože jejich největší společný dělitel je roven 1:*

$$\gcd(e_m, e_n) = 1.$$

4.2 Nejvyšší společný dělitel

V celočíselné aritmetice dělíme se zbytkem: je

$$a = qb + r.$$

Pro dvojici celých čísel a a b má smysl hledat nejvyšší celé číslo d, které obě čísla dělí beze zbytku.

Definice 4.8 (Největší společný dělitel)**.** ***Nejvyšší společný dělitel*** *dvou nenulových celých čísel $a \in \mathbb{Z}$ a $b \in \mathbb{Z}$ je největší nenulové přirozené číslo $d \in \mathbb{Z} - 0$ takové, že $d|a \wedge d|b$.*

Zapisujeme $\gcd(a, b) = d$.

Definice 4.9. *Nesoudělná čísla<2-> Čísla $a \in \mathbb{Z}$ a $b \in \mathbb{Z}$ nazýváme* ***nesoudělná*** *angl. relative primes, pokud* $\gcd(a, b) = 1$.

Euklidův algoritmus

Původně formulován geometricky cca 300 př.n.l. Eukleidés hledal nejdelší úsečku, která by se beze zbytku vešla do dvou delších úseček.

Metoda nalezení **největšího společného dělitele** (NSD ≡ angl. *greatest common divisor, GCD*) spočívá v jednoduchém pozorování, že největší společný dělitel dvou čísel $a > b$ je shodný s největším společným dělitelem čísel $a - b$ a b.

Tento poznatek již stačí k sestavení algoritmu.

Důkaz. Důkaz

Algoritmus 4.1 Zápis postupu výpočtu největšího společného dělitele čísel a a b Euklidovým algoritmem.

Require: $a, b \in \mathbb{Z}$
Ensure: $\gcd(a, b)$
  repeat
    if $a < b$ **then**
      $c \leftarrow a$; $a \leftarrow b$; $b \leftarrow c$;
    end if
    $a \leftarrow a - b$;
  until $a = 0$
  return b

1. Nechť a a b jsou nenulová celá čísla, jejichž gcd() počítáme.
2. Pokud $a > b$, platí $a = b + r$, kde $r = a - b$.
3. Pokud $a = s \cdot d$ a $b = t \cdot d$ bude $r = (s - t) \cdot d$.
4. Je tedy $\gcd(a, b) = \gcd(b, r)$ a stačí tedy hledat $\gcd(b, r)$.
5. Hodnota r postupně klesá a výpočet v konečném počtu kroků skončí stavem $r = 0$.

□

4.3 Dělitelnost

Definice 4.10. *Na množině celých čísel $\mathbb{Z}$ mějme definována dvě čísla: a, b.*

*Říkáme, že a **dělí** b, pokud existuje libovolné $c \in \mathbb{Z}$ takové, že $b = ac$. Pro zkrácený zápis toho vztahu používáme symbol $a \mid b$.*

V případě, že $a \nmid b$, dělíme se zbytkem: platí

$$b = q \cdot a + r,$$

kde $q \in \mathbb{Z}$ a $r \in \mathbb{N}$ je zbytek po dělení. Všimněte si, že pro zbytek platí $r < a$ a že $r = 0$ pouze v případě, kdy $a \mid b$.

Příklad 4.4. *Pro čísla 7 a 8 platí $8 = 1 \cdot 7 + 1$, tedy $7 \nmid 8$. Zbytek po dělení je 1.*
Pro čísla 7 a 71 platí $8 = 10 \cdot 7 + 1$, tedy $7 \nmid 71$. Zbytek po dělení je opět 1.

Příklad 4.5 (Dělitelnost záporných čísel)**.** *Pro čísla 7 a 8 platí $8 = 1 \cdot 7 + 1$, tedy $7 \nmid 8$. Zbytek po dělení je 1.*
Pro čísla 7 a -8 platí $-8 = -2 \cdot 7 + 6$, tedy $7 \nmid -8$. Zbytek po dělení je ovšem 6!

Největší společný dělitel

Pro **společný dělitel** c čísel a a b platí, že $c \mid a$ a zároveň $c \mid b$.

Definice 4.11 (Největší společný dělitel)**.** *Číslo d označujeme jako **největšího společného dělitele** čísel a a b a zapisujeme $d = \gcd(a, b)$, pokud platí, že*

- *číslo d je společný dělitel a a b, a*
- *pokud existuje nějaké $c \neq d$ takové, že $c \mid a$ a zároveň $c \mid b$, pak také $c \mid d$.*

Číslo $\gcd(a, b)$ je tedy největším kladným celým číslem jež dělí jak a, tak i b, s výjimkou $\gcd(0, 0) = 0$.

Připomínáme, že čísla $a \in \mathbb{Z}$ a $b \in \mathbb{Z}$ nazýváme **nesoudělná** *(relative primes)*, pokud $\gcd(a, b) = 1$. Nejvyšší společný dělitel dvou čísel lze efektivně spočítat například Eukleidovým algoritmem, uvedeným v minulé přednášce (pro úplnost jej uvádíme i zde jako Algoritmus 4.2).

Algoritmus 4.2 Zápis postupu výpočtu největšího společného dělitele čísel a a b Euklidovým algoritmem.

Require: $a, b \in \mathbb{Z}$
Ensure: $\gcd(a, b)$
  repeat
    if $a < b$ **then**
      $c \leftarrow a$; $a \leftarrow b$; $b \leftarrow c$;
    end if
    $a \leftarrow a - b$;
  until $a = 0$
  return b

Připomeňme, že pro $a, b, q \in \mathbb{Z}$ a $r \in \mathbb{N}$ platí

$$b = q \cdot a + r.$$

Definice 4.12 (Modulo)**.** *Zbytek po dělení dvou čísel označíme **modulo**, zapisujeme $b \bmod a$. Platí*

$$b = q \cdot a + r \Leftrightarrow r = b \bmod a.$$

Příklad 4.6. *Je $23 \bmod 4 = 3$, neboť $23 = 5 \cdot 4 + 3$.*

Všimněte si, že původní Eukleidův algoritmus nahrazuje dělení opakovaným odčítáním – pokud je rozdíl mezi čísly a a b dostatečně velký, počítáme postupně s páry (a, b), $(a - b, b)$, $(a - b - b, b)$ a tak dále. Tato posloupnost končí

v okamžiku, kdy by další odečtení čísla b způsobilo změnu znaménka. V ten okamžik je vlastně $a = k \cdot b + r$.

Toto pozorování vede na modifikaci původního Eulerova algoritmu, kde opakované odčítání nahradíme výše definovanou operací modulo. Tento postup uvádíme v Algoritmu 4.3.

Algoritmus 4.3 Modifikovaný postupu výpočtu největšího společného dělitele čísel a a b Euklidovým algoritmem s operací modulo.

Require: $a, b \in \mathbb{Z}$
Ensure: $\gcd(a, b)$
  repeat
    $b \leftarrow b \bmod a$
    $c \leftarrow a$; $a \leftarrow b$; $b \leftarrow c$;
  until $a = 0$
  return b

Mezivýsledky?

Pozorujeme-li mezivýsledky jednotlivých kroků Eulerova algoritmu, nahlédneme, že jde vždy o lineární kombinace čísel a a b s celočíselnými koeficienty. To vede k následujícímu pozorování.

Definice 4.13 (Bézoutova rovnost)**.** *Nejvyšší společný dělitel celých čísel a a b lze vyjádřit jako*

$$a \cdot x + b \cdot y = \gcd(a, b),$$

kde $x, y \in \mathbb{Z}$.

Hodnoty x a y lze spočíst **rozšířeným Eukleidovým algoritmem** uvedeným v Algoritmu 4.4.

4.4 Modulární aritmetika

Modulární aritmetika je aritmetikou na množině celých čísel $\mathbb{Z}$ v níž se čísla opakují po dosažení určité hodnoty n, již nazýváme **modul**.

Na rozdíl od běžných celočíselných operací se zde po každé operaci provede ještě *celočíselné dělení* modulem n a výsledkem operace je *zbytek* po tomto dělení.

Příklad 4.7. *V modulární aritmetice modulo 7 mají čísla 8 a 71 shodné reprezentace, protože* $8 \bmod 7 = 1$ *a zároveň* $71 \bmod 7 = 1$.

Celočíselná aritmetika v počítačích je modulární.

Algoritmus 4.4 Rozšířený Eukleidův algoritmus pro výpočet Bézoutovy rovnosti $a \cdot x + b \cdot y = \gcd(a, b)$.

Require: $a, b \in \mathbb{Z}$
Ensure: $\gcd(a, b), x, y$
  $a_x \leftarrow 1$; $a_y \leftarrow 0$;
  $b_x \leftarrow 0$; $b_y \leftarrow 1$;
  repeat
    $m \leftarrow b \operatorname{div} a$
    $b \leftarrow b - m \cdot a$
    $b_x \leftarrow b_x - m \cdot a_x$
    $b_y \leftarrow b_y - m \cdot a_y$
    $a \Longleftrightarrow b$; $a_x \Longleftrightarrow b_x$; $a_y \Longleftrightarrow b_y$;
  until $a = 0$
  return b, b_x, b_y

Příklad 4.8 (Aritmetika osmibitových čísel)**.** *Výsledkem operace 250+10 v osmibitové aritmetice je 4 (tedy* $260 \bmod 2^8$*). 12-16 dá v osmibitové aritmetice číslo 252 (což je* $-4 \bmod 2^8$*).*

Praktické aplikace modulární aritmetiky:

- **přenos zpráv** – ochrana zpráv proti chybám, komprese, zajištění integrity, utajování,
- **výpočetní technika** – hašovací funkce, pseudonáhodná čísla, dvojková komplementární reprezentace celých čísel, aritmetika s VELKÝMI celými čísly.

Kongruence

Uvažujme libovolný modul n takový, že $n \in \mathbb{N}$ a zvolme si dvě celá čísla $a, b \in \mathbb{Z}$.

Definice 4.14 (Kongruence)**.** *Pokud v modulární aritmetice platí, že* $a \bmod n$ *a* $b \bmod n$ *jsou si rovny (mají stejný zbytek po dělení* n*), říkáme, že že* a *je kongruentní s* b *modulo* n *a zapisujeme*

$$a \equiv b \pmod{n}.$$

Příklad 4.9. *Je tedy* $8 \equiv 71 \pmod 7$*,* 8 *je kongruentní s* 71 *modulo* 7*. Pozor na záporná čísla:* $-1 \equiv 7 \pmod 8$*.*

Označme si m onen zbytek po dělení $a \bmod n$ a $b \bmod n$. Bude potom platit, že

$$a = i \cdot n + m$$
$$b = j \cdot n + m$$

pro nějaká $i, j \in \mathbb{Z}$. V příkladu, uvedeném výše, je

$$8 = 1 \cdot 7 + 1$$
$$71 = 10 \cdot 7 + 1$$
$$-6 = -1 \cdot 7 + 1.$$

Příklad 4.10. *Mějme abecedu velkých písmen české abecedy,* $\{\mathsf{A}, \acute{\mathsf{A}}, \mathsf{B}, \ldots, \mathsf{Z}, \check{\mathsf{Z}}\}$*, reprezentovanou numerickými hodnotami* $\{1, 2, \ldots, 42\}$*. Nad touto abecedou provádíme všechny matematické operace modulárně, s modulem 42.*

V takové modulární aritmetice jsou si rovny například reprezentace celých čísel -41*,* 43 *a* 320328919*, protože zbytek po dělení 42 je vždy 1:*

$$\begin{aligned} -41 \equiv 43 \,(\text{mod } 42) &\quad\Leftrightarrow\quad -41 = 43 + 42 \cdot (-2), \\ -41 \equiv 320328919 \,(\text{mod } 42) &\quad\Leftrightarrow\quad -41 = 320328919 + 42 \cdot (-7626880), \\ 320328919 \equiv 43 \,(\text{mod } 42) &\quad\Leftrightarrow\quad 320328919 = 43 + 42 \cdot 7626878. \end{aligned}$$

Znak A *může tedy reprezentovat libovolné z čísel* -41*,* 43 *a* 320328919*.*

Třída kongruence

Množinu všech celých čísel, která jsou kongruentní s nějakým m modulo n je zvykem nazývat **třída kongruence** a zapisovat ji $[m]_n$, nebo (a to hlavně v anglosaské literatuře) bez uvedení modulu kongruence jako $\overline{m}$.

Příklad 4.11. *Například číslo 3 v modulu 5 může zastupovat i všechna čísla s ním kongruentní* $(\ldots, -7, -2, 3, 8, 13, \ldots)$*. V textech bude tato třída kongruence označována jako* $[3]_5$ *nebo jako* $\overline{3}$*.*

Vlastnosti kongruence modulo n umožňují počítat pouze se zbytky po dělení tímto modulem a výsledek pak zobecnit na všechna čísla.

Vlastnosti čísel v modulární aritmetice

Modulární aritmetika je uzavřená vůči operacím sčítání a násobení:

$$[a]_n + [b]_n = [a + b]_n,$$
$$[a]_n - [b]_n = [a - b]_n,$$
$$[a]_n \cdot [b]_n = [a \cdot b]_n.$$

Příklad 4.12. *V aritmetice modulo 7 by mělo platit* $[2]_7 + [6]_7 = [1]_7$. *Pro* $9 \in [2]_7$ *a* $-1 \in [6]_7$ *je výsledek* $9 - 1 = 8 \in [1]_7$. *Zcela obecně je*

$$a = i \cdot 7 + 2,$$
$$b = j \cdot 7 + 6$$

a potom

$$a + b = (i \cdot 7 + 2) + (j \cdot 7 + 6) = (i + j) \cdot 7 + 8 = (i + j + 1) \cdot 7 + 1.$$

Podobně zkuste v aritmetice modulo 7 ověřit $[2]_7 \cdot [6]_7 = [5]_7$.

Sčítání a násobení v modulární aritmetice je komutativní a asociativní:

$$[a]_n + [b]_n = [b]_n + [a]_n,$$
$$[a]_n \cdot [b]_n = [b]_n \cdot [a]_n,$$
$$([a]_n + [b]_n) + [c]_n = [a]_n + ([b]_n + [c]_n),$$
$$([a]_n \cdot [b]_n) \cdot [c]_n = [a]_n \cdot ([b]_n \cdot [c]_n).$$

Pro sčítání a násobení v modulární aritmetice existuje identita, pro sčítání i inverze:

$$[0]_n + [a]_n = [a]_n,$$
$$[a]_n + [-a]_n = [0]_n,$$
$$[1]_n \cdot [a]_n = [a]_n.$$

Příklad 4.13 (Dva příklady)**.** *V modulární aritmetice modulo 7 je* $28 \in [0]_n$ *a* $15 \in [1]_n$. *Pro jejich součet platí* $(28 + 15) \bmod 7 = 43 \bmod 7 = 1$.
V modulární aritmetice modulo 3 je $10 \in [1]_n$ *a* $8 \in [2]_n$. *Pro jejich součin platí* $(10 \cdot 8) \bmod 3 = 80 \bmod 3 = 2$.

Jak dopadne součet 57 a -73 v aritmetice modulo 8?

Modulární krácení

Pokud

$$a \cdot d \equiv b \cdot d \pmod{n}$$

obecně neplatí, že také

$$a \equiv b \pmod{n}$$

Příklad 4.14. *Kongruenci* $8 \equiv 14 \pmod 6$ *sice můžeme rozložit na* $4 \cdot 2 \equiv 7 \cdot 2 \pmod 6$, *ale rozhodně neplatí, že* $4 \equiv 7 \pmod 6$. *Všimněte se ale, že*

$$8 \equiv 14 \pmod 6$$
$$4 \cdot 2 \equiv 7 \cdot 2 \pmod{3 \cdot 2}$$
$$4 \equiv 7 \pmod 3.$$

Zdá se tedy, že v případech, kdy je modul kongruence také dělitelný d, je třeba vykrátit d nejenom z ekvivatelntních tříd, ale i z modulu kongruence.

Jsou dvě varianty

1. Pro $\gcd(d, n) = 1$ je opravdu $a \equiv b \pmod{n}$.

2. Pro $\gcd(d, n) = k, k > 1$ je $d = k \cdot x$ a $n = k \cdot y$ a kongruence se postupně změní na

$$\begin{aligned} akx &\equiv bkx \pmod{ky} \\ ax &\equiv bx \pmod{y} \\ a &\equiv b \pmod{y}. \end{aligned}$$

Příklad 4.15 (Modulární krácení pro d a n nesoudělná)**.** *Pro* $170 \equiv 35 \pmod{3} \longrightarrow 5 \cdot 34 \equiv 5 \cdot 7 \pmod{3}$ *je* $34 \equiv 7 \pmod{3}$*, protože 3 a 5 jsou nesoudělná čísla.*

Příklad 4.16 (Modulární krácení pro obecné $d \neq 0$)**.** *Z kongruence* $10 \equiv 6 \pmod{4} \longrightarrow 5 \cdot 2 \equiv 3 \cdot 2 \pmod{2 \cdot 2}$ *plyne* $5 \equiv 3 \pmod{2}$.

Co vyjde pro $10 \equiv 6 \pmod{3}$?

Jak už bylo naznačeno výše, na množině celých čísel nelze použít operaci dělení tak, jak ji známe z množiny reálných čísel – celá čísla umíme dělit pouze se zbytkem po dělení. Pokud řešíme v $\mathbb{R}$ rovnici

$$5x = 6$$

můžeme proměnnou x osamostatnit tak, že levou i pravou stranu rovnice vynásobíme **inverzním prvkem** čísla 5 vůči operaci násobení, číslem 5^{-1}, tedy zlomkem $1/5$.

V modulární aritmetice, definované na množině celých čísel, ale se zlomky pracovat neumíme. Přesto i kongruenci

$$5x \equiv 6 \pmod{11}$$

umíme převést na tvar

$$x \equiv 6 \cdot 5^{-1} \pmod{11},$$

kde inverzní prvek 5^{-1} označuje tak zvanou **multiplikativní inverzi** čísla 5 v aritmetice modulo 11.

Multiplikativní inverze

Pro $a \in \mathbb{Z}$ a $n \in \mathbb{N}$ je celé číslo x **multiplikativní inverzí** a, pokud splňuje podmínku

$$a \cdot x \equiv 1 \pmod{n}. \tag{4.5}$$

Z kongurence (4.5) lze snadno odvodit, že pokud opravdu lze multiplikativní inverzi x nalézt, lze těchto x pro dané a najít na množině celých čísel $\mathbb{Z}$ nekonečně mnoho. Vzhledem k tomu, že se ve výše uvedeném případě pohybujeme v množině čísel, dané zbytky po celočíselném dělení číslem n, označované jako $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, bude nutně množina $\mathbb{Z}_n$ obsahovat jedno číslo, jež je také hledanou modulární inverzí: Pro **nejmenší multiplikativní inverzi** platí, že x je nejmenší možnou kladnou multiplikativní inverzí k a a označujeme ji a^{-1}.

Všimněte si, že multiplikativní inverze existuje pouze pro čísla, jež jsou nesoudělná s modulem n: přepíšeme-li kongruenci (4.5) do zápisu pomocí dělení se zbytkem, dotaneme pro nějaké $q \in \mathbb{Z}$

$$\begin{aligned} a \cdot x &= 1 + n \cdot q \\ a \cdot x - n \cdot q &= 1 \end{aligned}$$

a po substituci $y = -q$ dostáváme Bézoutovu rovnost

$$\begin{aligned} a \cdot x + n \cdot y &= 1 \\ a \cdot x + n \cdot y &= \gcd(a, n) \end{aligned}$$

a srovnáním obou řádků je jasné, že musí platit $gcd(a, n) = 1$.

4.5 Malá Fermatova věta

Definice 4.15. *Pro $a \in \mathbb{Z}$ a prvočíslo $p \in \mathbb{N}$ takové, že $p \nmid a$ platí*

$$a^{p-1} \equiv 1 \pmod{p} \qquad \textit{resp.} \qquad a^p \equiv a \pmod{p}$$

Malá Fermatova věta[1] je základním stavebním kamenem algoritmu generování šifrovacího klíče asymetrické šifry RSA. Je také nutnou podmínkou pro prvočísla a základním kamenem **Fermatova testu prvočíselnosti**.

Z Malé Fermatovy věty přitom plyne, že

$$a^{-1} \equiv a^{p-2} \pmod{p} \tag{4.6}$$

pro $a \in \mathbb{Z}$ a prvočíselná $p \in \mathbb{N}$ taková, že $p \nmid a$.

[1] Ve skutečnosti je $a^{\phi(n)} \equiv 1 \pmod{n}$, kde $\phi(n)$ je takzvaná **Eulerova funkce**. O té si ale budeme povídat později.

Příklad 4.17 (Výpočet inverze)**.** *Chceme spočítat* a^{-1} *pro* $n = 11$ *a* $a = -3$*. Volíme postupně* $x = 1, 2, \ldots,$ *první kladné číslo* x *splňující vztah (4.5) je* $x = 7$*:* $-3 \cdot 7 \equiv 1 \pmod{11}$*.*

Příklad 4.18 (Výpočet inverze pomocí Malé Fermatovy věty)**.** *Použitím Malé Fermatovy věty (4.6) máme* $a^{-1} \equiv (-3)^{11-2} \pmod{11}$*, tedy* $a^{-1} \equiv -19683 \pmod{11}$ *což je to samé, jako* $a^{-1} \equiv 7 \pmod{11}$ *protože jde o stejnou třídu kongruence.*

Zkuste si to nyní sami pro $n = 7$ a $a = 5$.

4.6 Příklady

Opice a kokosy

Na pustém ostrově ztroskotají tři námořníci. Jediná potrava, kterou během dne našli, je hromada kokosových ořechů.

V noci se první námořník probudí, spravedlivě rozdělí hromadu na tři díly, přičemž jeden kokos zbyde – ten dostane opice. Svou třetinu námořník ukryje, zbytek navrší zpátky a jde zase spát. Postupně hromadu stejným způsobem „třetina pro mne, jeden kokos opici, zbytek vrátit“ zmenší jeho oba druhové.

Ráno si hromadu rozdělí na třetiny, opět zbyde jeden kokos, ten dostane opice.

Kolik musí být v původní hromadě kokosů, aby to fungovalo?

První námořník začíná s hromadou obsahující $n \equiv 1 \pmod 3$ kokosových ořechů.

Druhý námořník dělil hromadu s

$$m_1 = \frac{2(n-1)}{3} \equiv 1 \pmod 3$$

ořechy, třetí námořník přerozděloval

$$m_2 = \frac{2(m_1-1)}{3} \equiv 1 \pmod 3$$

ořechů a ve zbylé hromadě jich muselo zůstat

$$m_3 = \frac{2(m_2-1)}{3} \equiv 1 \pmod 3.$$

Hodnotu m_3 spočteme jako

$$m_3 = \frac{2}{3}m_2 - \frac{2}{3} = \cdots = \frac{8}{27}n - \frac{38}{27} \equiv 1 \pmod 3$$

a rovnici v modulární aritmetice řešíme pro n

$$8n - 38 \equiv 27 \,(\mathrm{mod}\, 81) \Rightarrow 8n \equiv 65 \,(\mathrm{mod}\, 81).$$

Dělit osmi nemůžeme, můžeme ale násobit multiplikativní inverzí (pro jejíž výpočet nelze použít Fermatovu větu – proč?):

$$n \equiv 8^{-1} \cdot 65 \equiv 71 \cdot 65 \equiv 79 \,(\mathrm{mod}\, 81).$$

Nejmenší počet kokosů v hromadě je tedy 79 (ale může být i 160, 241, ...).

Kontrolní součty

Má ho každá kniha, identifikuje zemi či region původu, nakladatele a vydání. Existuje ve verzi ISBN-10 a ISBN-13. Na poslední pozici každého ISBN je *kontrolní cifra*.

Příklad 4.19 (Výpočet kontrolní cifry ISBN-10)**.** *Mějme ISBN 0-552-13105-9. Kontrolní cifra ISBN-10 se počítá v modulu 11, pro případ zbytku 10 se použije znak X. Kontrolní součet je* $0 \cdot 10 + 5 \cdot 9 + 5 \cdot 8 + 2 \cdot 7 + 1 \cdot 6 + 3 \cdot 5 + 1 \cdot 4 + 0 \cdot 3 + 5 \cdot 2 + 9 \cdot 1 = 143 \,\mathrm{mod}\, 11 = 9$*. Uvedené ISBN je opravdu platné.*

Což takhle 80-85609-70-3?

Čísla bankovních účtů

Číslo bankovního účtu v ČR má tvar 123456-1234567890/1234, kde první a druhá část čísla účtu jsou chráněny proti překlepům opět algoritmem váženého ciferného součtu mod 11. Kontroluje se pouze tzv. předčíslí a vlastní číslo účtu, váhy jednotlivých číslic v předčíslí jsou zleva $10, 5, 8, 4, 2, 1$ a váhy cifer v čísle účtu jsou zleva $6, 3, 7, 9, 10, 5, 8, 4, 2, 1$. Těmito vahami se vynásobí jednotlivé číslice předčíslí, resp. čísla účtu, výsledek se sečte a určí se zbytek po dělení 11. Pokud tento zbytek vyjde 0, jedná se o korektní číslo účtu.

Příklad 4.20 (Kontrola čísla účtu)**.** *Mějme číslo účtu 19-2000145399/0800, jež pro účely kontroly přepíšeme na 000019-2000145399/0800. Kontrolní součet první části je* $0 \cdot 10 + 0 \cdot 5 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 9 \cdot 1 = 11 \,\mathrm{mod}\, 11 = 0$*. Kontrolní součet druhé části je* $2 \cdot 6 + 0 \cdot 3 + 0 \cdot 7 + 0 \cdot 9 + 1 \cdot 10 + 4 \cdot 5 + 5 \cdot 8 + 3 \cdot 4 + 9 \cdot 2 + 9 \cdot 1 = 121 \,\mathrm{mod}\, 11 = 0$.

Rodné číslo

Jednoznačný identifikátor občanů ČR a SR obsahující údaj o datumu narození, pohlaví a do roku 2004 i lokalitě porodnice.

Příklad 4.21 (Výpočet kontrolní cifry)**.** *Muž narozen 22. února 1959, rozlišující trojčíslí 177 (Zlín?). Poslední cifra rodného čísla zajišťuje dělitelnost ciferného součtu jedenácti, musí mít proto hodnotu* $590222177 \bmod 11 = 6$. *Odpovídající rodné číslo má tvar 590222/1776.*

Možná vás napadlo, jak to udělat, když na posledním místě rodného čísla může být teoreticky 11 různých číslic (tedy $0, 1, \ldots, 10$). Řešení bylo (po roce 2004 už je na podobné speciality pamatováno) takové, že místo cifry 10 se na konec rodného čísla napsala nula a takové rodné číslo neprošlo validací. Podle různých neustále se přesouvajících zdrojů z oblasti informačního systému státní správy bylo takových rodných čísel do roku 1985 vydáno cca 1000.

Jakkoliv je Wikipedie značně živelný a často zavádějící zdroj informací, v případě rodného čísla je odpovídající heslo Wikipedie [1] asi nejlepší přehledovou informací, která je o tématu dostupná.

Pseudonáhodná čísla

Pro různé druhy stochastických simulací a Monte Carlo výpočtů potřebujeme vhodný zdroj náhodných čísel z určitého rozdělení pravděpodobnosti – potřebujeme generovat sekvenci hodnot, kde hodnota prvku x_{k+1} nezávisí na žádném z prvků $x_0, x_1, \ldots, x_k$.

Takovou sekvenci v počítači vyrobit neumíme, umíme se jí ale docela věrně na počítači pro omezený počet hodnot nabpodobit pomocí generátoru pseudonáhodných čísel, jenž většinou generuje čísla z rovnoměrného rozdělení $U(0, 1)$. Ostatní rozdělení lze potom různými technikami simulovat pomocí přepočtu hodnot z rozdělení rovnoměrného.

Jednou z možností, jak pseudonáhodná čísla v $U(0, 1)$ generovat, je **lineární kongruentní generátor** *(LCG, Linear Congruence Generator).*

Příklad 4.22 (Jak funguje LCG)**.** *Uživatel zvolí* x_0 *(pevné nebo třeba odvozené od aktuálního času). Potom* $x_{k+1} = (a \cdot x_k + b) \bmod m$*, kde* a, b *a* m *jsou zvolené parametry určující kvality generátoru. Jedna z možných voleb je třeba* $a = 1664525$*,* $b = 1013904223$ *a* $m = 2^{32}$.

LCG jsou *velmi citlivé na volby parametrů.* Pokud dodržíme jisté předpoklady, generátor pracuje s periodou m, ale i to je v mnoha případech statistických výpočtů (například u vícerozměrné Monte Carlo integrace) žalostně málo.

Aritmetika velkých čísel

Registry v dnešních procesorech jsou většinou 32 nebo 64 bitové:

- největší binární číslo, s nímž počítač dokáže *pohodlně* pracovat, je tedy 2^{32} respektive 2^{64},

- největší binární číslo, jež můžeme reprezentovat v 1GB operační paměti, je $2^{1099511627776}$ ... jak rychle s ním ale budeme schopni počítat?

Jak se ale algoritmy typu RSA efektivně vypořádávají se sčítáním či násobením celých čísel v aritmetice velkých modulů (třeba 340282366920938463463374607431768211507)? Jak provádět operace s třídami čísel, která se do paměti počítače prostě nevejdou?

Literatura

[1] Rodné číslo.

KAPITOLA 5

Šifrování veřejným klíčem

doplnit úvod

5.1 Čínská věta o zbytcích

Více vzájemně ekvivalentních tvrzení z algebry a teorie čísel. Nejstarší zmínka z Číny ve 3. století našeho letopočtu.

Příklad 5.1. *Jak najít x, jenž je řešením více kongruencí najednou, například*

$$\begin{aligned} x &\equiv 2 \pmod 3, \\ x &\equiv 3 \pmod 5, \\ x &\equiv 2 \pmod 7? \end{aligned}$$

Zbytkové třídy jsou $[2]_3$, $[3]_5$ a $[2]_7$, výsledné řešení musí spadat do všech tří z nich.

Výsledkem bude opět kongruence, jejíž modul je dán násobkem tří dílčích modulů soustavy kongruencí,

$$\begin{aligned} x &= 3i + 2 = 5j + 3 = 7k + 2 \\ x &= \dots \end{aligned}$$

Zbytková třída výsledku bude tedy $3 \cdot 5 \cdot 7 = 105$.

Pokud zvolíme řešení jako lineární kombinaci třech dílčích řešení pro jednotlivé kongruence, tedy

$$x \equiv 2\kappa_1 + 3\kappa_2 + 2\kappa_3 \pmod{105}$$

stačí pro dodržení ekvivalencí v soustavě kongruencí zaručit, aby se κ_1 chovalo jako 1 ve zbytkové třídě 3 a jako 0 ve zbytkových třídách 5 a 7, a aby se κ_2

chovalo jako 1 ve zbytkové třídě 5 a jako 0 ve zbytkových třídách 3 a 7, a obdobně aby κ_3 bylo ve zbytkových třídách $[1]_7$, $[0]_3$ a $[0]_5$. Potom bude platit

$$\begin{aligned} 2\kappa_1 + 3\kappa_2 + 2\kappa_3 &\equiv 2 \pmod 3, \\ 2\kappa_1 + 3\kappa_2 + 2\kappa_3 &\equiv 3 \pmod 5, \\ 2\kappa_1 + 3\kappa_2 + 2\kappa_3 &\equiv 2 \pmod 7. \end{aligned}$$

V prvním kroku hledáme nulové a jednotkové zbytkové třídy pro kombinace původních modulů. V našem případě platí

$$\begin{aligned} \kappa_1 = 70 \equiv 0 \pmod{5 \cdot 7} \quad &\wedge \quad \kappa_1 = 70 \equiv 1 \pmod 3, \\ \kappa_2 = 21 \equiv 0 \pmod{3 \cdot 7} \quad &\wedge \quad \kappa_2 = 21 \equiv 1 \pmod 5, \\ \kappa_3 = 15 \equiv 0 \pmod{3 \cdot 5} \quad &\wedge \quad \kappa_3 = 15 \equiv 1 \pmod 7. \end{aligned}$$

Řešením dané soustavy kongruencí je v takovém případě číslo

$$\hat{x} = 2 \cdot 70 + 3 \cdot 21 + 2 \cdot 15 = 233.$$

Minimální hodnota x je dána třídou kongruence modulo $3 \cdot 5 \cdot 7 = 105$, tedy

$$x = 233 \bmod 105 = 23.$$

Vlastní tvrzení

Nechť $n_1, n_2, \ldots, n_k$ jsou navzájem nesoudělná přirozená čísla, $n_i \geq 2$ pro všechna $i = 1, \ldots, k$. Potom řešení soustavy rovnic

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

existuje a je určeno jednoznačně v modulo $n = n_1 \cdot n_2 \cdot \ldots \cdot n_k$.

Díky nesoudělnosti existuje ve třídě operací modulo n_i ke každému $N_i = n/n_i$ jeho multiplikativní inverze M_i, tedy

$$M_i \cdot N_i \equiv 1 \pmod{n_i}$$

a platí

$$x = \sum_{i=1}^{k} a_i M_i N_i.$$

Ve výše uvedeném případě se zbytkovými třídami $[2]_3$, $[3]_5$ a $[2]_7$ je

$$x = 2 \cdot 2 \cdot 35 + 3 \cdot 1 \cdot 21 + 2 \cdot 1 \cdot 15 = 233.$$

Výpočty modulo velké M lze převést na výpočty modulo menší součinitelé čísla M – zrychlení výpočtu.

Lze generalizovat pro soudělná čísla.

Význam hlavně v šifrovacích systémech.

Problém nůše s vejci

V nůši je v vajec. Pokud z ní odebíráme vejce po dvou, třech a pěti najednou, v nůši nakonec zůstane 1, 2, respektive 4 vejce. Pokud odebíráme vejce po sedmi kusech, v nůši nakonec nezůstane vejce žádné.

Jaká je nejmenší hodnota v pro niž může uvedená situace nastat?

Zbytkové třídy jsou $[1]_2$, $[2]_3$, $[4]_5$ a $[0]_7$.

Hledáme řešení soustavy

$$\begin{aligned} v &\equiv 1 \pmod 2 \\ v &\equiv 2 \pmod 3 \\ v &\equiv 4 \pmod 5 \\ v &\equiv 0 \pmod 7 \end{aligned}$$

Výsledek bude nějaká třída kongruence modulo 210.

Pro jednotlivé ekvivalence máme

i	n_i	N_i	M_i	a_i
1	2	105	1	1
2	3	70	1	2
3	5	42	3	4
4	7	30	4	0

$$\begin{aligned} v &= (1 \cdot 1 \cdot 105 + 2 \cdot 1 \cdot 70 + 4 \cdot 3 \cdot 42 + 0 \cdot 4 \cdot 30) \bmod 210 \\ &= (105 + 140 + 504 + 0) \bmod 210 = 749 \bmod 210 = 119 \end{aligned}$$

5.2 Modulární mocnění

Neefektivně lze opakovaným násobením a redukcí:

$$c = \underbrace{b\,[\,b\,[\ldots[\,b \cdot b}_{r-\text{krát}} \bmod n\,]\ldots]\bmod n\,]\bmod n$$

Jde to ale i lépe. Postup si ukážeme na následujících příkladech.

Příklad 5.2 (Mocnění opakovaným násobením)**.** *Mám-li počítat* $a \equiv 21^{41} \pmod{43}$*, mohu sice postupně vyčíslovat*

$$\begin{aligned} a_1 &= 21, \\ a_2 &= a_1 \cdot 21 \bmod 43, \\ a_3 &= a_2 \cdot 21 \bmod 43, \\ &\vdots \\ a = a_{41} &= a_{40} \cdot 21 \bmod 43, \end{aligned}$$

půjde ale o poměrně pracný postup – představte si, jak dlouho takto budete počítat kongruenci $a \equiv 97011687217^{98764321261} \pmod{225898512559}$*, a že takovýchto kongruencí potřebujete spočítat milióny.*

Vylepšení celého algoritmu spočívá v jednoduchém pozorování: Každý mocnitel r z množiny přirozených čísel můžeme reprezentovat jako součet mocnin čísla 2 a každou vysokou mocninu čísla b lze poskládat z násobků čísel $b^{(2^i)}$. Opět je asi vhodnější si celý postup ukázat na konkrétním příkladu:

Příklad 5.3 (Mocnění opakovaným kvadrátem)**.** *V našem ilustrativním případě* $a \equiv 21^{41} \pmod{43}$ *je mocnitel* $41 = 32 + 8 + 1$ *a platí tedy* $a \equiv 21^{41} \pmod{43} \equiv 21 \cdot 21^8 \cdot 21^{32} \pmod{43}$*, kde pro výpočet dvojkových mocnin čísla 21 potřebujeme pouze opakovaně umocňovat na druhou:*

$$\begin{aligned} 21^2 &\equiv (21^1)^2 \pmod{43} \equiv 11 \pmod{43}, \\ 21^4 &\equiv (21^2)^2 \pmod{43} \equiv 11^2 \pmod{43} \equiv 35 \pmod{43}, \\ 21^8 &\equiv (21^4)^2 \pmod{43} \equiv 35^2 \pmod{43} \equiv 21 \pmod{43}, \\ 21^{16} &\equiv (21^8)^2 \pmod{43} \equiv 21^2 \pmod{43} \equiv 11 \pmod{43}, \\ 21^{32} &\equiv (21^{16})^2 \pmod{43} \equiv 11^2 \pmod{43} \equiv 35 \pmod{43}. \end{aligned}$$

Výsledek obdržíme jako $a \equiv 21 \cdot 21^8 \cdot 21^{32} \pmod{43} \equiv 21 \cdot 21 \cdot 35 \pmod{43}$ *a po úpravách* $a \equiv 41 \pmod{43}$*.*

To vede na algoritmus opakovaného kvadrátu, jenž můžete shlédnout v Algoritmu 5.1.

Počet kroků nutných pro umocnění b^r opakovaným kvadrátem je $\log_2 r$ oproti r krokům potřebným pro opakované násobení.

Přibližme si celý postup, popsaný v Algoritmu 5.1, ještě jedním příkladem.

Algoritmus 5.1 Efektivní algoritmus mocnění pomocí opakovaného kvadrátu.

Require: $b \in \mathbb{Z}$, $r, n \in \mathbb{N}$
Ensure: $c \equiv b^r \pmod{n}$
  Nechť $r = \sum_{j=0}^{k} a_j \cdot 2^j, a_j \in \{0, 1\}$
  $c \leftarrow 1 + a_0 \cdot (b - 1)$; $b_0 \leftarrow b$
  for $j = 1$ **to** k **do**
    $b_j \leftarrow b_{j-1}^2 \bmod n$
    if $a_j > 0$ **then**
      $c \leftarrow c \cdot b_j \bmod n$
    end if
  end for
  return $c \equiv b^r \pmod{n}$

Příklad 5.4 (Spočtěte $c = 3^{17} \bmod 7$)**.** *Nejprve rozložíme* $r = 17 = 10001_{\mathrm{b}}$. *Je* $a_0 = 1$ *a proto prvotní hodnota* $c = b = 3$ *a* $b_0 = 3$. *Potom*

$$b_1 = 3^2 \bmod 7 = 9 \bmod 7 = 2, a_1 = 0,$$
$$b_2 = 2^2 \bmod 7 = 4 \bmod 7 = 4, a_2 = 0,$$
$$b_3 = 4^2 \bmod 7 = 16 \bmod 7 = 2, a_3 = 0,$$
$$b_4 = 2^2 \bmod 7 = 4 \bmod 7 = 4, a_4 = 1.$$

Nyní přepočteme $c = 3 \cdot 4 \bmod 7 = 12 \bmod 7 = 5$. *Další binární cifry už v* r *nejsou, výsledkem je proto* $c = 5$.

Kontrola: $3^{17} = 129140163 \bmod 7 = 5$.

5.3 Eulerova funkce

Definice 5.1 (Eulerova věta)**.** *Malou Fermatovu větu lze zobecnit na tvar*

$$a^{\phi(n)} \equiv 1 \pmod{n},$$

kde $\phi(n)$ *je tak zvaná* ***Eulerova funkce****, která udává počet přirozených čísel* $1 \leq x \leq n$*, jež jsou s* n *nesoudělná.*

Někdy $\phi(n)$ označuje názvem *totient.*

Pro prvočísla je

$$\phi(p) = p - 1,$$

pro nesoudělná x a y platí

$$\phi(x \cdot y) = \phi(x) \cdot \phi(y)$$

a proto pro prvočísla p a q také

$$\phi(p)\phi(q) = (p-1)(q-1).$$

Pro libovolné přirozené n platí také

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right).$$

Pozorování 5.2. *Zvolíme-li $n = p \cdot q$, kde p i q jsou prvočísla, bude podle Eulerovy věty*

$$a^{\phi(n)} = a^{\phi(p \cdot q)} = a^{(p-1)(q-1)} \equiv 1 \pmod{p \cdot q},$$

a tedy také

$$\left(a^{p-1}\right)^{q-1} \equiv \left(a^{q-1}\right)^{p-1} \equiv 1 \pmod{p \cdot q}.$$

a protože p a q jsou nesoudělná, tak také

$$a^{\phi(n)} \equiv 1 \pmod{p} \equiv 1 \pmod{q}.$$

5.4 Šifrování

Symetrické a asymetrické šifry

Existují dvě základní skupiny šifrovacích algoritmů:

- **Symetrické šifry** u nichž se ten samý klíč používá jak k šifrování, tak i k dešifrování zprávy. Odesílatel i příjemce musí mít k dispozici identické klíče. Příkladem je DES, 3DES, AES.
- **Asymetrické šifry** u nichž se šifruje jiným klíčem, než je klíč určený k dešifrování. Odesílatel po zašifrování již nemá možnost zprávu dešifrovat. Příkladem je RSA (PGP), GnuPG, ElGamal.

Symetrické šifry jsou při stejné délce šifrovacího klíče výrazně bezpečnější, než šifry asymetrické ...

Výměna klíčů

... ale symetrické šifrování má *základní problém*: distribuci klíčů.

Diffie a Hellman, 1976

Alice a Bob se na klíči mohou dohodnout přes nezabezpečený komunikační kanál. Je pouze třeba zajistit, aby operace, jež Alice a Bob provádějí, *nebyly výpočetně snadno invertovatelné*.

Diffieho-Hellmanova výměna klíčů

Veřejně známé prvočíslo p a $\alpha \in \{2, \ldots, p-2\}$. Oba jako klíč použijí $\alpha^{xy} \bmod p$ – Alice si vymyslí veliké $x \in \mathbb{N}$ a Bobovi pošle $\alpha^x \bmod p$, Bob pošle Alici $\alpha^y \bmod p$. Alice pak provede $(\alpha^y)^x \bmod p$, Bob obdobně.

Hodnota α se v praxi volí 2 nebo 5.

Výměna klíčů je založena na faktu, že v modulární aritmetice modulo p se velmi těžko hledá **diskrétní logaritmus** celého čísla, tedy číslo $x = \log_g(h)$[1] takové, že $g^x \equiv h \pmod p$.

Příklad 5.5 (Diskrétní logaritmus)**.** *Uvažujme kongruenci* $3^x \equiv 15 \pmod{19}$*. Jedním z možných řešení je* $x = 5$*, neboť* $3^5 \equiv 15 \pmod{19}$*, není to ale řešení jediné. Z Malé Fermatovy věty totiž plyne* $3^18 \equiv 1 \pmod{19}$*, a proto má řešená kongruence nekonečně mnoho řešení ve tvaru* $3^{5+18k} \equiv 15 \pmod{19}$ *pro* $k \in \mathbb{N}$*. Zadanou kongruenci tedy splňují všechna* x*, jež jsou řešeními kongruence* $x \equiv 5 \pmod{18}$ *a ze zveřejněné hodnoty* $3^x \bmod 19$ *lze jen s velkou náhodou určit jedno konkrétní* x*, zvolené při výměně klíče Alicí.*

Vzhledem k tomu, že $[a]_p \cdot [b]_p = [ab]_p$ platí pro Alicí přijaté Bobovo $\alpha^y \bmod p$ následující:

$$\alpha^y \bmod p \equiv [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p,$$

a po umocnění na x-tou:

$$\left([\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p\right)^x = \underbrace{[\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p \cdot [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p \cdots [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{y-\text{krát}}]_p}_{x-\text{krát}} \equiv$$

$$\equiv [\underbrace{\alpha \cdot \alpha \cdots \alpha}_{xy-\text{krát}}]_p.$$

Recipročně to platí i pro Bobem přijaté Aličino $\alpha^x \bmod p$.

Příklad výměny pro $p = 17$ *a* $\alpha = 5$

Alice si zvolí $x = 1039$.

Bob si zvolí $x = 1271$.

Po nezašifrovaném spojení pošle Alice Bobovi $5^{1039} \bmod 17 = 7$ a Bob pošle Alici $5^{1271} \bmod 17 = 10$.

Bob si spočte svůj klíč jako $7^{1271} \bmod 17 = 12$, Alice jako $10^{1039} \bmod 17 = 12$.

[1] Měli bychom ještě správně uvádět, že hledané číslo x je prvkem tzv. okruhu celých čísel modulo p, zapisujeme $x \in \mathbb{Z}/p\mathbb{Z}$

Ve skutečnosti budou p, α, x, y mnohem větší čísla (proč)?

Pro ilustraci situace útočníka si povšimněte, že platí $5^7 \equiv 5^{23} \equiv 5^{39} \equiv 5^{7+k\cdot 16} \equiv 10 \pmod{17}$ pro libovolné $k \in \mathbb{N}$ a že $1271 = 7 + 79 \cdot 16$. Stejně tak je $5^{15} \equiv 5^{31} \equiv 5^{47} \equiv 5^{15+k\cdot 16} \equiv 7 \pmod{17}$ a $1039 = 15 + 64 \cdot 16$. V našem ilustračním případě může útočník celkem lehce vyzkoušet všechny možné kombinace klíčů (bude jich jenom sedmnáct), ale v případě velkých prvočísel to už nebude praktické: Bude-li $p = 429183283$ a dokážeme-li otestovat 100 klíčů za vteřinu, bude prohledávání celého prostoru možných klíčů trvat přibližně 1192 hodin. Pro p z oblasti 64-bitových prvočísel by prohledávání celého prostoru možných klíčů rychlostí 10^6 klíčů/s mohlo trvat až 585 tisíc let (vyzkoušejte si to).

RSA (Rivest, Shamir a Adelman 1977)

V dnešní době asi nejznámějším algoritmem pro asymetrické šifrování (tedy pro šifrování veřejným klíčem) je algoritmus vyvinutý Ronem Rivestem, Adi Shamirem a Leonardem Adelmanem na MIT v roce 1977, označovaný zkratkou **RSA**. Až o dvacet let později vyšlo najevo, že britský matematik Clifford Cocks, zaměstnanec GHCQ,[2] navrhl v podstatě stejný systém už okolo roku 1973. GHCQ ovšem nemělo pro jeho vynález využití, a jako utajovaná informace zůstal 25 let archivován.

Šifra RSA vychází z předpokladu, že faktorizace součinu prvočísel p a q je časově náročná – všichni proto mohou znát šifrovací klíč e a šifrovací modul $n = p \cdot q$, ale nepomůže jim to ke zjištění dešifrovacího klíče d, založeného na p a q.

V praxi je šifrovací modul $n = p \cdot q \in \{0,1\}^{1024}$ až $\{0,1\}^{4096}$.

Poslední faktorizovaný RSA klíč je RSA-768 ($n = \{0,1\}^{768}$, 232 dekadických číslic) za necelé 3 roky na až 618 pracovních stanicích v roce 2009.

Ale pozor: Už v květnu 2007 padlo $M_{1039} = 2^{1039} - 1$ za 11 měsíců v laboratořích EPFL, Uni Bonn a NTT.

Faktorizovat 1024-bitový klíč, jenž se dnes stále běžně používá v každodenním šifrovaném provozu) by podle Kleinjunga a kolegů [2] bylo asi 1000× náročnější, než jejich faktorizace 768-bitového klíče. Vzhledem k tomu, je 768-bitový klíč je na faktorizaci několik tisíckrát složitější, než 512-bitový klíč, a že mezi faktorizacemi 512 a 768-bitového klíče uplynulo přibližně 10 let, lze očekávat, že kilobitový klíč bude považován za faktorizovatelný (a tedy prolomitelný) někdy okolo roku 2015.

[2] Anglicky *Government Communications Headquarters*. Je to britská obdoba NSA, vládní bezpečnostní a zpravodajská agentura zaměřená na ochranu datových komunikací a dešifrování zpráv.

Poslední faktorizovaný klíč je momentálně RSA-704 (212 dekadických číslic), faktorizovat jej trvalo necelý rok na několika sítích univerzitních počítačů ve Francii a Austrálii [1].

Generování veřejného a soukromého klíče

Proces tvorby veřejného a soukromého šifrovacícho klíče je poměrně jednoduchý:

1. Zvolíme nepříliš si blízká prvočísla p a q. Ideální délka každého prvočísla je v dnešní době alespoň 1024 bitů.
2. Spočteme **modul** šifrovací a dešifrovací transformace, $n = p \cdot q$. Tento modul bude mít tedy ideální cca 2048 bitů.
3. Vypočteme Eulerovu funkci pro n, $\phi(n) = (p-1)(q-1)$.
4. Zvolíme **šifrovací exponent** e takový, že $1 < e < \phi(n)$ a $\gcd(e, \phi(n)) = 1$.
5. Dopočteme **dešifrovací exponent** d tak, aby d bylo multiplikativní inverzí k e modulo $\phi(n)$, $d \cdot e \equiv 1 \pmod{\phi(n)}$.

Veřejný klíč pro zašifrování zprávy je (n, e), **soukromý klíč** pro dešifrování je (n, d).

Princip přenosu zprávy X je primitivní:

Šifrování

Po lince přenášíme šifrovaný text c, jenž vznikne jako

$$c = X^e \bmod n.$$

Dešifrování

Příjemce si z přijatého šifrovaného textu spočítá původní zprávu jako

$$X = c^d \bmod n.$$

Trik celého postupu spočívá v tom, že *z pouhé znalosti* (n, e) *nelze v rozumném čase určit* d. Nejde to z toho důvodu, že pro výpočet hodnoty d potřebujeme znát $\phi(n) = (p-1)(q-1)$, je tedy třeba nejprve faktorizovat šifrovací modul n. To je ovšem pro vhodně zvolená p a q časově velmi náročná úloha.

Důkaz RSA

Základní otázkou, kterou bychom si nyní měli položit, je: „Obdržíme dešifrováním opravdu původní text?“ Ač se to na první pohled nezdá z výše uvedených vzorců pravděpodobné, RSA vskutku funguje.

