Vybrané úlohy řešené na počítači: diskrétní a numerická matematika

Doc. Dr. rer. nat. Ing. Jan Valdman

Katedra matematiky
a
Katedra informatiky
Přírodovědecká fakulta
JU v České Budějovicích

2024

ISBN 978-80-7394-905-1

Snahou této publikace bylo vytvořit doplňkový text k přednáškám autora na PřF JU v Českých Budějovicích v období 2012 - 2023, který by čtenáře motivoval ke studiu vybraných matematických úloh za využití počítače. Věřím, že kombinace matematika-počítač přiláká pozornost studentů informatiky k vlastním počítačovým řešením matematických úloh a také studentů matematiky, kteří nemusí vždy jen nutně abstrahovat na papíře a mohou na počítači přiblížit svá řešení i nematematikům. Vývoj výpočetních a matematických systémů jde stále rychleji dopředu a počítačová gramotnost se očekává od všech absolventů. Rozsah tohoto textu pokrývá většinu praktických příkladů předmětů:

1. Diskrétní matematika (DM),
2. Numerická matematika 1 (NM1),
3. Numerická matematika 2 (NM2).

Předkládané úlohy jsou většinou známé, nová jsou vlastní počítačová řešení v systémech Matlab nebo Mathematica. Ty jsou k dispozici také ke stažení v úložišti GitHub (https://github.com/matlabfem/teaching). Počítačová řešení stačí vložit do příslušného systému a vyhodnotit. Většina kódů je psána tak, že jdou jednoduše modifikovat. Protože je u každé úlohy uvedeno pro jednoduchost řešení jen v jednom ze systémů, budu rád za Vaše vlastní řešení v systému druhém. Zde je třeba čtenáře varovat před pouhým slepým kopírováním a spouštěním počítačových kódů. Namísto toho doporučuji si nejdříve prostudovat komentovaná řešení, nebo se rovnou pokusit o řešení vlastní včetně počítačového. Pro ujištění, zda jsou vysvětlené postupy jasné, slouží úlohy k samostatnému řešení. Ty často také představují domácí úlohy výše zmíněných předmětů.

Korekce textu se také ujali studenti jednotlivých předmětů: Tomáš Vomáčka, Karolína Wolfová a Filip Nachtman (DM), Patrik Musil (NM1), Bc. Radka Šimerová, Bc. Lukáš Hronek (NM2). Pan Mgr. Alexej Moskovka mi pomohl s vizualizacemi v Mathematice. Všem srdečně děkuji. Další náměty k vylepšení skript mi posílejte na email: jvaldman@prf.jcu.cz .

Rád bych poděkoval Dr. Zuzaně Morávkové a prof. Radkovi Kučerovi z VŠB Ostrava za poskytnutí grafických formátů skript převzatých a modifikovaných z [1] a dále také doc. Janě Kalové za koordinaci v rámci grantu Rozvoj flexibilních forem vzdělávání na PřF JU, z něhož byla příprava 1. vydání skript v roce 2021 částečně financována.

V Českých Budějovicích, v červnu 2024,
Jan Valdman

Historie vydání:

- 1. vydání (leden 2022, celkem 50 stran) - obsahuje kapitoly Diskrétní matematika a Numerická matematika 1,
- 2. aktualizované vydání (srpen 2023, celkem 82 stran) - aktualizace obou kapitol a přidání kapitoly Numerická matematika 2.
- 3. aktualizované vydání (červen 2024, celkem 105 stran) - aktualizace kapitol Numerická matematika 1 a 2.