Při dešifrování $c \equiv X^e \pmod n$ máme

$$c^d \equiv (X^e)^d \equiv X^{ed} \pmod n \equiv X^{ed} \pmod{pq}.$$

Prozkoumáme vlastnosti $c^d \equiv X^{ed} \pmod p$ a $c^d \equiv X^{ed} \pmod q$ a zobecníme je na operace modulo n. ■ **Jde to i rovnou přes totient, zkoumáním jaké mociny X leží ve stejné třídě kongruence** ■

Z definice součinu ed v algoritmu RSA plyne

$$ed \equiv 1 \pmod{\phi(n)} \Rightarrow \exists g \in \mathbb{Z} : ed = 1 + g(p-1)(q-1),$$

což můžeme dále upravit na

$$ed = 1 + f(p-1)(q-1) = 1 + g(q-1) = 1 + h(p-1)$$

a tedy

$$ed \equiv 1 \pmod{\phi(n)} \equiv 1 \pmod{\phi(p)} \equiv 1 \pmod{\phi(q)}.$$

Důkaz. $ed \equiv 1 \pmod{\phi(n)} \equiv 1 \pmod{(p-1)(q-1)} \Leftrightarrow ed = 1 + g(p-1)(q-1)$ a tedy $ed = 1 + g_p(q-1)$ a $ed = 1 + g_q(p-1)$. □

Dokazujeme nadále p a q odděleně:

Pro $p \nmid X$ je podle Malé Fermatovy věty $X^{p-1} \equiv 1 \pmod p$ a tedy

$$X^{ed} = X^{1+h(p-1)} = X \cdot X^{h(p-1)} = X \cdot \left(X^{(p-1)}\right)^h \equiv X \cdot 1^h \equiv X \pmod p.$$

Pro $p \mid X$ je

$$X^{ed} \equiv 0^{ed} \pmod p \equiv X \pmod p$$

To samé platí pro q a tedy

$$X^{ed} \equiv X \pmod p$$
$$X^{ed} \equiv X \pmod q$$

Jedním z důsledků CRT je pro nesoudělná x a y ekvivalence ■ **lépe přesunout k CRT a ukázat, že řešení soustavy je vždy v mod x*y*z** ■

$$\begin{array}{rcl} a & \equiv & b \pmod x \\ a & \equiv & b \pmod y \end{array} \quad \Leftrightarrow \quad a \equiv b \pmod{xy}. \tag{5.1}$$

Proto také z

$$X^{ed} \equiv X \pmod{p}$$
$$X^{ed} \equiv X \pmod{q}$$

plyne

$$X^{ed} \equiv X \pmod{pq}.$$

Důkaz. Platnost ekvivalence (5.1) lze dokázat za předpokladu nesoudělnosti x a y tak, že uvážíme nejprve

$$a = k_1 x + b, \qquad a = k_2 y + b,$$
$$a - b = k_1 x, \qquad a - b = k_2 y.$$

Čísla x a y musí dělit výraz $a - b$ beze zbytku,

$$a - b = k_1 x = k_2 y,$$

a protože jde o nesoudělná čísla, musí zároveň platit, že $y \mid k_1$ a $x \mid k_2$,

$$k_1 x = k_2 y = (\kappa y)x = (\kappa x)y.$$

Můžeme tedy psát

$$a = \kappa \cdot xy + b$$

a tedy také

$$a \equiv b \pmod{xy}$$

□

CRT-RSA

V úvodu přednášky zmíněná Čínská věta o zbytcích má velmi zajímavé využití právě při výpočtech dešifrovací části šifry RSA.

Příklad 5.6 (Dešifrování RSA)**.** *Jak modul n, tak i dešifrovací exponent d jsou hodně velká čísla, a proces dešifrování*

$$X \equiv c^d \pmod{n}$$

trvá dlouho.

K urychlení dešifrovací transformace lze použít rozklad na výpočet s menšími moduly pomocí Čínské věty o zbytcích.

Pro $n = pq$ použijme již jednou provedený trik

$$X \equiv X_p \pmod{p} \equiv c^{d_p} \pmod{p},$$
$$X \equiv X_q \pmod{q} \equiv c^{d_q} \pmod{q},$$

přičemž

$$d_p \equiv d \pmod{\phi(p)} \equiv d \pmod{p-1} \Leftrightarrow d = d_p + j \cdot (p-1),$$
$$d_q \equiv d \pmod{\phi(q)} \equiv d \pmod{q-1} \Leftrightarrow d = d_q + k \cdot (q-1),$$

a tedy

$$c^d \equiv c^{d_p + j(p-1)} \pmod{p} \equiv c^{d_p} 1^j \pmod{p} \equiv c^{d_p} \pmod{p},$$
$$c^d \equiv c^{d_q + k(q-1)} \pmod{q} \equiv c^{d_q} 1^k \pmod{q} \equiv c^{d_q} \pmod{q}.$$

Zpráva X je tedy řešením soustavy dvou kongruencí sestavených pro c:

$$X \equiv c^{d_p} \pmod{p},$$
$$X \equiv c^{d_q} \pmod{q}.$$

Řešením je

$$X = \left[c^{d_p} M_p q + c^{d_q} M_q p \right] \bmod pq,$$

kde $M_p = q^{-1} \bmod p$ a $M_q = p^{-1} \bmod q$.

V tomto případě lze většinu hodnot předpočítat. Dešifrovací klíč je potom šestice $(p, q, d_p, d_q, M_p, M_q)$.

Prolomení RSA při nevhodné volbě p a q

Pokud zvolíme p a q nevhodně (blízko sebe, příliš malá, atd.), útočník využije znalosti (n, e):

1. Faktorizuje n na p a q.
2. Vypočte Eulerovu funkci pro n, $\phi(n) = (p-1)(q-1)$.
3. Dopočte **dešifrovací exponent** d tak, aby $d \cdot e \equiv 1 \pmod{\phi(n)}$.

Náš **soukromý klíč** pro dešifrování (n, d) v ten okamžik zná i útočník a může moje zprávy dešifrovat.

Příklad 5.7 (Příklad šifrování a dešifrování RSA)**.** *Chceme pomocí RSA zašifrovat a dešifrovat zprávu* ABORT *v anglické abecedě kódované podle Tabulky 5.1, přičemž pro generování RSA modulu použijeme* $p = 43$ *a* $q = 31$.

Nejprve vypočteme RSA modul a jeho totient,

$$n = p \cdot q = 1333,$$
$$\phi(n) = \phi(1333) = 42 \cdot 30 = 1260.$$

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Tabulka 5.1: Konverzní tabulka pro převod znaků anglické abecedy do 26 zbytkových tříd

Nyní musíme zvolit šifrovací exponent e tak, aby $1 < e < 1260$ *a zároveň aby exponent nebyl soudělný s 1260 (připomeňme si, že v opačném případě by nebyl exponent invertovatelný). Tato volba je na nás, nejčastěji se setkáváme s nízkými exponenty e – pro náš příklad si zvolíme*

$$e = 13,$$
$$d \equiv e^{-1} \pmod{\phi(n)} \equiv 13^{-1} \pmod{1260} \equiv 1153 \pmod{1260}.$$

Naši zprávu můžeme nyní například rozsekáme na bloky tak, aby celková bitová délka bloku byla kratší, než počet bitů v šifrovacím modulu n (ten má 11 bitů, naše reprezentace znaků anglické abecedy má 5 bitů na znak, do jednoho bloku tedy vložíme vedle sebe dva znaky).

text	*A*	*B*	*O*	*R*	*T*	
třída	*0*	*1*	*14*	*17*	*19*	
binárně	*00000*	*00001*	*01110*	*10001*	*10011*	
bloky	*00000 00001*		*01110 10001*		*10011 00000*	
číselně	*1*		*465*		*608*	

Další podobné příklady lze nalézt na různých interentových stránkách, například [3].

5.5 Příklady

V literatuře lze nalézt mnoho různých postupů šifrování veřejným klíčem. Nejznámější metody jsou:

- Diffie-Hellman
- DSS (Digital Signature Standard), which incorporates the Digital Signature Algorithm
- ElGamal

- Microsoft CAPI
- RSA encryption algorithm (PKCS)

Příkladem nevhodně navrženého algoritmu asymetrického šifrování je například vše, co vychází z Merkle-Hellmanových algoritmů balení batohu.

Asymetrické šifry se používají i v mnoha internetových protokolech, například:

- GPG (GNU Privacy Guard, jde o oimplementaci OpenPGP)
- PGP
- SSH
- SSL (Secure Socket Layer) používaný k zabezepečení protokolů HTTP, SMTP, POP, či IMAP. Nyní je implementován jako součást IETF standardu TLS.

Literatura

[1] S. Bai, E. Thomé, and P. Zimmermann. Factorisation of rsa-704 with cado-nfs, 2012.

[2] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit rsa modulus, 2010.

[3] Rsa (algorithm) – a working example.

KAPITOLA 6

Opakování teorie grafů

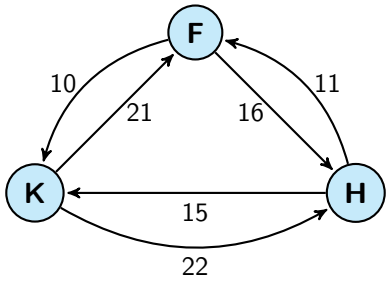
6.1 Úvod do teorie grafů

Teorie grafů je matematická a informatická disciplína zabývající se studiem matematických struktur popisujících párové reakce mezi objekty – obecných grafů.Grafy jsou jednou z klíčových oblastí, kterou studuje diskrétní matematika. K oblibě grafů v inženýrských disciplínách přispívá široká oblast možných aplikací: Obecné grafy lze použít k modelování mnoha typů vztahů a dynamických jevů ve fyzice, biologii, sociálních vědách či informačních systémech.

Některé aplikace grafů jsou zcela zjevné [1]: daný problém „ze života“ nejprve přeformulujeme na **grafovou úlohu**, tedy tak, aby celý problém popisoval nějaký typ obecného grafu (viz například Obrázek 6.1) a nějakou základní operaci na tomto grafu – o tom, jaké známe základní typy obecných grafů a jaké úlohy na nich typicky řešíme, si povíme později. Příslušný graf, modelující konkrétní situaci, potom nakreslíme nebo zadáme vhodným způsobem do počítače, grafovou úlohu nějakým známým postupem (jenž v tomto kontextu budeme nazývat **grafovým algoritmem**) vyřešíme a získané řešení opět z řeči grafů přeložíme zpět do běžné mluvy. Často jsou však aplikace grafů skryté: nikde se žádný graf explicitně nepoužije, graf zůstává skrytý v pozadí, ale použitý postup řešení má svou paralelu ve světě grafů a lze jej pomocí grafů zdůvodnit. ■ **Nějaký příklad?** ■

Umění aplikovat grafy zahrnuje dvě dílčí dovednosti [1]:

1. Musíme být schopni zkonstruovat graf, jenž modeluje danou situaci (tedy takový graf, na němž jsou zachyceny vztahy, o něž se zajímáme). Někdy to je zcela triviální (například při formulaci rozhodovací úlohy, jakým způsobem se nejrychleji přemístit z mé pražské kanceláře na výuku v Děčíně), jindy to vyžaduje značné úsilí. ■ **Příklad?** ■



Obrázek 6.1: Graf popisující možné cesty MHD mezi jednotlivými budovami ČVUT FD: **F** označuje budovu Na Florenci, **H** Horskou a **K** Konvikt. Ohodnocení hran je nejrychlejší cesta v kombinaci *pěšky* a *MHD* v minutách.

2. Musíme být schopni řešit alespoň základní grafové úlohy a musíme vědět, které z nich jsou snadno a rychle řešitelné a které jsou naopak komplikované a výpočetně složité. S úspěchem můžeme využít i schopnost převést nějakou neznámou grafovou úlohu na jinou, kterou již dovedeme vyřešit (toto umění převádět úlohy má mnoho společného s uměním najít vhodný grafový model).

Následující text je stručným výtahem z publikací pana Hliněného [2], pana Demela [1] a pánů Matouška a Nešetřila [3]. První text je spíše prakticky zaměřen, poslední dva texty považuji základní českou literaturu o grafech a doporučuji je ke studiu v případě, že se o grafech a grafových algoritmech potřebujete dozvědět více.

Základní definice

[

Základní poučka na úvod] Základní poučka na úvod zní: *Neplette si obecný graf s grafem funkce!*

Obecný graf se skládá z **vrcholů** a **hran**. Hrana vždy spojuje dva vrcholy a je buď **orientovaná**, nebo **neorientovaná**. U hran orientovaných rozlišujeme počáteční a koncový vrchol a říkáme, že hrana vede z počátečního do koncového vrcholu. Neorientované hrany chápeme jako symetrické spojení dvou vrcholů.

Pokud hrana spojuje nějaký vrchol se sebou samým, nazýváme ji **smyčkou**.

Orientovaný graf má všechny hrany orientované, **neorientovaný graf** má všechny hrany neorientované. Teoreticky existují i grafy smíšené, v nichž se vyskytují oba druhy hran, těmi se ale nebudeme zabývat.

Zkusme to nyní popsat formálněji.

Definice 6.1 (Neorientovaný graf)**.** *Neorientovaný graf je dvojice* $G = (\mathcal{V}, \mathcal{E})$ *tvořená neprázdnou konečnou množinou* $\mathcal{V}$*, jejíž prvky nazýváme vrcholy a konečnou množinou* $\mathcal{E}$*, jejíž prvky nazýváme orientovanými hranami. Každá hrana množiny* $\mathcal{E}$ *je přitom dvouprvkovou podmnožinou množiny* $\mathcal{V}$*,* $\forall e_{ij} \in \mathcal{E}$: $e_{ij} = \{v_i, v_j\}, v_i, v_j \in \mathcal{V}$.

Hranu mezi vrcholy i a j bývá zvykem značit $\{i, j\}$ nebo krátce e_{ij}. Množinu vrcholů grafu G označujeme $V(G)$, analogicky $E(G)$ označuje množinu hran tohoto grafu.

O hraně e_{xy} říkáme, že vede z vrcholu x do vrcholu y a také, že spojuje vrcholy x a y. O vrcholech x a y pak říkáme, že jsou incidentní (nebo že incidují) s hranou e_{xy} a také naopak můžeme říci, že hrana e_{xy} je incidentní s vrcholy x a y. Oba vrcholy x a y také souhrnně nazýváme krajními vrcholy hrany e_{xy}. Vrchol, který není incidentní s žádnou hranou, nazýváme izolovaným vrcholem.

Počet hran, incidentních s nějakým vrcholem $v \in V(G)$ nazýváme **stupněm vrcholu**. Izolovaný vrchol má stupeň 0.

Jestliže hrana spojuje vrchol se sebou samým (tedy například hrana e_{xx}), nazýváme ji (orientovanou) smyčkou.

Obecně je možné, aby několik hran mělo stejné počáteční a koncové vrcholy, tedy aby pro různé hrany (pozor, musíme změnit značení hran) $e_l \not\equiv e_2$ platilo $e_1 = \{x, y\}$ a zároveň $e_2 = \{x, y\}$. O takových hranách říkáme, že jsou rovnoběžné nebo též násobné. Množina hran grafu může být i prázdná.

Podgraf

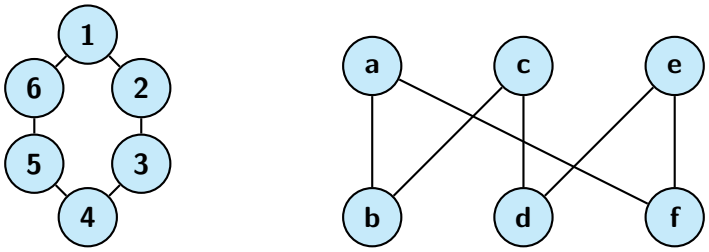
Každý graf (až na ten zcela triviální s jedním vrcholem a prázdnou množinou hran) lze rozdělit na podgrafy. **Podgraf** je, jednoduše řečeno, část grafu. Avšak tato slova musíme definovat poněkud přesněji, abychom předešli situacím, kdy by nám zbyly pouze hrany bez odpovídajících vrcholů.

Definice 6.2 (Podgraf)**.** *Podgrafem grafu* G *rozumíme libovolný graf* H *na podmnožině vrcholů* $V(H) \subseteq V(G)$*, jenž má za hrany libovolnou podmnožinu hran grafu* G *majících oba vrcholy ve* $V(H)$*. Píšeme* $H \subseteq G$*, tj. stejně jako množinová inkluze (ale význam je trochu jiný).*

Definice 6.3 (Indukovaný podgraf)**.** *Indukovaným podgrafem grafu* G *rozumíme takový podgraf* $H \subseteq G$*, kde podmnožina hran* $E(H)$ *zahrnuje všechny původní hrany mezi vrcholy z* $V(H)$*. Někdy se mu také říká plný podgraf.*

Isomorfismus

Co když vezmeme nějaký graf (třeba ten na Obrázku 6.2) a nakreslíme jej jednou tak, podruhé zase jinak — jedná se o tentýž graf nebo ne? Přísně



Obrázek 6.2: Oba grafy na obrázku popisují to samé – jsou izomorfní.

formálně řečeno, každé nakreslení jistého grafu, třeba toho na Obrázku 6.2, je jiným grafem. ale přitom bychom rádi řekli, že různá nakreslení téhož grafu jsou ekvivalentní – už jen proto, že grafy mají modelovat vztahy mezi dvojicemi objektů, ale tyto vztahy přece vůbec nezávisí na tom, jak si grafy nakreslíme. Pro tuto stejnost grafů se vžil pojem **isomorfní grafy**.

Pro správné pochopení a využití teorie grafů je tedy třeba nejprve dobře chápat pojem isomorfismu grafů.

Definice 6.4 (Izomorfismus)**.** *Izomorfismus grafů G a H je bijektivní (vzájemně jednoznačné) zobrazení $f : V(G) \to V(H)$, pro které platí, že každá dvojice vrcholů $(x, y) \in E(G)$ je spojená hranou v G právě tehdy, když je dvojice $f(u), f(v) \in V(H)$ spojená hranou v H.*

Grafy G a H jsou izomorfní, pokud mezi nimi existuje izomorfismus. Značíme $G \simeq H$. Izomorfní grafy mají stejný počet vrcholů i hran. Vrcholu stupně k lze izomorfismem přiřadit pouze vrchol stejného stupně k. Dvojici sousedních vrcholů může být izomorfismem přiřazena opět jen dvojice sousedních vrcholů.

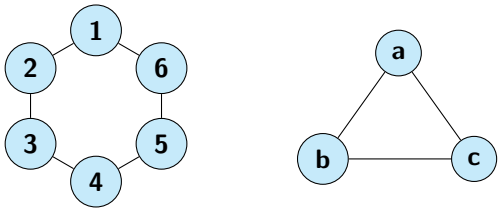
Zvláštní typy grafů

Některé často se vyskytující typy grafů je zvykem nazývat specifickými názvy. Nejdůležitější jsou patrně ty následující:

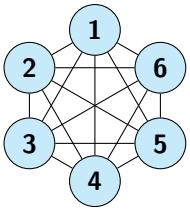
Definice 6.5 (Kružnice)**.** *Kružnice C_n délky n je graf, jenž má n vrcholů spojených do jednoho cyklu n hranami, $n \geq 3$. Podgrafu $H \subseteq G$, který je isomorfní nějaké kružnici, říkáme kružnice v G.*

Kružnici délky 3 říkáme trojúhelník.

Definice 6.6 (Cesta)**.** *Cesta P_n délky n má $n+1$ vrcholů spojených za sebou n hranami. Podgrafu $H \subseteq G$, který je izomorfní nějaké cestě, říkáme cesta v G.*



Obrázek 6.3: Kružnice a trojúhelník



Obrázek 6.4: Úplný graf, jenž vznikl doplněním hrad do kružnice na Obrázku 6.3

Definice 6.7 (Úplný graf)**.** *Úplný graf K_n má $n \geq 2$ vrcholů, všechny navzájem pospojované.*[1] *Grafy P_1 (cesta délky 1) a C_3 (trojúhelník) jsou zároveň úplnými grafy.*

Definice 6.8 (Klika)**.** *Podgrafu $H \subseteq G$, který je isomorfní nějakému úplnému grafu, říkáme klika v G.*

Někdy se za kliku považuje pouze takový úplný podgraf, který je maximální vzhledem k inkluzi.

Orientované grafy

V některých případech (například u toků v sítích) potřebujeme u každé hrany vyjádřit její směr. To vede na definici orientovaného grafu, ve kterém hrany jsou uspořádané dvojice vrcholů. V obrázcích kreslíme orientované hrany se šipkami.

Definice 6.9 (Orientovaný graf)**.** *Orientovaný graf je uspořádaná dvojice $D = (\mathcal{V}, \mathcal{A})$, kde $\mathcal{A} \subset \mathcal{V} \times \mathcal{V}$ je množina orientovaných hran mezi vrcholy grafu.*

Orientované grafy odpovídají relacím, které nemusí být symetrické: Hrana $\{x, y\}$ v orientovaném grafu D začíná ve vrcholu x a končí ve vrcholu v. Opačná hrana $\{y, x\}$ není totožná s hranou $\{x, y\}$.

[1] Takový graf má celkem $\binom{n}{2}$ hran.

Nejprve bychom si měli přesně ujasnit, jak se pohybujeme grafem, tedy co je vlastně procházkou v grafu. Tento pojem by měl postihnout základní věc, že v grafu procházíme hranami vždy z vrcholu do sousedního vrcholu, a přitom ponechat dostatek volnosti pro vracení se a zacyklení procházek.

Definice 6.10 (Sled)**.** *Sledem délky n v grafu G rozumíme posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, ve které vždy hrana e_i má koncové vrcholy v_{i-1}, v_i, tedy $e_i = \{v_{i-1}, v_i\}$.*

Sled je vlastně procházka po hranách grafu z vrcholu v_x do vrcholu v_y. Příkladem sledu může být průchod IP paketu internetem (včetně cyklení).

Věta 6.11. *Pokud mezi dvěma vrcholy grafu G existuje sled, pak mezi nimi existuje cesta.*

Důkaz. Nechť $u = v_0, e_1, v_1, \ldots, e_n, v_n = v$ je sled délky n mezi vrcholy u a v grafu G. Začneme budovat nový sled W z vrcholu $w_0 = u$, který už bude cestou: Předpokládejme, že nový sled W už obsahuje nějakou sekvenci vrcholů a hran, $w_0, e_1, w1, \ldots, w_i$ (na začátku pro $i = 0$ jde pouze o vrchol w_0 bez hran), kde $w_i = v_j$ pro některé $j \in \{0, 1, \ldots, n\}$. Najdeme největší index $k \geq j$ takový, že $v_k = v_j = w_i$, a sled W pokračujeme krokem ...,$w_i = v_j = v_k, e_{k+1}, w_{i+1} = v_{k+1}$. Zbývá dokázat, že nový vrchol $w_{i+1} = v_{k+1}$ se ve sledu W neopakuje. Pokud by tomu ale tak bylo $w_l = w_{i+1}$, $l \leq i$, pak bychom na se na vrchol w_{i+1} dostali už dříve z vrcholu w_l, což je v rozporu z naším předpokladem, že jde o nový vrchol. Budování sledu ukončíme v okamžiku, kdy $w_i = v$. □

Definice 6.12 (Souvislý graf)**.** *Graf G je souvislý pokud je G tvořený nejvýše jednou komponentou souvislosti, tedy pokud každé dva vrcholy G jsou spojené cestou.*

6.2 Implementace grafů

Mějme jednoduchý graf G na n vrcholech a značme vrcholy jednoduše čísly $V(G) = \{0, 1, \ldots, n-1\}$. Pro počítačovou implementaci grafu G pomocí *statických datových struktur* se nabízejí dva základní způsoby:

- **Maticí sousednosti $\mathbf{G}_{n\times n}$** ve které $\mathbf{G}_{ij} = 1$ znamená hranu mezi vrcholy i a j. Matici sousednosti implementujeme jako dvourozměrné pole binárních (nebo celočíselných) hodnot.
- **Výčtem sousedů $\mathbf{S}_{n\times m}$**, kde počet sloupců m je dán nejvypšším počtem sousedících vrcholů grafu C. Prvky $\mathbf{S}_{ij}$ v tomto pojetí udávají j-tý sousední vrchol vrcholu i.

Obrázek

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{S} = \begin{pmatrix} 0 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & -1 & -1 \end{pmatrix}$$

Eulerovské grafy

Snad nejstarší výsledek vůbec v oblasti teorie grafů pochází od Leonharda Eulera, jenbž tak de facto položil základ celému oboru. Jedná se o slavný *problém sedmi mostů v Královci* (původně Königsbergu, dnešním ruském Kaliningradě). Toto pruské město leží na řece Pregole, která vytváří dva ostrovy. Ostrovy byly s ostatním městem spojeny sedmi mosty, vyznačenými na Obrázku 6.5.

O jaký problém se tehdy jednalo? Městští radní chtěli vědět, zda mohou suchou nohou přejít po každém ze sedmi mostů právě jednou. Euler dokázal, že tomu tak není, a aby důkaz mohl formulovat, celý problém převedl do abstraktní roviny – z mostů vytvořil hrany a z ostrovů a pevniny uzly tehdy neznámé matematické struktury, dnes nazývané graf. Eulerovo pozorování, že základem problému je počet mostů a umístění jejich koncových bodů bez toho, abychom se museli zajímat o přesnou geometrickou polohu, je předzvěstí nástupu dalšího nového matematického oboru – topologie.

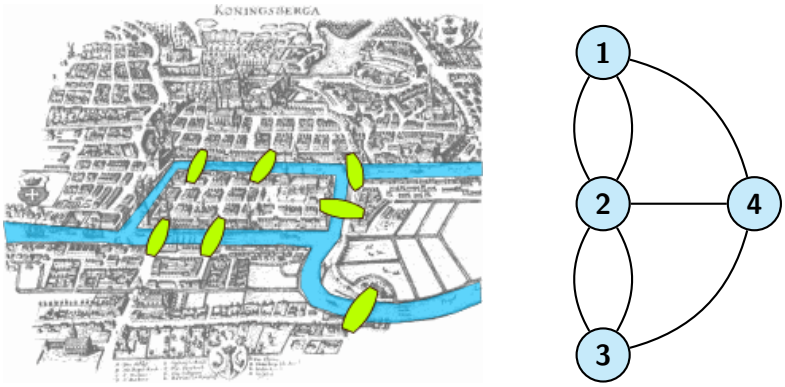
Rozbor problému mostů v Královci vedl Eulera k následující definici a odpovědi.

Definice 6.13 (Otevřený a uzavřený tah)**.** *Tah je sled v grafu bez opakování hran. Uzavřený tah je tahem, který končí ve vrcholu, ve kterém začal. Otevřený tah je tahem, který končí v jiném vrcholu, než ve kterém začal.*

Nejstarší výsledek teorie grafů od Leonarda Eulera poté zní:

Věta 6.14 (Eulerovský tah)**.** *Graf G lze nakreslit jedním uzavřeným tahem právě tehdy, když G je souvislý a všechny vrcholy v G jsou sudého stupně.*

Důsledkem této věty je také zjištění, že graf G lze nakreslit jedním otevřeným tahem právě tehdy, když G je souvislý a všechny vrcholy v G až na dva jsou sudého stupně.



Obrázek 6.5: Sedm mostů v Královci. Upravená dobová rytina. Převzato z Wikipedia Commons [4] (vlevo). Odpovídající graf (vpravo)

6.3 Základní grafové úlohy

Procházení grafem

Zcela základní grafovou úlohou je úloha, kdy chceme postupně navštívit všechny uzly grafu a nějakým blíže nespecifikovaným způsobem aktualizovat či ověřit ohodnocení hran či uzlů. Tuto úlohu nazýváme úlohou **procházení grafem**.

Při procházení grafem z vrcholu v_i si máme dvě možnosti, jak postupovat:

- **procházení do hloubky** -– před zpracováním sousedních uzlů v_j navštívíme všechny jejich nenavštívené potomky, projdeme tedy nejprve co nejhlouběji to v daném grafu jde a pak se postupně vracíme zpět,
- **procházení do šířky** – zpracujeme nejdříve všechny sousedních uzly v_j a až poté pokračujeme s jejich potomky, procházíme tedy nejprve co nejširší množinu bezprostředních sousedů.

Hledání nejkratší cesty

Příklad 6.1 (Nejkratší cesta v grafu)**.** *Najděte takovou cestu C z vrcholu u do v v grafu $G(V, E, w)$, jejíž ohodnocení je minimální možné.*

Pro nalezení nejkratší (vážené) cesty mezi dvěma vrcholy kladně váženého grafu se používá tradiční Dijkstrův algoritmus či jeho vhodná vylepšení (A*). Takové algoritmy se například používají při vyhledávání vlakových spojení. Pravděpodobně se i vy někdy dostanete do situace, kdy budete nejkratší cestu hledat, proto si popsaný algoritmus včetně jeho vylepšení A* zapamatujte.

Dijkstrův algoritmus

Dijkstrův algoritmus, popsaný v Algoritmu 6.1, je variantou na procházení grafu do šířky (jak uvidíme, nejde ovšem o čisté procházení do šířky), kdy u každému vrcholu grafu ještě přiřadíme proměnnou, udávající vzdálenost od výchozího bodu cesty (tedy délku nejkratšího sledu, kterým jsme se do tohoto vrcholu zatím dostali). V dalším kroku algoritmu procházíme úschovnu nalezených nových vrcholů a z ní vždy vybíráme vrchol s nejmenší vzdáleností k nějakému z již nalezených vrcholů (do takového vrcholu se žádnou kratší cestou už dostat nelze, všechny ostatní cesty budou delší). Na konci zpracování tyto proměnné vzdálenosti udávají správně nejkratší vzdálenosti z počátečního vrcholu do všech ostatních vrcholů.

Poznamenejme, že pokud necháme tento algoritmus proběhnout až do zpracování všech vrcholů, získáme ve vzdal[i] nejkratší vzdálenosti z počátečního vrcholu do všech ostatních vrcholů. Všimněme si dále, že Algoritmus 6.1 počítá stejně dobře nejkratší cestu i v orientovaném grafu.

Celkový počet kroků Dijkstrova algoritmu nutný k nalezení nejkratší cesty z uzlu a do uzlu b je přibližně $\mathcal{O}(|V(G)|^2)$, počet kroků tedy roste kvadraticky s počtem vrcholů grafu. Při efektivnější implementaci úschovny nezpracovaných vrcholů (můžeme například použít haldu indexovanou hodnotou vzdálenosti) lze na řídkých grafech, s nimiž často pracujeme, dosáhnout i mnohem rychlejšího běhu.[2]

Algoritmus A*

Algoritmus A* je rozšíření původního Dijkstrova algoritmu, postavené na heuristickém výběru následujícího vrcholu. Při *vhodně* zvolené heuristice je A* algoritmus výrazně rychlejší, než Dijkstrův algoritmus.

Nechť heuristika $h(x)$ udává libovolný dolní odhad vzdálenosti z vrcholu x do cíle v. Každá hrana $e_{xy} = \{x, y\}$ grafu $G(V, E, w)$ dostane nové délkové ohodnocení $w'(e_{xy}) = w(e_{xy}) + h(y) - h(x)$. Přípustná je taková heuristika, kde všechna upravená ohodnocení jsou nezáporná, neboli $w(e_{xy}) \geq h(x) - h(y)$. Všimněte si, že vzhledem k tomu, že se hodnoty $h(x)$ a $h(y)$ v obecném případě liší,A* algoritmus každý graf implicitně převádí na graf orientovaný – $w'(e_{xy}) \neq w'(e_{yx})$.

Pro použití při navigaci v mapě[3] může heuristika $h(x)$ udávat například Euklidovskou vzdálenost bodů x a v. Taková heuristika je podle trojúhelníkové nerovnosti vždy přípustná.

Použijeme-li na graf $G(V, E, w')$ s takto upraveným ohodnocením hran původní Dijkstrův algoritms, bude tento algoritmus silně preferovat hrany ve-

[2] Udává se čas téměř úměrný počtu hran prohledávaného grafu.
[3] Velmi častá aplikace A* algoritmu

Algoritmus 6.1 Dijkstrův pro nejkratší cestu v grafu. Tento algoritmus nalezne nejkratší cestu mezi vrcholy x a y kladně váženého grafu G, daného seznamem sousedních vrcholů.

Require: $G(V, E, w)$ na n vrcholech popsaný seznamem sousedů $s_{i,j}$ a délek hran $d_{i,j}$
Require: Počáteční uzel $x \in V(G)$, koncový uzel $y \in V(G)$
Ensure: Nejkratší cesta z x do y
  for $i = 1$ **to** n **do**
    $c_i \leftarrow \infty$ {zatím nalezená délka nejkratší cesty do tohoto uzlu}
    $z_i \leftarrow$ **false** {vrchol nebyl zatím zpracován}
  end for
  $c_x \leftarrow 0$ {výchozí uzel bude označen jako zpracovaný v prvním cyklu}
  while $z_y \neq$ **true do**
    $j \leftarrow n$
    for $i = 1$ **to** $n - 1$ **do**
      if $z_i =$ **false and** $c_i < c_j$ **then**
        $j \leftarrow i$
      end if
    end for{zde jsme našli nejbližší nezpracovaný vrchol j, ten teď zpracujeme}
    if c_j **is** ∞ **then**
      return Mezi vrcholy x a y není cesta.
    end if
    $z_j \leftarrow$ **true** {označíme jako zpracovaný}
    for $k = 1$ **to** $\|s_j\|$ **do** {projdeme všechny sousedy k uzlu j}
      if $c_j + d_{j,k} < c_{s_{j,k}}$ **then** {a pokud je cesta z x přes j kratší}
        $a_{s_{j,k}} \leftarrow= j$ {zapamatujeme si to}
        $c_{s_{j,k}} \leftarrow c_j + d_{j,k}$ {a uložíme vzdálenost od x}
      end if
    end for{zpracovali jsme všechny sousedy uzlu j}
  end while
  return Cesta délky c_y, uložená v poli a.

doucí ve směru k cíli v – ohodnocení takových hran bude totiž velmi nízké, zatímco ohodnocení hran vedoucích od cílového vrcholu směrem k počátečnímu vrcholu se téměř zdvojnásobí. Výsledkem bude výrazně menší počet prohledávaných vrcholů grafu (do nalezení cesty do cíle v) a také menší potřebná velikost úschovny vrcholů.

Kostry grafů

Kromě stromů samotných se zabýváme i stromy, které jsou obsaženy jako podgrafy ve větších grafech.

Definice 6.15 (Kostra grafu)**.** *Kostrou souvislého grafu G je podgraf v G, který je sám stromem a obsahuje všechny vrcholy grafu G.*

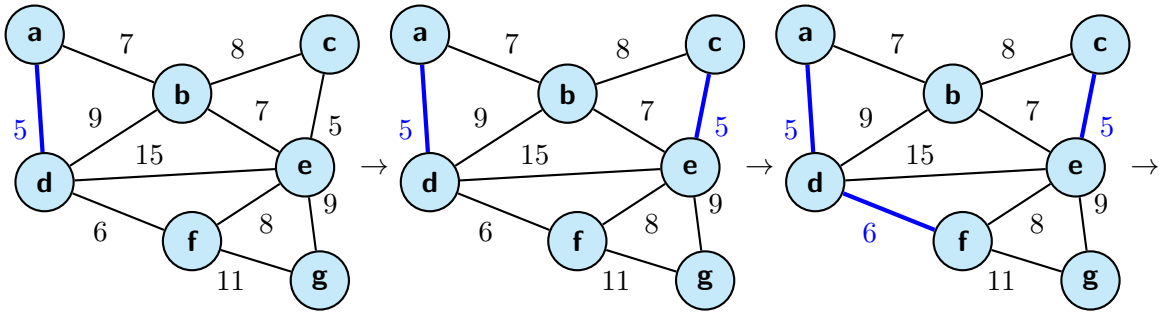
Příklad 6.2 (Problém minimální kostry grafu)**.** *Je dán (souvislý) ohodnocený graf $G = (V, E, w)$ s nezáporným ohodnocením hran w. Otázkou je najít takovou kostru T v G, jež má nejmenší možné celkové ohodnocení.*

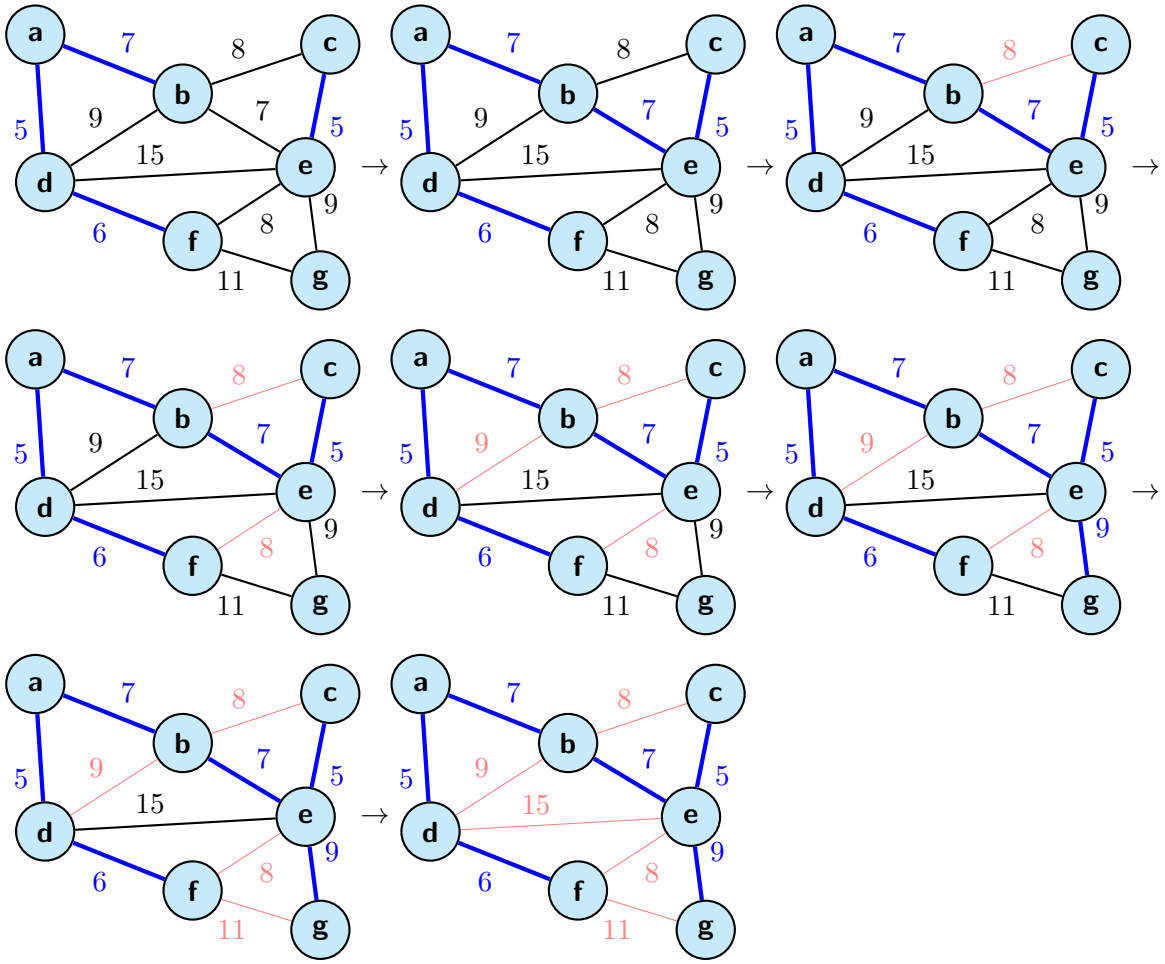
Kruskalův algoritmus

Algoritmus 6.2 Kruskalův „hladový“ algoritmus hledání minimální kostry grafu

Require: $G(V, E, w)$ s n hranami, souvislý, $\forall e_i : w(e_i) \geq 0$
Ensure: minimální kostra grafu T
  seřadíme hrany grafu G vzestupně podle jejich ohodnocení {bude $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_n)$}
  $T = \{\}$ {kostra je prázdná}
  for $i = 1$ **to** n **do**
    if $T \cup \{e_i\}$ nevytváří kružnici **then**
      $T \leftarrow T \cup \{e_i\}$
    end if
  end for
  množina T obsahuje hrany minimální kostry

Ukážeme si nyní, jak tento algoritmus funguje:





Prime-Jarníkův algoritmus

Algoritmus je velmi podobný Kruskalovu, hrany ale na začátku neseřazujeme, kostru začneme vytvářet z jednoho náhodně vybraného vrcholu a v každém kroku přidáme nejmenší z hran, které vedou z již vytvořeného podstromu do zbytku grafu.

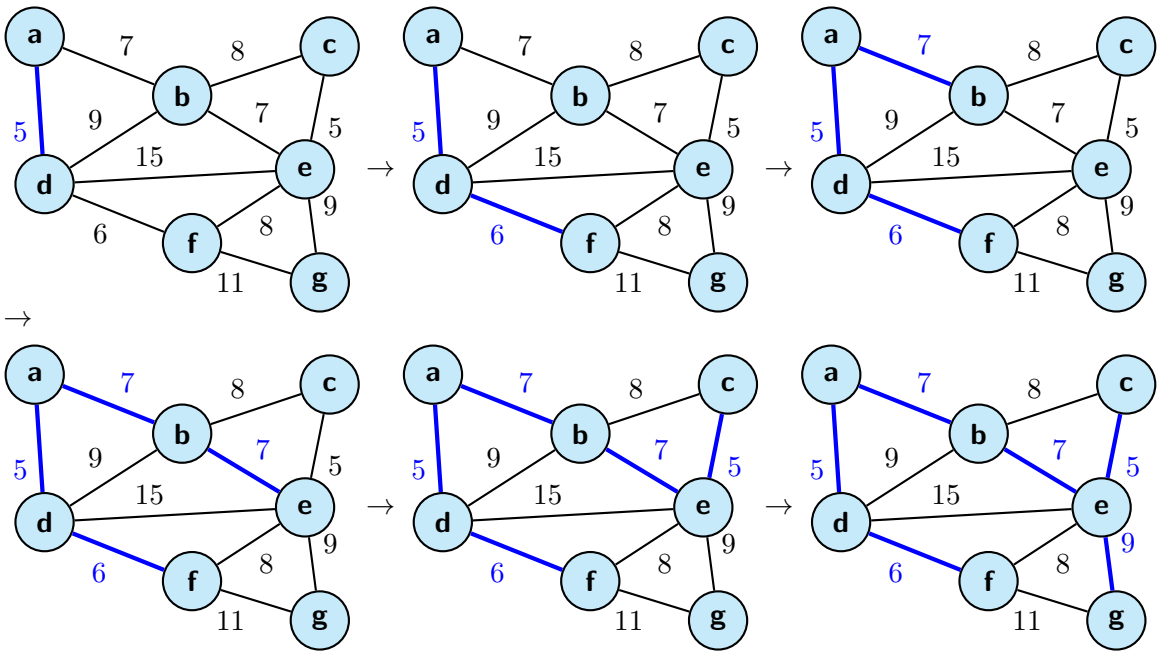
Tento algoritmus je velmi vhodný pro praktické výpočty a je dodnes široce používaný. Málokdo ve světě však donedávna věděl, že pochází od známého českého matematika Vojtěcha Jarníka – původní Jarníkova práce byla psána česky a ve světové literatuře se tak algoritmus obvykle připisuje Američanu Primovi, jenž jej nezávisle objevil až skoro 30 let po Jarníkovi.

Algoritmus 6.3 Prime-Jarníkův algoritmus hledání minimální kostry grafu

Require: $G(V, E, w)$ na n vrcholech popsaný seznamem sousedů $s_{i,j}$ a délek hran $d_{i,j}$

Ensure: Minimální kostra H

```
W ← {x}, kde x je libovolný prvek z V
F ← {} je prázdná množina hran kostry
while W ≠ V do
  d_min ← ∞
  for all x ∈ W do
    for all y ∈ (V \ W) do
      if d_{x,y} < d_min then
        d_min ← d_{x,y}
        z ← y
        φ ← {x, y} {zapamatujeme si nejbližší prvek ke komponentě}
      end if
    end for
  end for
  W ← W ∪ {z}
  F ← F ∪ {φ}
end while
return Minimální kostra H(W, F, ω).
```



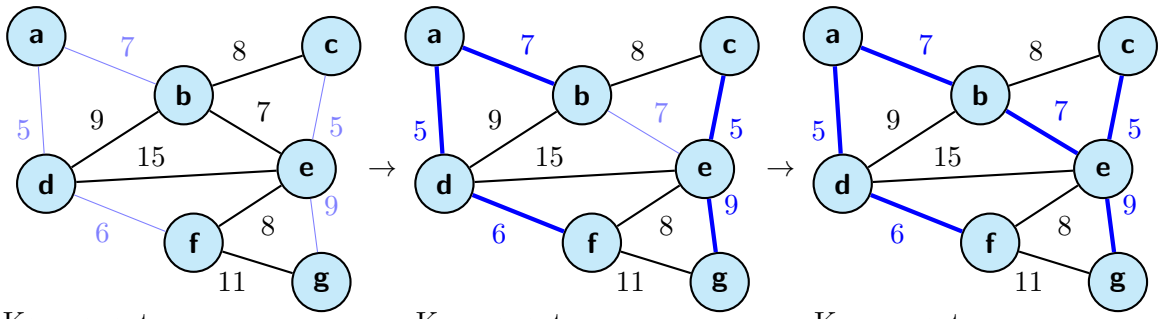
Sollinův-Borůvkův algoritmus

Historicky vůbec první algoritmus pro problém minimální kostry (z roku 1928) byl nalezen jiným českým (brněnským) matematikem a potkal jej snad ještě kurióznější osud, než algoritmus Jarníkův, byl totiž objeven několikrát (v Polsku, ve Francii), ale až pan Sollin jej popsal anglicky a proto většinou nese jeho jméno.

Jedná se o poněkud složitější algoritmus, chová se jako Jarníkův algoritmus spuštěný zároveň ze všech vrcholů grafu najednou. Detaily lze nalézt v literatuře [3, odstavec 4.5.3].

Algoritmus 6.4 Borůvkův algoritmus hledání minimální kostry grafu

Require: $G(V, E, w)$ na n vrcholech popsaný seznamem sousedů $s_{i,j}$ a délek hran $d_{i,j}$
Ensure: Minimální kostra H
  $W \leftarrow \{x\}$, kde x je libovolný prvek z V
  $F \leftarrow \{\}$ je prázdná množina hran kostry
  while $W \neq V$ **do**
    $d_{\min} \leftarrow \infty$
    for all $x \in W$ **do**
      for all $y \in (V \setminus W)$ **do**
        if $d_{x,y} < d_{\min}$ **then**
          $d_{\min} \leftarrow d_{x,y}$
          $z \leftarrow y$
          $\varphi \leftarrow \{x, y\}$ {zapamatujeme si nejbližší prvek ke komponentě}
        end if
      end for
    end for
    $W \leftarrow W \cup \{z\}$
    $F \leftarrow F \cup \{\varphi\}$
  end while
  return Minimální kostra $H(W, F, \omega)$.



Komponenty:
{a},{b},{c},{d},{e},{f},{g}

Komponenty:
{a,b,d,f},{c,e,g}

Komponenty:
{a,b,c,d,e,f,g}

Literatura

[1] Jiří Demel. *Grafy a jejich aplikace.* Academia, Praha, 1 edition, 2012.

[2] Petr Hliněný. *Základy teorie grafů.* Masarykova univerzita, Brno, 1 edition, 2010.

[3] Jiří Matoušek and Jaroslav Nešetřil. *Kapitoly z diskrétní matematiky.* Karolinum, Praha, 4., upr. a dopl. edition, 2009.

[4] Seven bridges of königsberg, 2013.

KAPITOLA 7

Jemný úvod do numerických metod

7.1 Úvod do numerické matematiky

Pro detailnější obeznámení s pojmy, uváděnými níže, doporučuji konzultovat monografii Michaela T. Heathe [1], případně nějaká z mnoha skript o numerické matematice, která v posledních letech vyšla – například [3] (části tohoto skripta jsou dostupné i on-line).

Matematické modelování

Termínem **matematické modelování** označujeme proces tvorby **matematického modelu**, jenž popisuje námi zkoumaný systém pomocí matematických pojmů a jim odpovídajícího zápisu. Připomeňme si, že jako **systém** chápeme část prostředí, kterou lze vnímat odděleně od jejího okolí. Systém od okolí odděluje nějaká hranice, ať už fyzická, či myšlenková.

Abychom mohli zkoumat chování nějakého systému, můžeme

- provádět **experimenty** anebo
- popsat systém matematicky – sestavit jeho **matematický model**.

Matematické modely mohou v závislosti na použitém matematickém aparátu nabývat mnoha různých forem. Nejčastěji se setkáme s diskrétními dynamickými systémy, statistickými modely, spojitými modely založenými na diferenciálních rovnicích, či s modely využívajícími poznatků teorie her. Na fakultě jste se s mnoha přístupy k matematickému modelování zajisté již setkali. Uveďme

jeden příklad za všechny: V rámci předmětu *Modelování systémů a procesů* [4] jsme si ukazovali různé modely, popisující chování systémů ve spojitém či diskrétním čase a popis systémů těmito modely dělili na *vnější* a *vnitřní* (stavový) popis.

Příklad 7.1 (Závaží na pružině)**.** *Netlumené kmity závaží na pružině popisuje homogenní diferenciální rovnice harmonických kmitů*

$$\frac{\mathrm{d}^2 y(t)}{\mathrm{d}t^2} + \omega^2 y(t) = 0.$$

V této rovnici označuje $y(t)$ polohu závaží v čase t a ω určuje úhlovou frekvenci kmitů. Počáteční podmínky této diferenciální rovnice potom udávají počáteční výchylku závaží od rovnovážné polohy a jeho počáteční zrychlení.

Příklad 7.2 (Model vývoje dluhu)**.** *Finanční model vývoje zadlužení může mít tvar diferenční rovnice*

$$y[n+1] = (1 + \alpha[n]) \cdot y[n] - u[n].$$

Zde je $y[n]$ posloupnost výšky dluhu (hodnota $y[0]$ bude odpovídat výšce původní půjčky), $\alpha[n]$ je v čase proměnná úroková míra a $u[n]$ označuje posloupnost splátek dluhu.

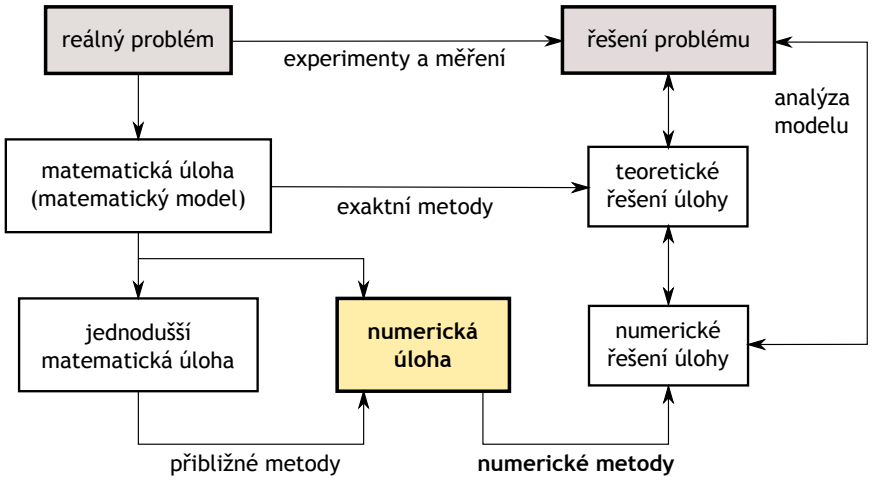
Ke zkoumání matematických modelů systémů jsme používali Matlab a Simulink.

V příštích přednáškách si stručně povíme

- co vlastně počítač musí umět, aby dokázal s dostatečnou přesností počítat s matematickými modely reálného světa,
- jaké matematické algoritmy se ve vybraných případech používají a
- proč není dobré počítači vždycky slepě věřit.

Numerická matematika

Reálný problém, který se snažíme vyřešit – například nastavení tuhosti odpružení podvozku nového modelu automobilu, nebo optimální alokaci skladů zásilkového obchodu –, můžeme zkoumat experimentální cestou, porovnáváním různých řešení a měřením výsledků bez detailní znalosti jeho modelu. Pokud ale potřebujeme určité záruky, že jsme danou úlohu vyřešili korektně a chceme dodat našemu způsobu řešení důvěryhodný základ, použijeme na vyřešení problému nějaký **matematický model** (o něm jsme se už zmiňovali výše). Kdo je zdrojem tohoto modelu, není v tento okamžik až tak důležité: v případě fyzikálních úloh to patrně bude někdo, kdo dokáže matematicky



Obrázek 7.1: Pozice numerické úlohy a numerických metod v kontextu matematického modelování.

popsat fyzikální stránku řešené úlohy, v případě chemických procesů pak to bude někdo znalý potřebné fyzikální chemie.

V procesu řešení matematického modelu zpravidla nemůžeme postupovat přímo, musíme se obrátit k jeho nahrazení jednodušší úlohou, tak zvaným **počítačovým modelem**. To je původní model upravený tak, že jej lze v konečném čase spočítat na počítači. Výsledky počítačového modelu jsou ale pouze přibližné (i když většinou dostatečně přesné) a slouží jako **přiblížení** neboli **aproximace** výsledků, které by byly přesným řešením uvažovaného matematického modelu. Dobré počítačové modely nám zároveň poskytují informaci o přesnosti získaných výsledků.

Takováto přibližná metoda řešení se v praxi používá velmi často, zapotřebí jí není spíše v malém počtu případů. Jako příklad nám můžou sloužit početné modely technických systémů založené na diferenciálních rovnicích. Úlohy pro diferenciální rovnice můžeme sice někdy řešit analyticky (tedy na papíře, užitím standardního matematického aparátu), ale u většiny diferenciálních rovnic takové analytické řešení není možné a musíme se spokojit s jeho aproximací.

Příklad 7.3 (Přibližná metoda řešení)**.** *Soustavu parciálních diferenciálních rovnic nelze zpravidla řešit přímo (podobně, jako jsme to zmiňovali u oby-*

čejné diferenciální rovnice). Pokud ale nahradíme parciální derivace diferencemi a celou původní doménu úlohy rozdělíme na síť rovinných či objemových elementů (použijeme postup, jenž se nazývá metoda konečných prvků*), lze původní soustavu převést na soustavu algebraických rovnic, lineárních nebo nelineárních – to je v našem případě ta jednodušší matematická úloha, uvedená na Obrázku 7.1. Soustavu lineárních rovnic můžeme být schopni vyřešit přesně například Gaussovou eliminací, ale tento proces je pro velké soustavy velmi pomalý a v mnoha případech se spokojíme s přibližným a mnohem rychlejším řešením některou iterační metodou.*

■ doplnit další příklady? ■

Numerická úloha

Protože počítač je konečný automat pracující pouze s konečným počtem vstupních a výstupních dat, zavádí se někdy také pojem numerické úlohy.

Numerická úloha – jasný a jednoznačný popis funkčního vztahu mezi *konečným* počtem vstupních a výstupních dat.

Některé matematické úlohy jsou rovnou numerickými úlohami – například již zmíněné řešení soustav lineárních rovnic.

Data – vyjádřitelná konečným počtem čísel.

⇒ Počítačový model je taková aproximace matematického modelu, jež může být v konečném čase realizována na počítači.

Poznamenejme, že ne všechny numerické úlohy lze na počítači vyřešit v konečném počtu kroků. Takové řešení lze provést přesně například u řešení soustav lineárních rovnic (nebereme-li v úvahu zaokrouhlovací chyby), ale ne už obecně při hledání vlastních čísel matic, což je také numerická úloha. U řady numerických úloh se tedy spokojíme s tím, že stanovíme nějaké jejich (vhodně definované) přibližné řešení (aproximaci). Taková řešení se pak počítají pomocí vhodných **numerických metod**. **■ O aproximaci se mluví výše. ■**

Numerický algoritmus – postup, kterým se v konečném počtu kroků řeší daná numerická úloha. Při studiu vlastností numerických algoritmů nás zajímá především realizace aritmetických operací s čísly, nikoliv logické operace. Algoritmus numerické metody (Newtonova metoda) – teoretická záležitost, nemusí být konečný.

Realizace algoritmu na počítači (též **implementace**), směřuje k programu – třeba algoritmus ve Fortranu, realizovatelný v konečném čase (má už epsilon a maximální povolený počet iterací); hledím, abych se vyhnul třeba odčítání sobě blízkých čísel.

Konečná fáze je **program** – žádná numerická metoda mně nic nespočítá,

vždycky je to až program.

Konstrukce a analýza metod a algoritmů pro realizaci numerických úloh na počítačích: **numerická matematika**.

Vstupní a výstupní data jsou vlastně danými a hledanými objekty numerické úlohy [3].

Příklad 7.4 (Numerická úloha)**.** *Přibližné řešení rovnice* $x^4+a_1x^2+a_2x+a_3 = 0$ *je možno počítat numericky pro konkrétní vstupní vektor* $\mathbf{a} = [a_1, a_2, a_3] \in \mathbb{R}^3$.

Výstupem numerické metody řešení bude vektor $\mathbf{x} = [x_1, x_2, x_3, x_4] \in \mathbb{C}^4$. *Řešení je v obecném případě garantováno pouze přibližné, při vhodné volbě vektoru parametrů* $\mathbf{a}$ *jej ale můžeme získat i přesně.*

Příklad 7.5 (Co není numerická úloha)**.** *Řešení rovnice* $y''(x) - y(x)^2 = 0$ *za daných počátečních podmínek nelze vyjádřit konečným počtem čísel a nelze jej tedy hledat numericky (řešením rovnice je funkce, nikoliv množina číselných hodnot).*

Numerický přístup lze použít pouze pro vyšetření hodnot ve vybraných bodech $x \in \{x_i\}_1^n$, *kde typicky* $x_i = x_0 + i \cdot \Delta$, *spojitý interval tedy aproximujeme diskrétním. Vyžaduje-li to přesnost výpočtu, nemusí být hodnoty* x_i *rozmístěny ekvidistantně – to uvidíme například při numerickém výpočtu integrálů.*

V okamžiku, kdy máme v ruce výsledky, musí ještě proběhnout **analýza modelu**, respektive **analýza řešení**. Účelem těchto analýz je **verifikovat** a **validovat** použitý model a metodu řešení a obdržet odpověď na následující otázky:

validace – Počítáme se správným modelem? Pokud jsou přípustné teploty kladné, nepředpovídá model zápornou teplotu? Má-li modelovaná veličina monotónně růst nebo klesat, nedochází ve výsledku k oscilacím?

verifikace – Počítá náš model správně? Nedochází v něm k nepřípustným chybám?

Zdrojům možných chyb při použití validního modelu se budeme věnovat v jednom z následujících odstavců. Nejprve si ale musíme vysvětlit, jak vlastně počítač reprezentuje reálná čísla.

7.2 Zobrazení čísel v počítači

Dnešní počítače pracují výlučně v binárním kódu – veškerá informace je ukládána v operační paměti a na vnější paměťová média ve formě jedniček a nul. Mimo jiné to znamená, že namísto desítkové soustavy, používané pro počítání na papíře, jsou v počítačích čísla reprezentována ve dvojkové soustavě.

Celá čísla – ekvivalenty ve dvojkové soustavě, jeden (nejvyšší) bit na znaménko. Celá čísla reprezentují i čísla přirozená ve dvojnásobném rozsahu, musíme si proto dávat pozor, s čím zrovna počítáme (v ANSI C deklarace `int` versus `unsigned int`).

Příklad 7.6. $66 = (01000010)_2$, $-126 = (11111110)_2$, *ovšem také* $(11111110)_2 = 254$

Pevná řádová čárka – pevný počet bitů pro celou a desetinnou část čísla. *Q notace* označuje počet bitů před a za desetinnou čárkou[5]: Q3.5 je osmibitové číslo, jež může nabývat hodnot $000{,}00000_2$ až $111{,}11111_2$, tedy $0{,}00000, 0{,}03125, \ldots 7{,}93750, 7{,}9687$

Příklad 7.7. $5{,}3100_{10} \approx 10101010_2 (= 101{,}01010_2)$, $7{,}5625_{10} = 11110010_2$

Pohyblivá řádová čárka – převod na tvar $a \cdot q^b$, kde $a \in \mathbb{R}$, $1 \leq a < 10$ je **mantisa**, $b \in \mathbb{Z}$ je **exponent** a $q \in \mathbb{N}$ je **základ**, typicky 2 (PC, IEEE 754) nebo 10 (kalkulačka).

Definice 7.1 (Semilogaritmický tvar)**.** *Číslo x lze reprezentovat v* ***semilogaritmickém tvaru s normalizovanou mantisou*** *jako*

$$x = \operatorname{sgn}(x) \cdot \left(\frac{a_1}{q} + \frac{a_2}{q^2} + \cdots + \frac{a_l}{q^l} \right) q^b,$$

kde $q \in \mathbb{N}$, $q > 1$ je základ*, $a_i \in \{0, 1, \ldots, q-1\}$, $a_1 \geq 1$, jsou* číslice mantisy *a $b \in \{m_1, \ldots, m_2\}$ $m_1, m_2 \in \mathbb{Z}$, obvykle $m_1 < 0$ a $m_2 > 0$ je* exponent*. Předpokládá se, že b je v intervalu $\langle m_1, m_2 \rangle$ reprezentováno rovnoměrně. Stejně tak se uvažuje pouze normalizovaná mantisa $m \in \langle 1, q)$, protože m a b nejsou jinak určeny jednoznačně.*

Všimněte si, že reprezentace x pokrývá pouze podmnožinu $\mathbb{R}$ – má pouze $2(q-1)q^{l-1}(m_2 - m_1 + 1) + 1$ prvků. Některá reálná čísla nelze tedy přesně reprezentovat. Jsou to například čísla s moc velkým a malým exponentem a čísla, která nemají konečný q rozvoj. Další limitující faktor je konečný počet desetinných míst mantisy – je jich pouze l.

Příklad 7.8 (Reprezentace 1/2 a 1/10)**.** *Budeme-li uvažovat $q = 2$, bude*

$$\frac{1}{2} = \left(\frac{1}{2} + \frac{0}{4} + \frac{0}{8} \cdots \right) 2^0$$

ale

$$\frac{1}{10} = \left(\frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \frac{1}{32768} \cdots \right) 2^{-4}$$

nelze reprezentovat konečným rozvojem.

Jedním z důsledků semilogaritmické reprezentace čísel v pohyblivé řádové čárce je i to, že v počítači existuje největší a nejmenší reprezentovatelné číslo.

Platí $a_1 \neq 0$, což při binární reprezentaci znamená ale $a_1 = 1$, a proto se a_1 vůbec při binární reprezntaci vůbec neukládá a získáme tak „zadarmo“ jeden bit přesnosti reprezentace navíc.

Jak zobrazení čísel, tak chování počítačové aritmetiky jsou dnes určovány všeobecně uznávanými normami, z nichž nejčastěji se užívá norma IEEE 754 [2]. Aritmetika podle této normy má tu vlastnost, že výsledek operace s dvěma strojovými čísly dává vždy nějaký strojově zobrazitelný výsledek. Tento požadavek vede k tomu, že v IEEE reprezentaci jsou potom speciální příznaky pro případy, kdy například dojde k podtečení a reálné číslo se reprezentuje v nižší přesnosti (n už nejde snížit) s hodnotou $a_1 = 0$ – tato čísla nazýváme subnormální (nebo denormalizovaná) **ubírá se mantisa, jsou pod `realmin`, ale to definujeme až později**. Podobně jsou zde symboly pro překročení číselného rozsahu směrem k nekonečnu (v Matlabu výsledek `Inf`) nebo pro výsledek, který matematicky nedává smysl (`NaN`, například výpočet 0/0).

Hezký interaktivní příklad výpočtu reprezentace decimálně zapsaných čísel podle IEEE 754 lze nalézt na stránkách http://babbage.cs.qc.cuny.edu/IEEE-754/.

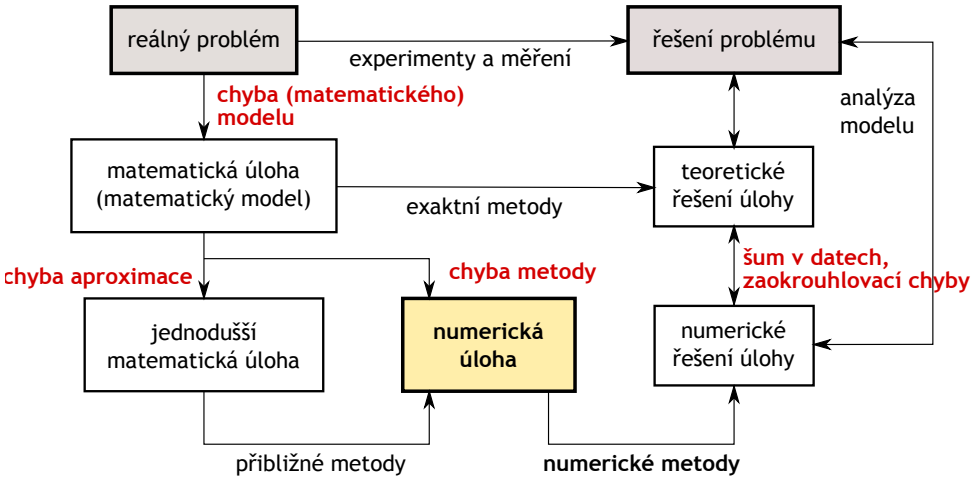
Když už nyní zhruba víme, jak se v počítači reprezentují čísla, podívejme se na možné zdroje chyb při matematickém výpočtu.

7.3 Typy chyb

V Odstavci **??** jsme se zmínili o tom, že při tvorbě matematického modelu problému z reálného světa jsme vždy nuceni provést určitá přiblížení a některé skutečnosti si idealizovat. Rozdíl řešení idealizovaného problému (matematické úlohy) a řešení reálného problému nazýváme **chybou matematického modelu** nebo také jenom **chybou modelu**. Z velikosti této chyby posuzujeme zpětně vhodnost či nevhodnost zvoleného modelu, tj. aproximace reality.

Jestliže k řešení matematické úlohy použijeme metodu, která nám neposkytne přesné (teoretické) řešení dané úlohy, pak chybu, které se dopustíme, nazýváme **chybou metody**. Typickým příkladem je chyba, které se dopustíme, když za limitu nekonečné posloupnosti vezmeme některý její člen s dostatečně velkým indexem. Často řešíme matematickou úlohu tím, že pomocí jistých metod ji nahradíme (aproximujeme) úlohou jednodušší – obvykle již úlohou numerickou – a rozdíl řešení těchto dvou úloh nazýváme **chybou aproximace**. Tato chyba se často bere jako součást chyby metody.

K posouzení přesnosti výsledku musíme ještě vzít v úvahu **chyby (šum) ve vstupních datech**, dané jednak chybami měření, jednak způsobené zobrazením vstupních dat do nesouvislé množiny reprezentující data v počítači.



Obrázek 7.2: Typy chyb v matematickém modelování

Poslední skupinou chyb jsou **chyby zaokrouhlovací**. Do této skupiny zahrnujeme všechny nepřesnosti způsobené realizací algoritmu v počítači včetně nepřesného provádění aritmetických operací.

V dalším textu budeme používat označení, v němž číslo x v numerickém algoritmu je reprezentováno přiblížením $\tilde{x}$.

Definice 7.2 (Absolutní a relativní chyba)**.** ***Absolutní chybou*** $\mathcal{A}(x)$ *aproximace čísla* x *číslem* $\tilde{x}$ *označujeme rozdíl*

$$\mathcal{A}(x) = |x - \tilde{x}|$$

Relativní chybou $\mathcal{R}(x)$ *aproximace čísla* x *číslem* $\tilde{x}$ *označujeme podíl*

$$\mathcal{R}(x) = \frac{\mathcal{A}(x)}{|x|} = \left|\frac{x - \tilde{x}}{x}\right|, \ x \neq 0$$

■ zmínit data naměřená s chybou? ■

Absolutní chyba tak není nazývána kvůli absolutní hodnotě, jedná se o absolutní odchylku mezi reprezentací čísla a jeho pravou hodnotou. Velikost absolutní chyby není postačujícím měřítkem pro posouzení přesnosti výpočtu, zde se nám naopak velmi hodí relativní chyba – ta bere v úvahu i velikost čísla,

s nímž se počítá. Pro x blízko 0 se ovšem může stát, že relativní chyba bude moc přísná (uvidíme to v další přednášce při výpočtu kořenů nelineární funkce Newtonovou metodou) a v této oblasti je vhodnější měřit absolutní chybou.

Připomeňme si, co jsme zmínili již v předešlém odstavci, když jsme se bavili o reprezentaci reálných čísel v pohyblivé řádové čárce: *Reálná čísla nejsou v počítači většinou reprezentována přesně.* Použijeme-li již zmíněný standard IEEE 754 [2], čísla v pohyblivé řádové čárce reprezentujeme standardně ve **dvojnásobné přesnosti** (tj. *IEEE 754 binary64* s 53 bity mantisy, z nichž je uloženo 52)[1] – relativní chyba této reprezentace je o malinko větší, než 10^{-16} (mantisa má 15,95 platných dekadických číslic). Pokud je to třeba, lze na úkor vyšších zaokrouhlovacích chyb volit **jednoduchou přesnost** (tj. *IEEE 754 binary32* s 24 bity mantisy, z nichž je uloženo 23)[2], kde je relativní chyba reprezentace o malinko nižší, než 10^{-7} (mantisa má 7,22 platných dekadických číslic).

Jak vidíme, je-li relativní chyba reprezentace čísla x hodnotou $\tilde{x}$ rovna $\mathcal{R}(x) = 10^{-d}$, znamená to, že x je má ve zvolené reprezentaci d platných číslic. Absolutní chyba $\mathcal{A}(x) = 10^{-d}$ udává počet platných desetinných míst.

Přesnost aritmetiky v Matlabu: `eps` (relativní chyba reprezentace čísla), `realmax` (největší zobrazitelné číslo), `realmin` (nejmenší plně reprezentovatelné – tedy nikoliv subnormální – číslo).

Vliv zahokrouhlovacích chyb

Nahromadění zaokrouhlovacích chyb během postupu výpočtu může nakonec vést k situaci, kdy výsledek výpočtu je zcela odlišný od správné hodnoty – a to může vést k velmi závažným následkům. Jedním z tragických příkladů takové chyby je selhání baterie protiraketových střel Patriot, chránících americkou základnu Dhahran v Saudské Arábii během války v Perském zálivu dne 25. února 1991, jež stálo život 28 amerických vojáků, dalších 98 bylo zraněno [6]. Střely Patriot se přitom v době konfliktu považovaly za velmi úspěšné prostředky ochrany proti raketovým útokům protivníka. V tomto případě ale baterie vůbec nevystřelila. Co se vlastně stalo?

Příklad 7.9 (Proč Patriot netrefí Scud)**.** *Hlavním důvodem selhání byl nepřesný výpočet časové základny v zaměřovacím systému baterie Patriot: Systém počítal s hodnotami času v desetinách sekundy, jeho autoři proto systémový čas v sekundách získávali prostým vynásobením hodnotou* 0,1*. Jak jsme viděli v Příkladě 7.8, většina desetinných zlomků nemá ale v pohyblivé řádové čárce s binárním základem konečný rozvoj, platí*

$$0{,}1 \approx (0{,}0001100110011001100110011001100110011\ldots)_2.$$

[1] V Matlabu i ANSI C/C++ `double`.
[2] V Matlabu `single`, v ANSI C/C++ `float`.

Interní registr v řídicím systému baterie pracoval ovšem pouze v jednoduché přesnosti, měl tedy 24 bitů mantisy a efektivně tak ukládal číslo

$$0{,}1 \approx (0{,}00011001100110011001100 11)_2 \approx 0{,}999999905,$$

tedy reprezentoval desetinu sekundy s absolutní chybou přibližně $0{,}000000095$.

Systém, chránící spojeneckou základu, byl v provozu nepřetržitě po dobu 100 hodin, a během této doby odchylka časové základy od referenčního času dosáhla $0{,}34$ *sekundy. Scud letí (či spíše jeho zbytky včetně hlavice padají) rychlostí okolo* $1700\,m/s$, *posun časové základny potom způsobil, že řídicí systém baterie jej po prvotním radarovém kontaktu hledal v bodě cca* $500\,m$ *za skutečnou polohou útočící střely. Vzhledem k tomu, že se v dané oblasti nic nevyskytovalo, prohlásil původní radarový kontakt za falešný poplach a proti blížícímu se Scudu nebyly odpáleny žádné rakety.*

K příkladu je třeba pro úplnost doplnit dvě věci: (i) Patriot je původně mobilní systém protiletadlové obrany, upravený na ničení taktických balistických střel. Při návrhu se nepočítalo s dlouhodobým nasazením při obraně statických cílů – nikdo proto nepředpokládal, že by celý systém byl aktivován po dobu několika dní. (ii) Řešení problému přitom bylo jednoduché (a navržené izraelskými odborníky asi dva týdny před incidentem): stačilo vždy po pár hodinách restartovat řídicí systém baterie. Opravený software byl přitom výrobcem dodán 26. února 1991, tedy den po incidentu v Dhahranu.

Vliv aritmetických operací na relativní chybu

Aritmetické operace mohou mít na nepřesné reprezentace čísel devastující vliv (například podíl velkého a malého čísla, ale i odčítání dvou sobě blízkých čísel stejného znaménka).

Relativní chyba se může výrazně zvětšit při odčítání dvou blízkých čísel:

$$\mathcal{R}(x \pm y) = \frac{\mathcal{A}(x \pm y)}{|x \pm y|}$$

Podmínkou ale je, že čísla x a y nejsou v počítači reprezentována přesně. Násobení ani dělení nemají na $\mathcal{A}(x)$ a $\mathcal{R}(x)$ výraznější vliv.

Příklad 7.10. *Mějme čísla* $x_1 = 758320$, $x_2 = 757940$, *a nechť jsou reprezentována jako* $\tilde{x}_1 = 758330$ *a* $\tilde{x}_2 = 757930$. *Platí* $\mathcal{A}(x_1) = 10$, $\mathcal{A}(x_2) = 10$,

$$\mathcal{R}(x_1) = \frac{10}{758320} \leq 1{,}32 \cdot 10^{-5}, \mathcal{R}(x_2) = \frac{10}{757940} \leq 1{,}32 \cdot 10^{-5}.$$

Máme tedy $v = x_1 - x_2 = 380$ *a je* $\tilde{v} = \tilde{x}_1 - \tilde{x}_2 = 400$. *Proto* $\mathcal{A}(v) = |v - \tilde{v}| = 20$ *a*

$$\mathcal{R}(v) = \frac{\mathcal{A}(v)}{|v|} = \frac{20}{380} \leq 0{,}053.$$

Relativní chyba rozdílu $v = x_1 - x_2$ je tedy o tři řády vyšší než relativní chyby obou operandů.

7.4 Typy numerických úloh

Mějme dány dva vektorové prostory $\mathcal{B}_x$ (vstupní data) a $\mathcal{B}_y$ (výstupní data).

Definice 7.3 (Matematická úloha)**.** ***Matematickou úlohou*** *rozumíme relaci*

$$y = U(x),\ x \in \mathcal{B}_x, y \in \mathcal{B}_y$$

Definice neříká nic jiného, než že matematická úloha transformuje posloupnost vstupních dat na posloupnost výsledků. Ne všechny úlohy, odpovídající formální definici uvedené výše, lze na počítači numericky řešit. Možné to je pouze pro podmnožinu všech úloh, kterou si můžeme definovat takto:

Definice 7.4 (Korektní úloha)**.** *Řekneme, že úloha je* ***korektní****, pokud*

1. *ke každému $x \in \mathcal{B}_x$ existuje právě jedno $y \in \mathcal{B}_y$,*
2. *řešení y spojitě závisí na datech, tedy pokud $x_n \to x$ a $U(x_n) = y_n$, pak také $y_n \to y = U(x)$.*

Zbylé matematické úlohy označujeme jako **nekorektní**. Jde například o nejednoznačně řešitelné problémy, intervalové odhady, úlohy s nevhodnou formulací zadání.

Příklad 7.11 (Korektní úloha)**.** *Jako příklad korektní úlohy může sloužit například výpočet integrálu z dané spojité a ohraničené funkce přes nějaký interval.*

Příklad 7.12 (Nekorektní úloha)**.** *Určete matici* $\mathbf{A}$ *splňující rovnici* $\mathbf{Ax} = \mathbf{b}$ *máte-li dány hodnoty* $\mathbf{x}$ *a* $\mathbf{b}$

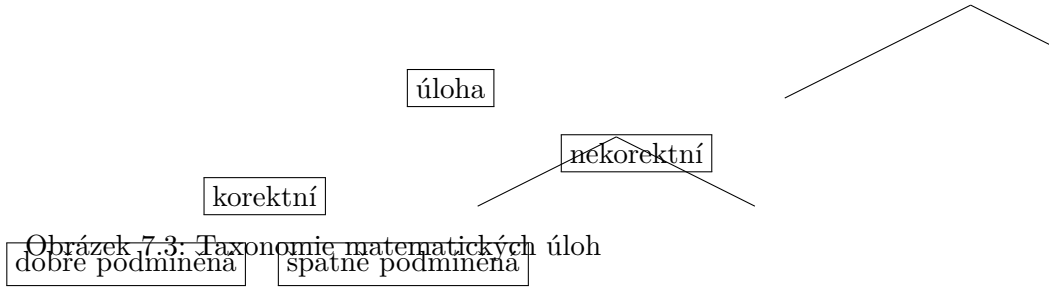
Příklad 7.13 (Jiná nekorektní úloha)**.** *Určete*

$$y = \int_{-1}^{1} 1/x \, dx.$$

Definice 7.5 (Číslo podmíněnosti)**.** *Podíl*

$$C_p = \frac{\dfrac{|\Delta x|}{|x|}}{\dfrac{|\Delta y|}{|y|}}$$

se nazývá ***číslo podmíněnosti*** *úlohy.*



Obrázek 7.3: Taxonomie matematických úloh

Číslo podmíněnosti nám udává, jak moc „silný“ je vliv změn ve vstupních datech úlohy na výstupní data. V případech, kdy $|x|$ nebo $|y|$ jsou malé, bývá namístě používat v definici čísla podmíněnosti v čitateli i jmenovateli nikoli relativní, ale absolutní chyby (mluvíme pak o **absolutním čísle podmíněnosti**).

Definice 7.6 (Dobře podmíněná úloha)**.** *Budeme říkat, že korektní úloha je* ***dobře podmíněná****, jestliže malá změna ve vstupních datech vyvolá malou změnu řešení (resp.* $C_p \approx 1$*).*

Dobře podmíněné úlohy se numericky řeší „snadno“, tedy s malou očekávanou chybou a s relativně malými problémy se zaokrouhlovacími chybami. Úlohy špatně podmíněné díky svým vlastnostem způsobují numerické problémy, je třeba je řešit „opatrně“ (používat vhodné algoritmy, jež třeba nejsou tak efektivní, jsou ale numericky stabilní) a nebo se pokusit je transformovat na lépe podmíněné úlohy.

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill, Boston, 2 edition, 2002.

[2] IEEE Std 754-2008. Ieee standard for binary floating-point arithmetic, 2008.

[3] Stanislav Míka and Marek Brandner. *Numerické metody I.* FAV ZČU, Plzeň, 2. edition, 2002.

[4] Modelování systémů a procesů.

[5] E. L. Oberstar. Fixed-point representation and fractional math, 2007.

[6] U.S. Government Accountability Office. Patriot missile defense: Software problem led to system failure at dhahran, saudi arabia., 1992.

KAPITOLA 8

Kořeny nelineárních funkcí

8.1 Řešení nelineárních rovnic

Formulace úlohy

Pro detailnější obeznámení s pojmy, uváděnými níže, doporučuji i zde konzultovat knihu Michaela T. Heathe [1], případně nějakou z českých učebnic či mnoha skript o numerické matematice, která v posledních letech vyšla – například [2], [3] (části tohoto skripta jsou dostupné i on-line). Mnohé ze zde použitých obrázků jsme převzali právě z [1].

Budeme se zde zabývat především numerickými metodami pro (přibližné) řešení **nelineární rovnice**

$$f(x) = 0, \tag{8.1}$$

kde $f : \mathbb{R} \to \mathbb{R}$ je reálná nelineární funkce jedné reálné proměnné. Řešit **lineární rovnice** tvaru $ax + b = c$, tj. $ax + b - c = 0$, jsme se naučili již na střední škole. Seznámili jsme se tam i s řešením některých nelineárních rovnic, například kvadratické rovnice $ax^2 + bx + c = 0$ nebo rovnice $\sin x = 0$. K řešení většiny nelineárních rovnic však potřebujeme použít některou vhodnou numerickou metodu.

Řešením nebo **kořenem** uvedené rovnice nebo také **nulovým bodem** funkce f nazýváme takové reálné číslo x^*, pro které platí $f(x^*) = 0$. Nelineární rovnice mohou mít právě jedno řešení (rovnice $x - \sin x = 0$), více řešení (rovnice $x^2 - 1 = 0$), nebo nemusí mít žádné řešení (rovnice $\sin x = 2$).

Budeme se také zabývat speciálním rovnicemi tvaru

$$x = g(x), \tag{8.2}$$

kde opět $g : \mathbb{R} \to \mathbb{R}$ a tedy $f(x) = x - g(x)$. Řešení takové rovnice se nazývá také **pevným bodem** funkce g.

Speciální situace nastává také, je-li uvažovaná funkce f polynom. Hledáme pak totiž často i jeho komplexní kořeny. Pro polynomy existují tedy i speciální numerické metody, jimiž se zde však nemůžeme extra zabývat. V praxi se setkáme i se soustavami nelineárních rovnic, jejichž řešením pak není číslo, ale vektor hodnot. Existují numerické metody i pro řešení takových soustav, z časových důvodů se jimi ale zde také zabývat nebudeme. Případné zájemce odkazujeme na Heathovu knihu [1] nebo na Míkovu učebnici [2] či skripta [3].

Existence a jednoznačnost

Existence a jednoznačnost řešení nelineárních rovnic je podstatně komplikovanější záležitost než je tomu u lineárních rovnic a jejich soustav. V mnoha případech je obtížné stanovit existenci nebo počet řešení nelineární rovnice. Zatímco u soustav lineárních rovnic musí být počet řešení roven nule, jedné nebo být nekonečný, nelineární rovnice mohou mít jakýkoli počet řešení, a to dokonce i pro jedinou rovnici.

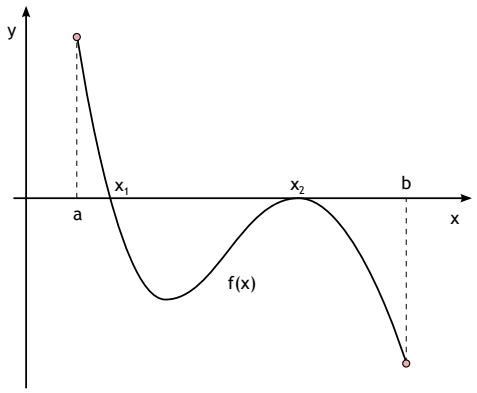
Příklad 8.1 (Existence kořenů)**.** *Uvažujeme-li kořeny následujících rovnic na celém $\mathbb{R}$, pak rovnice*

- $e^x + 1 = 0$ *nemá žádné řešení*
- $e^{-x} - x = 0$ *má právě jedno řešení*
- $x^2 - 4\sin x = 0$ *má dvě řešení*
- $x^3 + 6x^2 + 11x - 6 = 0$ *má tři řešení*
- $\sin x = 0$ *má nekonečně mnoho řešení*

Jakkoli je tedy obtížné získat jakákoli globální tvrzení o počtu řešení nelineární rovnice, máme přesto k dispozici některá užitečná lokální kritéria zaručující existenci aspoň jednoho kořene rovnice na daném intervalu. Jedno takové praktické kritérium je založeno na matematické větě, která pochází od B. Bolzana a říká:

Věta 8.1 (Bolzanova věta)**.** *Nechť funkce f je spojitá na uzavřeném omezeném intervalu $[a, b]$ a nechť platí $f(a) \cdot f(b) < 0$ (tj. $f(a)$ a $f(b)$ mají opačná znaménka). Pak funkce f má na intervalu (a, b) alespoň jeden nulový bod.*

Takový interval $[a, b]$, v jehož koncových bodech má funkce f opačná znaménka, budeme nazývat **uzávěrou** řešení nelineární rovnice $f(x) = 0$. Jak uvidíme později, v řadě numerických metod pro řešení nelineárních rovnic hraje důležitou roli právě postupné zužování takové předem nalezené uzávěry. Jak ovšem počáteční uzávěru najít, je víceméně záležitostí pokusů a omylů. Jedna z možností je odhadnout nějak počáteční interval, na němž budeme



Obrázek 8.1: Příklad nelineární funkce s kořeny x_1 a x_2 na intervalu $[a, b]$. Tento interval je pro danou funkci $f(x)$ uzávěrou kořene.

chování funkce f zkoumat (i když to ještě nebude uzávěra kořene), a pak procházet tímto intervalem po nějakých vhodně volených krocích, postupně počítat hodnoty $f(x)$ a sledovat, kdy v nich dojde ke změně znaménka.

Poznamenejme ještě, že právě zmíněné kritérium udává pouze postačující, nikoli nutnou podmínku existence kořene. Nemělo by nás tedy od hledání kořene předem odrazovat to, že jsme nenašli jeho uzávěru. V řadě případů totiž ani pro daný kořen uzávěra neexistuje. Jako příklad stačí vzít triviální rovnici $x^2 = 0$ s jediným kořenem $x = 0$. Zde pro všechna x máme $x^2 \geq 0$ a ke změně znaménka tedy nemůže dojít.

Obraťme svoji pozornost nyní k rovnici (8.2). Zde lze k důkazu existence pevného bodu na daném intervalu využít opět klasické matematiky, konkrétně tzv. věty o kontrakci.

Definice 8.2 (Kontrakce)**.** *Řekneme, že funkce $g : \mathbb{R} \to \mathbb{R}$ je na množině $S \subseteq \mathbb{R}$* ***kontrakce****, pokud existuje konstanta $L \in \mathbb{R}$, $0 < L < 1$ taková, že pro všechna $x, y \in S$ platí*

$$|g(x) - g(y)| \leq L|x - y|.$$

Věta 8.3 (O existenci pevného bodu)**.** *Jestliže je funkce g kontrakce na uzavřené množině $S \subseteq \mathbb{R}$ a $g(S) \subseteq S$, pak má g v S pevný bod, a to právě jeden.*

Pokud čtenáři vadí, že jsme výše uvedenou definici a větu formulovali pro obecnou množinu S, může si pod S představovat nějaký interval. Z uvedené věty ihned plyne, že pokud v rovnici (8.1) můžeme psát $f(x) = x - g(x)$, kde g je kontrakce na nějaké uzavřené množině S taková, že zobrazuje S do sebe samé, má rovnice $f(x) = 0$ na množině S právě jedno řešení, totiž pevný bod funkce g. Brzy uvidíme, že tato skutečnost nám dává možnost odvodit některé numerické metody pro řešení nelineárních rovnic. Poznamenejme ještě,

že pokud pro všechna $x \in S$ existuje derivace funkce g a platí $|g'(x)| < 1$, dá se ukázat, že g na S je kontrakce.

Z matematického či logického hlediska jsou naše úvahy o uzávěře spojité funkce či předpoklady věty o kontrakci pouze *postačující podmínky*, nikoli však podmínky nutné. Není tedy nikde psáno, že funkce f nemůže mít nulový bod v intervalu, který není uzávěrou, nebo že funkce g, která není kontrakce, nemůže mít pevný bod. V praxi tedy může být užitečné využít soudobých možností našeho softwarového vybavení a při řešení nelineárních rovnic si nejprve nechat vykreslit graf funkce f pro rovnici (8.1) nebo grafy $y = x$ a $y = g(x)$ pro rovnici (8.2).

Příklad 8.2 (Nesplněné postačující podmínky).

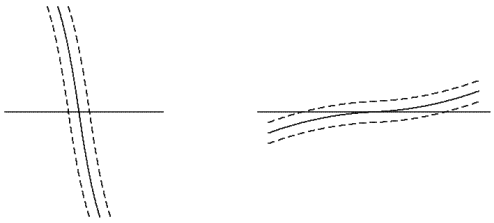
- *funkce* $f(x) = x^2$ *má nulový bod* $x = 0$ *na intervalu* $[-1, 1]$*, který není uzávěra*
- *funkce* $g(x) = \sin x$ *má pevný bod* $x = 0$*, ale v okolí tohoto bodu to není kontrakce*

Doposud jsme se soustředili převážně na existenci kořenů nelineárních rovnic a ne na jejich jednoznačnost, protože se obecně má za to, že nelineární rovnice mohou mít více než jedno řešení, přinejmenším globálně. Jednoznačnost kořene nás přesto může zajímat, alespoň lokálně, například na daném intervalu. Připomeňme si, že z lineární algebry víme, že soustava lineárních rovnic s regulární maticí má vždy právě jedno řešení. Pro nelineární funkce f platí podobné tvrzení o regularitě, přinejmenším lokálně. Pokud totiž funkce f má v daném bodě x^* nenulovou derivaci, pak existuje otevřený interval kolem tohoto bodu, v němž je funkce f ostře monotónní, tedy rostoucí nebo klesající. V takové situaci v okolí bodu x^* tedy může existovat nejvýše jeden kořen.

Pokud však v nějakém kořenu x^* má funkce f nulovou derivaci, má tento kořen jisté zvláštní vlastnosti, které ovlivňují jak podmíněnost řešené úlohy, tak také chování použité numerické metody. Nulový bod x^* funkce f, pro který platí zároveň $f(x^*) = 0$ a $f'(x^*) = 0$ se nazývá *násobný kořen* rovnice $f(x) = 0$. Geometricky to znamená, že graf funkce f má v tomto bodě vodorovnou tečnu splývající s osou x. Kořeny, které nejsou násobné, se nazývají *jednoduché*. Kořen x_1 z Obrázku8.1 je tedy jednoduchý, kořen x_2 je násobný. Pojem násobnosti lze pro hladké funkce f dále upřesnit. Pokud platí $f(x^*) = f'(x^*) = f''(x^*) = \cdots = f^{(m-1)}(x^*) = 0$, ale $f^{(m)} \neq 0$, řekneme, že násobnost kořene x^* je m.

Příklad 8.3 (Pevné body). *Ověřte si následující tvrzení:*

- *funkce* $g(x) = \sin x$ *má jediný pevný bod* $x = 0$
- *funkce* $g(x) = x^2$ *má pevné body* $x = 0$ *a* $x = 1$
- *kořen* $x = 0$ *rovnice* $\sin x = 0$ *je jednoduchý*



Obrázek 8.2: Podmíněnost kořenů nelineární rovnice $f(x) = 0$. Vlevo: dobře podmíněná úloha, vpravo: špatně podmíněná úloha.

- *kořen* $x = 0$ *rovnice* $x^2 = 0$ *je dvojnásobný*
- *kořen* $x = 1$ *rovnice* $x^3 - 3x^2 + 3x - 1 = 0$ *je trojnásobný*

Podmíněnost řešení

Abychom mohli kvantitativně měřit citlivost řešení nelineárních rovnic na data (funkční hodnoty), musíme pracovat s absolutním číslem podmíněnosti, což je obdoba již zavedeného čísla podmíněnosti z minulé přednášky, kde ale místo relativních změn v čitateli i jmenovateli vystupují absolutní odchylky. Je to dáno tím, že hodnota funkce f v kořenu rovnice je rovna nule. Dá se ukázat, že pokud má funkce f v okolí kořene x^* derivaci, je pak toto číslo podmíněnosti přibližně

$$C_{\mathrm{p,abs}} \approx \frac{1}{|f'(x^*)|}.$$

Pokud je ve jmenovateli $f'(x^*) = 0$ (násobný kořen), klademe $C_{\mathrm{p,abs}} = \infty$. Z definice čísla podmíněnosti pak vyplývá, že pokud najdeme bod $\tilde{x}$ takový, že $|f(\tilde{x})| < \epsilon$, může odchylka $|\tilde{x} - x^*|$ tohoto bodu od kořene rovnice $f(x) = 0$ mít velikost $\epsilon/|f'(x^*)|$. Pro malé hodnoty $|f'(x^*)|$ tedy může být tato odchylka od kořene velká, i když funkční hodnota sama je malá.

Celou situaci ilustruje Obrázek 8.2. Čárkované křivky vyznačují oblast nejistoty kolem každé plně nakreslené křivky, takže nulový bod dané funkce může být kdekoli mezi body, v nichž čárkované křivky protínají vodorovnou osu. Malý interval nejistoty pro nulový bod na levém obrázku je dán tím, že daná křivka strmě roste (takže převrácená hodnota derivace je malá), kdežto velký interval nejistoty pro nulový bod na pravém obrázku plyne z pomalého růstu (a tedy velké převrácené hodnoty derivace). Všimněte si také toho, že šíře pásu nejistot kolem funkčních hodnot je na obou obrázcích stejná.

Máme-li násobný kořen x^*, je $f'(x^*) = 0$, takže číslo podmíněnosti násobného kořene je nekonečné. To dává smysl, protože nepatrná změna v f může způsobit, že z násobného kořene se stane více než jeden kořen nebo naopak násobný

kořen zmizí. Stačí si k tomu nakreslit například funkci $f(x) = x^2$ a posunout ji o malé ϵ nahoru nebo dolů. ■ **příklad,obrázek** ■

Podmíněnost nelineární rovnice ovlivňuje náš pohled na přibližné řešení $\tilde{x}$: máme usilovat o to, aby $|f(\tilde{x})|$ byla malá, nebo spíše o to, aby bylo malé $|\tilde{x} - x^*|$, jakkoli přesné řešení x^* předem neznáme? Jak už to u numerických metod bývá, obě uvedené veličiny nejsou nutně malé současně, závisí to ještě na podmíněnosti. Tato skutečnost ovlivňuje volbu algoritmů numerických metod, o nichž budeme hovořit ve zbytku této přednášky. V každém případě je užitečné získat předem nějakou informaci o podmíněnosti řešené úlohy.

Iterační metody a jejich konvergence

Numerické metody pro řešení nelineárních rovnic jsou vesměs metody iterační. Iterace (z lat. *iterare*, opakovat) znamená postupné opakování určitého postupu, během kterého se postupně generuje posloupnost hodnot $x_0, x_1, \ldots, x_k, \ldots$ taková, že v našem případě (hledáme kořen x^* nelineární rovnice) postupně získávané hodnoty konvergují k hledanému řešení, $x_k \to x^*$ pro $k \to \infty$. Při skutečném výpočtu samozřejmě nemůžeme jít s k do nekonečna a iterační postup zastavíme po určitém dostatečně velkém počtu kroků pomocí vhodně zvoleného zastavovacího kritéria. Získáme tak přibližnou hodnotu hledaného kořene. Termín **iterace** se v numerické matematice používá nejen k označení výše uvedeného postupu jako celku, ale nazývá se tak také každý jeho krok a nazývají se tak také postupně počítané hodnoty x_k, tedy aproximace hledaného kořene.

Abychom mohli porovnávat efektivitu iteračních metod, potřebujeme nějak charakterizovat jejich rychlost konvergence, tj. rychlost konvergence posloupnosti iterací x_k k hledanému kořenu rovnice. *Chyba*(nepřesnost) k-té iterace, kterou budeme označovat e_k, se obvykle definuje jako $e_k = x_k - x^*$, kde x_k je aproximace (přiblížení) hledaného řešení získaná v iteraci k a x^* je skutečné (přesné) řešení. Některé z používaných metod neprodukují přímo konkrétní přibližné řešení x_k, ale pouze interval, který s určitostí obsahuje přesné řešení, přičemž délka tohoto intervalu se během iteračního procesu postupně zmenšuje. U takové metody pak e_k definujeme jako délku tohoto intervalu po k-té iteraci. V obou případech pak řekneme, že daná iterační **metoda konverguje s rychlostí** r (také: metoda je řádu r), jestliže pro nějakou konečnou kladnou konstantu $C > 0$ platí

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^r} = C.$$

Speciálně se rozlišují následující případy:

- pokud $r = 1$ a $C < 1$, je konvergence *lineární*,
- pokud $r > 1$, je konvergence *superlineární*,

- pokud $r = 2$, je konvergence *kvadratická*,
- pokud $r = 3$, je konvergence *kubická*, atd.

Jeden z důvodů, proč rozlišujeme mezi lineární a superlineární konvergencí, je ten, že, asymptoticky pro velká k, lineárně konvergentní posloupnost získává po každé iteraci jistý stále stejný počet přesných číslic, kdežto superlineárně konvergující posloupnost v jednotlivých iteracích získává počet přesných číslic, který stále roste. Přesněji můžeme říci, že lineárně konvergentní posloupnost získává v každé iteraci $-\log(C)$ přesných číslic, kdežto superlineárně konvergující posloupnost má po každé iteraci r-krát více přesných číslic než měla před touto iterací. Speciálně pak platí, že u kvadraticky konvergentní metody se počet přesných číslic po každé iteraci zdvojnásobí (pro dostatečně velká k).

Příklad 8.4 (Rychlosti konvergence)**.** *Jestliže členy následujících posloupností představují velikosti chyb postupně generovaných iteračních aproximací, jsou rychlosti konvergence takové, jak je u jednotlivých posloupností uvedeno.*

- $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, \ldots$ *lineární,* $C = 10^{-1}$
- $10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, \ldots$ *lineární,* $C = 10^{-2}$
- $10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}, \ldots$ *superlineární, ale ne kvadratická*
- $10^{-2}, 10^{-4}, 10^{-8}, 10^{-16}, \ldots$ *kvadratická*

V teorii numerických metod se dokazují věty o konvergenci, které nám umožňují říci, kdy pro danou rovnici ta či ona metoda konverguje a jak rychle. Nedávají nám ale explicitně pokyny pro to, kdy máme iterační proces zastavit a prohlásit výsledné přibližné řešení za „dostatečně přesné“. Navrhnout vhodné zastavovací kritérium je poměrně složitá záležitost, a to z řady důvodů. Díky teorii můžeme v zásadě vědět, že se chyba $|e_k|$ postupně zmenšuje, ale protože neznáme přesné řešení, není tu možnost přímo zjistit, jak veliké $|e_k|$ je. Rozumnou náhražkou tu může sloužit relativní změna v postupných iteracích, tedy

$$\frac{|x_{k+1} - x_k|}{|x_k|}.$$

Pokud se tato veličina stane dostatečně malou, znamená to, že se přibližné hodnoty řešení už přestaly významně měnit a nemá tedy cenu pokračovat. Na druhé straně bychom chtěli mít jistotu, že jsme skutečně získali dobré přibližné řešení a že tedy aspoň je hodnota $f(x_k)$ přiměřeně malá. Jak už jsme si ale mohli povšimnout, obě dvě tyto uvedené veličiny nemusí být malé současně, roli tu hraje podmíněnost úlohy. Dále se zde projevuje také případná změna měřítka u proměnné x a funkce f. Ze všech těchto důvodů je vytvoření zcela spolehlivého zastavovacího kritéria velmi obtížné a musíme se také spoléhat na

další informace, které o řešené úloze víme. U iteračních metod, které budeme vzápětí v této přednášce popisovat, proto zpravidla vynecháváme jakýkoli test na konvergenci a místo toho pouze naznačujeme jistý neurčený počet iterací s tím, že iterační proces je třeba ukončit poté, co se vyhoví určitému vhodnému kritériu, jehož volba je (bohužel) na uživateli.

8.2 Numerické metody řešení nelineárních rovnic

Budeme se zabývat numerickým (přibližným) řešením nelineární rovnice (8.1): pro danou spojitou funkci $f : \mathbb{R} \to \mathbb{R}$ hledáme bod $x^* \in \mathbb{R}$ takový, že $f(x^*) = 0$.

Metoda půlení intervalu neboli bisekce

V počítačové aritmetice s konečnou přesností nemusí existovat strojové číslo x^* takové, že $f(x^*)$ je přesně nula. Alternativní možnost je hledat nějaký velmi malý interval $[a, b]$, ve kterém f mění znaménko. Jak jsme již uvedli v odstavci 8.1, taková *uzávěra* zaručuje, že příslušná spojitá funkce musí někde uvnitř tohoto intervalu mít nulový bod. **Metoda půlení intervalu** neboli **metoda bisekce** začíná od nějaké počáteční uzávěry a postupně snižuje její velikost do té doby, až je řešení uzavřeno s požadovanou přesností (resp. tak, jak to aritmetika počítače dovolí). V každé iteraci se nejprve stanoví *střed* aktuálního intervalu a pro další iteraci se ponechá pouze jedna z polovin intervalu podle toho, jaké znaménko má funkční hodnota ve středu. Tato polovina pak tvoří opět (již kratší) uzávěr, s nímž vstupujeme do další iterace. Metodu bisekce formálně můžeme zapsat jako Algoritmus 8.1, v němž jako vstupní data figuruje funkce f, uzávěra $[a, b]$ a chybová tolerance Δ_{tol} pro délku výsledného intervalu obsahujícího kořen.

Algoritmus 8.1 Metoda půlení intervalu

Require: Funkce f, uzávěra $[a, b]$, chybová tolerance Δ_{tol}
Ensure: $x^* : f(x^*) \approx 0$
  while $(b - a) > \Delta_{\text{tol}}$ **do**
    $m \leftarrow a + \dfrac{b - a}{2}$
    if $\operatorname{sgn} f(a) = \operatorname{sgn} f(m)$ **then**
      $a \leftarrow m$
    else
      $b \leftarrow m$
    end if
  end while

Uvedeme ještě pár poznámek k implementaci metody bisekce ve výše uvedeném algoritmu:

Především, zdá se, že nejpřirozenější vzorec pro výpočet středu intervalu $[a, b]$ by byl $m = (a + b)/2$. Jenže v počítačové aritmetice není v extrémních případech zaručeno, že takto počítaný bod m vůbec padne do intervalu $[a, b]$. Komu se to zdá divné, může si v aritmetice se dvěma desítkovými číslicemi zkusit podle tohoto vzorce spočítat střed intervalu $[0.67, 0.69]$ (vyjde $m = 0.7$). Kromě toho může u tohoto vzorce mezivýsledek $a+b$ překročit rozsah počítače i v situacích, kdy střed intervalu v rozsahu počítače leží. Jakkoli jde o extrémní případy, je na tomto jednoduchém příkladu vidět, že počítačová implementace algoritmů není jen pouhé přepisování vzorečků v nějakém vhodném programovacím jazyce. Vzorec použitý v Algoritmu 8.1 se uvedeným problémům vyhýbá.

Dále, pokud jde o testování toho, zda dvě hodnoty $f(x_1)$ a $f(x_2)$ mají stejné znaménko, je na počítači lepší používat funkci signum než matematicky ekvivalentní testování, zda součin $f(x_1) \cdot f(x_2)$ je kladný nebo záporný. Takový součin může totiž také překročit rozsah počítače směrem k nekonečnu a v okolí kořene směrem k nule. Poznamenejme pro pořádek, že je $\operatorname{sgn}(x) = 1$ pro $x \geq 0$ a $\operatorname{sign}(x) = -1$ pro $x < 0$.

Příklad 8.5 (Metoda bisekce)**.** *Metodu půlení intervalu ukážeme na příkladu hledání kořene rovnice*

$$f(x) = x^2 - 4 \sin x = 0.$$

Jako počáteční uzávěru vezmeme interval $[a, b]$, kde $a = 1$ a $b = 3$. Záleží tu pouze na tom, aby se funkční hodnoty v těchto dvou bodech lišily ve znaménku. Vypočítáme hodnotu funkce ve středním bodě intervalu, tedy v $m = 2$ a zjistíme, že $f(m)$ má opačné znaménko než $f(a)$, takže si podržíme levou polovinu počátečního intervalu a položíme pro další krok $b = m$. Pak tento postup opakujeme tak dlouho, až se interval uzávěry zúží na požadovanou velikost. Následující tabulka ukazuje možnou posloupnost iterací.

a	$f(a)$	b	$f(b)$
1.000000	*-2.365884*	*3.000000*	*8.435520*
1.000000	*-2.365884*	*2.000000*	*0.362810*
1.500000	*-1.739980*	*2.000000*	*0.362810*
1.750000	*-0.873444*	*2.000000*	*0.362810*
1.875000	*-0.300718*	*2.000000*	*0.362810*
1.875000	*-0.300718*	*1.937500*	*0.019849*
1.906250	*-0.143255*	*1.937500*	*0.019849*
1.921875	*-0.062406*	*1.937500*	*0.019849*
1.929688	*-0.021454*	*1.937500*	*0.019849*
1.933594	*-0.000846*	*1.937500*	*0.019849*
1.933594	*-0.000846*	*1.935547*	*0.009491*
1.933594	*-0.000846*	*1.934570*	*0.004320*
1.933594	*-0.000846*	*1.934082*	*0.001736*

Interval, u kterého jsme iterace ukončili, má délku menší než 0.0005 a můžeme tedy říci, že nalezený kořen je s touto přesností roven $x^* \approx 1.934$.

Na závěr ještě několik poznámek k metodě půlení intervalu.

- V metodě půlení se nikde nevyužívají velikosti funkčních hodnot, pouze jejich znaménka.
- Pokud výpočet začneme s uzávěrou spojité funkce, pak metoda konverguje vždy, ale dosti pomalu.
- V každé iteraci se délka uzávěry snižuje na polovinu, takže rychlost konvergence je *lineární*, s $r = 1$ a $C = 0.5$.
- V každé iteraci bisekce získáváme jednu další přesnou dvojkovou číslici v přibližném řešení.
- Pro daný počáteční interval $[a, b]$ je délka intervalu po k iteracích rovna $(b - a)/2^k$, takže k dosažení chybové tolerance *tol* je zapotřebí zhruba

$$\log_2 \left(\frac{b-a}{tol} \right)$$

  iterací, nezávisle na vlastnostech použité funkce f.

Metoda postupných aproximací

Metoda postupných aproximací nebo také *metoda prosté iterace* slouží k hledání pevných bodů funkce g z rovnice (2). Připomeňme tedy, že pro funkci

$g : \mathbb{R} \to \mathbb{R}$ se *pevným bodem* nazývá takové číslo x^* (pokud existuje), pro které platí

$$x^* = g(x^*).$$

Důvodem tohoto názvu je skutečnost, že x^* se po aplikaci funkce g nezmění. Zatímco u nelineární rovnice $f(x) = 0$ hledáme bod, v němž graf funkce f protíná osu x (tedy přímku $y = 0$), při hledání pevného bodu funkce g chceme najít bod, v němž graf funkce g protne diagonální přímku $y = x$. Úlohy na hledání pevného bodu dost často pocházejí přímo z praxe, ale pro nás zde mají význam také z toho důvodu, že řešení nelineární rovnice (1) lze zpravidla převést na hledání pevného bodu odpovídající nelineární funkce g, tedy na řešení rovnice (2). Metoda postupných aproximací (prosté iterace) pro řešení této rovnice je založena na opakovaném (iteračním) použití vzorce

$$x_{k+1} = g(x_k)$$

s vhodně zvoleným *počátečním přiblížením* (počáteční aproximací) x_0.

Chceme-li řešit rovnici $f(x) = 0$ metodou postupných aproximací, pak ji nejprve musíme převést na úlohu o pevném bodu pro nějakou vhodně vybranou funkci g. Takových možností bývá pro danou f více, ale ne všechny jsou stejně vhodné pro získání iteračního schématu k řešení výchozí rovnice. Výsledná iterační metoda se pro různé volby g může lišit nejen co do rychlosti konvergence, ale také v tom, zda vůbec konverguje či nikoli.

Příklad 8.6 (Úlohy na pevný bod)**.** *Nelineární rovnice*

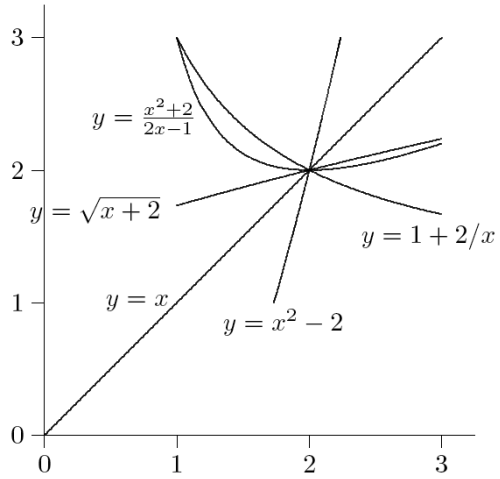
$$f(x) = x^2 - x - 2 = 0$$

má kořeny $x^* = 2$ a $x^* = -1$. Mezi ekvivalentní úlohy na hledání pevného bodu patří úlohy (2) s funkcemi (ověřte si to)

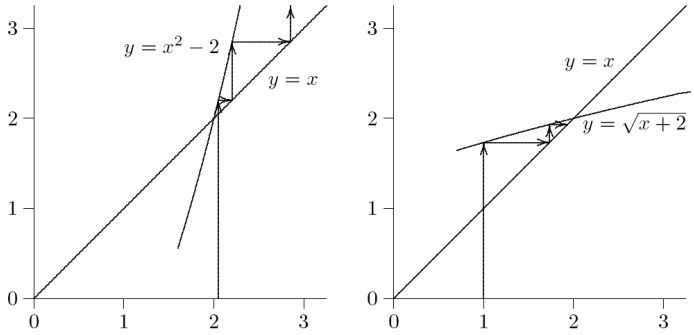
1. $g(x) = x^2 - 2$,
2. $g(x) = \sqrt{x + 2}$ *(ekvivalence pouze pro nezáporné pevné body, srv. (2)),*
3. $g(x) = 1 + (2/x)$,
4. $g(x) = (x^2 + 2)/(2x - 1)$.

Na obr. 8.3 je vykreslen průběh každé z těchto funkcí spolu s přímkou $y = x$. Všimněme si, že funkce g jsou konstruovány tak, že jejich grafy vesměs protínají přímku $y = x$ v pevném bodě $(2, 2)$.

Průběh příslušných iteračních schémat metody postupných aproximací je graficky znázorněn na Obrázcích 8.4 a 8.5. Šipka ve svislém směru odpovídá výpočtu hodnoty dané funkce v nějakém bodě a vodorovná šipka směřující k



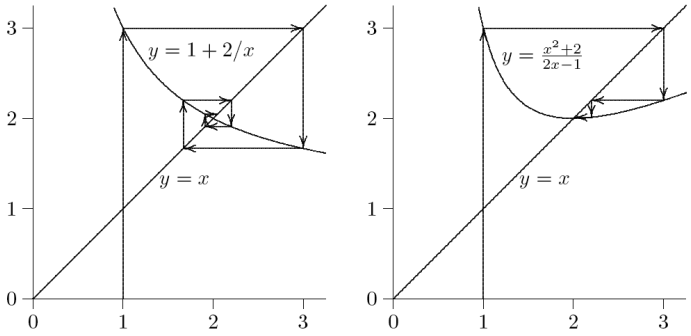
Obrázek 8.3: Pevný bod $(2, 2)$ nelineárních funkcí.



Obrázek 8.4: Metoda postupných aproximací pro první a druhou funkci g.

přímce $y = x$ vyznačuje, že se výsledek předchozího výpočtu hodnoty funkce g použije jeko argument pro příští výpočet funkční hodnoty. U první z uvedených funkcí vidíme, že i přes to, že počáteční bod je velmi blízko řešení, postupné aproximace divergují. U ostatních tří funkcí je vidět, že postupné iterace konvergují k pevnému bodu, i když byly odstartovány v bodě, který je od řešení relativně daleko. Zdá se přitom, že rychlosti konvergence pro tyto tři funkce se mohou lišit.

Jak lze z grafů funkcí na Obrázcích 8.4 a 8.5 vidět, chování metody prosté iterace se může značně odlišovat, od divergence přes pomalou konvergenci k rychlé konvergenci. Nejjednodušší (i když ne nejobecnější) způsob, jak charakterizovat chování iteračního schématu $x_{k+1} = g(x_k)$ pro řešení úlohy na pevný bod tvaru $x = g(x)$, je pokusit se vzít v úvahu derivaci funkce g v hledaném řešení x^* za předpokladu, že funkce g je hladká a tato derivace existuje. Dá



Obrázek 8.5: Metoda postupných aproximací pro třetí a čtvrtou funkci g.

se ukázat, že pokud $x^* = g(x^*)$ a $|g'(x^*)| < 1$, pak iterační schéma metody postupných aproximací **lokálně konverguje**. To znamená, že existuje nějaký interval obsahující x^* takový, že metoda prosté iterace s funkcí g konverguje, pokud je odstartována z nějakého x_0, jež leží uvnitř tohoto intervalu. Říkáme také, že metoda konverguje pro dostatečně blízké počáteční přiblížení. Naproti tomu pokud $|g'(x^*)| > 1$, pak metoda prosté iterace diverguje pro jakékoli počáteční přiblížení kromě x^*.

Důkaz tohoto tvrzení je založen na větě o střední hodnotě funkce, ale z časových důvodů jej zde neuvádíme, jakkoli není složitý (viz [2], [3] nebo [1]). Plyne z něj ale také to, že pokud metoda konverguje, je její asymptotická rychlost konvergence lineární s konstantou $C = |g'(x^*)|$. Čím menší je tato konstanta, tím je konvergence rychlejší, a ideální by tedy pro danou rovnici (8.1) bylo najít ekvivalentní formulaci (8.2) s funkcí g, pro niž by platilo $g'(x^*) = 0$. V takovém případě se dá pomocí Taylorova rozvoje opět poměrně snadno ukázat, že konvergence je nejméně kvadratická. V příštím odstavci si popíšeme jeden systematický způsob takové volby funkce g pro rovnici $f(x) = 0$.

Příklad 8.7 (Konvergence metody postupných aproximací)**.** *Pro čtyři úlohy na pevný bod z předcházejícího příkladu máme následující výsledky:*

1. $g'(x) = 2x$, *takže* $g'(2) = 4$ *a metoda postupných aproximací tedy diverguje.*
2. $g'(x) = 1/(2\sqrt{x+2})$, *takže* $g'(2) = 1/4$ *a metoda postupných aproximací konverguje linárně s konstantou* $C = 1/4$. *Kladné znaménko derivace* $g'(2)$ *vede k tomu, že se iterace přibližují k pevnému bodu z jedné strany.*
3. $g'(x) = -2/x^2$, *takže* $g'(2) = -1/2$ *a metoda postupných aproximací konverguje lineárně s konstantou* $C = 1/2$. *Záporné znaménko derivace* $g'(2)$ *vede k tomu, že se iterace přibližují k pevnému bodu po spirále, střídavě vždy z opačné strany.*

4. $g'(x) = (2x^2 - 2x - 4)/(2x - 1)^2$, *takže* $g'(2) = 0$ *a metoda postupných aproximací konverguje kvadraticky.*

Newtonova metoda

Metoda bisekce nepoužívá jiné vlastnosti funkčních hodnot než jejich znaménka, což vede k tomu, že konverguje vždy, ale pomalu. Pokud se využijí také velikosti funkčních hodnot, můžeme odvodit rychleji konvergující metody, které nám v každé iteraci budou dávat přesnější aproximaci kořene řešené rovnice. V první řadě se zde využívá aproximace funkce f vystupující v rovnici pomocí prvních dvou členů jejího Taylorova rozvoje, tedy

$$f(x + h) \approx f(x) + f'(x)h,$$

což je lineární funkce h, která aproximuje f v okolí bodu x. Nahradíme tudíž nelineární funkci f touto lineární funkcí, jejíž nulový bod v h se snadno vypočítá, je to $h = -f(x)/f'(x)$, pokud ovšem $f'(x) \neq 0$. Je jasné, že kořeny obou těchto funkcí nejsou obecně identické, takže popsaný postup musíme iteračně opakovat. To vede k iterační metodě, které se říká **Newtonova metoda** (nebo také *Newtonova-Raphsonova*), jejíž algoritmus uvádíme jako Algoritmus 2.

Algoritmus 2 Newtonova metoda

$x_0 =$ počáteční aproximace
for $k = 0, 1, 2, \ldots$
    $x_{k+1} = x_k - f(x_k)/g'(x_k)$
end

Ne obrázku 8.6 ukazujeme, že Newtonova metoda se dá interpretovat jako aproximace funkce f poblíž x_k tečnou ke grafu funkce vedenou v bodě $(x_k, f(x_k))$. Jako další aproximaci řešení pak bereme nulový bod této lineární tečné funkce a proces postupně opakujeme. Někdy se Newtonově metodě proto také říká metoda tečen.

Příklad 8.8 (Newtonova metoda)**.** *Newtonovu metodu předvedeme opět na hledání kořene rovnice*

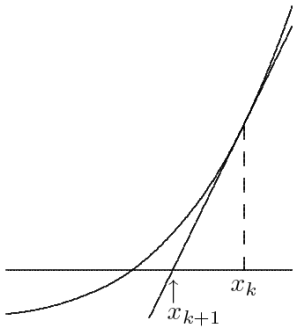
$$f(x) = x^2 - 4\sin x = 0.$$

Derivace této funkce je

$$f'(x) = 2x - 4\cos x,$$

takže iterační schéma je dáno vzorcem

$$x_{k+1} = x_k - \frac{x_k^2 - 4\sin x_k}{2x_k - 4\cos x_k}.$$



Obrázek 8.6: Newtonova metoda pro řešení nelineární rovnice.

Jako počáteční přiblížení zvolíme $x_0 = 3$ a postupně obdržíme posloupnost iterací, která je uvedena dále. Přitom $h_k = -f(x_k)/(x_k)$ označuje změnu x_k v každé iteraci. Iterační proces můžeme ukončit, když bude $|h_k|/|x_k|$ nebo $|f(x_k)|$, nebo obojí, menší než námi předepsaná tolerance.

k	x_k	$f(x_k)$	$f'(x_k)$	h_k
0	*3.000000*	*8.435520*	*9.959970*	*-0.846942*
1	*2.153058*	*1.294772*	*6.505771*	*-0.199019*
2	*1.954039*	*0.108438*	*5.403795*	*-0.020067*
3	*1.933972*	*0.001152*	*5.288919*	*-0.000218*
4	*1.933754*	*0.000000*	*5.287670*	*0.000000*

Na Newtonovu metodu se můžeme také dívat jako na speciální způsob převodu nelineární rovnice $f(x) = 0$ na úlohu o pevném bodě pro jistou funkci g, tedy $x = g(x)$, kde za funkci g volíme

$$g(x) = x - f(x)/f'(x).$$

a pevný bod hledáme metodou postupných aproximací. Abychom vyšetřili konvergenci metody, potřebujeme tedy nejprve znát derivaci funkce g, což je po úpravě

$$g'(x) = f(x)f''(x)/(f'(x))^2$$

(pokud $f'(x) \neq 0$). Je-li tedy x^* jednoduchý kořen, tj. $f(x^*) = 0$ a $f'(x^*) \neq 0$, pak $g'(x^*) = 0$. Newtonova metoda má tedy pro jednoduché kořeny asymptoticky kvadratickou rychlost konvergence, tedy $r = 2$.

Kvadratická rychlost konvergence Newtonovy metody znamená, že asymptoticky (v blízkosti kořene) se chyba metody po každé iteraci umocní na dru-

hou. Jinak také můžeme říci, že se počet přesných (správných) číslic přibližného řešení po každé iteraci zdvojnásobí. Naproti tomu pro násobné kořeny je Newtonova metoda pouze lineárně (lokálně) konvergentní s konstantou $C = 1 - (1/m)$, kde m je násobnost počítaného kořene. Opět ale musíme zdůraznit, že tyto úvahy o konvergenci platí pouze lokálně v nějakém větším nebo menším okolí hledaného kořene a že Newtonova metoda, která není odstartována dostatečně blízko ke kořeni, nemusí konvergovat vůbec. Jednoduchý příklad je situace, kdy někdy během iterací bude $f'(x_k)$ relativně malé (graf funkce f bude mít v bodě x_k téměř vodorovnou tečnu) a v důsledku toho bude následující iterace mít tendenci ležet někde daleko od posledního přiblížení.

Příklad 8.9 (Newtonova metoda pro násobný kořen)**.** *Následující dva příklady ukazují oba typy výše popsaného chování Newtonovy metody. První z nich ukazuje kvadratickou konvergenci k jednoduchému kořenu, druhý lineární konvergenci k násobnému kořenu. Násobnost kořene ve druhém z uvedených příkladů je 2, takže* $C = 1/2$.

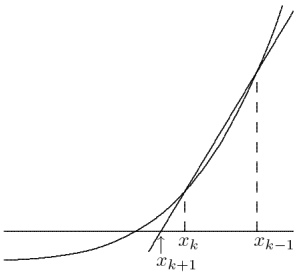
	$f(x) = x^2 - 1$	$f(x) = x^2 - 2x + 1$
k	x_k	x_k
0	*2.0*	*2.0*
1	*1.25*	*1.5*
2	*1.025*	*1.25*
3	*1.0003*	*1.125*
4	*1.00000005*	*1.0625*
5	*1.0*	*1.03125*

Metoda sečen

Jistou nevýhodou Newtonovy metody je, že za její kvadratickou konvergenci platíme tím, že v každém iteračním kroku musíme kromě funkční hodnoty počítat také hodnotu derivace. Výpočet hodnot derivace přitom může být nepohodlný nebo časově náročný, takže bychom mohli uvažovat o tom, že hodnoty derivací budeme nahrazovat diferenčními podíly vyplývajícími z definice derivace funkce, tedy bychom mohli pro vhodné dostatečně malé h klást například

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

To by ovšem znamenalo počítat v každé iteraci jednu funkční hodnotu navíc, a to jen proto, abychom získali přibližnou informaci o hodnotě derivace. Lepší je založit podobnou diferenční aproximaci derivace na funkčních hodnotách,



Obrázek 8.7: Metoda sečen pro řešení nelineární rovnice.

které jsme už během iterací stejně vypočítali, a klást

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

Tento postup vede k *metodě sečen*, jejíž algoritmus uvádíme jako Algoritmus 3. Na obrázku 8.7 vidíme, že metoda sečen se dá interpretovat jako aproximování funkce f přímkou procházející předchozími dvěma iteracemi, tedy sečnou, přičemž za nové přiblížení bereme nulový bod této lineární funkce. Na rozdíl od Newtonovy metody zde ovšem potřebujeme dvě počáteční aproximace.

Algoritmus 3 Metoda sečen

$x_0, x_1 =$ počáteční aproximace
for $k = 0, 1, 2, \ldots$
$\quad x_{k+1} = x_k - f(x_k)(x_k - x_{k-1})/(f(x_k) - f(xk - +))$
end

Příklad 8.10 (Metoda sečen)**.** *Metodu sečen budeme ilustrovat opět na hledání kořene rovnice*

$$f(x) = x^2 - 4\sin x = 0.$$

Za potřebná dvě počáteční přiblížení vezmeme $x_0 = 1$ a $x_1 = 3$, vypočítáme příslušné funkční hodnoty a za další přibližné řešení vezmeme průsečík přímky spojující tyto dvě funkční hodnoty s nulou. Celý postup pak opakujeme, přičemž použijeme toto nově získané přiblížení kořene a tu novější ze dvou předcházejících iterací, takže v každém iteračním kroku potřebujeme vypočítat pouze jednu novou funkční hodnotu. Posloupnost provedených iterací je uvedena v tabulce, kde h_k označuje změnu x_k v příslušné iteraci.

k	x_k	$f(x_k)$	h_k
0	*1.000000*	*-2.365884*	
1	*3.000000*	*8.435520*	*-1.561930*
2	*1.438070*	*-1.896774*	*0.286735*
3	*1.724805*	*-0.977706*	*0.305029*
4	*2.029833*	*0.534305*	*-0.107789*
5	*1.922044*	*-0.061523*	*0.011130*
6	*1.933174*	*-0.003064*	*0.000583*
7	*1.933757*	*0.000019*	*-0.000004*
8	*1.933754*	*0.000000*	*0.000000*

Protože každé nové přibližné řešení, které dává metoda sečen, závisí na *dvou* předchozích iteracích, je vyšetřování konvergence metody o něco složitější a detaily jsme nuceni zde vypustit. Uvádíme alespoň, že se dá dokázat, že chyby metody splňují pro jistou kladnou konstantu $c > 0$ vztah

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k| \cdot |e_{k-1}|} = c,$$

což znamená, že posloupnost iterací metodou sečen lokálně konverguje a rychlost konvergence je superlineární. Přesněji (viz [2], [3] a [1]) se dá ukázat, že asymptotická rychlost konvergence metody sečen je[1]

$$r = \frac{1+\sqrt{5}}{2} \approx 1{,}618.$$

Stejně jako u Newtonovy metody je i u metody sečen ke konvergenci nutno iterace odstartovat dostatečně blízko kořene.

Porovnáme-li metodu sečen s Newtonovou metodou, vidíme, že metoda sečen má výhodu v tom, že v každé iteraci potřebuje vypočítat pouze jednu novou funkční hodnotu. Za nevýhodu bychom mohli považovat to, že vyžaduje dvě startovací hodnoty a že vůči Newtonově metodě konverguje pomaleji, i když stále superlineárně. Menší pracnost provedení jedné iterace vyváží u metody sečen zpravidla to, že k dosažení konečného výsledku musíme provést větší počet iterací. Dá se tedy říci, že nalezení přibližné hodnoty řešení nelineární rovnice metodou sečen je často méně pracné než použití Newtonovy metody.

[1] Pokud vám to číslo připadá povědomé, připomínám, že je to hodnota zlatého řezu.

8.3 Dodatky

Bezpečné metody

Rychle konvergující metody pro numerické řešení nelineárních rovnic jako jsou například Newtonova metoda či metoda sečen (další takové metody lze najít v literatuře [2, 3, 1]) nejsou bezpečné v tom smyslu, že pokud nejsou odstartovány dostatečně blízko kořene, nemusí konvergovat. Bezpečnou metodou v tomto smyslu je metoda půlení intervalu, která je ale pomalá a tedy nákladná. Jakou metodu tedy volit?

Řešením tohoto dilematu jsou hybridní metody, které jsou zahrnuty ve většině moderního matematického softwaru a které v sobě kombinují vlastnosti obou výše popsaných typů metod. Jejich algoritmy jsou ale ovšem složitější. Tyto metody mohou například pracovat s rychle konvergentní metodou a přitom docílit toho, že iterace zůstávají uvnitř počáteční uzávěry kořene. Pokud následující aproximace řešení rychlým algoritmem padne mimo interval uzávěry, vrátíme se a provedeme jednu iteraci bezpečnou metodou, například bisekcí. Pak se může zkusit opět použitá rychlá metoda, tentokrát ovšem už na menším intervalu a s větší nadějí na úspěch. ke konci výpočtu už by měly iterace běžet tou rychlou metodou. Uvedený postup je jen zřídka horší než použitá pomalá metoda, zpravidla je mnohem rychlejší.

Populární implementace výše popsaného hybridního postupu dnes pochází od Brenta (v literatuře také tedy Brentova metoda) a kombinuje v sobě bezpečí bisekce s rychlejší konvergencí tzv. **inverzní kvadratické interpolace** (více k tomu viz [1]). Díky tomu, že se zde vyhýbáme Newtonově metodě, nejsou k výpočtu zapotřebí hodnoty derivace. Soudobý kvalitní software musí při implementaci metody vzít v úvahu také to, že se její algoritmus realizuje v počítačové aritmetice, tedy např. ohlídat možná překročení rozsahu počítače nebo nepřiměřeně přísné požadavky na přesnost výsledku. Dobrou implementaci výše popsaného postupu představuje například funkce `fzero` v Matlabu.

Poznamenáváme ještě, že jakousi kombinací metody bisekce a metody sečen je metoda *regula falsi* (z lat., doslova pravidlo falše). Každý její krok začíná tím, že body x_k a x_{k-+} tvoří uzávěru hledaného kořene, ale místo aby se v každém kroku interval uzávěry půlil, vypočítá se nejprve x_{k+1} pomocí vzorce metody sečen. Průběh funkce se tedy na daném intervalu opět nahradí sečnou. Pak se z takto získaných tří bodů zachovají ty dva, v nichž má funkce f opačná znaménka, a postup se opakuje. Metoda regula falsi je další vždy konvergentní metodou, musíme ji ovšem odstartovat z uzávěry kořene. Její konvergence je pouze lineární a může, ale nemusí být rychlejší než metoda půlení. Lze také ukázat, že v některých případech může jeden z krajních bodů uzávěry zůstávat během iterací trvale beze změny a ačkoli druhý bod konverguje ke kořenu rovnice, uzávěra se nemůže zmenšit pod jistou mez.

Numerický výpočet kořenů polynomu

Až dosud jsme se zabývali metodami pro nalezení jednoho nulového bodu obecné reálné funkce jedné reálné proměnné. Pokud je uvažovaná funkce polynom $p(x)$ stupně n, pak potřebujeme často najít všechny jeho nulové body, z nichž některé mohou být komplexní, i když polynom sám má reálné koeficienty. O kořenech polynomů nám algebraická teorie říká podrobnější informace než známe o nulových bodech obecných funkcí. Především je zde tzv. *základní věta algebry*, podle níž každý polynom stupně n má v komplexní rovině právě n nulových bodů (kořenů), pokud každý z nich počítáme tolikrát, kolik činí jeho násobnost. Dále se dá ukázat, že pokud má reálný polynom komplexní kořeny, vyskytují se tyto kořeny vždy ve dvojicích komplexně sdružených čísel, tedy jako $x \pm ßy$.

Pro hledání kořenů polynomů není nezbytné používat komplexní aritmetiku, leckdy lze počítat jejich reálné a imaginární části x a y odděleně. Pro výpočet kořenů polynomů existuje řada možností:

- Použijeme některou z popsaných obecných metod(např. Newtonovu metodu) a nalezneme jeden kořen x_1. Pak dále pracujeme s redukovaným polynomem $p(x)/(x - x_1)$, jehož stupeň je o jedničku nižší. Postup opakujeme tak dlouho, dokud nestanovíme všechny kořeny. Metoda se komplikuje, pokud narazíme na komplexní kořen.
- K danému polynomu sestavíme jeho *doprovodnou matici*, což je speciální matice mající vlastní čísla shodná s kořeny polynomu. Pak nějakou vhodnou numerickou metodou algebry stanovíme jako kořeny daného polynomu vlastní čísla této matice. Tento postup, který je použit ve funkci `roots` v Matlabu, je spolehlivý, ale není tak efektivní jako použití numerických metod odvozených speciálně pro výpočet kořenů polynomu.
- Použijeme některou ze speciálních metod pro výpočet nulových modů polynomů. Najdou se mezi nimi jak bezpečné metody, které izolují kořeny například ve sjednoceni disků v komplexní rovině (ty jsou ovšem podobně jako bisekce pouze lineárně konvergentní), tak rychle konvergující metody (i rychlejší než je Newtonova metoda). O těchto speciálních metodách se lze poučit například v [2, 3].

Numerické řešení soustav nelineárních rovnic

Řešení soustav nelineárních rovnic je obtížnější, než je tomu u jedné rovnice, a to z řady důvodů:

- Chování soustavy může být mnohem rozmanitější než chování jedné rovnice (a jejich kořenů). Teoretická analýza existence a počtu řešení je tak mnohem složitější.

- Konvenční metody používané pro jednu rovnici se leckdy dají víceméně přímočaře zobecnit i pro soustavy, ale u soustav není jednoduchý způsob, jak zobecnit pojem uzávěry řešení, takže zde není jednoduché sestrojit bezpečné, globálně konvergující metody. Určité možnosti zde ale existují, nicméně se vymykají možnostem tohoto textu a nenajdou se ani v běžných učebnicích. Nicméně v Matlabu je pro řešení soustav nelineárních rovnic k dispozici vcelku spolehlivá funkce `fsolve`.

- Pracnost numerického řešení soustav nelineárních rovnic roste nelineárně s počtem neznámých. Tak například jeden iterační krok Newtonovy metody pro soustavu o n neznámých znamená obecně výpočet n^2 hodnot derivací a jedno řešení souatavy n lineárních rovnic o n neznámých, což je samo o sobě obecně řádově n^3 aritmetických operací. Také organizační struktura algoritmů je mnohem složitější.

Jak jsme uvedli již na začátku tohoto textu, studium problematiky soustav se zde vymyká našim časovým možnostem. Úvodní informace může zájemce najít v doporučené literatuře [2], [3], [1].

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill, Boston, 2 edition, 2002.

[2] Stanislav Míka. *Numerické metody algebry.* MVŠT. SNTL, Praha, 1982.

[3] Stanislav Míka and Marek Brandner. *Numerické metody I.* FAV ZČU, Plzeň, 2. edition, 2002.

KAPITOLA 9

Numerická integrace

Pro detailnější obeznámení s pojmy, uváděnými níže, doporučuji i zde konzultovat knihu Michaela T. Heathe [1], případně nějakou z českých učebnic či mnoha skript o numerické matematice, která v posledních letech vyšla – například [2], [3], [4] (části tohoto skripta jsou dostupné i on-line). Mnohé ze zde použitých obrázků jsme převzali právě z [1].

9.1 Numerické metody výpočtu jednorozměrných integrálů

V této přednášce se budeme zabývat numerickými metodami pro (přibližný) výpočet jednorozměrných integrálů s konečnými mezemi, tedy integrálů $I(f)$ tvaru

$$I(f) = \int_a^b f(x)\mathrm{d}x, \tag{9.1}$$

kde $f : \mathbb{R} \to \mathbb{R}$ je reálná funkce jedné reálné proměnné, definovaná a integrovatelná na intervalu $[a, b]$ a a, b jsou daná reálná čísla. Počítat některé takové integrály explicitně v ruce jsme se naučili možná již na střední škole a pokud ne, pak v základních kursech matematiky na škole vysoké. Pro porozumění podstatě metod pro numerický výpočet integrálu je výhodné podívat se znovu na to, jak se definuje Riemannův jednorozměrný integrál funkce. Stejně jako v této definici, kde se hodnota integrálu definuje jako limita jistých vážených průměrů funkčních hodnot, se totiž většina numerických metod pro výpočet integrálů (říká se také **numerická kvadratura**) konstruuje jako vhodně sestavený vážený průměr určitého počtu navzorkovaných funkčních hodnot. Hlavním problémem je zde tedy volba bodů, v nichž se počítají (vzorkují) funkční hodnoty (říká se jim **uzly kvadratury** nebo **kvadraturního vzorce**) a stanovení vhodných koeficientů pro jejich lineární kombinaci

ve tvaru váženého průměru (**váhy kvadraturního vzorce**). Formálně má tedy obecný kvadraturní vzorec $Q_n(f)$ s n uzly tvar

$$Q_n(f) = \sum_{i=1}^{n} w_i f(x_i), \tag{9.2}$$

kde w_i jsou **váhy** nebo také **koeficienty** uvažovaného vzorce a kde budeme předpokládat, že pro uzly x_i plati $a \leq x_1 < x_2 < \cdots < x_n \leq b$. Říkáme, že kvadraturní vzorec je **otevřený**, pokud $a < x_1$ a $x_n < b$, a že je **uzavřený**, jestliže $a = x_1$ a $x_n = b$. Kvadraturní vzorec (9.2) nazýváme také n-*bodový kvadraturní vzorec*.

Formulace úlohy

Numerický výpočet integrálu tedy spočívá v tom, že řešení matematické úlohy (9.1), která má infinitesimální charakter (obecnou funkci nelze charakterizovat konečným počtem parametrů, limita v definici) přibližně nahradíme (aproximujeme) řešením numerické úlohy, totiž výpočtem hodnoty vhodného kvadratického vzorce (9.2), který má konečný počet uzlů a vah. Hlavním cílem numerických metod pro výpočet integrálu je pak volit uzly a váhy takovým způsobem, abychom dosáhli požadované úrovně přesnosti a přitom vynaložili pouze rozumné výpočetní úsilí, které je charakterizované především počtem výpočtů hodnot integrandu. Popravdě řečeno se velká část integrálů, které se vyskytují v praxi, v té či oné formě aproximuje numericky. Integrály, které jsme počítali v základním kurzu matematické analýzy, byly spíše ukázkové příklady na procvičení. Tak například už jednoduše vypadající integrál

$$I(f) = \int_0^1 \mathrm{e}^{-x^2} \mathrm{d}x$$

neumíme vypočítat analyticky.

Užitečnost numerického výpočtu integrálů vidíme na první pohled v oblasti geometrie a mechaniky, což jsou oblasti v nichž vlastně pojem integrálu vznikl, ale k aplikacím patří také mnohé další oblasti vědy a techniky, jako

- integrální transformace, jako je třeba Laplaceova transformace,
- výpočet hodnot speciálních funkcí v aplikované matematice a matematické fyzice (gamma funkce, Besselovy funkce, funkce chyb atd.), z nichž mnohé lze vyjádřit pomocí integrálů,
- metoda konečných a hraničních prvků pro řešení diferenciálních rovnic,
- integrální rovnice a variační metody,
- pravděpodobnost a statistika, kde jsou mnohé základní pojmy definovány pomocí integrálů,

- klasická a kvantová fyzika

a jistě i jiné.

Numerické metody výpočtu integrálu: podmíněnost a stabilita

Přirozený princip numerických metod pro výpočet integrálu vychází z našich znalostí o aproximaci funkcí (k tématu se vrátíme v pozdější části těchto textů). Postupujeme tak, že danou funkci f nahradíme nějakpu její aproximací φ, jejíž integrál umíme vypočítat analyticky a jako přibližnou hodnotu integrálu $I(f)$ použijeme integrál $I(\varphi)$. Jako aproximující funkce se typicky používají polynomy, a to jednak proto, že je snadné je explicitně integrovat, jednak také díky tzv. *Weierstrassově větě*, která velmi zhruba říká, že každá funkce spojitá na uzavřeném intervalu se dá libovolně přesně nahradit vhodným polynomem dostatečně vysokého stupně. A funkce, pro něž jsou běžné numerické kvadratury určeny, jsou především funkce spojité nebo po částech spojité.

Není těžké ukázat, že je-li aproximující funkce φ dobrým přiblížením funkce f na celém intervalu $[a, b]$, je integrál z φ dobrou aproximací integrálu z f, neboť

$$\left| \int_a^b f(x)\mathrm{d}x - \int_a^b \varphi(x)\mathrm{d}x \right| \leq \int_a^b |f(x) - \varphi(x)|\mathrm{d}x \leq (b-a) \sup_{x\in[a,b]} |f(x) - \varphi(x)|.$$

Odsud také plyne, že *absolutní číslo podmíněnosti* výpočtu integrálu vzhledem k poruchám ve funkčních hodnotách je $(b - a)$ a integrace je tedy vnitřně v tomto smyslu *dobře podmíněná*. Dá se ukázat, že pro relativní číslo podmíněnosti odsud ale dostáváme odhad

$$\mathrm{cond}(I(f)) \leq \frac{(b-a)\sup_{x\in[a,b]} |f(x)|}{|I(f)|},$$

který může nabývat velkých hodnot, jestliže počítáme integrál o malé absolutní hodnotě z funkce, která má velké funkční hodnoty. Je ovšem otázkou, zda v takovém případě (jmenovatel blízký nule), bychom i zde neměli používat spíše absolutní číslo podmíněnosti. Pokud jde o podmíněnost vzhledem k poruchám v integračních mezích, zde pouze řekneme (viz k tomu [1]), že absolutní podmíněnost je v zásadě dobrá, s výjimkou případů, kde funkce f má vně intervalu $[a, b]$ singularity poblíž koncových bodů (což nepřekvapuje).

Kromě přesnosti kvadraturního vzorce, kterou se budeme zabývat později (až popíšeme konkrétní vzorce), je třeba se zabývat také **stabilitou výpočtu**, tedy vlivem zaokrouhlovacích chyb a jiných poruch na výsledek výpočtu podle kvadraturních vzorců. Tato analýza stability se dá v daném případě provést

obecně. Jesliže $\hat{f}$ je funkce f porušená nějakými chybami, pak platí

$$\begin{aligned} |Q_n(\hat{f}) - Q_n(f)| &= |Q_n(\hat{f} - f)| \\ &= \left| \sum_{i=1}^{n} w_i(\hat{f}(x_i) - f(x_i)) \right| \\ &\leq \sum_{i=1}^{n} \Big(|w_i| \cdot |\hat{f}(x_i) - f(x_i)| \Big) \\ &\leq \left(\sum_{i=1}^{n} |w_i| \right) \sup_{x \in [a,b]} |\hat{f}(x) - f(x)|. \end{aligned}$$

Odsud je vidět, že absolutní číslo podmíněnosti kvadraturního vzorce je nanejvýš $\sum_{i=1}^{n} |w_i|$.

Je přirozené od kvadraturních vzorců požadovat, aby dávaly přesnou hodnotu integrálu alespoň pro konstantní funkce. Díky linearitě integrálu i kvadraturních vzorců stačí, aby tuto vlastnost měly pro funkci identicky rovnou jedné na $[a, b]$. Integrál z takové funkce je $b - a$, takže pro použitelné kvadraturní vzorce musí platit

$$\sum_{i=1}^{n} w_i = b - a.$$

Řekneme, že numerický algoritmus je **stabilní**, jestliže jeho podmíněnost je stejná jako je podmíněnost řešené úlohy nebo je s ní srovnatelná. Stabilní algoritmy tedy nezhoršují citlivost řešení na poruchy ve vstupních datech a během výpočtu. Pokud jde o námi uvažované kvadraturní vzorce, pak pokud jsou všechny váhy nezáporné, je tedy jeho absolutní číslo podmíněnosti nanejvýš $b - a$, což je srovnatelné s podmíněností řešené úlohy na výpočet integrálu. Kvadraturní vzorce s nezápornými vahami jsou tedy numericky stabilní. Na druhé straně, budou-li některé váhy záporné (takové vzorce se také vyskytují), může být absolutní číslo podmíněnosti vzorce mnohem větší a takový kvadraturní vzorec je pak nestabilní.

Algebraická přesnost kvadraturních vzorců

K dosažení požadované přesnosti za rozumnou cenu se musíme při konstrukci kvadraturních vzorců zabývat dvěma otázkami:

- Jak by měly být zvoleny vzorkovací body (uzly vzorce)?
- Jaké váhy bychom měli přisoudit jednotlivým vzorkům (funkčním hodnotám v uzlech)?

Vzhledem k tomu, co jsme řekli o aproximaci spojitých funkcí polynomy, je přirozené, že se při konstrukci kvadraturních vzorců používají dva základní postupy:

- snažíme se při předem daných uzlech stanovit váhy tak, aby vzorec přesně integroval polynomy co nejvyššího stupně; protože vzorec Q_n má při pevně daných uzlech n volných parametrů (vah), sestrojíme jej pokud možno tak, aby přesně integroval polynomy do stupně $n-1$ (ty mají totiž právě n koeficientů); pokud připustíme i libovolnou volbu uzlů, má Q_n celkem $2n$ parametrů a snažíme se, aby sestrojený vzorec přesně integroval polynomy do stupně $2n-1$,

- nebo sestavíme předem aproximaci obecné integrované funkce polynomem dostatečně vysokého (v praxi ale často i nízkého) stupně a tu pak zintegrujeme; použitá aproximace bude přirozeně využívat funkční hodnoty $f(x_i)$ v jistých uzlech x_i a dá se ukázat, že její integrál pak bude mít obecně opět tvar (2).

V souvislosti s tím, co jsme právě řekli, se zdá být užitečné zavést pojem algebraické přesnosti kvadraturního vzorce, který bude udávat maximální stupeň přesně integrovaných polynomů pro daný vzorec. Řekneme, že kvadraturní vzorec Q_n *má algebraickou přesnost* d (nebo také *je řádu* d), jestliže integruje přesně (tedy s nulovou chybou) všechny polynomy stupně d, ale není už přesný pro nějaký polynom stupně $d+1$. Ukážeme nyní, že při daném n lze vždy sestrojit kvadraturní vzorec algebraické přesnosti alespoň $n-1$.

Zvolme libovolně n uzlů kvadratury a pokusme se stanovit váhy $w_i, i = 1, 2, \ldots, n$, tak aby náš kvadraturní vzorec integroval přesně všechny polynomy až do stupně $n-1$. Protože každý polynom stupně $n-1$ je lineární kombinací bázových funkcí $1, x, x^2, \ldots, x^{n-1}$ a jak výpočet integrálu, tak výpočet kvadratury je lineární záležitost, stačí, aby náš vzorec integroval přesně tyto bázové funkce (říká se jim také *monomy*). Tento požadavek nám ihned dává soustavu n lineárních algebraických rovnic pro váhy w_i (pamatujme, že uzly jsme pevně zvolili předem; jediný požadavek je, aby byly vzájemně různé), kterým se také říkává *momentové rovnice*:

$$\begin{aligned} w_1 \cdot 1 + w_2 \cdot 1 + \cdots + w_n \cdot 1 &= \int_a^b 1 \mathrm{d}x = b - a, \\ w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n &= \int_a^b x \mathrm{d}x = (b^2 - a^2)/2, \\ &\vdots \\ w_1 \cdot x_1^{n-1} + w_2 \cdot x_2^{n-1} + \cdots + w_n \cdot x_n^{n-1} &= \int_a^b x^{n-1} \mathrm{d}x = (b^n - a^n)/n. \end{aligned} \tag{9.3}$$

Čtvercová matice této soustavy (napište si ji) se nazývá **Vandermondova matice** a je o ní známo, že pokud jsou čísla x_i navzájem různá, je regulární. Soustava (9.3) má tedy právě jedno řešení a jejím vyřešením můžeme získat hledané váhy a dokončit tak konstrukci kvadraturního vzorce $Q_n(f)$ s alge-

braickou přesností nejméně $n-1$ (díky dalším vlastnostem získaného vzorce může být jeho řád i o něco vyšší).

Příklad 9.1 (Výpočet vah kvadraturního vzorce)**.** *Právě popsaný postup založený na řešení soustavy* (9.3) *popíšeme na odvození tříbodového kvadraturního vzorce*

$$Q_3(f) = w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3)$$

pro integraci přes interval $[a,b]$ *Za tři uzly vzorce vezmeme dva krajní body s střed intervalu, tj. položíme* $x_1 = a, x_2 = (a+b)/2, x_3 = b$. *Soustavu 3 lineárních rovnic pro* w_1, w_2, w_3 *zde nevypisujeme, čtenář si ji snadno může sestavit sám podle (**??**). Soustavu vyřešíme, například bez problémů Gaussovou eliminací, a dostaneme váhy*

$$w_1 = \frac{1}{6}(b-a), w_2 = \frac{2}{3}(b-a), w_3 = \frac{1}{6}(b-a).$$

Výsledný kvadraturní vzorec je znám jako Simpsonovo pravidlo a dá se ukázat, že (díky své symetrii) je přesný dokonce pro polynomy třetího stupně.

Pokud uzly nezadáme předem pevně a necháme je také jako volné parametry, nebude soustava (9.3) už soustavou lineárních rovnic, ale budeme místo ní mít soustavu nelineárních rovnic s neznámými w_i i x_i. Protože je zde více volných parametrů, bude také více těchto rovnic, pokud se nám je ale podaří analyticky nebo numericky vyřešit, můžeme při daném n dosáhnou v podstatě dvojnásobné algebraické přesnosti. Na takovém přístupu jsou založeny Gaussovy kvadraturní vzorce, k nimž se ještě krátce vrátíme později.

Při pevně zvolených uzlech je ale místo výše popsaného a příkladem ilustrovaného postupu běžnější používat postup druhý, totiž zaměřit se na náhradu integrované funkce vhodným aproximujícím polynom a ten integrovat. Na tomto principu jsou založeny Newtonovy-Cotesovy vzorce, které v další kapitole popíšeme podrobněji. Patří k nim také Simpsonovo pravidlo z našeho předchozího příkladu.

Konvergence kvadraturních vzorců

Vzpomeneme-li si na definici Riemannova integrálu, kde hodnota integrálu je u integrovatelné funkce limitou Riemannových vážených průměrů při počtu vzorkovacích bodů rostoucím do nekonečna, můžeme čekat, že podobně se budou chovat i alespoň některé posloupnosti kvadraturních vzorců při $n \to \infty$. Při dané funkci f budeme posloupnost kvadraturních vzorců $\{Q_n(f), n = 1, 2, \dots\}$ také nazývat **kvadraturou**. Řekneme pak, že daná posloupnost vzorců tvoří na intervalu $[a,b]$ **konvergentní kvadraturu**, jestliže pro *každou* funkci f, která je na $[a,b]$ spojitá, platí

$$\lim_{n\to\infty} Q_n(f) = \int_a^b f(x)\mathrm{d}x.$$

Pro pořádek a pro ty, kdo o aproximaci funkcí již vědí více, uvádíme, že pojem konvergentní kvadratury vyžaduje platnost uvedeného limitního vztahu pro každou funkci, která je spojitá, a další požadavky se zde na ni nekladou. Pokud budeme vědět, že funkce f má například ohraničené všechny derivace na $[a, b]$, mohou pro ni konvergovat i takové posloupnosti kvadraturních vzorců, jež v našem smyslu konvergentní kvadraturu netvoří. Posloupnost Riemannových součtů je tedy z našeho hlediska konvergentní kvadratura.

V předchozích odstavcích jsme naznačili, že je v zásadě možné konstruovat posloupnosti kvadraturních vzorců $\{Q_n(f), n = 1, 2. \dots\}$ takové, že s roustoucím n jejich algebraická přesnost poroste. Bylo by proto přirozené čekat, že vezmeme-li takovou posloupnost, bude tvořit konvergentní kvadraturu. To ale není obecně pravda, ukazuje se, že podstatnou roli přitom hraje to, jakým způsobem na $[a, b]$ rozmisťujeme vzorkovací body (uzly kvadratury). O příkladech konvergentních a nekonvergentních kvadratur se zmíníme ještě později.

V praxi se při výpočtu snažíme minimalizovat výpočetní práci, takže zde posloupnosti kvadraturních vzorců volíme především tak, aby vzorce byly do sebe vnořené. Přesněji, říkáme, že daná kvadratura je **vnořená** nebo **progresívní**, jestliže při $m > n$ tvoří uzly Q_n podmnožinu uzlů Q_m. To tedy znamená, že již spočítaných n funkčních hodnot můžeme v $Q_m(f)$ znovu použít a potřebujeme tedy vypočítat pouze $m - n$ nových funkčních hodnot, čímž šetříme.

Dá se ukázat, že kromě zvyšování řádu kvadratury zvyšováním počtu uzlů je smysluplné také postupovat způsobem obdobným, jako použil Riemann ve své definici integrálu, kde mezi každými dvěma dělicími body aproximoval funkci se stále stejnou řádovou přesností (konstantou), a nezvyšoval tedy algebraickou přesnost, ale pouze počet vzorkovacích bodů. Přesnost lze tedy zvyšovat nejen zvyšováním algebraického řádu, ale také tak, že vyjdeme z jednoho *základního* kvadraturního vzorce, interval integrace postupně dělíme na malé části (tak zvané **panely**) a daný základní kvadraturní vzorec aplikujeme postupně na každém panelu. Se zjemňováním sítě panelů tak můžeme docílit požadované přesnosti, aniž bychom zvyšovali řád kvadratury. Tímto způsobem (rozmyslete si) dostáváme opět posloupnost kvadraturních vzorců, která je založena na jednom vzorci základním. Takto zkonstruovaným kvadraturním vzorcům se pak říká vzorce **složené**. Zaměříme se nyní nejprve na popis vybraných základních kvadraturních vzorců.

9.2 Newtonovy-Cotesovy vzorce

Nejjednodušší způsob, jak rozložit vzorkovací body na intervalu $[a, b]$ je bezesporu rozložit je rovnoměrně, *ekvidistantně* (ve stejné vzdálenosti). Při takovémto rozložení uzlů pak vznikají kvadraturní vzorce, kterým se říká **Newtonovy-Cotesovy vzorce**. Budeme-li chtít sestrojit n-bodový **otevřený** Newtonův-

Cotesův vzorec, použijeme jako uzly body

$$x_i = a + i(b-a)/(n+1),\ i = 1, \ldots, n,$$

kdežto **uzavřený** n-bodový Newtonův-Cotesův vzorec bude mít uzly

$$x_i = a + (i-1)(b-a)/(n-1),\ i = 1. \ldots, n.$$

Příklady Newtonových-Cotesových vzorců

Uvádíme tři příklady nejjednodušších a nejznámějších často používaných Newtonových-Cotesových vzorců.

Příklad 9.2 (Obdélníkové pravidlo)**.** *Pokud integrovanou funkci nahradíme na $[a, b]$ konstantou (tedy polynomem nultého stupně) rovnou funkční hodnotě ve středu intervalu a tuto konstantu zintegrujeme přes $[a, b]$, dostaneme jednobodový otevřený Newtonův-Cotesův vzorec*

$$M(f) = (b-a) f\left(\frac{a+b}{2}\right),$$

kterému se říká ***obdélníkové pravidlo*** *(angl. midpoint rule).*

Příklad 9.3 (Lichoběžníkové pravidlo)**.** *Pokud integrovanou funkci nahradíme na $[a, b]$ lineární funkcí spojující její hodnoty v krajních bodech (přímkou, tedy polynomem prvního stupně) a tuto lineární funkci zintegrujeme, dostaneme dvoubodový uzavřený Newtonův-Cotesův vzorec*

$$T(f) = \frac{b-a}{2}(f(a) + f(b)),$$

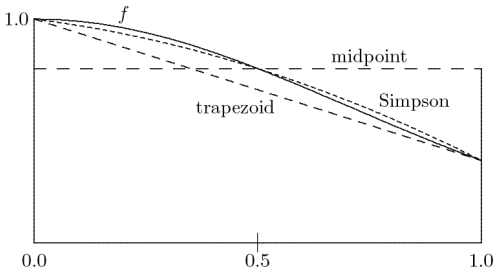
kterému se říká ***lichoběžníkové pravidlo*** *(angl. trapezoid rule).*

Příklad 9.4 (Simpsonovo pravidlo)**.** *Pokud integrovanou funkci nahradíme na $[a, b]$ kvadratickou funkcí (tedy parabolou, polynomem druhého stupně), která má stejné hodnoty jako f v krajních bodech intervalu $[a, b]$ a v jeho středu, a tento polynom druhého stupně zintegrujeme, dostaneme tříbodový uzavřený Newtonův-Cotesův vzorec*

$$S(f) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right),$$

kterému se říká ***Simpsonovo pravidlo****. Setkali jsme se s ním již v příkladu 9.1.*

Použití tří uvedených Newtonových-Cotesových kvadraturních vzorců ilustrujeme na příkladu.



Obrázek 9.1: Integrace funkce $f(x) = \mathrm{e}^{-x^2}$ Newtonovými-Cotesovými kvadraturními pravidly.

Příklad 9.5 (Newtonova-Cotesova kvadratura)**.** *Budeme aproximovat integrál*

$$I(f) = \int_0^1 \mathrm{e}^{-x^2}\,dx$$

pomocí každého z tří jednoduchých Newtonových-Cotesových vzorců, jež jsme právě popsali. Dostaneme

$$\begin{aligned} M(f) &= (1-0)\exp(-0{,}25) \approx 0{,}778801, \\ T(f) &= \frac{1}{2}(\exp(0)+\exp(-1)) \approx 0{,}683940, \\ S(f) &= \frac{1}{6}(\exp(0)+4\exp(-0{,}25)+\exp(-1)) \approx 0{,}747180. \end{aligned}$$

Na Obrázku 9.1 je zobrazen průběh integrandu a tří použitých aproximujících polynomů. Přesná hodnota integrálu zaokrouhlená na 6 platných cifer je 0,746824*. Může se zdát poněkud překvapivým, že velikost chyby lichoběžníkového pravidla (*0,062884*) je asi dvakrát tak velká, jako je tomu u obdélníkového pravidla (*0,031997*); k tomu se ještě vzápětí vrátíme. Simpsonovo pravidlo s chybou* 0,000356 *se zdá být pozoruhodně přesné, uvážíme-li, že je použito na poměrně velkém intervalu délky 1.*

Vlastnosti Newtonových-Cotesových vzorců

Pro chybu Newtonových-Cotesových vzorců se u hladkých funkcí s dostatečným počtem spojitých derivací na $[a, b]$ dají odvodit obecné odhady. Odvození se provádí tak, že se integrovaná funkce rozvine do Taylorovy řady; nebudeme jej zde však provádět a uvedeme pouze některé výsledky.

Pro obdélníkové pravidlo dostaneme (značíme zde $m = (a+b)/2$)

$$I(f) = f(m)(b-a) + \frac{f''(m)}{24}(b-a)^3 + \frac{f^{(4)}(m)}{1920}(b-a)^5 + \cdots = M(f) + E(f) + F(f) + \cdots,$$

kde $E(f)$ a $F(f)$ reprezentují první dva členy rozvoje chyby pro obdélníkové pravidlo.

Pro lichoběžníkové pravidlo nám podobným způsobem vyjde

$$I(f) = T(f) - 2E(f) - 4F(f) - \cdots$$

a pro Simpsonovo pravidlo dostaneme

$$I(f) = S(f) - \frac{2}{3}F(f) + \cdots .$$

Odečtením rozvojů pro lichoběžníkové pravidlo a obdélníkové pravidlo odsud dostaneme praktický asymptotický vzorec pro odhad hlavního členu chyb u těchto dvou kvadraturních vzorců. Vyjde nám totiž (po úpravě a zanedbání členů vyšších řádů)

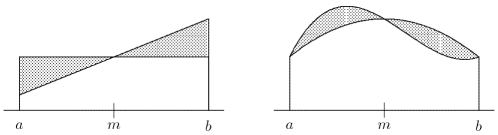
$$E(f) \approx \frac{T(f) - M(f)}{3}. \tag{9.4}$$

Tento vzorec ovšem platí za předpokladu, že délka intervalu integrace je malá (tak, aby platilo $(b-a)^5 \ll (b-a)^3$) a že funkce f je taková, že její čtvrtá derivace $f^{(4)}$ se chová „rozumně“. Za těchto předpokladů pak můžeme z dosavadních úvah pro tyto dva kvadraturní vzorce dospět k následujícím závěrům:

- Obdélníkové pravidlo je zhruba dvakrát tak přesné jako pravidlo lichoběžníkové (viděli jsme to již v příkl.9.5), přesto, že je založeno na aproximaci funkce f polynomem menšího stupně.
- Rozdíl hodnot získaných obdélníkovým a lichoběžníkovým pravidlem se dá využít o odhadu chyby každého z těchto kvadraturních vzorců.
- Snížíme-li délku integračního intervalu na polovinu, zmenší se chyba aproximace u každého z těchto vzorců faktorem cca 1/8.

Příklad 9.6 (Odhad chyby)**.** *Vraťme se k příkl. 9.5, kde jsme obdélníkovým a lichoběžníkovým pravidel počítali přibližné hodnoty integrálu. Dosadíme-li získané hodnoty do přibližného vzorce (9.4), dostaneme jako odhad hlavního členu chyby* $E(f) \approx -0.031620$*, což na tři desetinná místa dobře souhlasí se skutečnými velikostmi chyb uvedenými v příkl. 9.5.*

V předchozím jsme ukázali, že n-bodový kvadraturní vzorec lze sestrojit tak, aby jeho algebraická přesnost byla nejméně $n-1$. Mohli bychom tedy očekávat, že algebraická přesnost obdélníkového pravidla bude nula, lichoběžníkového pravidla jedna, Simpsonova pravidla dvě atd. To koneckonců souhlasí s tím, že tyto vzorce jsme odvodili pomocí náhrady integrované funkce polynomy stupně nula, jedna a dvě. Podíváme-li se ale na výše uvedené rozvoje chyb, vidíme, že chyba obdélníkového pravidla závisí na derivacích řádu dvě a vyšších, které jsou ale rovny nule nejen pro konstantu, ale i pro polynomy



Obrázek 9.2: Kancelace chyb u obdélníkového (vlevo) a Simpsonova (vpravo) pravidla.

prvního stupně. Obdélníkové pravidlo tedy integruje přesně nejen konstanty, ale i lineární funkce a jeho řád je tedy o jedničku větší než nula. Podobně u Simpsonova pravidla závisí chyba na derivacích integrandu řádu čtyři a vyšších, které se anulují nejen pro kvadratické, ale i pro kubické polynomy, takže Simpsonovo pravidlo je řádu tři, nikoli pouze dva (to také vysvětluje překvapivě dobrý výsledek získaný v příkl. 9.5).

Obecně se dá ukázat, že pro každé liché n má n-bodový Newtonův-Cotesův vzorec algebraickou přesnost n a nikoli $n-1$. Tento jev, který plyne z rozvojů pro chybu, je také možné vykládat jako *kancelaci* (vzájemné rušení) kladných a záporných složek chyby aproximace, což ilustrujeme na obr.9.2 pro případ obdélníkového a Simpsonova pravidla. Na obrázku vidíme vlevo lineární polynom a konstantní funkci danou jeho hodnotou ve středu intervalu, vpravo je kubický polynom a kvadratická funkce, která se s ním shoduje v krajních bodech a ve středu. Integrace lineárního polynomu obdélníkovým pravidlem vede ke dvěma trojúhelníkovým oblastem, které jsou stejně velké. Vliv jednoho z těchto trojúhelníků se přesně vyruší s vlivem trojúhelníku druhého. Podobně je tomu u kubického polynomu, kde obě stínované oblasti mají rovněž stejný obsah, takže se jejich vliv vzájemně vyruší. K takové kancelaci ale nedochází u Newtonových-Cotesových vzorců se sudým počtem uzlů. Souhrnně tedy můžeme říci, že algebraická přesnost n-bodového Newtonova-Cotesova kvadraturního vzorce je $n-1$ při n sudém, ale je rovna n při n lichém.

Newtonovy-Cotesovy vzorce se poměrně snadno odvozují a používají, ale mají také jisté závažné nevýhody. Příčinou těchto nevýhod je především skutečnost, že aproximace spojitých funkcí polynomy vysokých stupňů, které nabývají stejných hodnot jako aproximovaná funkce na rovnoměrné síti uzlů, mohou vykazovat nežádoucí oscilace. To pak vede k tomu, že Newtonova-Cotesova kvadratura při $n \to \infty$ nepředstavuje obecně (pro každou spojitou funkci) konvergentní kvadraturu. Pro konečná n je dále známo, že každý n-bodový Newtonův-Cotesův kvadraturní vzorec má při $n \geq 11$ alespoň jednu zápornou váhu. Není u nich tedy zaručena stabilita. Skutečnost je ještě horší, neboť se dá ukázat, že při $n \to \infty$ platí $\sum_{i=1}^{n} |w_i| \to \infty$, což znamená, že při růstu počtu uzlů a odpovídajícím růstu řádu Newtonových-Cotesových kvadraturních vzorců se libovolně zhoršuje jejich podmíněnost a tedy stabilita výpočtu. Přítomnost velkých kladných a záporných vah také znamená, že se hodnota

integrálu počítá jako součet velkých hodnot majících opačná znaménka, takže zde v konečné počítačové aritmetice může docházet k výrazné kancelaci.

Z důvodů, které jsme právě uvedli, je vidět, že nemůžeme čekat, že bychom na daném intervalu dosáhli libovolně velké přesnosti tak, že bychom postupně zvětšovali počet uzlů (vzorků) a používali Newtonovy-Cotesovy vzorce stále vyšších řádů. V praxi se proto p5i používání Newtonových-Cotesových vzorců obvykle omezujeme na základní vzorce s nevelkým počtem uzlů a pokud požadujeme vyšší přesnost, dělíme interval integrace na dílčí subintervaly (panely) a zvolený kvadraturní vzorec pak aplikujeme na každém z těchto panelů samostatně (k takovým postupům se ještě později vrátíme), takže tak vytváříme složený kvadraturní vzorec. Z tohoto hlediska je pozitivním rysem Newtonových-Cotesových kvadraturních vzorců, že jsou progresívní, takže při zvyšování počtu uzlů můžeme k jemnějšímu vzorkování využít funkční hodnoty jiže vypočítané dříve. Na druhé straně ale nemají Newtonovy-Cotesovy vzorce při daném n (a tedy daném počtu vzorků) největší možnou algebraickou přesnost (a tedy ani přesnost obecně), což je dáno tím, že jsme z $2n$ parametrů vzorce n parametrů (uzly) pevně zvolili předem. Popíšeme si nyní kvadraturní vzorce, které jsou z tohoto ohledu mnohem lepší.

9.3 Gaussovy kvadraturní vzorce

V kvadraturních vzorcích, které jsme dosud viděli, bylo všech n uzlů zadáno předem a n odpovídajících vah se pak hledalo tak, abychom dosáhli co největší algebraické přesnosti. Poněvadž jsme tedy měli pouze n volných parametrů, byl výsledný řád vzorce obecně $n - 1$. Pokud ale uvolníme také rozmístění uzlů, budeme mít $2n$ volných parametrů, takže by mělo být možné dosáhnout algebraické přesnosti $2n - 1$.

Odvození Gaussových kvadraturních vzorců

U *Gaussovy kvadratury* se volí jak váhy, tak uzly tím způsobem, že ve výsledném kvadraturním vzorci je dosaženo maximálního možné algebraické přesnosti. Při daném počtu uzlů tedy Gaussovy vzorce poskytují maximální možnou přesnost, ale na druhé straně je podstatně obtížnější je odvodit než tomu bylo při pevně zvolených uzlech u Newtonových-Cotesových vzorců. Důvodem je skutečnost, že soustava rovnic pro uzly a váhy má sice stejný tvar jako (9.3) (rovnic je ovšem dvakrát tolik, máme dvojnásobek neznámých), ale díky tomu, že neznámými jsou také uzly, je tato soustava momentových rovnic tentokrát nelineární. Řešit tuto soustavu pro obecný interval $[a, b]$ je nepohodlné, a proto se základní Gaussovy vzorce odvozují pro některý konkrétní interval, typicky třeba pro $[-1, 1]$. Na obecný interval $[a, b]$ se pak snadno transformují jednoduchou lineární transformací, ke které se vrátíme později. Při konstrukci Gaussových kvadraturních vzorců nejde jenom o to soustavu momentových

rovnic vyřešit. Je zde předem nutno zodpovědět některé teoretické otázky týkající se této soustavy, totiž:

- Má daná soustava momentových rovnic řešení?
- Je toto řešení jediné? Pokud ne, jsou jednotlivá řešení pouze permutacemi řešení ostatních?
- Jsou tato řešení reálná a padnou uzly do intervalu $[-1, 1]$?
- Jaká znaménka mají získané váhy?

Odpovědi na tyto otázky jsou vesměs příznivé a ukazuje se, že pro každé n existuje právě jeden Gaussův kvadraturní vzorec a všechny jeho váhy jsou kladná čísla.

Příklad 9.7 (Gaussův kvadraturní vzorec)**.** *Odvodíme dvoubodový Gaussův kvadaturní vzorec na intervalu* $[-1, 1]$

$$I(f) = \int_{-1}^{1} f(x)dx \approx w_1 f(x_1) + w_2 f(x_2),$$

kde se uzly x_1, x_2 *a stejně tak váhy* w_1, w_2 *budou volit tak, aby se maximalizoval řád vzorce. Máme čtyři volné parametry, takže budeme požadovat, aby vzorec integroval přesně první čtyři monomy a tím pádem všechny polynomy do třetího stupně. Stejně jako dříve sestavíme čtyři momentové rovnice*

$$\begin{aligned} w_1 + w_2 &= \int_{-1}^{1} 1\,dx &&= 2, \\ w_1 x_1 + w_2 x_2 &= \int_{-1}^{1} x\,dx &&= 0, \\ w_1 x_1^2 + w_2 x_2^2 &= \int_{-1}^{1} x^2\,dx &&= \frac{2}{3}, \\ w_1 x_1^3 + w_2 x_2^3 &= \int_{-1}^{1} x^3\,dx &&= 0. \end{aligned}$$

Jedním řešením této nelineární soustavy rovnic jsou hodnoty

$$x_1 = -1/\sqrt{3}, x_2 = 1/\sqrt{3}, w_1 = 1, w_2 = 2.$$

Existuje ještě jedno řešení, které získáme prohozením znamének u x_1 *a* x_2*, takže toto řešení dává identický Gaussův vzorec. Dvoubodový Gaussův kvadraturní vzorec má tedy tvar*

$$G_2(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3})$$

a jeho algebraická přesnost je tři.

Alternativní způsob, jak předem získat uzly Gaussových kvadraturních vzorců spočívá ve využití *ortogonálních polynomů*. Řekneme, že dva polynomy $p(x)$ a $q(x)$ jsou na intervalu $[a, b]$ *ortogonální*, jestliže platí

$$\int_a^b p(x)q(x)\mathrm{d}x = 0.$$

Nechť p je polynom stupně n takový, že je na $[a, b]$ ortogonální ke všem monomům menšího stupně, neboli nechť platí

$$\int_a^b p(x)x^k\mathrm{d}x = 0,\ k = 0, \ldots, n-1,$$

takže p je na $[a, b]$ ortogonální vzhledem ke všem polynomům stupně menšího než n. Pak se dá ukázat, že platí:

1. Všechny kořeny polynomu p jsou jednoduché (je jich tedy n různých), reálné a leží v otevřeném intervalu (a, b).

2. Při uzlech zvolených jako kořeny polynomu p lze sestrojit již popsaným způsobem kvadraturní vzorec, jehož váhy jsou řešením lineární soustavy momentových rovnic (**??**). Tyto váhy jsou kladné a řád takto získaného kvadraturního vzorce je $2n - 1$; je to tedy nutně jednoznačně určený n-bodový Gaussův kvadraturní vzorec.

Teorie a konstrukce ortogonálních polynomů jsou v matematice dobře zpracovány, ale téma se vymyká možnostem tohoto textu. Pro zájemce pouze uvádíme, že vhodnými ortogonálními polynomy jsou zde tzv. Legendrovy polynomy P_n a že se díky tomu vzniklé kvadraturní vzorce nazývají také Gaussovy-Legendrovy kvadraturní vzorce. Jakkoli jsou tedy Legendrovy polynomy známou věcí, zbývá zde ještě provést výpočet jejich kořenů; teprve pak můžeme stanovit váhy kvadratury ze soustavy momentových rovnic. Touto tématikou, která je také dobře zpracována teoreticky i algoritmicky, se zde však opět nemůžeme podrobněji zabývat. Zájemce odkazujeme na dostupnou literaturu, například na [2], [3], [4] nebo [1].

Příklad 9.7 je typický v tom smyslu, že pro všechna n jsou gaussovské uzly rozloženy symetricky kolem středu intervalu; pro lichá n je střed intervalu vždy sám také uzlem. Příklad 9.7 je typický také v tom, že uzly jsou většinou iracionální čísla, i když koncové body a a b jsou racionální. Dá se říci, že tento rys může činit Gaussovu kvadraturu poněkud nepohodlnou pro ruční výpočty, pokud ji totiž srovnáme s jednoduchými Newtonovými-Cotesovými vzorci. Ovšem na počítači jsou obvykle uzly a váhy Gaussových kvadraturních vzorců tabelovány předem a obsaženy v podprogramech, které se podle potřeby vyvolávají, takže uživatel ani nemusí znát jejich hodnoty natož je počítat.

Gaussovy kvadraturní vzorce pro obecný interval

Použití Gaussových kvadraturních vzorců je poněkud komplikovanější než je tomu u vzorců Newtonových-Cotesových také proto, že jejich váhy a uzly se odvozují a udávají pro konkrétní interval, jako je například $[-1,1]$. Tím pádem je třeba obecný interval integrace $[a,b]$ transformovat na tento standardní interval, pro nějž jsou tabelovány hodnoty uzlů a vah. Pokud tedy chceme použít (platí to obecně, nejen pro Gaussovu kvadraturu) kvadraturní vzorec, jehož parametry jsou udány pro interval $[\alpha,\beta]$,

$$\int_\alpha^\beta f(x)\mathrm{d}x \approx \sum_{i=1}^n w_i f(x_i),$$

k aproximaci integrálu přes interval $[a,b]$,

$$I(g) = \int_a^b g(t)\mathrm{d}t,$$

musíme provést substituci, jež bude transformovat x v intervalu $[\alpha,\beta]$ na t v intervalu $[a,b]$. Takových substitucí existuje celá řada, ale my budeme používat jednoduchou lineární transformaci

$$t = \frac{(b-a)x + a\beta - b\alpha}{\beta-\alpha},$$

která zobrazuje oba intervaly na sebe vzájemně jednoznačně a má tu přednost, že zachovává řád kvadraturního vzorce. Počítaný integrál je pak

$$\begin{aligned} I(g) &= \frac{b-a}{\beta-\alpha}\int_\alpha^\beta g\left(\frac{(b-a)x+a\beta-b\alpha}{\beta-\alpha}\right)\mathrm{d}x \\ &\approx \frac{b-a}{\beta-\alpha}\sum_{i=1}^n w_i g\left(\frac{(b-a)x_i+a\beta-b\alpha}{\beta-\alpha}\right). \end{aligned}$$

Příklad 9.8 (Změna intervalu)**.** *Jako ilustraci právě popsaného postupu použijeme dvoubodový Gaussův kvadraturní vzorec G_2 z příkl. 9.7 odvozený pro interval $[-1,1]$ k přibližnému výpočtu integrálu*

$$I(f) = \int_0^1 \mathrm{e}^{-t^2}\,dt$$

z příkladu 9.5. Právě popsaná lineární transformace má v tomto případě tvar

$$t = \frac{x+1}{2},$$

takže náš integrál se aproximuje jeko $G_2(g) =$

$$\frac{1}{2}\left[\exp\left(-\left(\frac{(-1/\sqrt{3})+1}{2}\right)^2\right)+\exp\left(-\left(\frac{(1/\sqrt{3}*1)}{2}\right)^2\right)\right] \approx 0.746595,$$

což je o něco přesnější výsledek než ten, který jsme v příkl. 9.5 pro tento integrál obdrželi Simpsonovým pravidlem, přestože jsme zde použili pouze dvě funkční hodnoty místo tří.

Některé vlastnosti Gaussových kvadraturních vzorců

Shrneme zde některé důležité vlastnosti Gaussovy kvadratury, o nichž jsme se dosud zmínili:

- Gaussovy kvadraturní vzorce lze sestrojit pro každé n, a to právě jedním způsobem
- Gaussovy kvadraturní vzorce mají při daném počtu uzlů maximální možný řád, a tedy optimální přesnost
- váhy Gaussových vzorců jsou pro všechna n vždy kladné, takže algoritmy výpočtu podle Gaussových vzorců jsou numericky stabilní
- navíc se dá ukázat, že Gaussovy kvadraturní vzorce tvoří konvergentní kvadraturu

Gaussovy kvadraturní vzorce mají také ale jednu vážnou nevýhodu: pro $m \neq n$ nemají vzorce G_m a G_n žádné společné uzly (s výjimkou středu intervalu, pokud jsou m i n lichá čísla). Gaussova kvadratura tedy *není progresívní*, což znamená, že pokud zvýšíme počet uzlů řekněme z n na m, musíme počítat navíc m funkčních hodnot namísto $m - n$ hodnot. Tento nedostatek se však dá obejít.

Progresívní Gaussova kvadratura

Jak jsme se právě zmínili, Gaussovy kvadraturní vzorce nejsou progresívní: pokud volíme všechny uzly a váhy volně tak, aby se při daném počtu uzlů maximalizovala algebraická přesnost, nebudou mít vzorce s různým počtem uzlů v podstatě žádné uzly společné, což znamená, že funkční hodnoty integrandu vypočítané pro jeden soubor uzlů se nedají v jiném vzorci s odlišným počtem uzlů znovu využít.

Kronrodovy kvadraturní vzorce se takové práci navíc vyhýbají. Jsou to dvojice vnořených kvadraturních vzorců: jeden člen dvojice je obvyklý n-bodový Gaussův kvadraturní vzorec G_n a druhý z nich je $(2n + 1)$-bodový Kronrodův kvadraturní vzorec K_{2n+1}, jehož uzly se opět volí optimálně tak, aby se maximalizovala algebraická přesnost, ovšem za podmínky, že se v $K2n + 1$ znovu použijí všechny uzly z G_n. Mezi uzly K_{2n+1} je tedy n uzlů pevně dáno předem, takže jako volné parametry máme zbývajících $n + 1$ uzlů a rovněž všechny váhy pro všechny uzly dohromady, kterých je $2n + 1$. Celkem je zde tedy $3n + 2$ volných parametrů a ty se určí opět tak, aby se maximalizoval řád vzorce. Algebraická přesnost vzorce K_{2n+1} je tedy rovna $3n + 1$, kdežto standardní Gaussův kvadraturní vzorec s $2n + 1$ uzly by měl algebraickou přesnost $4n + 1$. Jak vidíme, je zde jistý kompromis mezi přesností a efektivitou.

Jedním z hlavních důvodů, proč používáme dvojice vnořených kvadraturních vzorců je to, že pomocí rozdílu mezi přibližnými hodnotami integrálu získanými oběma vzorci můžeme odhadnout chybu metody. Použijeme-li Gaussův-Kronrodův pár vzorců, vezmeme jako aproximaci integrálu hodnotu K_{2n+1} a realistickým a přitom konzervativním odhadem chyby slouží veličina

$$(200|G_n - K_{2n+1}|)^{1.5}.$$

Tento na první pohled podivný vzorec plyne zčásti z teorie kvadratury, zčásti z praktických zkušeností (není tedy exaktní). Gaussovy-Kronrodovy kvadraturní vzorce patří k nejefektivnějším numerickým metodám výpočtu integrálu, neboť efektivně poskytují jak vysokou přesnost, tak spolehlivý odhad chyby. Obecně se dnes používá především dvojice (G_7, K_15).

Dříve než opustíme toto téma se ještě zmíníme o jednom mírnějším rozšíření Gaussových kvadraturních vzorců, které se také používá a může být užitečné. Jak jsme se již zmínili, pravý Gaussův kvadraturní vzorec je vždy *otevřený*, jeho uzly tedy neobsahují krajní body intervalu integrace. Ale z jistých důvodů je někdy rozumné koncové body intervalu jako uzly ve vzorci mít. U *Gaussových-Lobattových* kvadraturních vzorců se oba koncové body předepisují jako předem pevně dané uzly, takže nám zbývá $n-2$ uzlů a n vah jako volné parametry, které se volí tak, aby se maximalizovala algebraická přesnost. Výsledný n-bodový Gaussův-Lobattův kvadraturní vzorec je pak řádu $2n-3$.

9.4 Složené kvadraturní vzorce

Až dosud jsme se zabývali *základními* kvadraturními vzorci založenými na aproximaci integrandu jedním polynomem na celém intervalu integrace. Přesnost takového vzorce lze svýšit a jeho chybu lze odhadnout tak, že zvýšíme počet uzlů kvadratury a tím i řád aproximujícího polynomu. Jinou možností je rozdělit interval integrace na dva nebo více subintervalů a některý základní kvadraturní vzorec aplikovat na každém takovém subintervalu odděleně. Sečtením takto získaných dílčích výsledků pak dostaneme aproximaci integrálu přes celý interval.

Složený kvadraturní vzorec na daném intervalu $[a, b]$ dostaneme tak, že tento interval rozdělíme na k subintervalů (budeme jim říkat *panely*), které mají typicky stejnou délku $h = (b-a)/k$, na každém z těchto panelů aplikujeme nějaký n-bodový základní kvadraturní vzorec Q_n a jako přibližnou hodnotu celkového integrálu pak vezmeme součet takto získaných dílčích výsledků. Jestliže je použitý kvadraturní vzorec Q_n otevřený, bude takový výpočet vyžadovat kn výpočtů funkčních hodnot. Je-li naproti tomu vzorec Q_n uzavřený, pak se ve složeném vzorci některé body opakují, takže je třeba vypočítat pouze $k(n-1)+1$ funkčních hodnot. Uvedeme příklady složených kvadraturních vzorců založených na jednoduchých základních Newtonových-Cotesových vzorcích.

Příklad 9.9 (Složené kvadraturní vzorce)**.** *Rozdělíme interval* $[a, b]$ *na* k *panelů délky* $h = (b-a)/k$ *a položíme* $x_j = a+jh, j = 0, \ldots k$*. Složené obdélníkové pravidlo je pak*

$$M_k(f) = \sum_{j=1}^{k} (x_j - x_{j-1}) f\left(\frac{x_{j-1} + x_j}{2}\right) = h \sum_{j=1}^{k} f\left(\frac{x_{j-1} + x_j}{2}\right)$$

a složené lichoběžníkové pravidlo je

$$\begin{aligned} T_k(f) &= \sum_{j=1}^{k} \frac{(x_j - x_{j-1})}{2} (f(x_{j-1}) + f(x_j)) \\ &= h(\tfrac{1}{2} f(a) + f(x_1) + \cdots + f(x_{k-1} + \tfrac{1}{2} f(b)). \end{aligned}$$

Pokud je použitý základní kvadraturní vzorec stabilní, je stabilní i vzniklý složený kvadraturní vzorec. Pokud má použitý základní kvadraturní vzorec řád alespoň nula (tj. integruje přesně konstanty), dá se ukázat, že z něj postupně při $k \to \infty$ vznikající složené kvadraturní vzorce tvoří konvergentní kvadraturu. V zásadě tedy platí, že pokud zvolíme dostatečně velké k, můžeme dosáhnout libovolné přesnosti (ta je ovšem v praxi omezena přesností počítačové aritmetiky) i tehdy, je-li samotný základní kvadraturní vzorec nízkého řádu. Nemusí to být ovšem ten nejefektivnější způsob, jak požadované přesnosti dosáhnout, v praxi je zpravidla nutno zvolit vhodný kompromis mezi řádem základního vzorce n a počtem panelů k. Složené vzorce také jaksi navíc poskytují obzvláště jednoduchý způsob odhadu chyby, při němž se zpravidla využijí přibližné hodnoty integrálu získané na k panelech a na rozpůlených $2k$ panelech. Podrobnosti nejsou složité a lze je najít v běžné literatuře, například v[2], [3], [4] nebo [1].

9.5 Dodatky

Numerická integrace je tématem, které je v numerické matematice dobře zpracované a je k dispozici kvalitní numerický software pro přibližný výpočet integrálů, především integrálů jednorozměrných. Řadu témat jsme zde nuceni vynechat, uvádíme pouze pár stručných informací o některých z nich.

Adaptivní kvadratura

Stručně se zde zmíníme o principech současného softwaru pro výpočet jednorozměrných integrálů. Složený kvadraturní vzorec doplněný o odhad chyby poskytuje možnost sestrojit jednoduchý *automatický* algoritmus pro přibližný výpočet integrálu se zadanou požadovanou přesností: postupně pokračujeme s půlením panelů tak dlouho, až celková odhadovaná chyba klesne pod požadovanou úroveň. Pro mnohé integrandy je ale vysoce neefektivní udržovat na

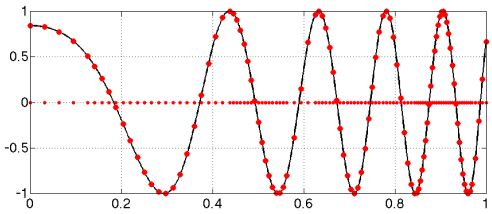
celém $[a, b]$ stejně veliké panely, protože by se tak vynaložil výpočet velkého počtu funkčních hodnot v částech intervalu, kde se integrand dobře chová a kde se dá snadno vyhovět toleranci na chybu. Používá se proto inteligentnější přístup, *adaptivní kvadratura*, při němž se interval integrace dělí na panely selektivně tak, aby to odpovídalo charakteru integrované funkce. Dělení na panely tak bude obecně nerovnoměrné.

Typická adaptivní strategie funguje následujícím způsobem. Především potřebujeme mít k dispozici vhodně vybranou dvojici základních kvadraturních vzorců, řekněme Q_{n_1} a Q_{n_2}, jejichž rozdíl nám poskytuje požadovaný odhad chyby. jako jednoduchý příklad tu může posloužit lichoběžníkové a obdélníkové pravidlo, jejichž rozdíl je zhruba trojnásobkem chyby přesnějšího z nich, jak jsme viděli v odst. 9.2. Větší efektivity se obvykle ale dosáhne použitím vzorců vyššího řádu, jako je například Gaussův-Kronrodův pár (G_7, K_15). Jinou alternativou je použít jeden základní vzorec na dvou rozdílných úrovních dělení. V tomto směru je oblíbené Simpsonovo pravidlo. Ve všech případech ovšem je k minimalizaci počtu potřebných funkčních hodnot třeba, aby použitá dvojice kvadraturních vzorců byla progresívní.

Postup adaptivního algoritmu je pak v principu prostý: nejprve oba vzorce Q_{n_1} a Q_{n_2} aplikujeme na počáteční interval integrace $[a, b]$. Pokud se takto získané přibližné hodnoty integrálu liší o více, než je předepsaná tolerance, rozdělíme interval integrace na dva nebo více panelů a uvedený postup opakujeme na každém z nich. Pokud se na některém panelu dosáhne splnění chybové tolerance, pak se tento panel už dále nedělí. Pokud na daném panelu odhad chyby přesahuje předepsanou toleranci, pokračuje se v dělení, a to tak dlouho, až je předepsané toleranci na chybu vyhověno na všech panelech. Taková strategie vede k tomu, že vzorkovací body integrandu budou obecně rozloženy nerovnoměrně, přičemž jich bude více tam, kde se daná funkce integruje obtížně, a relativně málo tam, kde se integruje snáze. Typické rozložení vzorkovacích bodů (uzlů složeného kvadraturního vzorce), které takový algoritmus sám generuje, je na obr. 9.3. Funkce zobrazená na obrázku je $f(x) = (1 - 30x^2)$, interval integrace je $[0, 1]$ a chybová tolerance je 10^{-4}. Program založený na adaptivní Simpsonově kvadratuře vygeneroval celkem 109 uzlů, které jsou znázorněny jak na grafu funkce, tak na ose x.

V Matlabu jsou k dispozici tři kvalitní funkce pro adaptivní numerický výpočet jednorozměných integrálů. Je to jednak funkce `quad`, založená na Simpsonově pravidlu, dále funkce `quadl`, která používá dvojice vnořených Gaussových-Lobattových vzorců, a konečně funkce `quadgk` založená na Gaussově-Kronrodově páru (G_7, K_15).

I když se adaptivní kvadraturní algoritmy v praxi velmi dobře osvědčují, ve výjimečných případech mohou také selhávat: výsledná přibližná hodnota integrálu i odhad chyby mohou být výjimečně i zcela chybné. Důvodem je tu skutečnost, že integrand se vzorkuje pouze v konečném počtu bodů, takže



Obrázek 9.3: Typické rozložení uzlů u adaptivní kvadratury.

může dojít i k tomu, že se nějaký jeho podstatný rys pomine. Může se například stát, že interval integrace je velmi dlouhý, ale veškeré „zajímavé“ chování integrandu je soustředěno do jeho velmi úzké části. V takovém případě může dojít k tomu, že adaptivní algoritmus při svém vzorkování tuto zajímavou část integrandu zcela pomine a výsledná hodnota integrálu vyjde zcela chybně. Je proto rozumné nespokojit se při výpočtu integrálů s jedním získaným výsledkem, ale porovnávat výsledky získané s různými tolerancemi. Další užitečnou informaci poskytují ty algoritmy, které na výstupu udávají také počet použitých funkčních hodnot. Je totiž (známe takový příklad z praxe) například nemožné, abychom rozumně vypočítali integrál z funkce $\sin 100x$ na intervalu $[0, 2]$ pomocí pouhých 33 vzorků (proč?).

Dvojné integrály

Až dosud jsme se zabývali pouze jednorozměrnými integrály, kde z geometrického hlediska chceme určit obsah oblasti pod nějakou křivkou na nějakém intervalu. U dvojrozměrného neboli *dvojného integrálu* si přejeme vypočítat objem tělesa pod nějakou plochou na danou rovinnou oblastí Ω. Pro obecnou oblast $\Omega \subseteq \mathbb{R}^2$ takový integrál má tvar

$$\int\int_{\Omega} f(x, y) \mathrm{d}\Omega.$$

V případě, že integrujeme přes obdélníkovou oblast $R = [a, b] \times [c, d]$ se dvojný integrál dá často převést na *dvojnásobný integrál*, v němž jsou vlastně do sebe vnořeny dva integrály jednorozměrné:

$$\int\int_{R} f(x, y) \mathrm{d}R = \int_a^b \left(\int_c^d f(x, y) \mathrm{d}y \right) \mathrm{d}x.$$

Analogicky jako tomu bylo se zavedením pojmu numerická kvadratura, se numerická aproximace dvojrozměných integrálů někdy nazývá *numerická kubatura*.

K výpočtu dvojných integrálů se dá použít řada postupů, ale jejich popis se vymyká rozsahovým možnostem tohoto textu. Poznamenáváme pouze, že

často i dvojné integrály se dají počítat pomocí algoritmů pro integrály jednorozměrné, a to tak, že například pro výše popsaný integrál přes obdélník použijeme jeden kvadraturní vzorec pro vnější integrál a druhý (třeba i identický) pro vnitřní integrál. Pokaždé, když vnější rutina bude vyvolávat integrovanou funkci, bude se tak vyvolávat rutina pro výpočet vnitřního integrálu. Další detaily k výpočtu dvojných integrálů lze najít v literatuře, viz například [1]. Také Matlab má k dispozici řadu funkcí pro výpočet takových integrálů, a to i přes obecné dvojrozměrné oblasti.

Vícerozměrné integrály

K výpočtu vícerozměrných integrálů ve více než dvou dimenzích lze sice v principu využít postupů používaných pro dvojné integrály, ale náklady na výpočet rostou s počtem dimenzí nelineárně. Jediným obecně užitečným přístupem k výpočtu složených integrálů ve více dimenzích se zdá být *metoda Monte Carlo*. Postupuje se tak, že se integrovaná funkce vyvzorkuje v n bodech, které jsou (pseudo)náhodně rozloženy v oblasti integrace, vypočítá se střední hodnota vzorů a ta se vynásobí objemem integrační oblasti, čímž dostaneme odhad integrálu. Chyba této aproximace klesá k nule jako $1/\sqrt{n}$ při $n \to \infty$, což znamená například, že abychom získali jednu další přesnou desítkovou číslici ve výsledku, musíme počet vzorkovacích bodů zvýšit stokrát. Z tohoto důvodu není u integrace metodou Monte Carlo žádnou vzácností, jestliže se počet použitých funkčních hodnot pohybuje v řádu milionů.

Metoda Monte Carlo nemůže soutěžit s ostatními metodami u jednorozměrných a dvojrozměrných integrálů, její krása ale spočívá v tom, že rychlost konvergence u ní nezávisí na počtu dimenzí. Tím pádem například milion vzorkovacích bodů v šesti dimenzích vlastně znamená pouze 10 bodů na každou dimenzi, a to je podstatně méně, než by vyžadovala k získání požadované přesnosti jakákoli konvenční kvadratura. Existuje řada účelných způsobů vzorkování, které mají zvýšit efektivitu metody, ale zde můžeme zájemce pouze odkázat na literaturu, např. [1]. Generování pseudonáhodných čísel ke užitečné i jinde než při numerické integraci a k tomuto tématu se ještě vrátíme na závěr kurzu.

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill, Boston, 2 edition, 2002.

[2] Petr Přikryl. *Numerické metody matematické analýzy.* MVŠT. SNTL, Praha, 1985.

[3] Petr Přikryl. *Numerické metody matematické analýzy.* MVŠT. SNTL, Praha, 2., opr. a dopl. edition, 1988.

[4] Petr Přikryl and Marek Brandner. *Numerické metody II.* FAV ZČU, Plzeň, 2001.

KAPITOLA 10

Řešení soustav lineárních rovnic

10.1 Soustavy lineárních algebraických rovnic

Mnohé matematické a počítačové modely fyzikálních a technických jevů vedou v konečné fázi na řešení soustav **lineárních algebraických rovnic**.Velmi často jsou tyto soustavy rozsáhlé, mají běžně statisíce nebo miliony rovnic a neznámých.Z tohoto důvodu se jak v teoretické, tak v praktické numerické matematice (tvorbě matematického softwaru) tématice numerické lineární algebry věnovala a věnuje značná pozornost a tato oblast je dnes důkladně prozkoumána.

Používají se jak **přímé**, tak **iterační** metody řešení soustav lineárních rovnic. Přímé metody dávají přesné řešení soustavy, alespoň teoreticky, v konečném počtu kroků. Z tohoto hlediska je tedy řešení soustavy lineárních rovnic typická **numerická úloha**. Při realizaci přímé metody na počítači ovšem musíme brát v úvahu také vlastnosti počítačové aritmetiky, především zaokrouhlovací chyby.

Protože velká část používaných počítačových modelů vychází z matematických modelů formulovaných typicky jako okrajové úlohy pro diferenciální rovnice a řešené soustavy lineárních rovnic jsou pouze aproximacemi těchto okrajových úloh, je často zbytečné usilovat o jejich přesné vyřešení. Stačí nám přibližné řešení získané vhodnou iterační metodou s přesností srovnatelnou s přesností aproximace teoretické okrajové úlohy. Takové přibližné řešení se dá iteračními metodami zpravidla získat mnohem úsporněji (uvažte velikost řešených soustav!) než řešení vypočítané přímou metodou.

Ale i v jednotlivých krocích iteračních metod se může přímá metoda vyskytnout, navíc jsou tyto metody velmi užitečné při řešení soustav s maticemi menších rozměrů. Také teorie přímých metod, alespoň ta, kterou vyložíme

zde, je podle našeho názoru snazší než je tomu u metod iteračních a patří víceméně ke všeobecnému vzdělání inženýra. V této části kurzu se proto budeme zabývat přímými metodami a ohledně iteračních metod, na které nám již nezbude čas a prostor, čtenáře musíme odkázat na literaturu.

Pro detailnější obeznámení s pojmy, uváděnými níže, doporučuji i zde konzultovat knihu Michaela T. Heathe [1] nebo monografii Nicka Highama [2]. Čtenář může případně použít i některou z českých učebnic či mnoha skript o numerické matematice, která v posledních letech vyšla – například [3, 4, 5] (části skripta [5] jsou dostupné i on-line). Mnohé ze zde použitých materiálů jsme převzali právě z [1].

Formulace úlohy

Zopakujeme nejprve stručně formulaci úlohy a některé základní informace o řešení soustav a o jejich maticích, které by čtenář měl již znát z kurzu lineární algebry. V lineární algebře jsme se dozvěděli, že pohodlný způsob, jak vyjádřit lineární zobrazení (transformaci) mezi dvěma konečně-rozměrnými vektorovými prostory, spočívá v použití **matic**.V tomto textu se budeme zabývat pouze soustavami n rovnic o n neznámých, zmíněné vektorové prostory tedy budou mít stejnou dimenzi a příslušné matice budou čtvercové.Poznamenejme, že pod pojmem vektor zde rozumíme vždy *sloupcový* vektor, pokud výslovně neuvedeme jinak. V maticovém zápisu má soustava lineárních algebraických rovnic tvar

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{10.1}$$

kde $\mathbf{A}$ je čtvercová matice **řádu** n (nebo také typu $n \times n$), a $\mathbf{x}, \mathbf{b}$ jsou n-složkové vektory.

Taková soustava lineárních rovnic vlastně před nás klade otázku „Je možné vektor $\mathbf{b}$ vyjádřit jako lineární kombinaci sloupců matice $\mathbf{A}$?“. Nebo, což je totéž, „Leží vektor $\mathbf{b}$ v prostoru $\text{span}(\mathbf{A}) = \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\}$?“

Pokud tomu tak je, soustava má zřejmě řešení a nazývá se **konzistentní**. Řešení soustavy může i nemusí existovat, a pokud existuje, může i nemusí být jediné. Detaily jsou studovány v lineární algebře, ale zde se omezíme pouze na soustavy rovnic s regulárními maticemi, kde je otázka existence a jednoznačnosti řešení jednoduše vymezena. Pro jednoduchost se omezíme také pouze na soustavy lineárních algebraických rovnic s reálnými koeficienty, jejichž řešeními jsou pak nutně opět reálné vektory. Ovšem komplexní soustavy lineárních rovnic mohou být zpracovány zcela obdobně.

Existence a jednoznačnost řešení

Soustavy lineárních algebraických rovnic se čtvercovými regulárními maticemi mají vždy řešení (pro každou pravou stranu $\mathbf{b}$) a toto řešení je jediné.

Definice 10.1 (Regulární matice)**.** *Čtvercová matice* $\mathbf{A}$ *řádu* n *je* ***regulární****, jestliže vyhovuje kterékoli z následujících ekvivalentních podmínek:*

1. *k matici* $\mathbf{A}$ *existuje inverzní matice* $\mathbf{A}^{-1}$ *(pro takovou matici platí* $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$*, kde* $\mathbf{I}$ *je jednotková matice řádu* n*);*
2. $\det(\mathbf{A}) \neq 0$ *(tj. determinant matice je nenulový);*
3. *hodnost matice* $\mathbf{A}$ *je rovna* n *(****hodnost matice*** *je maximální počet lineárně nezávislých sloupců nebo řádků, které matice obsahuje);*
4. *soustava* $\mathbf{A}\mathbf{z} = \mathbf{o}$*, kde* $\mathbf{o}$ *je nulový vektor, má pouze triviální řešení (řešením je opět pouze nulový vektor).*

Matice, která není regulární, se nazývá **singulární**.

Existence a jednoznačnost řešení soustavy (10.1) jednoduchým způsobem závisí na regularitě matice $\mathbf{A}$: Jestliže je matice $\mathbf{A}$ regulární, pak existuje její inverzní matice $\mathbf{A}^{-1}$ a soustava (10.1) má vždy právě jedno řešení $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ nezávisle na hodnotě pravé strany $\mathbf{b}$. Jestliže je naproti tomu matice $\mathbf{A}$ singulární, pak je řešitelnost soustavy (10.1) závislá na pravé straně $\mathbf{b}$; pro některou pravou stranu nemusí mít soustava žádné řešení, ale pokud existuje alespoň jedno řešení, pak jich existuje nekonečně mnoho. Řešení tedy v takovém případě není určeno jednoznačně.

Geometricky ve dvou dimenzích každá z rovnic soustavy 2×2 představuje rovnici přímky a řešením soustavy je tedy společný bod obou přímek. Jsou-li tyto dvě přímky různoběžné, mají právě jeden průsečík, řešení existuje a je jediné (regulární případ), Jsou-li obě přímky rovnoběžky, pak buď nemají průsečík vůbec nebo splývají a pak je zde nekonečně mnoho společných bodů, řešení (singulární případ).

Podmíněnost soustav lineárních rovnic, odhady chyb

Poté, co jsme zhruba popsali otázky existence a jednoznačnosti řešení lineárních soustav, se nyní zaměříme na citlivost řešení $\mathbf{x}$ na poruchy ve vstupních datech řešené numerické úlohy, což jsou zde matice $\mathbf{A}$ a vektor pravých stran $\mathbf{b}$. Abychom takové vektorové a maticové poruchy mohli měřit, budeme potřebovat pro vektory a matice nějak zobecnit pojem velikosti, jako který nám u reálných a komplexních čísel slouží jejich absolutní hodnota. Příslušným zobecněním, které by již čtenář mohl znát z kurzu lineární algebry, je pojem **normy**. Pro jistotu zde příslušné informace uvedeme.

Vektorové normy

Budeme zde používat vektorové normy, které jsou vesměs speciálními případy tzv. p-norem, které jsou pro kladné celé číslo $p > 0$ a n-složkový vektor $\mathbf{x}$

definovány jako

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Důležité speciální případy představují

- *jedničková norma*

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|,$$

- *euklidovská norma*

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_I|^2 \right)^{1/2},$$

  která odpovídá obvyklému pojmu vzdálenosti v euklidovských prostorech (nebo chcete-li, délce vektoru), a

- ∞*-norma* (čteno nekonečno norma)

$$\|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i|,$$

  která je limitní hodnotou p-normy při $p \to \infty$ a říká se jí také *Čebyševova norma.*

Při práci se všemi těmito normami dostáváme kvalitativně podobné výsledky, ovšem jednotlivé normy se můžou analyticky nebo výpočetně zpracovávat snáze nebo hůře. Při analýze podmíněnosti soustav se většinou pracuje s jedničkovou nebo Čebyševovou normou. Euklidovská norma se v této souvislosti používá méně, ale nachází účelnou aplikaci v jiných oblastech numerické lineární algebry.

Dá se ukázat, že při pevně daném n se libovolné dvě z uvedených norem liší nanejvýš nevelkou multiplikativní konstantou, která sice závisí na n, ale nezávisí na konkrétním měřeném vektoru. Tím pádem jsou všechny tři normy ekvivalentní v tom smyslu, že pokud je jedna z nich malá, jsou úměrně malé i ty zbývající. V dané situaci tedy můžeme při našich úvahách volit tu z norem, s níž se nám pohodlně pracuje. Z tohoto důvodu budeme v další části tohoto textu index u norem používat pouze tam, kde to bude hrát nějakou roli, a tam, kde bude jedno, která norma se použije, budeme index vynechávat.

Každá vektorová p-norma má pro libovolné dva vektory $\mathbf{x}, \mathbf{y}$ tyto důležité vlastnosti:

1. $\|\mathbf{x}\| \ge 0$ a $\|\mathbf{x}\| = 0$ právě pro $\mathbf{x} = \mathbf{o}$.

2. $\|\alpha\mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$ pro každý skalár (číslo) α.

3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (trojúhelníková nerovnost).

Pro výpočet p-norem vektorů je v Matlabu k dispozici funkce `norm`.

Maticové normy

Budeme potřebovat také nějaký způsob, jak posuzovat „velikost“ matic. Jsou zde sice možné rozličné definice norem, ale ty normy, které budeme používat, budou definovány prostřednictvím již zavedených vektorových norem.

Definice 10.2 (Maticová norma matice $\mathbf{A}$)**.** *Při dané vektorové normě budeme maticovou normu matice* $\mathbf{A}$ *definovat jako*

$$\|\mathbf{A}\| = \max_{\mathbf{x} \neq \mathbf{o}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

O takové maticové normě se říká, že je **indukovaná** příslušnou vektorovou normou nebo že je této normě **podřízená**. Intuitivně se na tuto definici můžeme dívat tak, že norma matice měří to, jak transformace danou maticí může maximálně protáhnout ten který vektor, když délku vektorů měříme použitou vektorovou normou.

S některými maticovými normami se pracuje snáze než s jinými.

Příklad 10.1 (Maticová norma odpovídající jedničkové vektorové normě)**.** *Tato norma se rovná jednoduše maximální hodnotě sloupcových součtů prvků matice braných v absolutních hodnotách,*

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|.$$

Příklad 10.2 (Maticová norma odpovídající Čebyševově vektorové normě)**.** *Maticová norma odpovídající Čebyševově vektorové normě je prostě maximální řádkový součet prvků matice braných v absolutních hodnotách,*

$$\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|.$$

Maticová norma odpovídající euklidovské vektorové normě se dá definovat pomocí vlastních čísel matic, její explicitní výpočet je tudíž složitější a my se jí zda zabývat nebudeme.

Takto definované maticové normy mají pro libovolné matice $\mathbf{A}, \mathbf{B}$ tyto důležité vlastnosti:

1. $\|\mathbf{A}\| \geq 0$ a $\|\mathbf{A}\| = 0$ právě když $\mathbf{A} = \mathbf{O}$ (nulová matice).
2. $\|\alpha\mathbf{A}\| = |\alpha| \cdot \|\mathbf{A}\|$ pro každý skalár (číslo) α.
3. $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$.
4. $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|$.
5. $\|\mathbf{Ax}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|$ pro každý vektor $\mathbf{x}$.

Poznamenejme, že obecně se za maticovou normu považuje jakákoli funkce matic, která má první tři právě uvedené vlastnosti. Normy, které mají navíc poslední dvě vlastnosti, se pak nazývají **submultiplikativní** nebo **konzistentní**. Zdůrazňujeme znovu, že maticové normy indukované vektorovými p-normami jsou s nimi vždy konzistentní, mají tedy všech pět výše uvedených vlastností.

Námi popsané maticové normy se v Matlabu dají snadno počítat pomocí funkce `norm`.

Podmíněnost matic

Číslo podmíněnosti regulární čtvercové matice $\mathbf{A}$ se v dané maticové normě definuje jako

$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|.$$

Pro singulární matici $\mathbf{A}$ se úmluvou klade $\mathrm{cond}(\mathbf{A}) = \infty$. Při hlubším studiu se dá ukázat, že takto zavedené číslo podmíněnosti je konzistentní s definicí podanou v Přednášce 8, a to v tom smyslu, že ohraničuje poměr mezi relativní změnou řešení soustavy lineárních rovnic a danou relativní změnou ve vstupních datech.

Z definice je zřejmé, že hodnota čísla podmíněnosti závisí na použité maticové normě, což se někdy označuje tím, že se použije příslušný index; píšeme pak například $\mathrm{cond}_1(A)$ nebo $\mathrm{cond}_\infty(A)$.

Z toho, co jsme říkali v Odstavci 10.1 ale vyplývá, že hodnoty čísla podmíněnosti se při různých normách mohou lišit maximálně nevelkou multiplikativní konstantou (která závisí na n) a jsou tedy jako měřítka podmíněnosti stejně užitečné. Jako zajímavost uvádíme, že se dá ukázat, že číslo podmíněnosti také měří poměr mezi maximálním relativním protažením a maximálním relativním zkrácením libovolných dvou n-složkových vektorů poté, co na ně aplikujeme matici $\mathbf{A}$. Z definice čísla podmíněnosti matic lze vcelku snadno odvodit následující jeho důležité vlastnosti, které platí při použití jakékoli námi zavedené normy:

1. Pro všechny matice $\mathbf{A}$ platí $\mathrm{cond}(\mathbf{A}) \geq 1$.

2. Pro jednotkovou matici je $\mathrm{cond}(\mathbf{I}) = 1$.

3. Pro jakoukoli matici $\mathbf{A}$ a každé číslo α různé od nuly je $\mathrm{cond}(\alpha\mathbf{A}) = \mathrm{cond}(\mathbf{A})$.

4. Pro každou diagonální matici $\mathbf{D} = \mathrm{diag}(\mathbf{d})$ platí $\mathrm{cond}(\mathbf{D}) = (\max_i |d_i|)/(\min_j |d_j|)$.

Číslo podmíněnosti je mírou toho, jak blízko je daná matice k tomu, aby byla singulární: matice s velkým číslem podmíněnosti (co to znamená, řekneme v následujícím odstavci) je téměř singulární, zatímco matice, jejíž číslo podmíněnosti je řádu jednotek, má k singularitě daleko. Z definice je také zřejmé, že regulérní matice a její inverze mají stejné číslo podmíněnosti.

Všimněme si také toho, že přestože determinant singulární matice je nulový, není hodnota determinantu sama o sobě měřítkem toho, jak blízko nebo daleko má daná matice $\mathbf{A}$ k singulární matici. Vezměme si jako příklad matici $0{,}1\,\mathbf{I}$, jejíž determinant je roven $0{,}1^n$, což už u malé matice s $n = 30$ dává jako hodnotu determinantu 10^{-30}, tedy velmi malé číslo. Přitom je tato matice velmi dobře podmíněná, její číslo podmíněnosti je 1 v jakékoli normě.

Brzy uvidíme, že užitečnost čísla podmíněnosti se ukáže při posuzování přesnosti řešení soustav lineárních rovnic. V definici čísla podmíněnosti ovšem vystupuje inverzní matice, takže počítat jeho hodnotu by byla obvykle netriviální záležitost. Ve skutečnosti by výpočet čísla podmíněnosti z jeho definice vyžadoval podstatně více výpočetní práce než samotné řešení posuzované soustavy lineárních rovnic. V praxi lze naštěstí jako vedlejší produkt při řešení soustav získat alespoň dobrý odhad čísla podmíněnosti, souhlasící s jeho skutečnou hodnotou přinejmenším řádově. V Matlabu je pro tento účel k dispozici funkce `cond` a některé další funkce.

Příklad 10.3 (Odhad čísla podmíněnosti)**.** *Uvažujme matici*

$$\mathbf{A} = \begin{bmatrix} 0{,}913 & 0{,}659 \\ 0{,}457 & 0{,}330 \end{bmatrix}.$$

Odhadneme její čísla podmíněnosti v Matlabu; použijeme k tomu funkci `cond`*. Dostaneme*

$$\mathrm{cond}_1(\mathbf{A}) = \mathrm{cond}_\infty(\mathbf{A}) \approx 1{,}6958 \cdot 10^4,$$

$$\mathrm{cond}_2(\mathbf{A}) \approx 1{,}2485 \cdot 10^4.$$

Odhady se řádově shodují. Vzhledem k nevelkému řádu matice a počtu uvedených platných cifer můžeme danou matici považovat za špatně podmíněnou. Jak uvidíme v příštím odstavci, pokud by prvky této matice byly naměřeny s chybou velikosti řádově 10^{-4}*, nemohli bychom při řešení soustav s touto maticí zaručit ve výsledku ani jednu platnou číslici.*

Odhady chyb

Kromě toho, že je číslo podmíněnosti spolehlivým indikátorem blízkosti k singulární matici, nám dává také užitečné kvantitativní odhady chyby pro vypočítaná řešení lineárních soustav.Uvedeme zde pouze výsledky, jakkoli matematika za nimi skrytá není nikterak obtížná. Případné zájemce v tomto směru odkazujeme například na [1, 2].

Uvažujme nejprve poruchy v pravé straně soustavy lineárních rovnic. Nechť $\mathbf{x}$ je řešení regulární soustavy lineárních rovnic $\mathbf{A}\mathbf{x} = \mathbf{b}$ a nechť $\hat{\mathbf{x}}$ je řešením soustavy $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b} + \Delta\mathbf{b}$ s porušenou pravou stranou. Označme $\Delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}$. Pak se dá odvodit odhad

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \mathrm{cond}(\mathbf{A})\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Číslo podmíněnosti matice je tedy jakýsi „amplifikační faktor“, který pomáhá odhadnout maximální relativní změnu řešení způsobenou danou relativní poruchou ve vektoru pravých stran (porovnejte to s obecně zavedeným pojmem čísla podmíněnosti z Přednášky 8).

Podobný výsledek se dá odvodit pro relativní změny v prvcích matice $\mathbf{A}$. Jestliže $\mathbf{A}\mathbf{x} = \mathbf{b}$ a $(\mathbf{A} + \mathbf{E})\hat{\mathbf{x}} = \mathbf{b}$, pak

$$\frac{\|\Delta\mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \leq \mathrm{cond}(\mathbf{A})\frac{\|\mathbf{E}\|}{\|\mathbf{A}\|}.$$

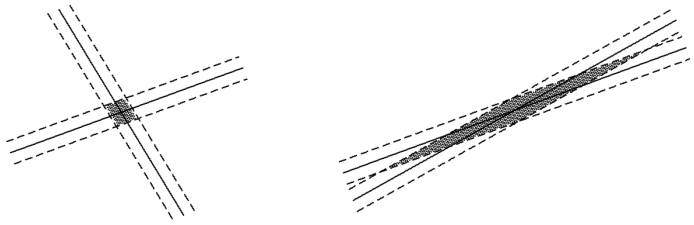
Obecně se pak dá říci, že při současných změnách prvků matice i pravé strany platí (až na veličiny vyšších řádů)

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \mathrm{cond}(\mathbf{A})\left(\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} + \frac{\|\mathbf{E}\|}{\|\mathbf{A}\|}\right).$$

Vidíme tedy opět, že relativní změna řešení je ohraničená součinem čísla podmíněnosti a relativní změny v datech úlohy.

Geometrická interpretace popsaných výsledků ve dvou dimenzích je ta, že jsou-li přímky definované dvěma rovnicemi téměř rovnoběžné, pak pokud mohou být tyto přímky zatíženy zaokrouhlovacími chybami nebo chybou měření, není jejich průsečík definován ostře. Pokud ale tyto přímky vůbec nejsou rovnoběžné, jsou třeba téměř na sebe kolmé, je jejich průsečík relativně ostře vymezen. Oba tyto případy ilustrujeme na Obrázku 10.1, kde čárkované přímky vyznačují oblast nepřesnosti pro každou plně vytaženou přímku, takže průsečík skutečných přímek (které přesně neznáme) by mohl být kdekoli ve vystínovaném rovnoběžníku. Velké číslo podmíněnosti je tedy spojeno s velkou nepřesností získaného řešení.

Máme-li tedy shrnout to, jsme se zatím naučili, pak v případě, že jsou vstupní data přesná na strojovou přesnost (relativní chyba v Matlabu tedy cca 10^{-16}),



well-conditioned ill-conditioned

Obrázek 10.1: Dobře podmíněná (vlevo) a špatně podmíněná (vpravo) soustava dvou lineárních rovnic o dvou neznámých.

můžeme za rozumný odhad relativní chyby ve vypočítaném řešení považovat

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \lessapprox \mathrm{cond}(\mathbf{A})\, \epsilon_{\mathrm{mach}}$$

kde ϵ_{mach} je relativní přesnost aritmetiky počítače (angl. *unit roundoff*, definici a výpočet máte popsány v podkladech pro 8. cvičení a také v [1] **dopsat do přednášky**).Jednoduchý způsob interpretace získaných výsledků je ten, že vypočítané řešení soustavy ztrácí během řešení zhruba $\log_{10}(\mathrm{cond}(\mathbf{A}))$ desítkových číslic přesnosti vůči přesnosti vstupních dat. Podíváme-li se na matici z Příkladu 10.3, vidíme, že její číslo podmíněnosti je řádově 10^4, takže při řešení soustavy dvou rovnic s touto maticí nemůžeme v výsledku očekávat jedinou správnou číslici, pokud prvky matice nejsou přesné na více než čtyři platné číslice a pokud řešení nepočítáme v aritmetice používající alespoň čtyři platné desítkové číslice.

Jako kvantitativní míra citlivosti tak číslo podmíněnosti matice hraje při řešení soustav lineárních rovnic stejnou roli, jako tomu bylo u obecně zavedeného čísla podmíněnosti v Přednášce 8. Důležitý rozdíl je zde ale v tom, že maticové číslo podmíněnosti nemůže nikdy být menší než 1.

Reziduum

Jedním ze způsobů, jak ověřit řešení rovnice, je dosadit je do řešené rovnice a podívat se, jak se po dosazení shoduje její levá a pravá strana. **Reziduum** přibližného řešení $\hat{\mathbf{x}}$ lineární soustavy rovnic $\mathbf{A}\mathbf{x} = \mathbf{b}$ je rozdíl

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}.$$

Pokud je $\mathbf{A}$ regulární, platí pro chybu teoreticky, že $\|\Delta\mathbf{x}\| = \|\hat{\mathbf{x}} - \mathbf{x}\| = 0$ tehdy a jen tehdy, je-li $\|\mathbf{r}\| = 0$. V praxi však nemusí být obě tyto veličiny malé současně. Všimněme si především toho, že vynásobíme-li soustavu $\mathbf{A}\mathbf{x} = \mathbf{b}$ libovolnou nenulovou konstantou, její řešení se nezmění, ale reziduum samé se násobí stejným číslem. Reziduum tedy můžeme udělat libovolně velké

nebo malé podle toho, jakým faktorem soustavu vynásobíme (přeškálujeme ji, změníme měřítko), a tím pádem je velikost rezidua nesmyslné kritérium, pokud ji nějak nevztáhneme k velikosti dat úlohy a řešení. Z tohoto důvodu se zavádí pojem **relativní reziduum**, které se definuje jako

$$R(\hat{\mathbf{x}}) = \frac{\|\mathbf{r}\|}{\|\mathbf{A}\| \cdot \|\hat{\mathbf{x}}\|}.$$

Abychom uvedli relativní reziduum do vztahu s chybou získaného přibližného řešení, všimneme si, že

$$\|\Delta\mathbf{x}\| = \|\hat{\mathbf{x}} - \mathbf{x}\| = \|\mathbf{A}^{-1}(\mathbf{A}\hat{\mathbf{x}} - \mathbf{b})\| = \| - \mathbf{A}^{-1}\mathbf{r}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\mathbf{r}\|.$$

Vydělíme-li obě strany získané nerovnosti $\|\hat{\mathbf{x}}\|$ a použijeme definici čísla podmíněnosti matice $\mathbf{A}$, dostáváme odsud odhad

$$\frac{\|\Delta\mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \leq \mathrm{cond}(\mathbf{A}) \frac{\|\mathbf{r}\|}{\|\mathbf{A}\| \cdot \|\hat{\mathbf{x}}\|} = \mathrm{cond}(\mathbf{A}) \cdot R(\hat{\mathbf{x}}).$$

Malé relativní reziduum tedy znamená malou relativní chybu vypočítaného řešení tehdy a jen tehdy, když je $\mathbf{A}$ dobře podmíněná matice.

Pro algoritmy přímých metod řešení soustav lineárních rovnic se dá ukázat, že jimi vypočítané řešení $\hat{\mathbf{x}}$ je *přesným* řešením nějaké porušené soustavy

$$(\mathbf{A} + \mathbf{E})\hat{\mathbf{x}} = \mathbf{b}.$$

Popravdě řečeno je takových porušených soustav celá řada, stačí si uvědomit, že při daných $\mathbf{A}$ a $\hat{\mathbf{x}}$ máme ve výše uvedeném vztahu určit n^2 prvků matice $\mathbf{E}$ z n lineárních rovnic. Ukazuje se však, že mezi poruchami $\mathbf{E}$ se dá vybrat porucha s minimální maticovou normou, a té se pak říká *zpětná chyba* vypočítaného řešení (na rozdíl od *přímé chyby* $\Delta\mathbf{x}$). Odhady zpětné chyby jsou pro algoritmy přímých metod k dispozici a můžeme u nich tedy říci, že jimi vypočítané numerické řešení je přesným řešením původní soustavy porušené o poruchu, jejíž velikost (v normě) umíme odhadnout shora.

Výše provedené úvahy o relativním reziduu nám snadno ale dávají dolní odhad takové zpětné chyby, Je totiž

$$\|\mathbf{r}\| = \|\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\| = \|\mathbf{E}\hat{\mathbf{x}}\| \leq \|\mathbf{E}\| \cdot \|\hat{\mathbf{x}}\|,$$

odkud plyne po vydělení normou $\mathbf{A}$ nerovnost

$$\frac{\|\mathbf{r}\|}{\|\mathbf{A}\| \cdot \|\hat{\mathbf{x}}\|} \leq \frac{\|\mathbf{E}\|}{\|\mathbf{A}\|}.$$

Velké relativní reziduum tak implikuje velkou zpětnou chybu, což znamená, že algoritmus použitý k výpočtu řešení nebyl stabilní a získané řešení nemůže být kvalitní. Na druhé straně, stabilní algoritmy nutně produkují malá relativní rezidua bez ohledu na podmíněnost úlohy, a tím pádem malé ■ **relativní?** ■ reziduum vypovídá o kvalitě získaného řešení jen málo.

Příklad 10.4 (Malé reziduum)**.** *Uvažujme soustavu lineárních rovnic*

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 0{,}913 & 0{,}659 \\ 0{,}457 & 0{,}330 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0{,}254 \\ 0{,}127 \end{bmatrix} = \mathbf{b},$$

s jejíž maticí jsme se již setkali v Příkladu 10.3. Řekněme, že máme k dispozici dvě přibližná řešení

$$\hat{\mathbf{x}}_1 = \begin{bmatrix} -0{,}0827 \\ 0{,}5 \end{bmatrix} \quad \text{a} \quad \hat{\mathbf{x}}_2 = \begin{bmatrix} 0{,}999 \\ -1{,}001 \end{bmatrix}.$$

Normy odpovídajících reziduí jsou

$$\|\mathbf{r}_1\|_1 = 2{,}1 \cdot 10^{-4} \quad a \quad \|\mathbf{r}_2\|_1 = 2{,}4 \cdot 10^{-2}.$$

Které z obou řešení tedy máme považovat za přesnější? Díky mnohem menšímu reziduu bychom mohli mít sklon k tomu, abychom řekli, že je to $\hat{\mathbf{x}}_1$*. Ale přesné řešení této soustavy je (dá se to snadno ověřit) vektor*

$$\mathbf{x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

takže $\hat{\mathbf{x}}_2$ *je ve skutečnosti mnohem přesnější než* $\hat{\mathbf{x}}_1$*. Důvodem této možná překvapivé skutečnosti je, že matice* $\mathbf{A}$ *je špatně podmíněná, jak jsme ostatně viděli v Příkladu 10.3. Díky jejímu velkému číslu podmíněnosti zde malé reziduum neznamená, že máme přesné řešení.*

■ **Relativní rezidua jsou R1=2.6456e-04 a R2=8.6095e-04.** ■

10.2 Numerické metody pro řešení soustav lineárních rovnic

Přímé numerické metody pro řešení soustav lineárních algebraických rovnic, kterými se nyní budeme zabývat, nejsou vlastně ničím jiným, než zobecněním postupů známých ze střední školy. Tam jsme u soustavy dvou rovnic o dvou neznámých tím či oním způsobem soustavu upravili: z jedné rovnice jsme vyloučili (eliminovali) druhou neznámou a převedli ji tak na rovnici pro jedinou neznámou. Tuto rovnici jsme vyřešili, výsledek dosadili do některé z původních rovnic a z ní pak vypočítali zbývající neznámou. Ukazuje se, že všechny takové úpravy se dají přehledně zapsat ve formě maticových transformací. Numerické metody, které zde budeme popisovat, vlastně provádějí na danou soustavu (její matici) určitou posloupnost úprav, které se dají chápat jako maticové transformace a jejichž cílem je danou soustavu převést na soustavu jednodušší (s maticí speciálního tvaru), která je s původní soustavou ekvivalentní, ale řeší se podstatně přímočařeji.

Transformace řešené úlohy

Prozatím zde pouze popíšeme maticové transformace, které nemění řešení soustavy, a ukážeme, že k nim mimo jiné patří ty transformace, které si lze představit jako elementární operace s rovnicemi soustavy, speciálně jako vynásobení rovnice nenulovou konstantou nebo jako přičtení násobku jedné rovnice k rovnici jiné. Naším cílem bude posléze převést řešenou soustavu (její matici) ne takový tvar, v němž ji půjde už snadno a přímo vyřešit. Abychom nebyli příliš tajemní, poznamenáváme již zde, že naším cílem bude získat ekvivalentní soustavu, jejíž matice bude trojúhelníková.

Je především jasné, že pokud obě strany soustavy $\mathbf{Ax} = \mathbf{b}$ s regulární maticí vynásobíme zleva jakoukoli regulární maticí $\mathbf{M}$, na řešení soustavy to nebude mít vliv. Abychom se o tom přesvědčili, všimněme si, že řešení transformované soustavy

$$\mathbf{MAz} = \mathbf{Mb}$$

je dáno jako

$$\mathbf{z} = (\mathbf{MA})^{-1}\mathbf{Mb} = \mathbf{A}^{-1}\mathbf{M}^{-1}\mathbf{Mb} = \mathbf{A}^{-1}\mathbf{b} = \mathbf{x}.$$

Permutace

Důležitým příkladem regulární transformace je skutečnost, že řádky matice $\mathbf{A}$ a odpovídající složky ve vektoru $\mathbf{b}$ lze přeházet, aniž bychom změnili řešení $\mathbf{x}$. Intuitivně je to zřejmé: všechny rovnice soustavy musí být splněny současně, takže na pořadí, v jakém je zapíšeme, nezáleží. Formálně se takové přehazování řádků dá v maticovém tvaru zapsat jako násobení obou stran soustavy zleva **permutační maticí P**, což je čtvercová matice (opět řádu n), mající v každém řádku a sloupci právě jednu jedničku a všechny ostatní prvky rovné nule (je to vlastně jednotková matice s přeházenými řádky a sloupci). Podívejme se, jak permutační matice může například přeházet řádky sloupcového vektoru:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_3 \\ v_1 \\ v_2 \end{bmatrix}.$$

Permutační matice je vždy regulární; ve skutečnosti je její inverzí prostě její transpozice, $\mathbf{P}^{-1} = \mathbf{P}^{\mathrm{T}}$. Pro pořádek poznamenejme, že *transpozicí* matice $\mathbf{M}$, kterou značíme $\mathbf{M}^{\mathrm{T}}$, rozumíme matici, jejímiž sloupci jsou řádky matice $\mathbf{M}$. Je-li tedy $\mathbf{N} = \mathbf{M}^{\mathrm{T}}$, platí $n_{ij} = m_{ji}$.

Řádkové škálování

Jiným jednoduchým, ale důležitým případem regulární maticové transformace je **diagonální škálování**. Připomínáme, že matice $\mathbf{D}$ je **diagonální**, jestliže v ní pro všechna $i \neq j$ platí $d_{ij} = 0$, což znamená, že jedinými prvky, které v

ní mohou být nenulové, jsou prvky na *hlavní diagonále* $d_{ii}, i = 1, \ldots, n$. Determinantem diagonální matice je součin jejích prvků na hlavní diagonále, takže regulární diagonální matice má na hlavní diagonále pouze nenulové prvky.

Vynásobíme-li obě strany soustavy $\mathbf{Ax} = \mathbf{b}$ zleva regulární diagonální maticí,

$$\mathbf{DAx} = \mathbf{Db}$$

je to totéž, jako bychom každý z řádků matice soustavy a pravé strany vynásobili odpovídajícím diagonálním prvkem d_{ii}, a mluvíme zde proto o **řádkovém škálování**. Řádky, kterým v použité diagonální matici odpovídají na hlavní diagonále jedničky, se přitom nemění. Řádkové škálování může být v praxi někdy užitečné, protože nemění řešení soustavy a přitom může pozitivně ovlivnit chování použitého algoritmu na počítači.

Lineární kombinace řádků matice

Další ekvivalentní úprava soustavy lineárních rovnic spočívá v tom, že provedeme určitou lineární kombinaci řádků soustavy a přičteme ji k některé jiné rovnici. Speciální případ, který zde popíšeme, protože hraje významnou roli v numerických metodách řešení soustav, je ten, kdy určitý násobek jednoho řádku matice přičteme k jinému řádku (nebo jej od něj odečteme).

Ukážeme, že taková operace se dá opět zapsat jako transformace jistou regulární maticí, takže nemění řešení soustavy (koneckonců se to zdá být zřejmé intuitivně). Především je vhodné si zopakovat definici násobení matic a uvědomit si, že prvek na místě ij ve výsledné matici není nic jiného než skalární součin i-tého řádku prvního činitele s j-tým sloupcem činitele druhého. Pak už bychom měli snadno pochopit, že přičtení m-násobku řádku i v dané matici k jejímu j-tému řádku se dá maticově zapsat tak, že danou matici vynásobíme maticí $\mathbf{M}$, která se od jednotkové matice liší pouze tím, že má navíc na místě ji číslo m. Matice M je tedy trojúhelníková matice, která má například pro $i < j$ tvar podobný

$$\mathbf{M} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & m & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Jestliže je v uvedené matici prvek m ve sloupci k a řádku $k + 1$, pak $\mathbf{MA}$ realizuje přičtení m-násobku řádku k v matici $\mathbf{A}$ k řádku $k + 1$ v téže matici.

Příklad 10.5 (Lineární kombinace řádků matice)**.** *Mějme transformační matici* $\mathbf{M}$ *a matici* $\mathbf{A}$ *zadané jako*

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & m & 1 \end{bmatrix} \quad a \quad \mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & 1 \\ -1 & 2 & 1 \end{bmatrix}.$$

Potom

$$\mathbf{MA} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & 1 \\ -1+0m & 2-1m & 1+1m \end{bmatrix}$$

Matice podobného typu (třeba i s více nenulovými prvky m v některém sloupci) budeme v dalším nazývat **elementární eliminační matice**. Všimněme si, že elementární eliminační matice mají na diagonále samé jedničky, takže jejich determinant je roven jedné a matice jsou regulární. Popsané kombinování řádků soustavy tedy nemění její řešení.

Řešení soustav rovnic s trojúhelníkovou maticí

První otázka, kterou si při konstrukci přímých metod řešení soustav lineárních rovnic klademe je, jaké soustavy leze řešit nejsnáze. Na ně se pak budeme obecné soustavy snažit ekvivalentními úpravami převést.

Představme si, že máme soustavu $\mathbf{Ax} = \mathbf{b}$, v níž se najde rovnice, která obsahuje jenom jednu neznámou (jednu složku hledaného řešení). To znamená, že v daném řádku matice $\mathbf{A}$ je jednom jeden nenulový prvek. Z takové rovnice se pak dá snadno (prostým dělením) vypočítat příslušná složka řešení. Teď si představme, že v soustavě je ještě jiná rovnice, v níž vystupují pouze dvě neznámé, a že jedna z nich je ta, kterou jsme právě vypočítali. Jestliže nyní do této druhé rovnice dosadíme hodnotu právě vypočítané složky řešení, můžeme z ní opět snadno vypočítat tu druhou. Pokud je struktura matice i nadále podobná, to jest pokud nám v každé rovnici přibude pouze jedna nová neznámá, dají se tak postupně snadno vypočítat všechny složky řešení.

Matici s touto speciální vlastností se říká **trojúhelníková**, a to z důvodu, který bude brzy zřejmý. Protože soustavy lineárních rovnic s trojúhelníkovými maticemi se snadno řeší, jsou právě ony cílem úprav prováděných v přímých numerických metodách na obecných soustavách rovnic.

Pro výpočetní účely se ukazuje jako užitečné zavést trojúhelníkové matice dvou speciálních tvarů. Řekneme, že matice $\mathbf{L}$ je **dolní trojúhelníková**, jestliže všechny její prvky nad hlavní diagonálou jsou nulové (tj. $\ell_{ij} = 0$ pro $i < j$). Podobně řekneme, že matice $\mathbf{U}$ je **horní trojúhelníková**, jestliže všechny její prvky pod hlavní diagonálou jsou nuly (tj. $u_{ij} = 0$ pro $i > j$. Poznamenáváme, že trojúhelníkovou matici definovanou výše v obecném smyslu můžeme vždy

permutacemi řádků nebo sloupců převést na dolní nebo horní trojúhelníkovou matici.

Příklad 10.6 (Horní a dolní trojúhelníková matice)**.**

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}. \quad a \quad \mathbf{U} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Jakkoli je způsob řešení soustav s trojúhelníkovými maticemi nyní již snad zřejmý, považujeme za užitečné algoritmy pro jejich řešení popsat podrobněji. Proces řešení dolní trojúhelníkové soustavy $\mathbf{Lx} = \mathbf{b}$ postupným dosazováním se nazývá **přímá substituce** a matematicky se dá zapsat jako

$$x_1 = b_1/\ell_{11}, \quad x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) / \ell_{ii}, \quad i = 2, \ldots, n.$$

Formální zápis příslušného algoritmu může vypadat například následujícím způsobem.

Algoritmus 10.1 Přímá substituce pro soustavu s dolní trojúhelníkovou maticí

```
for j = 1 to n do {cyklus přes sloupce}
  if ℓ_jj = 0 then {stop při singulární matici}
    return error
  end if
  x_j ← b_j/ℓ_jj {výpočet složky řešení}
  for i = j + 1 to n do
    b_i ← b_i − ℓ_ij x_j {aktualizace pravé strany}
  end for
end for
```

Podobně se u horní trojúhelníkové soustavy $\mathbf{Ux} = \mathbf{b}$ proces řešení postupným dosazováním nazývá **zpětná substituce** a matematicky se dá zapsat jako

$$x_n = b_n/u_n, \quad x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}, \quad i = n-1, \ldots, 1.$$

Uvádíme jeden ze způsobů algoritmické realizace zpětné substituce.

V obou těchto algoritmech jsme zvolili pořadí indexů v cyklech tak, že se k prvkům matic přistupuje po sloupcích (a ne po řádcích) a že operace, kterou provádí vnitřní cyklus, je skalár krát vektor plus vektor, operace zkracovaná **saxpy** (z angl. *scalar times a vector plus a vector*) a používaná v základních knihovnách numerické lineární algebry. Ukládání maticí po sloupcích je

Algoritmus 10.2 Zpětná substituce pro soustavu s horní trojúhelníkovou maticí

```
for j = n to 1 do {zpětný cyklus přes sloupce}
  if u_jj = 0 then {stop při singulární matici}
    return error
  end if
  x_j ← b_j/u_jj {výpočet složky řešení}
  for i = 1 to j − 1 do
    b_i ← b_i − u_ij x_j {aktualizace pravé strany}
  end for
end for
```

typické pro některé programovací jazyky, jako je například Matlab nebo Fortran. Naproti tomu třeba v jazyku C se matice ukládají po řádcích a při větších soustavách by výše popsané algoritmy bylo třeba přepsat tak, aby se v nich k maticím přistupovalo po řádcích. To nepředstavuje zásadní problém, zmiňujeme se o tom pouze proto, abychom čtenáře upozornili, že způsob implementace matematického algoritmu může podstatně ovlivnit jeho efektivitu, a to v závislosti na použitém programovacím jazyku a výpočetním systému. Poznamenáváme ještě, že výskyt nulového prvku na diagonále bude znamenat selhání uvedených algoritmů, což ovšem není překvapivé, protože trojúhelníková matice s nulovým prvkem na diagonále má nulový determinant a je tedy singulární.

Příklad 10.7 (Soustava rovnic s trojúhelníkovou maticí)**.** *Mějme soustavu rovnic s horní trojúhelníkovou maticí*

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -1 & -6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}.$$

Z poslední rovnice je dáno $x_3 = 1$*. Tuto hodnotu pak můžeme dosadit do druhé rovnice a z ní vypočítat* $-x_2 - 6 = -6$*,* $x_2 = 0$*. Nakonec do první rovnice dosadíme jak* x_3 *tak* x_2 *a vypočítáme z ní* $x_1 + 0 + 2 = 3$*,* $x_1 = 1$*.*

Gaussova eliminační metoda

Naší strategií nyní je navrhnout regulární lineární transformaci, která bude převádět danou obecnou soustavu lineárních algebraických rovnic na ekvivalentní soustavu s trojúhelníkovou maticí, již pak budeme moci snadno řešit postupnými substitucemi. Budeme tedy potřebovat transformaci, která by nahrazovala vybrané nenulové prvky dané matice nulami (vyeliminovala je).

Uvidíme, že se toho dá dosáhnout prováděním vhodných lineárních kombinací řádků dané matice.

Základem přímých numerických metod pro řešení obecných soustav lineárních algebraických rovnic je tak postup, který nyní popíšeme na příkladu soustavy čtyř rovnic o čtyřech neznámých a který se nazývá **Gaussova eliminační metoda**. Uvažujme tedy soustavu lineárních rovnic

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4 \end{aligned}$$

Položíme

$$m_{i1} = a_{i1}/a_{11}, \quad i = 2, 3, 4,$$

a odečteme m_{i1} krát první rovnici od i-té rovnice pro $i = 2, 3, 4$. Dostaneme tak upravenou ekvivalentní soustavu

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 + a'_{24}x_4 &= b'_2 \\ a'_{32}x_2 + a'_{33}x_3 + a'_{34}x_4 &= b'_3 \\ a'_{42}x_2 + a'_{43}x_3 + a'_{44}x_4 &= b'_4 \end{aligned}$$

kde

$$a'_{ij} = a_{ij} - m_{i1}a_{1j} \quad \text{a} \quad b'_i = b_i - m_{i1}b_1.$$

Vidíme, že neznámá x_1 byla z posledních tří rovnic vyeliminována. Protože při eliminaci se čísly m_{i1} násobí první rovnice říká se jim **multiplikátory** (z angl. *multiply*, násobit). Nyní položíme

$$m_{i2} = a'_{i2}/a'_{22}, \quad i = 3, 4,$$

a odečteme m_{i2} krát druhou rovnici od i-té rovnice ($4i = 3, 4$). Výsledkem je soustava

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 + a'_{24}x_4 &= b'_2 \\ a''_{33}x_3 + a''_{34}x_4 &= b''_3 \\ a''_{43}x_3 + a''_{44}x_4 &= b''_4 \end{aligned}$$

kde

$$a''_{ij} = a'_{ij} - m_{i2}a'_{2j} \quad \text{a} \quad b''_i = b'_i - m_{i2}b'_2.$$

A nakonec položíme

$$m_{i3} = a''_{i3}/a''_{33}, \quad i = 4,$$

a odečteme m_{i3} krát třetí rovnici od rovnice čtvrté. Výsledkem je soustava s horní trojúhelníkovou maticí

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 + a'_{24}x_4 &= b'_2 \\ a''_{33}x_3 + a''_{34}x_4 &= b''_3 \\ a'''_{44}x_4 &= b'''_4 \end{aligned}$$

kde

$$a'''_{ij} = a''_{ij} - m_{i3}a''_{3j} \quad \text{a} \quad b'''_i = b''_i - m_{i3}b''_{33}.$$

Protože má tato soustava horní trojúhelníkovou matici, lze ji snadno vyřešit zpětnou substitucí.Postupu, kterým jsme původní soustavu lineárních rovnic převedli na ekvivalentní soustavu s trojúhelníkovou maticí se také říká **přímý chod** a řešení výsledné trojúhelníkové soustavy **zpětný chod** Gaussovy eliminace.Prvkům matic, které postupně objevují na hlavní diagonále, tedy v našem případě $a_{11}, a'_{22}, a''_{33}, a'''_{44}$, se říká **hlavní prvky** nebo také **pivoty**.

Popsaný postup je proveditelný pouze, pokud jsou při něm všechny hlavní prvky nenulové. Pokud během eliminace narazíme na nulový hlavní prvek, postupujeme tak, že rovnici s tímto prvkem prohodíme z některou ze zbývajících rovnic, která má v daném sloupci nenulový koeficient. Pokud by ovšem v daném sloupci matice už byly všechny prvky pod diagonálou nulové (včetně hlavního prvku), není těžké ukázat, že by to znamenalo, že matice řešené soustavy je singulární (a tedy by soustava buď neměla řešení nebo by jich měla nekonečně mnoho).

Gaussovu eliminační metodu ilustrujeme na jednoduchém konkrétním příkladu řešení soustavy tří rovnic o třech neznámých.

Příklad 10.8 (Gaussova eliminace)**.** *Budeme řešit soustavu lineárních rovnic*

$$\begin{aligned} x_1 + 2\,x_2 + 2\,x_3 &= 3, \\ 4\,x_1 + 4\,x_2 + 2\,x_3 &= 6, \\ 4\,x_1 + 6\,x_2 + 4\,x_3 &= 10, \end{aligned}$$

která se dá maticově zapsat jako

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 10 \end{bmatrix} = \mathbf{b}.$$

Pro účely Gaussovy eliminace je vhodné zapsat data úlohy ve formě tak zvané ***rozšířené matice*** *soustavy*

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 3 \\ 4 & 4 & 2 & 6 \\ 4 & 6 & 4 & 10 \end{array}\right]$$

a postup Gaussovy eliminace provádět na této matici; je to tak přehlednější. Po prvním kroku eliminace, během kterého odečteme čtyřnásobek prvního řádku rozšířené matice od druhého a třetího řádku, rozšířená matice přejde na

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 3 \\ 0 & -4 & -6 & -6 \\ 0 & -2 & -4 & -2 \end{array}\right].$$

Abychom nyní vynulovali člen pod diagonálou v druhém sloupci získané rozšířené matice, odečteme druhý řádek vynásobený jednou polovinou od třetího řádku a dostaneme tak matici

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 3 \\ 0 & -4 & -6 & -6 \\ 0 & 0 & -1 & 1 \end{array}\right].$$

To je již rozšířená matice trojúhelníkové soustavy rovnic, z níž můžeme zpětnou substitucí dostat hledané řešení jako

$$\mathbf{x} = \begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}.$$

Algoritmus, který jsme výše obecně popsali pro soustavy čtyř rovnic, se dá snadno zobecnit pro soustavy libovolného řádu. Ukážeme nyní, že výše odvozený postup Gaussovy eliminační metody se dá elegantně zapsat pomocí maticových transformací, v nichž opět figurují trojúhelníkové matice.

LU rozklad matice

Ukážeme opět na příkladu obecné soustavy 4×4, že během Gaussovy eliminace vlastně konstruujeme dolní trojúhelníkovou matici $\mathbf{L}$ a horní trojúhelníkovou matici $\mathbf{U}$ takové, že $\mathbf{A} = \mathbf{LU}$.

Položíme $\mathbf{A}_1 = \mathbf{A}$ a dále

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -m_{21} & 1 & 0 & 0 \\ -m_{31} & 0 & 1 & 0 \\ -m_{41} & 0 & 0 & 1 \end{bmatrix},$$

kde čísla m_{ij} jsou multiplikátory zavedené v předchozím odstavci. Pak se dá snadno ověřit (vzpomeňte si na Odstavec 10.2), že platí

$$\mathbf{A}_2 = \mathbf{M}_1\mathbf{A}_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{bmatrix},$$

což je matice soustavy, která výše vznikla po prvním kroku Gaussovy eliminační metody. Nyní položíme

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -m_{32} & 0 & 0 \\ 0 & -m_{42} & 0 & 1 \end{bmatrix}.$$

Opět nyní není těžké ukázat, že

$$\mathbf{A}_3 = \mathbf{M}_2\mathbf{A}_2 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & a''_{43} & a''_{44} \end{bmatrix},$$

což je matice soustavy, kterou jsme výše obdrželi po provedení druhého kroku eliminace. Konečně položíme

$$\mathbf{M}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -m_{43} & 1 \end{bmatrix}.$$

Potom

$$\mathbf{U} = \mathbf{M}_3\mathbf{A}_3 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & 0 & a'''_{44} \end{bmatrix}$$

je horní trojúhelníková matice výsledné soustavy získané Gaussovou eliminační metodou. Jak jsme již poznamenali v Odstavci 10.2, matice $\mathbf{M}_k$, které se během právě popsaného postupu vyskytly, se v literatuře nazývají **elementární eliminační matice**.

Zapíšeme-li vše maticově, vidíme, že pro horní trojúhelníkovou matici $\mathbf{U}$ platí

$$\mathbf{U} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{A}.$$

Položíme-li tedy

$$\mathbf{L} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{M}_3^{-1},$$

pak

$$\mathbf{A} = \mathbf{L}\mathbf{U}.$$

Protože inverze dolní trojúhelníkové matice je opět dolní trojúhelníková a protože součin dolních trojúhelníkových matic je opět dolní trojúhelníková matice, je výše sestrojená matice $\mathbf{L}$ dolní trojúhelníková matice, a ukázali jsme tak tedy, že Gaussova eliminační metoda vlastně realizuje LU rozklad matice

A na součin dolní a horní trojúhelníkové matice. Této skutečnosti se hojně využívá při realizaci algoritmů Gaussovy eliminace na počítači.

Dosud jsme se nijak nezabývali výpočtem prvků matice **L**. Jejich stanovení je však překvapivě jednoduché, neboť se dá ukázat, že matice **L** má na hlavní diagonále jedničky a pod diagonálou má v pozici $\ell_{ij}, i > j$, multiplikátor m_{ij}. Není tak těžké to ověřit. Poznamenáváme opět, že zde popsané úvahy o trojúhelníkovém rozkladu matice nezávisejí na řádu matice a dají se snadno zobecnit pro obecné n.

Algoritmus LU rozkladu

Pokud již máme k dispozici rozklad $\mathbf{A} = \mathbf{LU}$, můžeme snadno převést úlohu řešit soustavu $\mathbf{Ax} = \mathbf{b}$ na řešení dvou soustav rovnic s trojúhelníkovými maticemi. Je to okamžitě vidět z řešené soustavy, kam za **A** dosadíme její trojúhelníkový rozklad. Nejprve vyřešíme soustavu

$$\mathbf{Lz} = \mathbf{b},$$

jejíž řešení **z** je vlastně eliminací transformovaná pravá strana původní soustavy, a pak soustavu

$$\mathbf{Ux} = \mathbf{z},$$

jejímž řešením je řešení původní soustavy rovnic. Poznamenáváme ještě, že pokud jsme pro danou matici jednou stanovili její trojúhelníkový rozklad, můžeme tímto způsobem řešit s menší pracností i více soustav rovnic, které se liší pouze pravou stranou. To bývá někdy v praxi užitečné.

Zbývá nám pro případné zájemce popsat algoritmus výpočtu matic **L** a **U**. Na počítači zpravidla prvky těchto matic ukládáme postupně na místa původně obsazená prvky matice **A**. Co do počtu míst to vychází, protože víme, že diagonální prvky matice **L** jsou jedničky a nemusíme je tedy ukládat. Příslušný algoritmus má řadu variant, uvádíme zde jednu z možností.

Výběr hlavního prvku

Gaussova eliminační metoda v sobě nese jeden zřejmý problém, o kterém jsme se již zmínili, ale také jeden další, méně zřetelný problém. Možným zřejmým problémem je to, že celý proces selže, pokud se během eliminace vyskytne nulový hlavní prvek, protože by se při výpočtu multiplikátorů v odpovídajícím sloupci dělilo nulou. Ale řešení tohoto problému je téměř stejně zřejmé: pokud je v kroku k eliminace na diagonále nulový hlavní prvek, prohodíme řádek k v soustavě (tedy v její matici i v její pravé straně, jinak řečeno, řádek k v rozšířené matici soustavy) s nějakým z pod ním ležících řádků, jehož prvek ve sloupci k je nenulový. Z Odstavce 10.2 již víme, že tím se řešení soustavy nezmění. S nenulovým diagonálním prvkem jako hlavním prvkem

Algoritmus 10.3 LU rozklad Gaussovou eliminací

```
for k = 1 to n − 1 do {cyklus přes sloupce}
  if a_kk = 0 then
    return {stop, je-li hlavní prvek nulový}
  end if
  for i = k + 1 to n do {výpočet multiplikátorů pro daný sloupec}
    a_ik ← a_ik/a_kk
  end for
  for j = k + 1 to n do
    for i = k+1 to n do {aplikujeme transformaci na zbývající submatici}
      a_ij ← a_ij − a_ik a_kj
    end for
  end for
end for
```

pak celý postup pokračuje jako obvykle. Takové přehazování řádků se pak nazývá **výběr hlavního prvku** nebo také **pivotace**.

Příklad 10.9 (Pivotace a singularita)**.** *Případná nutnost provádět výběr hlavního prvku a přehazovat řádky nemá nic společného s tím, zda daná matice je singulární či nikoliv. Tak například matice*

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

je zřejmě regulární (spočítejte si determinant), ale pokud nepřehodíme řádky, nemá žádný LU rozklad. Naproti tomu singulární matice

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

má (bez přehazování řádků) LU rozklad

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \mathbf{LU}.$$

Ale co dělat v případě, že na hlavní diagonále ani pod ní ve sloupci k není žádný nenulový prvek? Pak v daném kroku není třeba provádět nic, protože všechny prvky, které bychom měli eliminovat jsou nulové, a přejdeme tedy prostě k dalšímu sloupci (je tedy $\mathbf{M}_k = \mathbf{I}$). Všimněme si toho, že tento krok zanechá na diagonále nulový prvek, takže výsledná matice $\mathbf{U}$ bude singulární, nicméně LU rozklad lze stejně dokončit. Znamená to ovšem, že s takovou maticí selže zpětný chod, protože se v něm vyskytuje dělení každým z diagonálních prvků matice $\mathbf{U}$, ale to není až tolik překvapivé, protože v takovém

případě je nutně singulární už původní matice **A** (stačí spočítat si její determinant z determinantů jejích trojúhelníkových faktorů, což jsou součiny jejich diagonálních prvků).

Oním méně zřetelným problémem je to, že v počítačové aritmetice nám na diagonále nemusí vyjít přesná nula, ale pouze nějaké velmi malé číslo, což nás vede k následující úvaze.

Jako hlavní prvek se při výpočtu multiplikátorů dá v zásadě využít jakékoli nenulové číslo, ale v počítačové aritmetice s konečnou přesností bychom se měli zajímat také o to, jak minimalizovat šíření zaokrouhlovacích chyb během výpočtu. Konkrétně je naším cílem omezit velikosti multiplikátorů, takže se již existující zaokrouhlovací chyby vzniklé v předchozích krocích eliminace nebudou zesilovat při následujících transformacích realizovaných elementárními eliminačními maticemi. Absolutní hodnota multiplikátorů zaručeně nikdy nepřevýší jedničku, jestliže v každém sloupci vybereme za hlavní prvek ten z prvků na diagonále nebo pod ní, který má největší absolutní hodnotu. Takové strategii se říká **částečný výběr hlavního prvku** a v praxi je tento výběr pivota podstatný pro to, abychom mohli sestrojit numericky stabilní implementaci Gaussovy eliminace pro obecné matice. Poznamenejme pro pořádek, že slovo *částečný* je zde použito proto, že vhodný hlavní prvek hledáme pouze v prvním sloupci dosud nevyeliminované submatice. Mohli bychom jej vybírat i v celé této submatici, pak bychom ovšem možná museli permutovat i některé sloupce. Navíc tento způsob volby pivota (**úplný výběr hlavního prvku**) nepřináší žádné podstatné výhody.

Ukážeme nyní na extrémním, ale jednoduchém příkladu, jak se může špatná volba pivota při výpočtu projevit. Podobný efekt bychom mohli pozorovat s méně přehnanou volbou pivota u větších soustav či matic.

Příklad 10.10 (Malé hlavní prvky)**.** *Pracujeme-li v počítačové aritmetice, která má pouze konečnou přesnost, musíme se vyhýbat nejen nulovým hlavním prvkům, ale také hlavním prvkům, které jsou malé vzhledem k ostatním prvkům daného sloupce, a to proto, abychom zabránili nepřijatelnému růstu chyb. Ukážeme to na velmi zjednodušeném příkladu, kde symbol* $\approx$ *znamená výsledek počítačové operace. Označíme zde* ϵ *jakékoli kladné číslo, které je menší než strojové epsilon* ϵ_{mach}*, tedy jakékoli dostatečně malé číslo* x*, pro které platí* $1 + x \approx 1$*. Pokud chce čtenář být konkrétní, může vzít v jednoduché aritmetice* $\epsilon = 10^{-8}$ *a v dvojnásobné přesnosti, se kterou normálně pracuje Matlab, vzít* $\epsilon = 10^{-16}$*.*

Vezměme nyní matici

$$\mathbf{A} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}.$$

To je regulární matice, jejíž determinant je zhruba roven -1 *a která je dobře podmíněná, má číslo podmíněnosti řádu jednotek. Pokud nepřehodíme řádky,*

bude hlavním prvkem ϵ, takže výsledný multiplikátor je $-1/\epsilon$ a

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}.$$

Jako výsledná horní trojúhelníková matice po eliminaci nám na počítači vyjde

$$\mathbf{U} = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix} \approx \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}.$$

Vynásobíme-li ale nyní matice $\mathbf{L}$ *a* $\mathbf{U}$ *získané na počítači, dostaneme*

$$\mathbf{LU} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \neq \mathbf{A}.$$

Použití malého prvku a tím pádem velkého multiplikátoru způsobilo velkou ztrátu informace ve výsledné matici $\mathbf{U}$.

Přehodíme-li řádky, bude nyní hlavní prvek roven 1 *a odpovídající multiplikátor bude $-\epsilon$, takže*

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix}.$$

Jako výsledná horní trojúhelníková matice nám na počítači tentokrát vyjde

$$\mathbf{U} = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{bmatrix} \approx \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Vynásobením takto získaných matic $\mathbf{L}$ *a* $\mathbf{U}$ *nyní dostaneme*

$$\mathbf{LU} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix},$$

což je přesně původní matice $\mathbf{A}$ *s přehozenými řádky.*

Dá se ukázat, že přehazování řádků během přímého chodu Gaussovy eliminace se dá shrnout jako provedení jedné permutační transformace na původní matici a že tedy na rozdíl od přímého LU rozkladu matice $\mathbf{A}$, který nemusí vždy existovat (viz výše), ke každé čtvercové matici $\mathbf{A}$ existuje permutační matice $\mathbf{P}$ tak, že k permutované matici lze LU rozklad nalézt, tedy

$$\mathbf{PA} = \mathbf{LU}.$$

Algoritmus LU rozkladu popsaný v předchozím odstavci se při výběru hlavních prvků příliš nemění, je třeba si pouze uvědomit, že nyní místo soustavy $\mathbf{Ax} = \mathbf{b}$ řešíme ekvivalentní soustavu $\mathbf{PAx} = \mathbf{Pb}$. Po získání LU rozkladu matice $\mathbf{PA}$ (matice $\mathbf{P}$ vlastně vzniká teprve během tohoto rozkladu) nejprve

řešíme soustavu $\mathbf{Lz} - \mathbf{Pb}$ s dolní trojúhelníkovou maticí přímou substitucí a poté řešíme zpětnou substitucí soustavu $\mathbf{Ux} = \mathbf{z}$ s horní trojúhelníkovou maticí. V numerické lineární algebře se ukazuje, že existují třídy matic, pro které je výběr hlavního prvku zbytečné provádět a kde tedy výše uvedená permutační matice je jednotková. Patří se například pozitivně definitní symetrické matice, s nimiž se často setkáváme v odborné praxi. Pro nedostatek místa zde explicitně algoritmus LU rozkladu s výběrem hlavního prvku neuvádíme.

Příklad 10.11 (Gaussova eliminace s částečnou pivotací)**.** *V Příkladu 10.8 jsme nepoužívali permutace řádků a v důsledku toho byly některé multiplikátory větší než jedna. Jako ilustraci nyní tento příklad zopakujeme, ale budeme tentokrát používat částečný výběr hlavního prvku.*

Soustava, kterou jsme řešili v Příkladu 10.8, je

$$\mathbf{Ax} = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 10 \end{bmatrix} = \mathbf{b}$$

a její rozšířená matice je

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 3 \\ 4 & 4 & 2 & 6 \\ 4 & 6 & 4 & 10 \end{array}\right].$$

Největší prvek v prvním sloupci je 4, takže prohodíme první dva řádky a permutovaná rozšířená matice bude

$$\left[\begin{array}{ccc|c} 4 & 4 & 2 & 6 \\ 1 & 2 & 2 & 3 \\ 4 & 6 & 4 & 10 \end{array}\right].$$

Abychom vynulovali prvky pod diagonálou v prvním sloupci, odečteme od druhého řádku 0,25 *krát první řádek a od třetího řádku první řádek krát jedna. jako transformovanou matici tak dostaneme*

$$\left[\begin{array}{ccc|c} 4 & 4 & 2 & 6 \\ 0 & 1 & 1{,}5 & 1{,}5 \\ 0 & 2 & 2 & 4 \end{array}\right].$$

Poslední prvek pod diagonálou v druhém sloupci je 2, takže přehodíme poslední dva řádky a dostaneme tak opět permutovanou matici

$$\left[\begin{array}{ccc|c} 4 & 4 & 2 & 6 \\ 0 & 2 & 2 & 4 \\ 0 & 1 & 1{,}5 & 1{,}5 \end{array}\right].$$

Abychom vynulovali subdiagonální prvek ve druhém sloupci, odečteme od třetího řádku $1/2$*-násobek druhého řádku a dostaneme tak konečný tvar vyeliminované matice jako*

$$\left[\begin{array}{ccc|c} 4 & 4 & 2 & 6 \\ 0 & 2 & 2 & 4 \\ 0 & 0 & 0{,}5 & -0{,}5 \end{array}\right].$$

Převedli jsme tak původní soustavu rovnic na ekvivalentní soustavu rovnic s horní trojúhelníkovou maticí, kterou můžeme nyní vyřešit zpětnou substitucí a dostáváme tak stejně jako v Příkladu 10.8

$$\mathbf{x} = \begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}.$$

Pro zájemce uvádíme ještě výsledek ve formě LU rozkladu. Výsledná permutační matice je

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

a máme

$$\mathbf{PA} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0{,}25 & 0{,}5 & 1 \end{bmatrix} \begin{bmatrix} 4 & 4 & 2 \\ 0 & 2 & 2 \\ 0 & 0 & 0{,}5 \end{bmatrix} = \mathbf{LU}.$$

10.3 Dodatky

Numerické chyby při Gaussově eliminaci

Jak jsme již uvedli v Odstavci 10.1, platí pro relativní reziduum vypočítaného řešení nerovnost

$$\frac{\|\mathbf{r}\|}{\|\mathbf{A}\| \cdot \|\hat{\mathbf{x}}\|} \leq \frac{\|\mathbf{E}\|}{\|\mathbf{A}\|},$$

kde $\mathbf{E}$ je zpětná chyba v matici $\mathbf{A}$. Ale jak velká asi bude $\|\mathbf{E}\|$ při praktických výpočtech? V numerické lineární algebře se ukazuje, že pro LU rozklad Gaussovou eliminací platí odhad tvaru

$$\frac{\|\mathbf{E}\|}{\|\mathbf{A}\|} \leq \rho\, n\, \epsilon_{\text{mach}},$$

kde ρ se nazývá **růstový faktor** a ϵ_{mach} je strojové epsilon charakterizující přesnost počítačové aritmetiky. V Matlabu je standardně strojové epsilon k dispozici jako proměnná `eps` a je řádové velikosti 10^{-16}. Pokud jde o růstový faktor, je to poměr největšího prvku matice $\mathbf{U}$ k největšímu prvku matice $\mathbf{A}$ braný v absolutních hodnotách.

Pokud neprovádíme výběr hlavního prvku, může být tento faktor ρ libovolně velký, a tím pádem není Gaussova eliminační metoda bez pivotace obecně numericky stabilní (nemusí poskytovat malou zpětnou chybu). Pokud provádíme částečný výběr hlavního prvku, může být růstový faktor teoreticky velikosti 2^{n-1} (protože v nejhorším možném případě se velikost prvků může v každém kroku eliminace zdvojnásobit), ale takové chování Gaussovy eliminační metody je v praxi nesmírně vzácné. V prakticky řešených příkladech dochází při částečném výběru hlavních prvků pouze k malému nebo žádnému růstu mezivýsledků, takže se dá přibližně brát jako odhad zpětné chyby

$$\frac{\|\mathbf{E}\|}{\|\mathbf{A}\|} \lessapprox n\,\epsilon_{\mathrm{mach}}.$$

Tento vztah znamená, že při řešení soustav lineárních rovnic Gaussovou eliminací s částečným výběrem hlavního prvku téměř vždy dostáváme velmi malá relativní rezidua, a to bez ohledu na to, jak špatně podmíněná řešená soustava je. Malé relativní reziduum tak nutně neznamená, že jsme získali přesný výsledek; to platí pouze u dobře podmíněných soustav. Doporučujeme čtenáři, aby se v této souvislosti znovu podíval na Příklad **??**.

Metoda LU rozkladu na počítači

Existuje několik způsobů jak implementovat metodu LU rozkladu na počítači, které se liší v prvé řadě tím, jak se přistupuje k prvkům matice $\mathbf{A}$ (po řádcích nebo po sloupcích). Účinnost těchto implementací se na daném počítači může výrazně lišit a jedna a ta samá implementace se může na různých výpočetních systémech chovat různě. Některé z variant Gaussovy eliminační metody si dokonce vysloužily vlastní pojmenování, takže máme například Croutovu metodu nebo Doolittlovu metodu. Speciální varianta metody LU rozkladu pro řešení soustav rovnic s třídiagonálními maticemi bývá v inženýrské praxi často nazývána Thomasovým algoritmem. Jakkoli se ale různé implementace LU rozkladu mohou významně lišit co do výkonnosti, produkují pro danou matici $\mathbf{A}$ v podstatě tentýž výsledný rozklad.

Pokud jde o hospodaření s pamětí, je běžné, že vytvářené trojúhelníkové faktory postupně přepisují původní prvky matice $\mathbf{A}$. Prostorově to přesně vychází, protože matice $\mathbf{L}$ má na diagonále jedničky a ty není třeba ukládat.

Aby se minimalizoval fyzický pohyb dat, neprovádí se přehazování řádků při výběru hlavních prvků explicitně. Místo toho řádky zůstávají na svých původních místech, ale k zaznamenávání jejich pořadí se používá pomocný celočíselný vektor indexů. Je totiž zřejmé, že konečný efekt všech záměn řádků je pouze nějaká permutace čísel $1, \ldots, n$.

V Matlabu je algoritmus metody LU rozkladu s částečným výběrem hlavního prvku (včetně řešení obou trojúhelníkových soustav) zabudován do operátoru \ (zpětné lomítko, angl. backslash). Řešení soustavy rovnic $\mathbf{Ax} = \mathbf{b}$ zapisujeme

v Matlabu jako `x = A\b`. Matlab má také k dispozici funkci `inv` pro výpočet inverzní matice. Výslovně ale upozorňujeme, že počítat řešení soustavy jako `x = inv(A)*b` je zcela nesprávné. Takovýto způsob výpočtu je totiž podstatně pracnější než použití zpětného lomítka a je také horší z hlediska numerické stability, zejména u větších matic. Zápis $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ je tedy užitečný pouze v teoretických úvahách na papíře.

Výpočetní složitost řešení lineárních soustav

Výpočet LU rozkladu matice řádu n Gaussovou eliminací vyžaduje zhruba $n^3/3$ násobení v pohyblivé řádové čárce a zhruba stejný počet sčítání. Řešení vzniklých trojúhelníkových soustav pro jednu pravou stranu si vyžádá asi n^2 násobení a podobný počet sčítání. Jestliže tedy řád matice n roste, začne v celkové výpočetní práci stále více převládat konstrukce LU rozkladu.

Z hlediska výpočetní složitosti je výpočet inverzní matice samotné asi třikrát náročnější než řešení soustavy s jednou pravou stranou. Dalším postupem, kterému je třeba se při výpočtech řešení soustav vyhnout, je Cramerovo pravidlo, v němž se každá složka řešení počítá jako podíl dvou determinantů. Pro větší matice (už jistě od řádu dvacet) je tento postup přímo astronomicky náročný na výpočetní práci a Cramerovo pravidlo se tak dá použít buď u matic velmi malého řádu nebo v teoretické lineární algebře.

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill, Boston, 2 edition, 2002.

[2] H. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 2 edition, 2002.

[3] Stanislav Míka. *Numerické metody algebry.* MVŠT. SNTL, Praha, 1982.

[4] Stanislav Míka. *Numerické metody algebry.* MVŠT. SNTL, Praha, 2., opr. edition, 1985.

[5] Stanislav Míka and Marek Brandner. *Numerické metody I.* FAV ZČU, Plzeň, 2000. zkonotrolovat.

KAPITOLA 11

Aproximace a interpolace

11.1 Numerické úlohy v analýze a aproximace funkcí

Předmětem studia matematické analýzy jsou funkce a operace s nimi. Typickými úlohami matematické analýzy jsou například výpočet integrálu z dané funkce přes daný interval, výpočet hodnoty derivace, řešení diferenciálních rovnic nebo i pouhé stanovení hodnoty funkce v některém bodě. Pro úlohy matematické analýzy je charakteristické, že zpravidla nejde o numerické úlohy, tj. takové úlohy, kde vstupní i výstupní data jsou vektory o konečném počtu složek. Protože na počítači můžeme přímo modelovat (naprogramovat) pouze úlohy numerické (příkladem mohou sloužit úlohy lineární algebry, viz Přednáška 11), bývá při řešení úloh matematické analýzy na počítači nezbytným přípravným krokem přibližné nahrazení (aproximace) dané matematické úlohy úlohou numerickou. Numerické metody matematické analýzy mají proto poněkud jiný charakter než numerické metody algebry.

Obsah této přednášky je podobně, ale mnohem podrobněji vyložen v učebnicích [2], [3] a skriptech [4]. Zájemce najde jiný styl výkladu a další podrobnosti například v Heathově učebnici [1].

Aproximace funkcí

Jedním ze základních úkolů numerických metod matematické analýzy je studium aproximací funkcí. Při numerickém řešení úloh matematické analýzy totiž často nahrazujeme danou funkci f, vystupující v řešené matematické úloze, jinou funkcí φ, která v nějakém vhodném smyslu napodobuje funkci f a snadno se přitom matematicky zpracovává či modeluje na počítači. Tuto funkci φ pak nazýváme **aproximací (přiblížením)** funkce f.

Oblasti matematiky, v nichž používáme aproximace, jsou značně různorodé. V této přednášce se budeme zabývat pouze těmi úlohami matematické analýzy, jejichž vstupními a výstupními daty jsou reálné funkce jedné reálné proměnné. Již pouhý výpočet funkčních hodnot takových funkcí na počítači se provádí užitím aproximací φ, jejichž hodnoty se dají vypočítat pomocí konečného počtu aritmetických a logických operací. Typicky se zde používají polynomy nebo racionální funkce; to jsou totiž nejobecnější funkce, jejichž hodnoty se dají na počítači přímo vyčíslit pomocí konečného počtu operací. Tyto aproximace jsou ovšem pro řadu funkcí již zabudovány do výpočetního systému a uživatel počítače si často ani neuvědomuje, že píše-li ve svém programu například `y = sin(x)`, nahrazuje výpočet hodnoty funkce $\sin x$ výpočtem hodnoty jistého polynomu, aproximace.

V této přednášce se z časových důvodů omezíme pouze na aproximace pomocí polynomů. Nebudeme se zde zabývat například aproximacemi pomocí racionálních funkcí, jakkoli mají své výhody (a nevýhody) a v posledních letech se jim věnuje v numerické matematice značná pozornost. Podobně vynecháváme aproximace pomocí trigonometrických polynomů a Fourierovu analýzu vůbec.

Typickým příkladem použití aproximací v matematické analýze jsou také numerické metody pro výpočet určitého integrálu z funkce f. Zde nahrazujeme funkci f funkcí φ, která se snadno integruje, například polynomem. Další oblastí numerické matematiky založenou na užití aproximací je zpracování výsledků měření. Hledáme tu zpravidla jednoduchý analytický výraz vyjadřující (přibližně) funkční závislost, zadanou tabulkou naměřených hodnot.

Při použití aproximací tedy místo původní úlohy, ve které vystupovala funkce f, řešíme úlohu, v níž místo f vystupuje její aproximace φ. To má ovšem za následek, že výsledkem výpočtu nebude přesné řešení původní úlohy. Úkolem numerických metod analýzy je proto také zkoumat, co můžeme říci o odchylce získaného přibližného řešení od přesného (teoretického) řešení dané úlohy. Takové odchylce se říká **chyba aproximace**.

Třídy aproximujících funkcí

V celé této přednášce se budeme zabývat aproximacemi spojitých reálných funkcí jedné reálné proměnné. Při výběru vhodné aproximace postupujeme v numerické analýze tak, že nejprve předem zvolíme tvar aproximující funkce, ve kterém vystupují některé proměnné parametry, a hodnoty těchto parametrů se pak snažíme určit tak, aby získaná aproximace vyhovovala našim konkrétním požadavkům.

Obecně se při hledání vhodné aproximace velmi často postupuje takto následujícím způsobem. Zvolíme nejprve pevně systém jednoduchých **bázových funkcí** $\varphi_0, \varphi_1, \ldots, \varphi_n$ takových, které se snadno matematicky zpracovávají nebo se s nimi dobře pracuje na počítači, a danou funkci f pak aproximujeme

lineární kombinací φ těchto základních funkcí. Klademe tedy

$$\varphi(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + \cdots + c_n\varphi_n(x). \tag{11.1}$$

Otázka volby aproximace k funkci f se tak převádí na určení hodnot parametrů $c_0, c_1, \ldots, c_n$ podle nějakého kritéria vhodného pro tu či onu konkrétní úlohu. Protože aproximace φ daná uvedeným výrazem závisí na parametrech $c_0, c_1, \ldots, c_n$ lineárně, říkáme o ní, že je **lineárního typu**. Při pevných bázových funkcích nazýváme množinu všech možných jejich lineárních kombinací **třídou aproximujících funkcí** (lineárního typu).

Příklady často používaných bázových funkcí jsou

1. $1, x, x^2, \ldots, x^n$,
2. $1, x - x_0, (x - x_0)^2, \ldots, (x - x_0)^n$, x_0 pevné,
3. $1, \cos x, \sin x, \ldots, \cos Lx, \sin Lx$ $(n = 2L)$,
4. $1, \mathrm{e}^{\mathrm{i}x}, \mathrm{e}^{\mathrm{i}2x}, \ldots, \mathrm{e}^{\mathrm{i}nx}$ $(\mathrm{i}^2 = -1)$.

Bázové funkce uvedené v bodech 1 a 2 vytvářejí **třídu polynomů** stupně nejvýše n. Pro bázové funkce uvedené v bodech 3 a 4 se aproximující funkce lineárního typu nazývají trigonometrické polynomy.

Příkladem tříd aproximujících funkcí, které nejsou lineárního typu, jsou třídy racionálních funkcí (to jsou podíly dvou polynomů). Díky tomu, že tyto funkce závisejí na svých parametrech nelineárně, je teorie a praxe takových aproximací ve srovnání s aproximacemi lineárního typu složitější. Pro některé úlohy však aproximace nelineárního typu dávají velmi dobré výsledky.

Výběr aproximující funkce

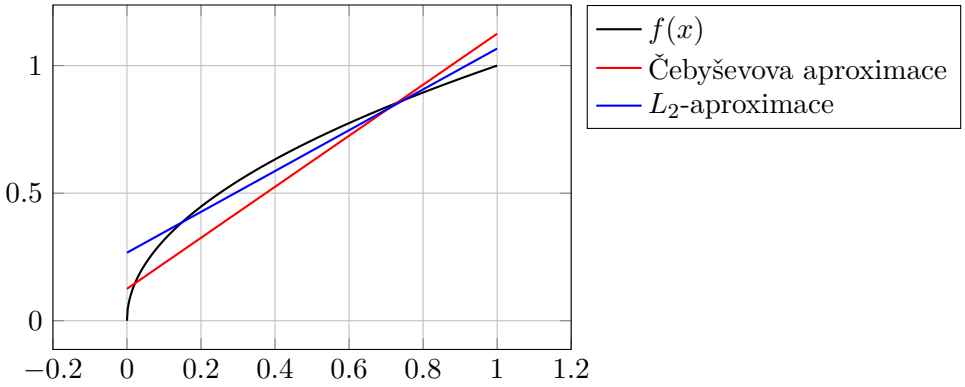
Definice 11.1 (Chyba aproximace)**.** *Budiž f daná funkce a φ její aproximace na intervalu $[a, b]$. Funkci E danou pro $x \in [a, b]$ vztahem*

$$E(x) = f(x) - \varphi(x)$$

*budeme nazývat **chybou aproximace**.*

Chceme-li vybrat funkci φ z dané třídy aproximujících funkcí tak, aby byla dobrou aproximací funkce f, budeme se jistě snažit o to, aby byla chyba aproximace v nějakém smyslu malá. Ve většině úloh nás přitom prakticky nezajímá znaménko chyby a při posuzování kvality aproximace tak pracujeme pouze s její absolutní hodnotou $|E(x)| = |f(x) - \varphi(x)|$.

Možností, jak posuzovat („měřit“) velikost funkce E na intervalu $[a, b]$, je ale celá řada. V první řadě je, podobně jako tomu bylo u vektorů, potřeba přesně



Obrázek 11.1: Původní funkce $f(x) = \sqrt{x}$ a její nejlepší Čebyševova a L_2-aproximace

říci, co rozumíme „velikostí“ funkce. Pojem absolutní hodnoty čísla se tak podobně jako tomu bylo u vektorů rozšiřuje na pojem **normy funkce**. Normu funkce F označujeme $\|F\|$ a některé používané normy zavedeme v příštích odstavcích. Jakou konkrétně zvolíme normu v daném případě závisí na tom, jaké požadavky na velikost chyby v řešené úloze klademe. Můžeme například požadovat, aby maximální hodnota $|E(x)|$ na intervalu $[a, b]$ byla menší než nějaké dané číslo ε. Jindy pro nás může být důležitá velikost $|E(x)|$ pouze v některých bodech intervalu $[a, b]$. Můžeme třeba požadovat, aby byl malý integrál z funkce $|E(x)|$ přes celý interval $[a, b]$. A jsou i další možné přístupy. To, jakým způsobem budeme chybu aproximace měřit, závisí tedy do značné míry na naší libovůli. Volbu vhodného kritéria pro měření velikosti chyby, podle něhož chybu měříme v té či oné normě, pak provádíme s přihlédnutím k řešené matematické úloze a způsobu zadání funkce f.

Příklad 11.1 (Nejlepší aproximace: spojitý případ)**.** *Ukážeme, že nejlepší aproximace se mohou pro různé normy použité k měření chyby aproximace podstatně lišit. Postup odvození nejlepších aproximací jsme zde nuceni vynechat, uvádíme pouze výsledky. Uvedeme zde dvě různé nejlepší aproximace funkce* $f(x) = \sqrt{x}$ *na intervalu* $[0, 1]$ *polynomem prvního stupně. Jako bázové funkce tedy volíme* $\varphi_0(x) = 1$, $\varphi_1(x) = x$ *a nejlepší aproximace hledáme ve třídě funkcí tvaru* $\varphi(x) = c_0 + c_1 x$.

Nejlepší Čebyševova aproximace φ^*_∞ *má tvar* $\varphi^*_\infty(x) = x + \frac{1}{8}$ *a Čebyševova norma její chyby je* $\|E\|_\infty = 0{,}125$. *Nejlepší* L_2*-aproximace* φ^*_L *má tvar* $\varphi^*_L(x) = \frac{4}{5}x + \frac{4}{15}$ *a* L_2*-norma její chyby je* $0{,}0471$. *Může být zajímavé nakreslit si graf aproximované funkce i obou nejlepších aproximací – pro kontrolu jej k nahlédnutí uvádíme na Obrázku 11.1.*

Za zmínku stojí ještě to, že L_2*-norma chyby Čebyševovy aproximace je* $0{,}0854$ *a Čebyševova norma nejlepší* L_2*-aproximace je* $0{,}267$. *Vidíme tedy, že při hle-*

dání optimální aproximace zřetelně záleží na našem hledisku, tj. na požadavcích, které na hledanou aproximaci klademe.

Měření chyby aproximace: diskrétní případ

Funkce f, kterou chceme aproximovat, může být zadána různými způsoby. V praxi je velmi častý případ, kdy je funkce f dána tabulkou

$$\{(x_i, f(x_i)), i = 0, 1, \ldots, m\}$$

$m+1$ funkčních hodnot $f(x_i)$ v bodech $x_i \in [a, b]$. V takových případech měříme chybu aproximace diskrétními obdobami kritérií popsaných pro spojitý případ, v nichž vystupuje pouze daný konečný počet hodnot funkce f. Spojité p-normy používané ve spojitém případě tak přecházejí v *diskrétní p-normy* dané jako

$$\|F\|_p^m = \left(\sum_{i=0}^{m} |F(x_i)|^p \right)^{1/p}, \quad p \geq 1.$$

Z těchto norem se v praxi opět velmi často používají především dvě následující diskrétní normy pro $p = 2$ a $p = \infty$. První z nich, nazývaná obvykle opět *L_2-norma*, je tedy

$$\|F\|_2^m = \left(\sum_{i=0}^{m} |F(x)|^2 \right)^{1/2}.$$

Další hojně používanou normou je *diskrétní nekonečno-norma*, které se také říká *diskrétní Čebyševova norma* a která je vlastně limitou diskrétních p-norem při $p \to \infty$. Je dána vztahem

$$\|F\|_\infty^m = \max_{i=0,1,\ldots,m} |F(x_i)|.$$

Jako měřítko kvality aproximace se tedy v diskrétním případě může užít například veličina

$$\|E\|_\infty^m = \|f - \varphi\|_\infty^m = \max_{i=0,1,\ldots,m} |f(x_i) - \varphi(x_i)|.$$

Nejlepší aproximace ve smyslu této normy se často nazývá *diskrétní Čebyševova aproximace*. V diskrétním případě se Čebyševova norma příliš často nepoužívá. Podle našeho názoru se v diskrétním případě čtenář častěji setká s měřením chyby v L_2-normě, která je zde dána vztahem

$$\|E\|_2^m = \|f - \varphi\|_2^m = \left(\sum_{i=0}^{m} |f(x_i) - \varphi(x_i)|^2 \right)^{1/2}.$$

Tato norma nijak nezdůrazňuje maximální absolutní hodnotu chyby aproximace a dá se o ní opět říci, že vyjadřuje jakousi střední hodnotu chyby. Nejlepší aproximaci ve smyslu této normy se říká také **aproximace metodou nejmenších čtverců**.

Poznamenáváme ještě na závěr, že všechny zde uvedené normy mají opět stejné základní vlastnosti, jako měly vektorové normy uvedené v Kapitole 10. Diskrétní normy jsou totiž vlastně vektorové normy vektorů funkčních hodnot v tabulkových bodech. Přesto, že vektorové normy jsou ve smyslu uvedeném v Kapitole 10 ekvivalentní, nejsou nejlepší aproximace sestrojené v různých normách obecně identické.

Jak ještě uvidíme, ve speciálním případě $n = m$ může být diskrétní norma chyby aproximace nulová, aniž by nutně platila rovnost $\varphi = f$ jako funkcí na celém intervalu $[a, b]$. Nulovost diskrétní p-normy chyby ovšem znamená nulovost všech sčítanců, a v takovém případě tedy platí $\varphi(x_i) = f(x_i), i = 0, 1, \ldots, m$. Interpolace s touto vlastností se pak nazývá **interpolační aproximace**. Poznamenáváme, že o chování interpolační aproximace mezi tabulkovými body však samotná nulovost diskrétní normy chyby nic neříká.

Příklad 11.2 (Nejlepší aproximace: diskrétní případ)**.** *Budeme opět vyšetřovat aproximaci funkce $f(x) = \sqrt{x}$ na intervalu $[0, 1]$ polynomem prvního stupně. Funkce f nechť je nyní zadána tabulkou.*

Máme-li funkci f zadánu tabulkou hodnot v bodech $x_0 = 0$ a $x_1 = 1$, takže $f(x_0) = 0, f(x_1) = 1$, má nejlepší diskrétní L_2-aproximace φ_L^* tvar $\varphi_L^*(x) = x$.

Že je φ_L^* nejlepší aproximace, je ihned vidět z toho, že pro její chybu E platí

$$\|E\|_2^1 = \|f - \varphi_L^*|_2^1 = \left(\sum_{i=0}^{1} |f(x_i) - \varphi_L^*(x_i)|^2 \right)^{1/2} = 0.$$

Jde tedy o interpolační aproximaci.

Bude-li funkce f zadána tabulkou hodnot ve třech bodech $x_0 = 0, x_1 = \frac{1}{2}, x_2 = 1$, bude mít nejlepší diskrétní L_2-aproximace φ_L^* tvar $\varphi_L^*(x) = x + \frac{1}{6}(\sqrt{2} - 1)$. Diskrétní norma chyby této aproximace již bude nenulová (kladná). Aproximace není interpolační.

11.2 Aproximace interpolačním polynomem

Jako aproximace funkcí se v numerické matematice velmi často používají polynomy. Důvodů pro používání polynomiálních aproximací je celá řada [1]. Patří sem především to, že polynomy se dají plně popsat konečným počtem údajů (stupeň, koeficienty) a že se jejich hodnoty dají bez problémů počítat konečným počtem aritmetických operací. Dalším důvodem pro užívání polynomů je to, že se s nimi snadno matematicky pracuje (derivování, integrování). Navíc,

jak jsme již uvedli v Přednášce 10 o numerické integraci, platí pro polynomy Weierstrassova věta, která v podstatě říká, že pomocí vhodně vybraných polynomů můžeme spojitou funkci aproximovat s libovolně vysokou přesností. Upozorňujeme ještě na to, že mnohé výsledky, které uvádíme v této kapitole pro interpolační aproximace pomocí polynomů, si zachovávají platnost i pro jiné třídy interpolačních aproximací lineárního typu (např. trigonometrické polynomy).

Obecně o interpolační aproximaci

Představme si, že máme dánu následující tabulku dat:

t	1,0	2,0	3,0	4,0	5,0	6,0
y	1,9	2,7	4,8	5,3	7,1	9,4

Taková data mohou být výsledkem laboratorních měření, přičemž t by mohlo představovat čas nebo teplotu a y by mohlo představovat vzdálenost nebo tlak. Nebo by uvedená data mohla představovat měření nějakého přírodního jevu, jako je třeba populace nějakého ohroženého druhu či křivku záření supernovy v čase. Nebo by data mohla představovat ceny akcií v různých dnech nebo úhrny prodejů za určitá období. A data by mohla také představovat hodnoty nějaké matematické funkce pro různé argumenty.

U takových dat existuje zde řada věcí, které bychom s nimi mohli chtít dělat. Mohli bychom chtít vynést data do grafu a nakreslit hladkou křivku procházející tabulkovými body. Mohli bychom chtít nějak odhadnou hodnoty dat mezi tabulkovými body nebo předpovědět hodnoty pro t ležící vně tabulky. Pokud data reprezentují nějaký fyzikální jev, třeba populaci, mohli bychom chtít stanovit některé důležité parametry, třeba tedy růst počtu narozených a zemřelých jedinců. Pokud tabulková data reprezentují hodnoty jisté matematické funkce, mohli bychom chtít najít přibližné hodnoty derivace nebo integrálu, popřípadě rychle vypočítat hodnotu funkce pro daný argument.

V této přednášce se budeme zabývat tím, jak taková diskrétní data vyjádřit pomocí poměrně jednoduchých funkcí, s nimiž se pak dá snadno manipulovat. Budeme se přitom nejprve snažit o to, aby tyto jednoduché funkce nejen aproximovaly celkový trend tabulkových dat, ale aby přímo procházely tabulkovými body, což znamená, že aproximující funkce musí nabývat hodnot daných tabulkou přesně. Znamená to, že diskrétní normy chyby aproximace brané přes tabulkové body budou rovny nule a budeme tedy hledat nejlepší aproximace ve smyslu těchto norem. Takovým funkcím se říká **interpolanty**. S interpolantami jsme se vlastně již v minulých přednáškách setkali. Tak třeba v metodě sečen pro řešení nelineárních rovnic jsme prokládali přímku dvěma funkčními hodnotami. Nebo, i když jsme to podrobně nepopisovali, je Simpsonovo pravidlo pro numerický výpočet integrálu výsledkem toho, že třemi

hodnotami integrované funkce proložíme kvadratický polynom. Pokusíme se nyní podat zde obecnější a systematický pohled na problematiku interpolace.

Uvidíme, že interpolační polynom stupně nejvýše n se dá k tabulkovým bodům jednoznačně sestrojit tehdy a jen tehdy, pokud je těchto bodů právě $n+1$. Budeme tedy předpokládat, že funkce f, kterou chceme aproximovat, je dána pouze svými hodnotami v $n+1$ bodech $x_0, x_1, \ldots, x_n$ a že chceme najít aproximaci φ takovou, která bude co nejlépe napodobovat chování funkce f v okolí všech těchto bodů. O tabulkových bodech $x_i, i = 0, 1, \ldots, n$, v nichž jsou zadány (tabelovány) hodnoty $f(x_i)$, zde budeme předpokládat, že jsou navzájem *různé*, a budeme jim říkat **uzly interpolace** nebo **interpolační uzly**. O **interpolaci** tedy hovoříme tehdy, jestliže se aproximující funkci φ snažíme volit tak, aby platilo

$$\varphi(x_i) = f(x_i), \quad i = 0, 1, \ldots, n,$$

a aby tedy přesně nabývala zadaných funkčních hodnot. Uvedeným podmínkám se říká **interpolační podmínky**.

Pokud se nechceme omezit pouze na interpolační polynomy, můžeme úlohu o interpolaci chápat i obecněji. Postupujeme tak, že nejprve stanovíme — s přihlédnutím k předpokládaným vlastnostem tabelované funkce f — vhodnou třídu aproximujících funkcí, v níž budeme aproximaci φ hledat. Prakticky to znamená, že vybereme vhodnou soustavu bázových funkcí. Pro polynomiální interpolaci jsou často používány jako bázové funkce **monomy**, tedy funkce $1, x, x^2, \ldots, x^n$, i když to není jediná možná volba. Konkrétní aproximující funkci z vybrané třídy, tedy vlastně koeficienty v jejím vyjádření jako lineární kombinace bázových funkcí, pak vybereme na základě vstupních dat naší úlohy, tedy pomocí interpolačních podmínek. Protože vstupní data naší úlohu jsou dána konečnou tabulkou a výstupní data jsou dána konečným počtem koeficientů $c_0, c_1, \ldots, c_n$, je úloha o interpolaci numerickou úlohou. Uvidíme, že interpolační aproximace se při pevně daných bázových funkcích dá najít v konečném počtu kroků.

Jak jsme již poznamenali, interpolační aproximace φ minimalizuje v uvažované třídě aproximujících funkcí generované n bázovými funkcemi diskrétní normy $\|E\|_p^m$ s $m = n$ pro libovolné $p \geq 1$, neboť tyto diskrétní normy jsou u interpolačních aproximací rovny nule. K otázce existence a jednoznačnosti interpolačních aproximací se za okamžik vrátíme, nyní zde ale učiníme ještě několik poznámek.

Tak především jsme v Příkladu 11.2 vlastně pro $n = 1, x_0 = 0, x_1 = 1$ uvedli interpolační aproximaci funkce $f(x) = \sqrt{x}$. Skutečně, nejlepší diskrétní L_2-aproximace $\varphi_L^*(x) = x$ z tohoto příkladu splňuje interpolační podmínky $x_i = \sqrt{x_i}, i = 0, 1$, a je tedy současně interpolační aproximací ze třídy polynomů nejvýše prvního stupně.

Ačkoliv je interpolace mocným nástrojem výpočtové matematiky, není požadavek přesného splnění interpolačních podmínek vždy oprávněný. Tak například, pokud jsou vstupní data zatížena chybami měření nebo jiným významným zdrojem chyb, je obvykle namístě dát přednost vyhlazení takového možného šumu postupem, který se používá v metodě nejmenších čtverců, tedy aproximovat $n+1$ tabulových dat aproximující funkcí, která má méně než $n+1$ volných parametrů c_i (tedy například polynomem stupně $m < n$). Chyba nejlepší aproximace je zde pak obecně nenulová, kladná. Příklad takové nejlepší L_2-aproximace funkce $\sqrt{x}$ jsme uvedli opět v Příkladu 11.2 pro $m = 2$ a $n = 1$.

Podobně se interpolační aproximace obecně nepoužívají v softwarových knihovnách pro výpočet elementárních a speciálních funkcí, které jsou zahrnuty do většiny programovacích jazyků (včetně Matlabu). V takové situaci je totiž důležité, aby aproximující funkce byla blízká aproximované funkci ve všech bodech intervalu $[a, b]$, a není přitom podstatné, aby se této funkci v několika vybraných bodech přímo přesně rovnala. Používá se zde tedy spíše spojitá Čebyševova aproximace.

Je dobré si uvědomit, že ve většině interpolačních problémů je jistá libovůle, protože k daném souboru tabulkových dat může existovat mnoho interpolant. I když vyžadujeme splnění interpolačních podmínek, zůstává zde ještě řada otevřených otázek:

- Jak má vůbec interpolanta vypadat, tj. jakou třídu aproximujících funkcí máme použít? Tady nám leckdy mohou pomoci vhodné matematické nebo fyzikální úvahy.
- Jak by se měla interpolanta chovat mezi tabulkovými body?
- Měla by interpolanta zachovávat některé vlastnosti dat, třeba jejich monotónii, konvexnost nebo periodicitu?
- Zajímají nás primárně hodnoty koeficientů c_i, které při zvolených bázových funkcích definují interpolační aproximaci, nebo nám jde spíše o výhodný výpočet hodnot interpolanty pro její vykreslování či podobné účely?
- Pokud se tabulková data a průběh interpolanty vynesou do grafu, má být tento graf vizuálně uspokojivý?

Volba třídy aproximujících funkcí tak při interpolaci závisí nejen na datech, která chceme popsat, ale i na odpovědích na takové otázky. Výběr interpolační aproximace je obvykle založen na tom,

- jak snadno se s interpolantou pracuje (jde tu o určení koeficientů z daných tabulkových dat, výpočet hodnot mimo tabulkové body, derivování nebo integrace interpolanty atd.),

- jak dobře souhlasí vlastnosti interpolanty s vlastnostmi tabulkových dat (hladkost, monotónie, konvexita, periodicita atd.).

Některé běžné třídy aproximujících funkcí, které se k interpolaci využívají, jsou

- polynomy,
- funkce, které jsou po částech polynomy (splajny),
- trigonometrické funkce,
- exponenciální funkce,
- racionální funkce.

Jak jsme již uvedli, v této přednášce se kromě určitých úvah na obecné úrovni soustředíme výhradně na interpolaci jednorozměrných dat prostřednictvím polynomů. Jiné možnosti jsou k nalezení v literatuře [2, 3, 4, 1].

Existence, jednoznačnost a podmíněnost

Otázka existence a jednoznačnosti interpolační aproximace se, jak uvidíme, vlastně redukuje na to, zda počet volných parametrů v obecné funkci z dané uvažované třídy aproximací souhlasí s počtem uzlů interpolace. Není-li tomu tak, pak buď interpolanta obecně neexistuje (parametrů méně než uzlů) nebo není určena jednoznačně (parametrů více než uzlů). Omezíme se proto v dalším na situaci, kdy počet uzlů i volných parametrů je stejný, tedy $n+1$.

Jak jsme uvedli již v úvodu, při daném souboru tabulkových dat $\{(x_i, f(x_i)), i = 0, 1, \ldots, n\}$ (v souladu s předchozím bereme nyní $m = n$) vybíráme interpolantu z třídy funkcí, které jsou lineárními kombinacemi daných **bázových funkcí** $\varphi_0, \varphi_1, \ldots, \varphi_n$. Hledaná interpolační aproximace φ tedy bude mít tvar

$$f(x) = \sum_{i=0}^{n} c_i \varphi_i(x),$$

kde c_i jsou volné parametry, které je třeba určit tak, aby byly splněny interpolační podmínky. Pro daná tabulková data tedy požadujeme, aby platilo

$$\varphi(x_i) = \sum_{i=0}^{n} c_i \varphi_i(x_j) = f(x_j) = y_j, \quad j = 0, 1, \ldots, n,$$

což je ale soustava $n+1$ lineárních algebraických rovnic pro $n+1$ neznámých $c_i, i = 0, 1, \ldots, n$, která se dá v maticovém tvaru zapsat jako

$$\mathbf{A}\mathbf{c} = \mathbf{y}, \tag{11.2}$$

kde prvky čtvercové matice $\mathbf{A}$ řádu $n+1$ jsou dány jako $a_{ij} = \varphi_{j-1}(x_{i-1})$ (tj. a_{ij} je hodnota $j-1$-té bázové funkce v $i-1$-tém uzlu), složky vektoru pravých stran $\mathbf{y}$ jsou známá data $y_j = f(x_{j-1})$, a složky vektoru neznámých $\mathbf{c}$ jsou hledané koeficienty c_i.

Existence a jednoznačnost řešení dané soustavy nyní závisí pouze na tom, zda je matice soustavy $\mathbf{A}$ regulární. V takovém případě zřejmě interpolační aproximace existuje a je jediná, jakkoli její konkrétní tvar závisí na tom, jak jsme konkrétně zvolili bázové funkce. A podobně, citlivost řešení c na poruchy v datech závisí na podmíněnosti matice $\mathbf{A}$. Uvažujeme-li nějakou předem danou třídu funkcí (třeba polynomy stupně nejvýše n), pak pro ní může existovat i řada různých systémů bázových funkcí. Konkrétní volba bázových funkcí pak ovlivňuje nejen podmíněnost soustavy $\mathbf{Ac} = \mathbf{y}$, ale také objem práce potřebné k jejímu vyřešení a snadnost výpočtu hodnot interpolanty či jiných manipulací s ní. Nyní již se budeme zabývat pouze konkrétně polynomiální interpolací. Jakkoli se zde přitom jsme nuceni omezit pouze na dva případy systému bázových funkcí, upozorňujeme, že existují a používají se i jiné přístupy, s nimž se lze seznámit v literatuře [2, 3, 4, 1].

Monomy jako interpolační bázové funkce

Protože interpolační podmínky představují $n+1$ podmínek na aproximující funkci, můžeme očekávat, že jednou z vhodných tříd aproximujících funkcí bude třída polynomů stupně nejvýše n, které mají $n+1$ koeficientů. Nejpřirozenější systém bázových funkcí pro tuto třídu tvoří prvních $n+1$ monomů, tedy funkcí

$$\varphi_i(x) = x^i, \quad i = 0, 1, \ldots, n.$$

Každá lineární kombinace těchto bázových funkcí je polynom

$$p_n(x) = c_0 + c_1 x + \cdots + c_n x^n$$

stupně nejvýše n. Zapíšeme-li nyní interpolační podmínky v maticovém tvaru tak, jak jsme to udělali v minulém odstavci, zjistíme, že vektor $\mathbf{c}$ koeficientů polynomu, který interpoluje tabulková data $\{(x_i, f(x_i)), i = 0, 1, \ldots, n\}$, je řešením soustavy $n+1$ rovnic o $n+1$ neznámých

$$\mathbf{Ac} = \begin{bmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} = \mathbf{y}.$$

S maticí tohoto tvaru jsme se již setkali v Odstavci 1.4 Přednášky 10, kde jsme se zabývali numerickými metodami pro přibližný výpočet integrálů; nazývá se **Vandermondova matice** a dá se o ní celkem snadno ukázat, že pokud jsou uzly kvadratury navzájem různé, je regulární. Pokud by tomu totiž tak

nebylo, měla by homogenní soustava rovnic $\mathbf{Ac} = \mathbf{o}$ netriviální řešení a tedy by existoval nenulový polynom stupně nejvýše n, který by měl $n+1$ kořenů. To ale není podle toho, co víme z algebry, možné.

Z regularity Vandermondovy matice pak plyne, že pro každá námi popsaná tabulková data existuje právě jeden interpolační polynom stupně nejvýše n. Čtenáře, který již třeba netrpělivě listoval některou učebnicí numerické matematiky, by tady mohlo zmást, že se setkal s interpolačními polynomy různých názvů (třeba Newtonův nebo Lagrangeův interpolační polynom). Nejde přitom o nic jiného než o různé způsoby zápisu téhož jediného interpolačního polynomu, jehož koeficienty jsou řešením výše uvedené soustavy s Vandermondovou maticí. Různé zápisy se mohou ukázat užitečnými v různých praktických situacích.

Všimněme si na tomto místě ještě dvou věcí, o kterých jsme se možná mohli zmínit již dříve. Především mohou být uzly interpolace (tabulkové body) rozloženy zcela libovolně, nemusí být ekvidistantní. Jediný požadavek je zde, aby byly vzájemně různé. Pro jistotu také vysvětlíme, proč zde všude hovoříme o polynomech stupně *nejvýše* n a ne přímo o polynomech stupně n. Je to proto, že některé z koeficientů c_i interpolačního polynomu nám mohou vyjít nulové a interpolační polynom pro našich $n+1$ dat může mít stupeň *menší* než n. Stačí si představit třeba 10 tabulkových dat ležících na přímce. Ta budou interpolována vždy polynomem prvního stupně (lineárním polynomem), ať už interpolační polynom hledáme třeba ve třídě polynomů stupně 9.

Příklad 11.3 (Interpolace polynomem)**.** *Jako ukázku sestrojení interpolačního polynomu pomocí monomiální báze odvodíme interpolační polynom druhého stupně, který bude interpolovat data* $(-2,-27),(0,-1),(1,0)$*. Víme již z předchozího, že existuje právě jeden kvadratický polynom*

$$p_2(x) = c_0 + c_1 x + c_2 x^2,$$

který interpoluje tři body $(x_0,y_0),(x_1,y_1),(x_2,y_2)$*. Použijeme-li monomiální bázi, pak jsou koeficienty tohoto polynomu řešením soustavy rovnic*

$$\mathbf{Ac} = \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \mathbf{y}.$$

Dosadíme-li sem naše konkrétní data, dostaneme soustavu

$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}.$$

Vyřešíme-li tuto soustavu (například Gaussovou eliminační metodou), dostaneme jako její řešení vektor $\mathbf{c} = [-1 \quad 5 \quad -4]^T$*, takže hledaný interpolační*

polynom je

$$p_2(x) = -1 + 5x - 4x^2.$$

Řešení soustavy $\mathbf{Ac} = \mathbf{y}$ pomocí některého standardního řešiče (v Matlabu třeba `c = A\y`) vyžaduje provést řádově n^3 aritmetických operací. Vzniká tak otázka, zda při vhodné volbě bázových funkcí bychom nemohli dospět k soustavě s nějakou maticí speciálního tvaru. Ukazuje se, že je to možné a uvidíme to například v příštím odstavci. Při použití monomiální báze se navíc ukazuje, že Vandermondovy matice jsou pro větší n často špatně podmíněné. Závisí to jistě také na rozložení uzlů interpolace, ale ve většině případů se ukazuje, že s rostoucím počtem tabulkových bodů číslo podmíněnosti Vandermondovy matice vzrůstá přinejmenším exponenciálním způsobem. Jakkoli tedy teoreticky je Vandermondova matice regulární, stává se při rostoucím n stále hůře podmíněnou a pro dostatečně velká n se na počítači v jeho konečné aritmetice může jevit jako singulární.

Poznamenáváme ale, že přes eventuální špatnou podmíněnost a z ní plynoucí nepřesné stanovení koeficientů interpolačního polynomu bude i tento polynom dobře splňovat interpolační podmínky. Plyne to z toho, že — jak jsme již uvedli u Gaussovy eliminace — Gaussova eliminační metoda s částečným výběrem hlavních prvků dává při řešení soustav lineárních rovnic vždy malá rezidua, nezávisle na podmíněnosti soustavy. Přitom ovšem hodnoty koeficientů polynomu samotné mohou být vypočítány s velkou chybou.

Na závěr v tomto odstavci ještě připomeneme jeden užitečný způsob výpočtu hodnot polynomu, který je vyjádřen pomocí monomiální báze, tedy polynomu tvaru

$$p_n(x) = c_0 + c_1 x + \cdots + c_n x^n.$$

Celý postup, kterému se říká **Hornerovo schéma** nebo někdy také **syntetické dělení**, spočívá vlastně pouze ve vhodném uzávorkování. Zapíšeme

$$p_n(x) = c_0 + x(c_1 + x(c_2 + x(\cdots(c_{n-1} + xc_n)\cdots)))$$

a vidíme, že výpočet jedné hodnoty polynomu tímto způsobem vyžaduje pouze n sčítání a n násobení. Podobný princip se používá i při sestavování Vandermondovy matice. Místo aby se explicitně počítaly mocniny uzlů, pracujeme se vztahem

$$a_{ij} = \varphi_{j-1}(x_{i-1}) = x_{i-1}^{j-1} = x_{i-1}\varphi_{j-2}(x_{i-1}) = x_{i-1}a_{i,j-1}, \quad j = 2, \ldots, n.$$

V Matlabu je pro výpočet hodnot polynomů k dispozici funkce `polyval`. Koeficienty interpolačního polynomu v monomiální bázi lze stanovit pomocí funkce `polyfit` (ta umožňuje pracovat i s metodou nejmenších čtverců, tedy prokládat daty polynomy, které jsou nižšího stupně než odpovídá počtu tabulkových dat) nebo pomocí funkce `interp1` (ta umožňuje provádět interpolaci i s jinými bázovými funkcemi než jsou polynomy).

Lagrangeův interpolační polynom

Jak jsme viděli již dříve, matice soustavy interpolačních podmínek (11.2) je dána hodnotami bázových funkcí v uzlech interpolace. Vzniká tedy otázka, jak vhodně volit polynomiální bázové funkce tak, aby matice $\mathbf{A}$ soustavy (11.2) měla jednoduchý tvar a tato soustava se snadno řešila. Nejjednodušší matici dostaneme postupem, který pochází od klasického francouzského matematika Lagrange. Tato matice $\mathbf{A}$ bude *jednotková*, nebude tedy co řešit a koeficienty u Lagrangeových bázových funkcí budou přímo funkční hodnoty.

Aby matice $\mathbf{A}$ vyšla jednotková, potřebujeme tedy zvolit bázové funkce $\varphi_i, i = 0, 1, \ldots, n$, tak, aby platilo $\varphi_i(x_i) = 1$ a $\varphi_i(x_k) = 0$ pro $i \neq k$. Snadno se ověří, že těmto podmínkám budou vyhovovat bázové funkce tvaru

$$\varphi_i(x) = \ell_i(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)},$$

kterým se říká **Lagrangeovy bázové polynomy** nebo někdy také **fundamentální polynomy**. Interpolační polynom zapsaný v těchto bázových funkcích se nazývá **Lagrangeův interpolační polynom** a podle toho, co jsme výše řekli o matici $\mathbf{A}$, má tvar

$$L_n(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x).$$

Je to tedy opět polynom stupně nejvýše n a vzhledem k jednoznačnosti interpolačního polynomu je pouze jiným zápisem polynomu, který bychom dostali s monomiální bází ve tvaru, na který jsme běžně zvyklí.

Použijeme-li Lagrangeovy bázové polynomy, je tedy snadné pro daná tabulková data data zapsat interpolační polynom a koeficienty v jeho vyjádření jsou naprosto dobře podmíněné. Ale hodnoty interpolačního polynomu v Lagrangeově tvaru je pracnější počítat než je tomu u polynomu v monomiální bázi, kde se dá použít Hornerovo schéma. Lagrangeovy polynomy se také hůře derivují, integrují atd. Z těchto důvodů se z Lagrangeových interpolačních polynomů používají spíše polynomy nižších stupňů.

Příklad 11.4 (Interpolace Lagrangeovým polynomem)**.** *Nechť je funkce f dána svými hodnotami ve čtyřech bodech ($n = 3$) $x_0 = 0, x_1 = 1, x_2 = -1, x_3 = 3$ a nechť funkční hodnoty v těchto bodech jsou $f(0) = 1, f(1) = 2, f(-1) = 2, f(3) = 0$. Vypočítáme přibližnou hodnotu $f(2)$ pomocí Lagrangeova interpolačního polynomu L_3, který při těchto datech bude třetího stupně.*

Máme nyní

$$\ell_0(x) = \frac{(x - 1)(x + 1)(x - 3)}{(0 - 1)(0 + 1)(0 - 3)} = \tfrac{1}{3}(x - 1)(x + 1)(x - 3)$$

a dále podobně obdržíme

$$\begin{aligned}\ell_1(x) &= -\tfrac{1}{4}x(x+1)(x-3),\\ \ell_2(x) &= -\tfrac{1}{8}x(x-1)(x-3),\\ \ell_3(x) &= \tfrac{1}{24}x(x-1)(x+1).\end{aligned}$$

Je $\ell_0(2) = -1, \ell_1(2) = \frac{3}{2}, \ell_2(2) = \frac{1}{4}, \ell_3(2) = \frac{1}{4}$, *a tedy*

$$L_3(2) = 1\cdot(-1) + 2\cdot\tfrac{3}{2} + 2\cdot\tfrac{1}{4} + 0\cdot\tfrac{1}{4} = \tfrac{5}{2} \approx f(2).$$

Samotný polynom L_3 *má tvar*

$$\begin{aligned}L_3(x) &= 1\cdot\ell_0(x) + 2\cdot\ell_1(x) + 2\cdot\ell_2(x) + 0\cdot\ell_3(x) =\\ &= \tfrac{1}{3}(x-1)(x+1)(x-3) - \tfrac{1}{2}x(x+1)(x-3) - \tfrac{1}{4}x(x-1)(x-3) =\\ &= -\tfrac{5}{12}x^3 + x^2 + \tfrac{5}{12}x + 1,\end{aligned}$$

kde poslední řádek jsme získali roznásobením a je identický s tvarem polynomu v monomiální bázi.

Lineární a kvadratická interpolace

Interpolujeme-li funkci f na základě jejích hodnot ve dvou bodech, aproximujeme ji polynomem prvního stupně ($n = 1$) a hovoříme o *lineární interpolaci*. Interpolační polynom prvního stupně L_1 má pak tvar

$$L_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1). \tag{11.3}$$

Právě lineární interpolace se občas používá při práci s tabulkami funkcí ve střední škole.

Je-li $n = 2$, hovoříme o *kvadratické interpolaci*. Interpolační polynom L_2 druhého stupně má tvar

$$L_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2). \tag{11.4}$$

Příklad 11.5 (Lineární interpolace)**.** *Máme dánu tabulku hodnot funkce* $f(x) = \sin x$ *na 5 desetinných míst:*

x	$20°$	$21°$	$22°$
$\sin x$	*0,34202*	*0,35837*	*0,37461*

Vypočítáme přibližnou hodnotu $\sin 20°18'$, *tj.* $\sin 20{,}3°$, *lineární interpolací z funkčních hodnot v bodech* $x_0 = 20°$ *a* $x_1 = 21°$. *Dostaneme*

$$\sin 20°18' \approx \frac{20{,}3 - 21}{20 - 21}\cdot 0{,}34202 + \frac{20{,}3 - 20}{21 - 20}\cdot 0{,}35837 \approx 0{,}34693.$$

Přesná hodnota $\sin 20°18'$ *zaokrouhlená na 5 desetinných míst je 0,34694. V tomto případě je tedy lineární interpolace zcela postačující, bylo by zbytečně pracné volit* $n > 1$.

Chyba aproximace interpolačním polynomem

Z toho, že při interpolaci pro chybu aproximace $E = f - p_n$ polynomem stupně n platí $E(x_i) = 0, i = 0, 1, \ldots, n$ (protože interpolační polynom splňuje interpolační podmínky), nemůžeme ještě dělat přímé závěry o hodnotě chyby aproximace mimo tabulkové body. Popravdě řečeno samotná tabulka o chování aproximované funkce mimo tabulkové body nic neříká a pro $x \neq x_i$ se aproximovaná funkce může chovat zcela libovolně. Abychom tedy mohli o chování chyby aproximace mimo tabulkové body vůbec něco říci, musíme mít k dispozici více informací o funkci, která má být tabulkou reprezentována. Příkladem takových informací je například informace o tom, že aproximovaná funkce f je na intervalu $[a, b]$ dostatečně „hladká“, tj. má tam určitý počet spojitých derivací.

V dalším výkladu budeme symbolem $\mathrm{int}(x_0, x_1, \ldots, x_n, x)$ označovat nejmenší otevřený interval takový, že uvedené body $x_0, x_1, \ldots, x_n, x$ leží uvnitř tohoto intervalu nebo na jeho hranici. V učebnicích numerické matematiky se pak dokazuje o chybě aproximace interpolačním polynomem následující tvrzení, zde uváděné bez důkazu.

Věta 11.2 (Chyba interpolační aproximace)**.** *Nechť $[a, b]$ je libovolný interval, který obsahuje všech $n + 1$ interpolačních uzlů $x_0, x_1, \ldots, x_n$. Nechť $f, f', \ldots, f^{(n)}$ existují a jsou spojité na $[a, b]$ a nechť pro všechna $x \in (a, b)$ existuje derivace $f^{(n+1)}(x)$.*

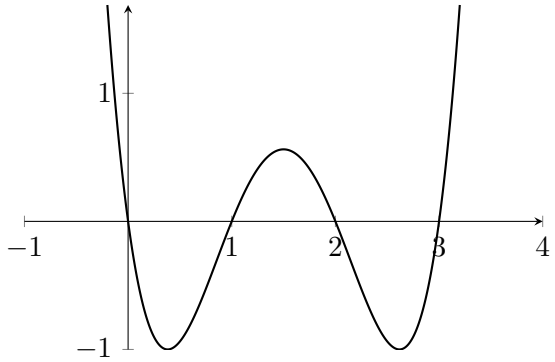
Pak pro chybu E aproximace funkce f interpolačním polynomem p_n platí na intervalu $[a, b]$ vzorec

$$E(x) = f(x) - p_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n) \frac{f^{(n+1)}(\xi)}{(n+1)!}, \qquad (11.5)$$

kde ξ je nějaké (blíže neurčené) číslo z intervalu $\mathrm{int}(x_0, x_1, \ldots, x_n, x)$, které tedy závisí na konkrétní hodnotě x.

Vidíme, že pokud se $f^{(n+1)}(x)$ na intervalu (a, b) příliš nemění, je pro průběh chyby aproximace rozhodující průběh polynomu $\omega_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$, Tento polynom nezávisí na interpolované funkci, ale pouze na bodech x_n. Vhodnou volbou uzlů $x_i, i = 0, 1, \ldots, n$, tedy můžeme chybu aproximace zmenšit. Bližší podrobnosti v tomto směru lze najít v literatuře [2], [3], [4], [1]. Zde pouze poznamenáváme, že optimální volbou uzlů interpolace jsou kořeny Čebyševových ortogonálních polynomů (viz [1]) a ne tedy rovnoměrně rozložené uzly.

Příklad 11.6. *Pro $n = 3$ a $x_i = i, i = 0, 1, 2, 3$, je průběh polynomu ω_3 znázorněn na obr. 11.2. Z obrázku vidíme, že se dá čekat, že vně intervalu $\mathrm{int}(x_0, x_1, \ldots, x_n)$ chyba aproximace interpolačním polynomem prudce poroste.*



Obrázek 11.2: Průběh polynomu ω_3 pro uzly $x_i = i$.

Odhad chyby aproximace

Vzorec (11.5) pro chybu aproximace se pro přímé praktické použití příliš nehodí, neznáme totiž předem hodnotu argumentu ξ, který ve vzorci vystupuje. Ze vzorce je ovšem vidět, že chyba aproximace je v interpolačních uzlech nulová, neboť to jsou kořeny polynomu $\omega_n(x)$. Dále ze vzorce také vidíme, že je-li aproximovaná funkce f sama polynomem nejvýše stupně n, aproximuje ji interpolační polynom $p_n(x)$ stupně n s nulovou chybou, přesně, a je tedy přímo $p_n(x) = f(x)$. To plyne ze skutečnosti, že v takovém případě je derivace funkce f řádu $n+1$ (a všechny vyšší derivace) identická nula pro všechny hodnoty argumentu.

Ze vzorce (11.5) ale plyne prakticky použitelný odhad chyby, alespoň v případech, kdy dokážeme hodnotu v něm vystupující derivace na intervalu $[a, b]$ nějak odhadnout. Pokud se nám totiž podaří určit konstantu M takovou, že pro všechna $x \in [a, b]$ je $|f^{(n+1)}(x)| \leq M$, pak je

$$|E(x)| \leq \frac{M}{(n+1)!}|\omega_n(x)| \leq \frac{M}{(n+1)!} \max_{x\in[a,b]} |\omega_n(x)$$

pro všechna $x \in [a, b]$.

Příklad 11.7 (Odhad chyby)**.** *Odhadneme chybu aproximace, které jsme se dopustili při lineární interpolaci v příkl. 11.5.*

Je $n = 1$ a $f''(x) = -\sin x$. Takto počítaná derivace se ale odvozuje pro x brané v obloukové míře, kde $360° = 2\pi$, takže musíme celý odhad provést na intervalu $[\pi/9, 7\pi/60]$, který v obloukové míře odpovídá intervalu $[20°, 21°]$. Z tabulky v příkl. 11.5 a na základě toho, co víme o průběhu funkce $\sin x$, vidíme, že na tomto intervalu máme na 5 platných číslic $|f''(x)| = |\sin x| \leq 0.35837$, a tedy (do $\omega_1(x)$ dosazujeme opět v obloukové míře!)

$$|E(20°18'')| \leq \frac{1}{2} \cdot 0.35837 \cdot \left(\frac{18}{60} \cdot \frac{\pi}{180}\right) \cdot \left(\frac{42}{60} \cdot \frac{\pi}{180}\right) \approx 1.1 \times 10^{-5}.$$

Skutečná chyba aproximace je (viz údaje v příkl. 11.5) přibližně rovna 1×10^{-5}. *Všimněme si ještě toho, že pokud bychom užili k odhadnutí druhé derivace pouze známý vztah* $|\sin x| \leq 1$, *dostali bychom odhad chyby aproximace zbytečně pesimistický.*

Extrapolace

Používáme-li interpolační polynom k výpočtu přibližných hodnot interpolované funkce vně intervalu $\text{int}(x_0, x_1, \ldots, x_n)$, říkáme, že provádíme *extrapolaci*. Jak je víceméně vidět z obr. 11.2, chyba aproximace může být ve větších vzdálenostech od koncových bodů interpolačního intervalu velká. Pro hodnoty x blízké koncovým bodům intervalu $\text{int}(x_0, x_1, \ldots, x_n)$ lze však přesto dostat dobré výsledky.

Příklad 11.8 (Lineární extrapolace)**.** *Lineární interpolace z hodnot* $\sin 20°$ *a* $\sin 21°$ *uvedených v tabulce v příkl. 11.5 dává podle vzorce (11.3) pro* $\sin 21°18'$ *přibližnou hodnotu* 0.36328 *s chybou asi* 3×10^{-5}, *což je zcela přijatelný výsledek. Lineární interpolace s uzly* $x_0 = 21°$ *a* $x_1 = 22°$ *by nám dala jako výsledek* $\sin 21°18' \approx 0.36324$ *s chybou přibližně* 1×10^{-5}.

11.3 Dodatky

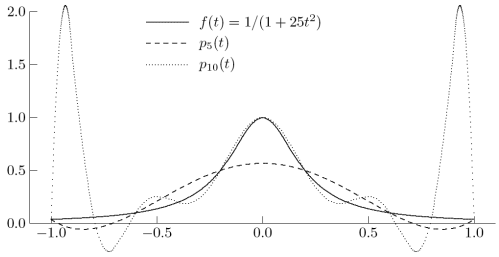
Konvergence interpolačního procesu

Podobně jak jsme se o tom už zmiňovali v Přednášce 10 u numerických kvadratur, má smysl i zde zabývat se otázkou, zda při interpolaci funkce f na intervalu $[a, b]$ s uzly $x_0, x_1, \ldots, x_n$ povede zvyšování počtu uzlů (a tedy aproximace interpolačními polynomy stále vyšších stupňů) ke stále lepší aproximaci funkce f. Odpověď na tuto otázku je v jistém smyslu negativní. Ať už totiž volíme uzly interpolace jakkoli, vždy se dá najít spojitá funkce taková, že námi generovaná posloupnost interpolačních polynomů k ní nebude konvergovat *stejnoměrně* na celém $[a, b]$.

Volíme-li uzly interpolace při tomto procesu ekvidistantně, dělají potíže už poměrně jednoduché spojité funkce jako $\sqrt{x}$ na $[0, 1]$ nebo $1/(1 + x^2)$ na intervalu $[-1, 1]$. Jako příklad problematické interpolace s ekvidistantními uzly uvádíme na obr. 11.3 interpolační polynomy stupně 5 a 10 pro Rungovu funkci $f(t) = 1/(1 + 25t^2)$ na intervalu $[-1, 1]$. Vidíme, že interpolace polynomem vyššího stupně zlepšuje kvalitu aproximace kolem středu intervalu, ale výrazně ji zhoršuje u krajů intervalu.

Volba uzlů interpolace

Jak jsme právě viděli, může interpolace spojitých funkcí při použití ekvidistantních uzlů dávat problematické výsledky. Není tomu tak naštěstí vždy, v



Obrázek 11.3: Interpolace Rungovy funkce - ekvidistantní uzly.

teorii interpolace se ukazuje, že interpolační proces s ekvidistantními uzly konverguje k aproximované funkci f stejnoměrně na intervalu $[a, b]$, je-li f celistvá analytická funkce (jde o pojem z teorie funkcí komplexní proměnné, v podstatě jde o funkci mající všechny derivace, která se dá rozvinout v mocninnou řadu). Příkladem takových funkcí jsou například $\sin x, \cos x$ nebo e^x.

Interpolace ekvidistantních dat polynomy vyšších stupňů je v některých případech také špatně podmíněná úloha, tj. malé nepřesnosti ve vstupních datech mohou působit velké chyby v hodnotách interpolačního polynomu. Jak jsme už viděli na obrázku, potíže vzikají především při aproximování funkce poblíž konců intervalu vytvářeného uzly interpolace, kde se pak navíc interpolační polynomy vyšších stupňů zpravidla značně „vlní“.

Chceme-li aproximovat nějakou funkci polynomem na celém intervalu a můžeme-li si vybrat body, v nichž počítáme nebo měříme funkční hodnoty, je proto vždy rozumnější *nevolit* uzly x_i ekvidistantně. Dobrá strategie je volit tabulkové body tak, aby byly rozloženy obdobně jako *kořeny Čebyševových polynomů*. O těchto ortogonálních polynomech, které v numerické matematice hrají roli i jinde než při interpolaci, se lze poučit v literatuře [2], [3], [4], [1]. Takovou volbou uzlů také minimalizujeme hodnotu $\max |\omega_n(x)|$ v odhadu chyby interpolace.

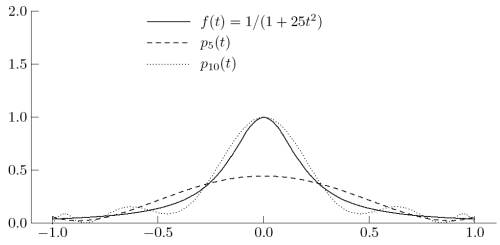
Jako informaci pro čtenáře uvádíme, že pracujeme-li s intervalem $[a, b] = [-1, 1]$, volíme při práci s kořeny Čebyševových polynomů jako uzly

$$x_i = \cos\left(\frac{2i+1}{n+1} \cdot \frac{\pi}{2}\right), \quad i = 0, 1, \ldots, n.$$

Na obecný interval $[a, b]$ tyto uzly zobrazíme jednoduchou lineární transformací

$$z = \tfrac{1}{2}[(b-a)x + (b+a)], \quad x \in [-1, 1], \quad z \in [a, b].$$

Volíme-li uzly interpolace tímto způsobem, má interpolační proces při $n \to \infty$ tu vlastnost, že vzniklé interpolační polynomy konvergují na $[a, b]$ stejnoměrně k aproximované funkci už při velmi mírných požadavcích na tuto funkci. Postačující podmínkou pro stejnoměrnou konvergenci je tu například již pouhá



Obrázek 11.4: Interpolace Rungovy funkce – Čebyševovy uzly.

existence spojité první derivace f' na $[a, b]$. Na závěr tohoto odstavce ještě na obrázku ukážeme, jak vypadají interpolační polynomy stupňů 5 a 10 pro Rungovu funkci při Čebyševově rozložení uzlů interpolace.

Metoda nejmenších čtverců

Je-li naším úkolem aproximovat data

$$\{(x_i, f(x_i)), i = 0, 1, \ldots, m\}$$

a jsou li hodnoty $f(x_i)$ zatíženy chybami (jde například o naměřené veličiny), není namístě požadovat splnění interpolačních podmínek a hledat jako aproximující funkcí polynom stupně $n = m$, který tabulkovými daty přesně prochází. Místo toho spíše volíme jako aproximující funkci polynom p_n nižšího stupně $n < m$ a jeho koeficienty se snažíme volit tak, abychom minimalizovali střední kvadratickou odchylku od tabulkových dat, tedy diskrétní L_2-normu chyby

$$\|E\|_2^m = \|f - p_n\|_2^m = \left(\sum_{i=0}^{m} |f(x_i) - p_n(x_i)|^2 \right)^{1/2}.$$

Tak například můžeme prokládat tabulkou dvaceti naměřených hodnot přímku $p_1(x) = ax + b$. V tomto případě předpokládáme, že charakter měřené závislosti je možno s dostatečnou přesností vystihnout právě polynomem prvního stupně. Obecně se dá říci, že situace, kdy se charakter většího počtu naměřených dat dá dobře vystihnout polynomem poměrně nízkého stupně, jsou v praxi dost běžné. Popsanému přístupu k aproximaci dat zatížených chybami se říká *metoda nejmenších čtverců*.

Aproximace získané metodou nejmenších čtverců mají jisté užitečné statistické vlastnosti a vyrovnávají vliv náhodných chyb v zadaných (naměřených) hodnotách. Aproximace získané metodou nejmenších čtverců jsou tedy také nejlepší diskrétní L_2-aproximace při $n < m$. Čtenáře možná nepřekvapí, že pro $n = m$ dává aproximace metodou nejmenších čtverců interpolační polynom stupně n. Metodě nejmenších čtverců a její teorii i praxi je v numerické

matematice věnována značná pozornost. Bohužel nám tento text nedává možnost se o ní podrobněji rozepsat. Zájemce tak odkazujeme především na [2], [3] a [4]; jiný pohled a nejnovější výsledky lze najít v [1].

V Matlabu umožňuje práci s metodou nejmenších čtverců již zde zmiňovaná funkce `polyfit`, kde jako jeden z parametrů můžeme volit stupeň aproximujícího polynomu. Pokud jde o vhodnou volbu stupně aproximujícího polynomu, doporučují statistické úvahy konstruovat metodou nejmenších čtverců postupně polynomy stupně $n = 0, 1, 2, \ldots$, počítat navíc pro každý polynom p_n hodnotu veličiny

$$\frac{(\|E_2^m\|)^2}{m-n}$$

a pokračovat se zvyšováním stupně tak dlouho, dokud tato veličina s rostoucím n významně klesá.

I v metodě nejmenších čtverců se stejně jako při interpolaci může ukázat, že se naměřené hodnoty pro aproximaci polynomem nehodí. V takovém případě je třeba přejít k jiným systémům základních funkcí nebo hledat aproximaci nelineárního typu. Zde jsme již ale opět nuceni odkázat případného zájemce na literaturu.

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill, Boston, 2 edition, 2002.

[2] Petr Přikryl. *Numerické metody matematické analýzy.* MVŠT. SNTL, Praha, 1985.

[3] Petr Přikryl. *Numerické metody matematické analýzy.* MVŠT. SNTL, Praha, 2., opr. a dopl. edition, 1988.

[4] Petr Přikryl and Marek Brandner. *Numerické metody II.* FAV ZČU, Plzeň, 2001.

KAPITOLA 12

Náhodná čísla a Monte Carlo

Obsah této přednášky čerpá z klasické přehledové publikace [2] a z možná praktičtěji zaměřené monografie [4]. K dispozici je samozřejmě i celá řada dalších publikací, okrajově je téma zahrnuto i v [1]. V českém jazyce se problematice v rozumném rozsahu věnují snad pouze skripta [7].

12.1 Stručná historie Monte Carlo metod

Podstatou metody Monte Carlo je simulace hazardních her, jejichž chování a výsledek může být použit ke studiu některých vědecky zajímavých jevů. Zatímco tato podstata není přímo spojena s počítači, efektivně využit výpočetně simulované hazardní hry jako prostředku seriózní vědecké a technické praxe výrazně je možné až s dostupností moderních výkonných digitálních počítačů. Je zajímavé, a může to někomu připadat až pozoruhodné, že hraní hazardních her nebo náhodný výběr vzorků bude produkovat něco užitečného. Ostatně někteří autoři v době zrodu Monte Carlo metod tvrdili, že Monte Carlo nikdy nebude metodou, použitelnou pro jiné než hrubé odhady číselných veličin.

Patrně první dokumentovaný pokus, jak použít náhodné vzorkování pro nalezení hodnoty integrálu, pochází již z roku 1777.

Příklad 12.1 (Buffonova jehla)**.** *Comte de Buffon tehdy navrhl tento experiment: Mějme jehlu délky ℓ, kterou opakovaně náhodně upustíme na papír s nakreslenou soustavou rovnoběžek, jež se nacházejí ve vzdálenosti $d > \ell$. Jaká bude pravděpodobnost p, že jehla protne některou z rovnoběžek? Comte de Buffon tento experiment provedl a poté také analyzoval matematicky a ukázal, že pravděpodobnost, že jehla protne některou s rovnoběžek, je*

$$p = \frac{2\ell}{\pi d}.$$

Po nějaké době navrhl Pierre de Laplace, že by tento experiment bylo možno použít na ověření hodnoty π: Označíme p_N poměr hodů, kdy jehla protla nějakou rovnoběžku, ku všem N hodům. Platí

$$\lim_{N\to\infty} p_N = \frac{2\ell}{\pi D}$$

a tedy

$$\pi = \lim_{N\to\infty} \frac{2\ell}{p_N D}$$

Toto lze opravdu považovat za Monte Carlo metodu na určení hodnoty π. Její rychlost konvergence je ovšem velmi nízká.

Vzorkování se přitom používá už od nepaměti – banky či starověcí výběrčí daní odhadovali počet mincí či jiných komodit tak, že zvážili vybraný vzorek a pak celou hromadu a z poměru usuzovali na celkový počet mincí.

Systematické použití Monte Carlo metod pro řešení reálných technických a vědeckých problémů se prvně objevuje v raných dobách výpočetní techniky a doprovází konstrukci prvního programovatelného počítač na světě (nazýval se MANIAC, z angl. *Mathematical Analyzer, Numerical Integrator and Computer*, a byl umístěn v Los Alamos). Vědci, pracující na projektu první atomové bomby (Stanisław Ulam, John von Neumann, Nicholas Metropolis, Enrico Fermi a další), na tomto počítači řešili stochastickou simulací mimo jiné problém týkající se rozptylu neutronů v různých návrzích struktury atomové bomby a problém odhadu vlastních čísel (kvantové) stacionární Schrödingerovy rovnice.

Dnes již považujeme za samozřejmé, že mnoho problémů v oblasti statistiky či fyziky nebo biologie lze přeformulovat na úlohy statistické inference (usuzování) – jako optimalizační úlohy, jako úlohy pro výpočet mnohorozměrných integrálů nebo jako stochastické simulace. Monte Carlo metody toto umožňují vyčíslit. Zvlášť efektivní jsou v případě vícerozměrných problémů.

Příklad 12.2 (Optimalizace)**.** *Optimalizační problémy jsou problémy typu*

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} E(\mathbf{x})$$
$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} f(\mathbf{x}|\mathbf{d})$$

kde $E(\mathbf{x})$ označuje očekávanou hodnotu náhodné vektorové proměnné $\mathbf{x}$ a $f(\mathbf{x}|\mathbf{d})$ je podmíněná hustota pravděpodobnosti náhodné vektorové proměnné $\mathbf{x}$ při pozorovaných datech $\mathbf{d}$.

Příklad 12.3 (Integrace)**.** *Dalším příklad využití Monte Carlo metod je výpočet integrálu*

$$I = \int_D g(\mathbf{x})\, d\mathbf{x}$$

kde doména integrace D je prostor značně vysoké dimenze (ve finančnictví či fyzice není výjimkou počítat s dimenzemi řádu stovek).

12.2 Stochastická simulace

Doposud jsme pro řešení matematických problémů používali pouze deterministické numerické metody. Alternativní přístup, jenž se začal rozvíjet společně s nástupem první výpočetní techniky, jsou **stochastické simulace**. Detailní popis tématu jde jako obvykle nad rámec tohoto textu, zmíníme se ale alespoň o základních metodách a o náhodných generátorech, na než tyto metody spoléhají.

Stochastické simulační metody se snaží napodobit nebo ve vybraných případech přímo kopírovat chování systému z reálného světa tak, že využívají náhodnosti jevů – výsledkem simulace jsou vzorky statistické distribuce všech možných výsledků simulace. Vzhledem k náhodnosti, jež je zapojena do celého simulačního procesu, se tyto metody často nazývají Monte Carlo metody.

Stochastické simulační metody jsou užitečné pro zkoumání [1]:

- Nedeterministických procesů
- Deterministických procesů, jejichž popis je tak složitý, že jej nelze sestavit analyticky ■ **tohle ale je numerika, ne? tady heath podle mne nemá pravdu** ■
- Deterministických procesů s vysokou dimenzionalitou (jde například o fyzikální simulace s mnoha stupni volnosti), kde standardní diskretizace, používaná v případě deterministických metod, způsobuje exponenciální nárůst složitosti

Hlavními dvěma požadavky na stochastickou simulaci jsou

- Znalost odpovídajících pravděpodobnostních distribucí
- Dostatečná zásoba náhodných čísel, na jejichž základě při simulaci činíme náhodná rozhodnutí

Znalost relevantních pravděpodobnostních distribucí záleží na naší schopnosti teoreticky popsat nebo empiricky odpozorovat chování simulovaného reálného systému. Jedním z prvních reálných úkolů Monte Carlo metod byla simulace rozptylu neutronů při jejich průchodu stínícím médidem.

Příklad 12.4 (Průchod neutronu materiálem)**.** *K simulaci interakce neutronu s materiálem, jež má fungovat jako stínění, je třeba znát tak zvaný účinný průřez σ, vyjadřující pravděpodobnost, s jakou bude částice z nalétávajícího*

svazku interagovat s částicí média. Tato hodnota nám v simulaci určí průměrnou délku volného letu neutronu médiem před tím, než dojde ke kolizi s atomy média. Cestu neutronu médiem potom simulujeme na základě posloupnosti náhodných jevů, škálovaných odpovídající pravděpodobností (kolize, pohlcení, odrazu). Simulací dostatečně rozsáhlého počtu trajektorií částic můžeme získat aproximaci výsledné statistické distribuce celkových výsledků, přičemž přesnost této aproximace (bohužel pomalu, ale přece jenom) roste s počtem simulovaných částic.

Náhodnost

Definovat jednoznačně pojem **náhodnost** je poněkud složité. Fyzikální procesy, o nichž si myslíme, že jsou náhodné (třeba vrh kostkou nebo hod mincí) jsou totiž ve své podstatě deterministické jevy, popsané složitou soustavou pohybových a momentových rovnic pro příslušné počáteční podmínky. V posledních letech se dříve jasná dělící čára mezi deterministickým a náhodným chováním začíná ve světle nových konceptů jako je chaos a chaotické chování dynamických systémů (vzpomeňte si na Lorenzův atraktor) rozmazávat. Složitější nelineární dynamické systémy mohou být totiž extrémně citlivé na své počáteční podmínky a jejich chování v reakci na drobnou změnu počátečních podmínek může být zcela nepředvídatelné i když systém sám o sobě je deterministický – jde o tak zvaný **efekt motýlího křídla** (angl. *butterfly effect*) ■ **vzdálená obdoba špatné podmíněnosti úlohy?** ■.

Příklad 12.5 (Předpověď počasí)**.** *Ačkoliv fyzika dokáže celkem přesně popsat, jak se chová proudění kapalin a tedy i vzdušné proudění naší planety, přesnou předpověď počasí stále nejsme schopni získat pro delší časové období než několik dní. Důvodem je právě citlivost celého systému na drobné změny počátečních podmínek.*

Náhodná čísla

Jedním ze způsobů, jak charakterizovat nepřevídatelnost, kterou spojujeme s náhodností, je říci, že posloupnost čísel je náhodná tehdy a jen tehdy, když pro ni neexistuje žádný kratší popis, než posloupnost sama.

Příklad 12.6 (Náhodná posloupnost)**.** *I když pravděpodobnosti výskytu posloupností* $\{1,2,3,4,5,6\}$*,* $\{1,1,1,1,1,1\}$*,* $\{1,2,1,2,1,2\}$ *či* $\{4,1,6,2,3,5\}$ *mohou být zcela stejné, pouze tu poslední můžeme označit za náhodnou.*

V mnoha případech za náhodné považujeme i ne zcela náhodné veličiny, jako je například v teorii hromadné obsluhy čas příchodu požadavku a doba nutná pro jeho zpracování. Sestavit přesný model těchto veličin je totiž příliš složité a navíc tyto veličiny mnohdy ani nelze určit přesně. Studium systémů zahrnu-

jících takové veličiny je potom možno provádět pouze pomocí stochastických simulačních metod.

Druhou významnou vlastností náhodnosti je **neopakovatelnost**. Od náhodné posloupnosti čísel rozhodně neočekáváme, že se bude po nějaké době opakovat: pokud si budeme házet kostkou, očekáváme, že posloupnost hozených hodnot bude zcela nepředvídatelná a že se nezačne v nějakém okamžiku opakovat. Vlastnost neopakovatelnosti u náhodných posloupností je ale u počítačových experimentů v mnoha případech nežádoucí: představte si, že potřebujete ověřit chování nějakého simulačního programu nebo v něm dokonce budete hledat nějaké chyby. V takovém případě potřebuje programátor porušit premisu neopakovatelnosti a stochastický simulační proces spouštět se stále stejnou sekvencí náhodných čísel.

Opakovatelnost je ale dvojsečná zbraň. Statistický významnost stochastikcých simulací závisí na nezávislosti jednotlivých pokusů (let neutronu překážkou v Příkladu 12.4 by měl pokaždé používat nezávislou posloupnosti náhodných čísel, jinak nebudou výsledky k ničemu). V raných dobách stochastických simulací se pro výběr náhodných čísel používaly knižní tabulky (jako příklad uveďme alespoň [6], až do padesátých let 20. století v podstatě standardní tabulky pro stochastické experimenty)[1] Problém s tabulkami, navíc tak mohutně rozšířenými, byl v tom, že pokud by všichni experimentátoři začali svoji simulaci s náhodnou posloupností začínající na prvním řádku tabulek, všechny stochastické simulace založené na této knize by používaly zcela stejnou náhodnou sekvenci a mohly by vést ke zcela nesmyslným výsledkům.

Jakým způsobem si vybrat vhodný počáteční bod a jak v dnešní době počítače samy generují posloupnosti čísel, jež jsou blízké těm náhodným si povíme v následujícím odstavci.

12.3 Generátory náhodných čísel

S rozvojem výpočetní techniky převzaly postupně roli generátorů náhodných čísel počítače. Počítačové algoritmy pro generování náhodných čísel jsou ovšem z podstaty výpočetní techniky deterministické, pouze mají tu vlastnosti, že výsledná posloupnost čísel vypadá dostatečně náhodně a neopakovatelně.

Ovšem algoritmus, jenž generuje takovou posloupnost čísel, poskytuje velmi jasný popis (a většinou také velmi krátký, protože chceme, aby generátor náhodných čísel byl co nejrychlejší) toho, jak daná posloupnost vzniká. Z definice náhodnosti je potom jasné, že taková sekvence nemůže být označena jako náhodná, proto se pro označení prvků takto generovaných posloupností vžilo přesnější označení **pseudonáhodná čísla**. Pseudonáhodná posloupnost sice vypadá náhodně, ve své podstatě jde ovšem o opakovatelnou a predikovatelnou

[1] Doporučuji si pro pobavení přečíst uživatelské recenze na www.amazon.com.

sekvenci čísel – pro počítačové odborníky je to příjemná vlastnost, zaručující v případě potřeby možnost ladění chyb v simulačních program a verifikaci výsledků simulace. Důležitým rysem pseudonáhodné poslooupnosti čísel je pak i to, že vzhledem ke konečnému počtu čísel, reprezentovatelných v počítači, se po nějaké periodě musí každá pseudonáhodná sekvence začít opakovat.

Dobrý náhodný generátor by měl mít co nejvíce vlastností z následujícího seznamu [1]:

Generuje náhodný vzorek Generátor by měl splnit statistické testy náhodnosti (například χ^2 test)

Dlouhá perioda Generátor by měl generovat co nejdelší posloupnost čísel před tím, než se hodnoty začnou opakovat (typicky alespoň 2^{32} či 2^{64})

Efektivita Generátor by měl být rychlý a vyžadovat minimum operační paměti (většina simulací potřebuje generovat milióny náhodných čísel)

Opakovatelnost Pro totožné počáteční podmínky generátor produkuje totožnou posloupnost čísel.

Přenositelnost Generátor musí běžet na různých hardwarových platformách a musí přitom poskytovat totožné výsledky.

Splnit všech pět podmínek, uvedených výše, je těžké. Některé běžně používané pseudonáhodné generátory produkují silně korelované posloupnosti, což lze demonstrovat graficky pokud po sobě následující hodnoty použijeme k vykreslení 2D nebo 3D bodového grafu. Takovéto generátory mohou zcela zruinovat význam výsledku stochastické simulace, pokud nejsou použity vhodným způsobem.

Rané pseudonáhodné generátory

■ doplnit! ■

Kongruenční pseudonáhodné generátory

Již v závěru čtvrté přednášky, když jsme si povídali o významu modulárních výpočtů a kongruencí, jsme si uváděli příklad dnes asi nejrozšířenější formy generátorů pseudonhodných čísel, tedy **lineárního kongruenčního generátoru** (LCG). Takový generátor má tvar

$$x[k+1] \equiv a \cdot x[k] + b \pmod{n},$$

kde a a b jsou nějaká přirozená čísla, počáteční podmínka $x[0]$ se nazývá v překladu náhodné semínko (angl. *seed*) a hodnota n je přibližně (nejčastěji zcela)

rovna 2^m, kde m je počet bitů reprezentace přirozeného čísla (m je tedy obvykle 32 nebo 64). Kongruenční generátory generují posloupnosti celých čísel v intervalu $[0, n-1]$ a pro získání hodnot například rovnoměrného rozdělení $\mathcal{U}(0,1)$ je třeba v každém kroku v pohyblivé řádové čárce vydělit $x[k+1]/n$.

Kvalita takového pseudonáhodného generátoru velmi závisí na volbě hodnot a a b a v žádném případě jeho perioda nemůže překročit n. LCG může být poskytovat poměrně kvalitní pseudonáhodné posloupnosti, je ale třeba velmi pečlivě vybírat hodnoty a a b. Většina pseudnáhodných generátorů v počítačových systémech je dnes odvozena od LCG a některé z nich jsou notoricky známé svými nedostatky.

Základním problémem při návrhu hodnot pro lineární kongruenční generátor je dosažení plné periody n. pracoval s plnou periodou n jsou tyto [3]:

1. Čísla a a n jsou nesoudělná.
2. Číslo $a-1$ je dělitelné všemi prvočíselnými faktory čísla n.
3. Číslo $a-1$ je násobek 4, jestliže n je násobek 4.

Kratší perioda LCG není ovšem ještě ten nejhorší problém. Ve většině případů nevhodné volby a a b dojde totiž k tomu, že výsledné hodnoty sekvence padnou do malého počtu hyperrovin v prostoru poměrně nízké dimenze (mnohdy i 3) [5].

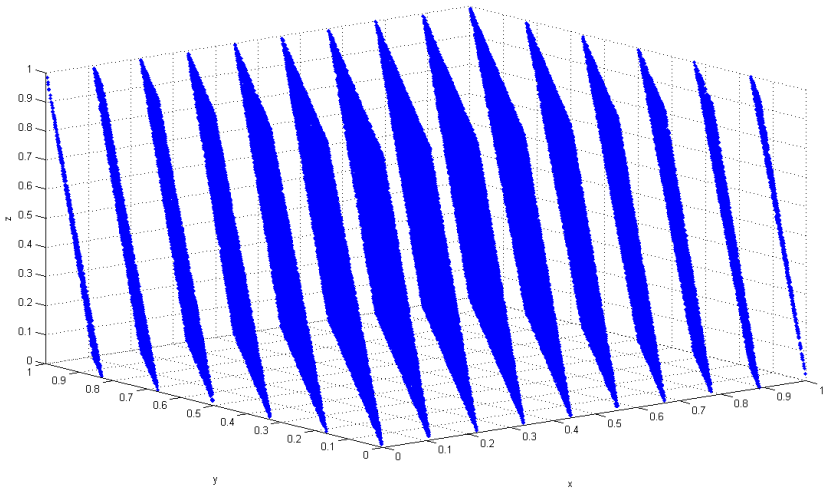
Příklad 12.7 (Multiplikativní kongruernční generátor RANDU)**.** *V případě $b = 0$ přejde LCG na multiplikativní generátor Lehmerova typu, používaný ve výpočetní technice už od konce čyřicátých let. Notoricky známým představitelem tohoto typu kongruenčních generátorů je RANDU, definovaný vztahem*

$$x[k+1] \equiv 65539 \cdot x[k] \pmod{2^{31}}.$$

Tento generátor je od sedmdesátých let 20. století uváděn jako odstrašující příklad nevhodného návrhu kongruenčního generátoru, neboť generuje velmi deterministickou posloupnost

$$x[k+2] \equiv 6\,x[k+1] - 9\,x[k] \pmod{2^{31}},$$

což má za následek, že body ve 3D, tvořené třemi po sobě vzatými hodnotami posloupnosti, leží na celkem 15 rovinách ve třídimenzionálním prostoru. Graf, znázorňující rozmístění jednotlivých bodů, uvádíme na Obrázku 12.1.



Obrázek 12.1: 100 002 hodnot vygenerovaných generátorem pseudonáhodných čísel RANDU a použitých po trojicích jako souřadnice bodů zobrazeno jako trojrozměrný graf. Převzato z Wikipedie

Fibonacciho generátory

Alternativn9 metodou, produkující rovnou rovnoměrně rozložená čísla v pohyblivé řádové čárce na intervalu 0,1 jsou Fibonacciho generátory, jež generují nové hodnoty jako součet, rozdíl či násobek předchozích hodnot. Typický příkladem je generátor

$$x[k+17] = x[k] - x[k+12].$$

Tento generátor má zpoždění 5 a 17. Na výběru zpoždění záleží, jaké bude mít generátor vlastnosti. Hodnota rozdílu může samozřejmě vyjít záporná, v takovém případě přičítáme jedničku, aby se výsledek opět ocitl v požadovaném intervalu 0,1.

Fibonacciho generátor pseudonáhodných čísel vyžaduje více paměti, než LCG. V případě, uvedeném výše, si musíme pamatovat celkem 17 předešlých hodnot. Tyto generátory také vyžadují specifický způsob startování (těch 17 hodnot nelze vybrat zcela náhodně). Na druhou stranu nevyžadují Fibonacciho generátory žádné pomalé dělení v pohyblivé řádové čárce, a dobře navržené Fibonacciho generátory mají velmi dobré statistické vlastnosti. Další výhodou Fibonacciho generátorů je jejich velmi dlouhá perioda, výrazně delší, než perioda kongruenčních generátorů. Důvod je jednoduchý: když u kongruenčního generátoru narazíme na stejný prvek posloupnosti, následné prvky se budou opakovat, v případě Fibonacciho generátoru tomu ale tak není – stejné následné prvky se v posloupnosti objeví až v okamžiku, kdy generátor obdrží

totožné všechny zpožděné prvky (v našem případě tedy prvky vzdálené celkem 12 kroků od sebe).

Nerovnoměrná náhodná čísla

Doposud jsme se zabývali generováním náhodných čísel pocházejících z rovnoměrného rozložení $\mathcal{U}(0,1)$. V případě, že potřebujeme rovnoměrně rozložená pseudonáhodná čísla na jiném intervalu $\mathcal{U}(a,b)$, lze původní hodnoty $x[k]$ jednoduše škálovat vztahem

$$x\prime[k] = (b-a)\,x[k] + a.$$

Výrazně složitější problém na nás ale čeká v případě, kdy bychom rádi vzorkovali z nějakého jiného rozdělení, než $\mathcal{U}(0,1)$. Velmi stručně si nyní ukážeme dva základní postupy, jimiž toho lze dosáhnout.

Metoda inverze

Pokud potřebuje vzorkovat z takového rozdělení pravděpodobnosti, jehož distribuční funkce $F(x)$ je snadno invertovatelná, můžeme použít jednoduchou transformaci hodnot: Nejprve vygenerujeme vzorek z rovnoměrného rozložení na intervalu [0,1] a tento vzorek potom pomocí inverzní distribuční funkce převedeme přímo na vzorek z odpovídajícího pravděpodobnostního rozdělení.

Příklad 12.8 (Exponenciální rozdělení)**.** *Uvažujme exponenciální rozdělení s hustotou pravděpodobnosti*

$$f(t) = \lambda e^{-\lambda t}, \quad t > 0$$

a s odpovídající distribuční funkcí

$$F(x) = \int_0^x f(t)\,dt = 1 - e^{-\lambda x} = y.$$

Inverzní distribuční funkce je potom

$$F^{-1}(y) = -\frac{\log(1-y)}{\lambda}.$$

Pro generování vzorků z $\mathcal{E}(\lambda)$ potom stačí vygenerovat $y[k] \sim \mathcal{U}(0,1)$ a spočíst $x[k] \sim \mathcal{E}(\lambda)$ jako

$$x[k] = -\frac{\log(1-y[k])}{\lambda}.$$

Zamítací metoda

Mnoho důležitých pravděpodobnostních rozložení má bohužel neinvertovatelnou distribuční funkci – asi nejprominentnějším zástupcem je normální rozdělení pravděpodobnosti, jehož distribuční funkci nelze vyjádřit v uzavřené formě (neznamená to ale, že distribuční funkci $\Phi(x) = 1/2 \cdot \mathrm{erfc}(-x/\sqrt{2})$ a její inverzi nelze spočítat přibližně nebo numericky s nějakou garantovanou chybou – metodu, popsanou v Odstavci 12.3 bychom tedy mohli použít, bude to ale velmi pomalé ■ **porovnat se Zigguratem** ■).

Jedním ze způsobů (nikoliv ale způsobem nejefektivnějším), jak generovat posloupnost hodnot z libovolného rozdělení $\mathcal{F}$, u něhož jsme schopni analyticky vyjádřit jeho hustotu pravděpodobnosti $f(x)$, ale distribuční funkci $F(x)$ již nikoliv, je tak zvaná **zamítací metoda**, jež pro vygenerování jedné náhodné hodnoty z libovolného rozdělení potřebuje vždy alespoň dvě náhodná čísla. Tato metoda postupuje následovně:

1. Z nějakého rozdělení pravděpodobnosti (může to být rovnoměrné rozdělení, ale nemusí, výhodnější je použít takové rozdělení $\mathcal{G}$, jehož hustota pravděpodobnosti $g(x)$ vhodně aproximuje $f(x)$ až na multiplikativní konstantu M tak, že $\forall x : f(x) \leq Mg(x)$■ **moc koncentrované, rozvést nebo vynechat** ■), navzorkujeme bod x
2. Spočteme hodnotu $f(x)$, neboť hustotu pravděpodobnosti máme dánu analyticky.
3. Vygenerujeme náhodné číslo $u \sim \mathcal{U}(0, 1)$
4. Pokud je $u \cdot Mg(x) < f(x)$, akceptujeme x jako vzorek z $\mathcal{F}$. V opačném případě se vracíme na začátek a vzorkujeme nový bod x.

Algoritmus 12.1 naznačuje, jak bychom mohli použít zamítací metodu pro vzorkování distribuce s hustotou pravděpodobnosti podobnou hustotě rovnoměrného rozdělení pravděpodobnosti.

Obdobně jako v případě rovnoměrného rozdělení pravděpodobnosti, i v případě normálního rozdělení stačí generovat vzorky ze „základního“ rozdělení $\mathcal{N}(0, 1)$. Vzorky $x \sim \mathcal{N}(\mu, \sigma^2)$ lze získat ze vzorků $y \sim \mathcal{N}(0, 1)$ transformací

$$x = \sigma y + \mu.$$

12.4 Monte Carlo integrace

V osmé přednášce jsme se zabývali metodami numerické integrace funkcí jedné proměnné a ukázali si základní kvadraturní vzorce. V metodách výpočetní fyziky či počítačové grafiky se velmi často setkáváme s nutností počítat složité

Algoritmus 12.1 Zamítací metoda vzorkování využívající pouze rovnoměrného rozdělení pravděpodobnosti. Jde pouze o ilustrativní příklad, vhodný pro vzorkování distribucí s „plochou“ hustotou, podobnou rovnoměrnému rozdělení. Vzorkování $\mathcal{N}(0,1)$ není tímto způsobem efektivní

Require: hustota pravděpodobnosti $f(x)$ rozdělení $\mathcal{F}()$
Ensure: $x \sim \mathcal{F}()$
  $f_{\max} \leftarrow \max_x f(x)$
  repeat
    $x \sim \mathcal{U}(-\infty, \infty)$
    $u \sim \mathcal{U}(0,1)$
  until $u \cdot f_{\max} < f(x)$

integrály funkcí mnoha proměnných. Pro tyto úlohy není standardní přístup s rovnoměrným rozdělením itegračního intervalu a případným zjemňováním příliš vhodný.

Příklad 12.9 (Počet vyčíslení vícerozměrné funkce)**.** *Budeme-li pro náš kvadraturní vzorec potřebovat vyčíslit v jedné dimenzi integrovanou funkci celkem m-krát, bude v $n \gg 1$ dimenzích tento počet vyčíslení roven m^n. Pokud tedy integrujeme $f(x)$ a vyhodnotíme ji desetkrát, v případě desetirozměrné fyzikální úlohy budeme funkci $f(\mathbf{x})$ vyhodnocovat celkem 10^{10}-krát.*

Jediný v praxi použitelný způsob, jak vyhodnocovat vícerozměrné integrály pro počet dimenzí větší, než dva, je **Monte Carlo integrace**. Princip je velmi jednoduchý: Integrovanou funkci $f(\mathbf{x})$ navzorkujeme v celkem m bodech, vybraných náhodně v doméně integrace D a spočteme průměrnou hodnotu $\bar{y}$. Výsledný odhad integrálu je potom roven $I \approx A(D)\bar{y}$, kde $A(D)$ je plocha domény integrace ■ **resp. objem resp. něco jiného** ■. Monte Carlo integrace koverguje velmi pomalu: chyba odhadu klesá s $\mathcal{O}(1/\sqrt{n})$, abychom tedy získali jedno další desetinné místo přesnosti, musíme použít stonásobný počet vzorků. Není proto neobvyklé, že počet vyhodnocení integrandu se pohybuje v řádech miliónů či miliard.

Proč se tedy vlastně Monte Carlo integrace používá, když jde o tak pomalu konvergující postup? (Von Neummann: MC is a method of last resort.) Již jsme si řekli, že pro integraci funkcí v jedné či dvou proměnných postrádá Monte Carlo integrace smysl. Její půvab ale spočívá v tom, že na rozdíl od deterministických postupů není její rychlost konvergence na dimenzionalitě řešeného problému. 1

Příklad 12.10 (Monte Carlo integrace vícerozměrné funkce)**.** *Pro $n = 10^6$ a doménu integrace $D \subset \mathbb{R}^6$ vychází u deterministického přístupu počet vzorků na jednu dimenzi roven 10. Budeme-li vyžadovat stejnou přesnost, budeme ale*

při použití deterministického přístupu muset použít vzorků výrazně větší počet ■ ***kolik?*** ■.

V nejjednodušším případě, kdy $D = [0,1]$ a $I = \int_0^1 f(x)\mathrm{d}x$, můžeme tento integrál aproximovat jako

$$\bar{I}_m = \frac{1}{m}\left(f(b_1) + \cdots + f(b_m)\right),$$

kde $b_j = j/m$. Jde o aproximaci integrálu, vycházející z původní Riemannovy definice: V každém z m panelů o délce $1/m$ použijeme k odhadu funkční hodnotu na jeho pravém okraji. Budeme-li předpokládat, že funkce f je spojitá a dostatečně hladká, pak se chyba Riemannovy aproximace klesá jako $\mathcal{O}(1/m)$, což je výrazně lepší, než $\mathcal{O}(1/\sqrt{m})$ v případě Monte Carlo integrace (pro dosažení chyby Riemannovy aproximace bychom museli navzorkovat celkem m^2 funkčních hodnot). Použijeme-li pokročilejší deterministická kvadraturní pravidla m-tého řádu, například složené obdélníkové či Simpsonovo pravidlo, bude chyba aproximace integrálu ještě nižší – pro obdélníkové pravidlo bude chyba klesat jako $\mathcal{O}(1/m^2)$, pro Simpsonovo pravidlo dokonce jako $\mathcal{O}(1/m^4)$ (blíže viz [1, odstavec 8.4.1] nebo ■ **desátá?** ■ přednáška MAG).

S rostoucím počtem dimenzí domény D ale deterministická kvadraturní pravidla ztrácí své kouzlo: V případě $D = [0,1]^{10}$ a $I = \int_{\mathbf{0}}^{\mathbf{1}} f(\mathbf{x})\mathrm{d}\mathbf{x}$ budeme v případě Riemannovy aproximace muset pro dosažení stejného řádu chyby jako výše vyhodnotit celkem $\mathcal{O}(m^{10})$ uzlových vektorů. Pokud použijeme Monte Carlo integraci, navzorkujeme celkem $\mathcal{O}(m^2)$ rovnoměrně distribuovaných vektorů $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{m^2}$ z domény D, tedy celkově $10m$ rovnoměrně distribuovaných hodnot, a chyba aproximace bude stále $\mathcal{O}(1/m)$ bez ohledu na dimenzi problému.

Poznamenejme, že ačkoliv chyba Monte Carlo integrace klesá stejným řádem nezávisle na dimenzi, při rostoucím počtu dimenzí se u této metody musíme potýkat se dvěma problémy, jež vyplývají z podstaty této metody:

1. Je-li doména D rozlehlá, rozptyl σ^2, tedy míra „uniformity“ integrované funkce f v celém regionu D, může být při rovnoměrném vzorkování velmi vysoký.
2. Vzorkovat rovnoměrně doménu D obecného tvaru nemusí být vždy možné.

Řešením těchto problémů je metoda **vzorkování podle významnosti** (angl. *importance sampling*), kdy vzorky $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ z domény D vzorkujeme z nerovnoměrného rozdělení pravděpodobnosti $\pi(\mathbf{x})$, jež s vyšší pravděpodobností generuje vzorky ve „významných“ částech regionu D. V takovém případě lze integrál I aproximovat jako

$$\hat{I}_m = \frac{1}{m}\sum_{j=1}^{m}\frac{f(\mathbf{x}_j)}{\pi(\mathbf{x}_j)},$$

přičemž rozptyl takového statistického odhadu bude $\sigma_\pi^2 = \mathrm{var}(f(\mathbf{x})/\pi(\mathbf{x}))$. V ideálním případě, kdy je f nezáporná a I konečné, bychom mohli zvolit $\pi(\mathbf{x}) \propto f(\mathbf{x})$. To ovšem znamená, že f známe, a v praxi tato situace není příliš pravděpodobná. Budeme proto spíše hledat takové π, jež je dostatečně podobné f a nasměruje více vozrků do oblastí, kde má f vysokou absolutní hodnotu. Generovat ovšem vzorky z takové spojité distribuce může být docela složité.

Efektivitu Monte Carlo integrace lze zvýšit několika dalšími různými metodami – použitím kvazináhodných posloupností, které rozmísťují vzorkovací body ve doméně integrace rovnoměrněji, než náhodně, ale nikoliv zcela deterministicky, či **vzorkováním po částech** (angl. *stratified sampling*), kdy doménu D dělíme buď předem na několik subdomén a celkový integrál počítáme jako součet dílčích integrálů, nebo doménu dělíme rekurzivně (obdobně jako u adaptivních kvadraturních pravidel) na základě odhadu rozptylu integrované funkce.

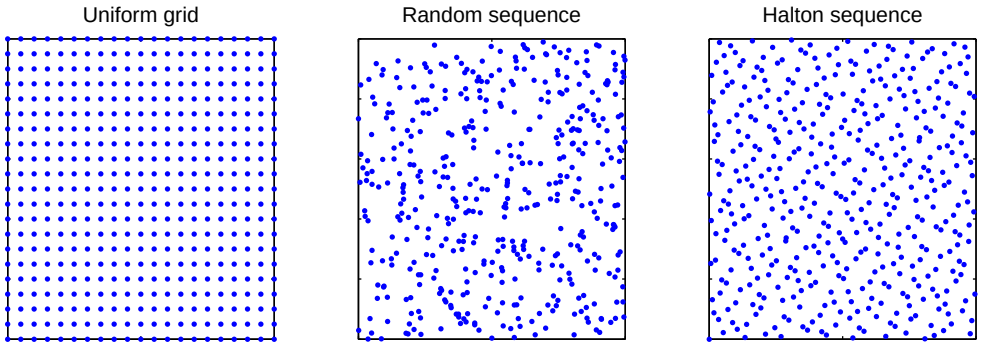
12.5 Kvazináhodná čísla

Přes všechnu snahu, kterou věnujeme vývoji algoritmů, generujících co nejnáhodnější pseudonáhodná čísla, je vhodné si uvědomit, že opravdová náhodnost není vždy nutností. Pro určité vypy aplikací, jako je například Monte Carlo integrace, je podstatně důležitější dosáhnout co nejrovnoměrnějšího náhodného pokrytí vzorkované domény integrace, než to, jestli jsou vzorky opravdu zcela náhodné. Jedním z problémů opravdu náhodných posloupností čísel je totiž to, že jejich hodnoty se mohou v některých oblastech náhodně shluknout a celkové pokrytí vzorkovaného intervalu je tak poměrně nerovnoměrné, viz Obrázek 12.2. Druhým extrémem je použití zcela rovnoměrné mřížky bodů (jako v případě deterministických kvadraturních algoritmů), jak jsme ale viděli v předchozím odstavci, tento postup nelze efektivně škálovat do vyšších dimenzí.

Kvazináhodné posloupnosti čísel (angl. *quasi-random sequences*) představují jakýsi kompromis mezi zcela rovnoměrným a zcela náhodným pokrytím. Tyto posloupnosti nelze považovat za náhodné, jde o deterministické posloupnosti, jež jsou zkonstruovány tak, aby poskytovaly pro daných k vzorků co nejrovnoměrnější pokrytí vzorkovaného intervalu a zároveň aby vypadaly „dostatečně náhodně“. Jednotlivé prvky kvazináhodných posloupností se sobě záměrně vyhýbají, aby se předešlo shlukům, jež pozorujeme u náhodných posloupností.

Rozdíly všech tří zmíněných přístupů ke vzorkování ukazuje Obrázek 12.2.

Kvazináhodné posloupnosti, někdy též označované za posloupnosti s nízkou diskrepancí (angl. *low-discrepancy series*) se v dnešní době s oblibou používají



Obrázek 12.2: 441 hodnot vygenerovaných jako rovnoměrná mřížka 21×21 bodů, psudonáhodným generátore a Haltonovou kvazináhodnou sekvencí

v Monte Carlo metodách, vyžadujících dostatečně rovnoměrné pokrytí vzorkované domény a nekladou takový důraz na statistické vlastnosti vzorkovaného souboru – jde například o integraci či náhodné prohledávání. Nejsou naopak vhodné pro statistické simulace, kde velmi záleží na nezávislosti jednotlivých vzorků.

V případě integrace zvýší použití kvazináhodné posloupnosti významně rychlost konvergence, chyba nyní klesá s $\mathcal{O}(1/n)$, bohužel rychlost konvergence se stává závislou na dimenzi řešené úlohy (teoretická horní hranice chyby je pro d dimenzí nepříjemných $\mathcal{O}((\ln n)^d/n)$, v praxi ale v mnoha případech vycházejí i pro $d > 100$ chyby blíže k $\mathcal{O}(1/n)$) a reálná rychlost konvergence proto s rostoucí počtem rozměrů klesá ■ **jak? doplnit** ■.

Literatura

[1] Michael T. Heath. *Scientific computing: an introductory survey*. McGraw-Hill, Boston, 2 edition, 2002.

[2] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*. Wiley-VCH, Weinheim, 2 edition, 2008.

[3] Donald E. Knuth. *The art of computer programming*, volume Volume 2: Seminumerical Algorithms. Addison-Wesley, Upper Saddle River, 3 edition, 1998.

[4] Jun S. Liu. *Monte Carlo strategies in scientific computing*. Springer, New York, 2008.

[5] George Marsaglia. Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences*, 61(1):25–28, 1968.

[6] The RAND Corporation. *A million random digits with 100,000 normal deviates*. RAND Corp., Santa Monica, CA, 2001.

[7] Miroslav Virius. *Metoda Monte Carlo*. ČVUT (Pražská technika?), Praha, 2010.

Rejstřík

a dělí b, 37
úplný výběr hlavního prvku, 157
Číslo podmíněnosti, 140
Řešením, 91
částečný výběr hlavního prvku, 157
čísla
    nesoudělná, 36
    prvočísla, 27
    složená, 27
číslo podmíněnosti, 89
řádkovém škálování, 147
řádu, 136
šifrovací exponent, 57

Absolutní chybou, 86
absolutním čísle podmíněnosti, 90
algoritmus
    vlastnosti, 8
Algoritmus A*, 71
analýza řešení, 83
analýza modelu, 83
aproximace, 81, 163
    lineárního typu, 165
aproximace metodou nejmenších čtverců, 168
asymptoticky blíží, 28

bázových funkcí, 172

Celá čísla, 83
chyba aproximace, 164
chybou aproximace, 85, 165
chybou matematického modelu, 85
chybou metody, 85
chybou modelu, 85
chyby (šum) ve vstupních datech, 85
chyby zaokrouhlovací, 86

dělitel
    největší společný, 36
dešifrovací exponent, 57
diagonální, 146
diagonální škálování, 146
Dijkstrův algoritmus, 71
diskrétní logaritmus, 55
dobře podmíněná, 90
dolní trojúhelníková, 148
dvojnásobné přesnosti, 87

efekt motýlího křídla, 188
elementární eliminační matice, 148, 154
Eulerova funkce, 44
exponent, 84

Fermatova testu prvočíselnosti, 44
fundamentální polynomy, 176
funkce
    bázové, 164

Gaussova eliminační metoda, 151
grafovým algoritmem, 63
grafovou úlohu, 63

hlavní prvky, 152
hodnost matice, 137
horní trojúhelníková, 148
Hornerovo schéma, 175
hran, 64

indukovaná, 139

interpolační aproximace, 168
interpolační podmínky, 170
interpolační uzly, 170
interpolaci, 170
interpolanty, 169
inverzní kvadratické interpolace, 109
inverzním prvkem, 43
isomorfní grafy, 66
iterační, 135
iterace, 96

jednoduchou přesnost, 87

kořenem, 91
koeficienty, 114
kontrakce, 93
konvergentní kvadraturu, 118
konzistentní, 136, 140
korektní, 89
kvadraturního vzorce, 113
kvadraturou, 118
Kvazináhodné posloupnosti, 197

Lagrangeův interpolační polynom, 176
Lagrangeovy bázové polynomy, 176
lichoběžníkové pravidlo, 120
lineární kongruentní generátor, 47
lineární rovnice, 91
lineárních algebraických rovnic, 135
lineárního kongruenčního generátoru, 190
lokálně konverguje, 103

mantisa, 84
matematické modelování, 79
matematického modelu, 79
matematický model, 80
Matematickou úlohou, 89
matic, 136
Maticí sousednosti $\mathbf{G}_{n\times n}$, 68
metoda bisekce, 98
metoda konverguje s rychlostí, 96
Metoda půlení intervalu, 98
modul, 39, 57
Modulární aritmetika, 39
modulo, 38
monomy, 170
Monte Carlo integrace, 195
multiplikátory, 151
multiplikativní inverzí, 44
multiplikativní inverzi, 43

náhodnost, 188
Nederministický Turingův stroj, 24
nejmenší multiplikativní inverzi, 44
největšího společného dělitele, 38
nekorektní, 89
nelineární rovnice, 91
neopakovatelnost, 189
neorientovaná, 64
neorientovaný graf, 64
nesoudělná, 38
Newtonova metoda, 104
Newtonovy-Cotesovy vzorce, 119
normy, 137
normy funkce, 166
nulovým bodem, 91
Numerická úloha, 82
numerická úloha, 135
numerická kvadratura, 113
numerická matematika, 83
Numerický algoritmus, 82

obdélníkové pravidlo, 120
orientovaná, 64
Orientovaný graf, 64
otevřený, 114, 119

přímá substituce, 149
přímé, 135
přímý chod, 152
přiblížení, 81, 163
panely, 119
permutační maticí, 146
Pevná řádová čárka, 84
pevným bodem, 91
pivotace, 156
pivoty, 152
počítačovým modelem, 81

podřízená, 139
Podgraf, 65
Pohyblivá řádová čárka, 84
procházení do šířky, 70
procházení do hloubky, 70
procházení grafem, 70
progresívní, 119
pseudonáhodná čísla, 189

růstový faktor, 160
regulární, 137
Relativní chybou, 86
relativní reziduum, 144
Reziduum, 143
rozšířené matice, 152
rozšířeným Eukleidovým algoritmem, 39
rozhodovací úlohu, 23

saxpy, 149
semilogaritmickém tvaru s normalizovanou mantisou, 84
Simpsonovo pravidlo, 120
singulární, 137
složené, 119
smyčkou, 64
soukromý klíč, 57, 60
společný dělitel, 38
stabilitou výpočtu, 115
stabilní, 116
stochastické simulace, 187
stupněm vrcholu, 65
submultiplikativní, 140
syntetické dělení, 175
systém, 79

třída kongruence, 41
třídou aproximujících funkcí, 165
třídu polynomů, 165
trojúhelníková, 148

uzávěrou, 92
uzavřený, 114, 120
uzly interpolace, 170
uzly kvadratury, 113

váhy, 114
váhy kvadraturního vzorce, 114
Výčtem sousedů $\mathbf{S}_{n\times m}$, 68
výběr hlavního prvku, 156
validovat, 83
Vandermondova matice, 117, 173
Veřejný klíč, 57
verifikovat, 83
vnořená, 119
vrcholů, 64
vzorkování podle významnosti, 196
vzorkováním po částech, 197

základ, 84
zamítací metoda, 194
zpětná substituce, 149
zpětný chod, 152