OBSAH

KAPITOLA

1

DISKRÉTNÍ MATEMATIKA

1.1 Matematická indukce

Příklad 1.1.1 Dokažte, že pro všechna přirozená čísla $n \in \mathbb{N}$ platí

$$\sum_{k=1}^{n} k^3 = \frac{1}{4}n^2(1+n)^2. \tag{1.1}$$

Řešení: Indukční předpoklad, tedy tvrzení pro n=1 platí, neboť dosazením do vzorce (1.1) obdržíme rovnost $1 = 1$.
Sumu pro $n+1$ členů můžeme rozložit obecně jako sumu pro n členů a $(n+1)$-ho členu

$$\sum_{k=1}^{n+1} k^3 = (\sum_{k=1}^{n} k^3) + (n+1)^3.$$

Sumu pro n členů nahradíme pomocí indukčního předpokladu (je vyznačený modrou barvou) (1.1) a získáme

$$\sum_{k=1}^{n+1} k^3 = \frac{1}{4}n^2(1+n)^2 + (n+1)^3 = \frac{1}{4}(1+n)^2 \cdot (2+n)^2.$$

Poslední výraz představuje tvrzení (1.1) pro $n+1$ členů a tím je důkaz matematickou indukcí dokončen.
Na počítači dovedeme sumu spočítat přímo:

Kód - Wolfram Alpha 1.1: suma mocnin

```
1 Sum[k^3,{k,n}]
```

Poznámka k příkladu

Jednoduchou modifikací kódu výše lze získat vzorce

$$\sum_{k=1}^{n} k = \frac{1}{2}n \cdot (1+n), \tag{1.2}$$

$$\sum_{k=1}^{n} k^3 = \frac{1}{4}n^2(1+n)^2, \tag{1.3}$$

$$\sum_{k=1}^{n} k^4 = \frac{1}{30}n(n+1)(2n+1)\left(3n^2+3n-1\right), \tag{1.4}$$

$$\sum_{k=1}^{n} k^5 = \frac{1}{12}n^2(n+1)^2\left(2n^2+2n-1\right), \tag{1.5}$$

které si zkuste technikou matematické indukce také dokázat. Vzorce jsou speciálními případy obecné Faulhaberovy formule, kterou lze najít včetně důkazu v literatuře. V počítačových systémech je přímý výpočet sumy označován jako symbolická sumace (angl. symbolic summation).

Úloha k samostatnému řešení

Obecně platí (to neukazujeme) pro $p \in \mathbb{N}$, že

$$\sum_{k=1}^{n} k^p = a_{p+1}n^{p+1} + a_p n^p + \ldots a_1 n + a_0 \tag{1.6}$$

tj. výsledný vzorec pro sumu je polynom stupně $p+1$ v proměnné n. Dosazením konkrét-

ních hodnot pro $n \in \{1,\ldots,p+2\}$ do (1.6) získejte systém $p+2$ lineárních rovnic pro koeficienty $a_0,\ldots,a_{p+1}$. Vzniklé systémy lineárních rovnic zapište a pro $p \in \{1,2,3,4,5\}$ vyřešte. Vypočítané koeficienty dosaďte do polynomu na pravé straně výrazu (1.6) a tím získáte vzorce (1.1) - (1.5).

Příklad 1.1.2 Dokažte, že pro všechna $n \in \mathbb{N}$ platí

$$\sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}, \qquad q \neq 1. \tag{1.7}$$

Řešení: Číslo $q \in \mathbb{R}$ představuje tzv. kvocient geometrické posloupnosti

$$\{1, q, q^2, q^3, \ldots, q^n\} \tag{1.8}$$

a formule (1.7) přestavuje součet její řady. Tvrzení (1.7) pro $n = 0$ platí, neboť $1 = 1$.
Sumu pro $n+1$ členů můžeme rozložit jako

$$\sum_{k=0}^{n+1} q^k = \sum_{k=0}^{n} q^k + q^{n+1}$$

a dále upravit pomocí indukčního předpokladu (1.7), který opět vyznačíme modrou barvou, jako

$$\sum_{k=0}^{n+1} q^k = \frac{q^{n+1}-1}{q-1} + q^{n+1} = \frac{q^{n+1}-1+q^{n+1}(q-1)}{q-1} = \frac{q^{n+2}-1}{q-1},$$

což představuje tvrzení (1.7) pro $n+1$ členů. Tím je důkaz matematickou indukcí hotov.
Na počítači dovedeme takovou sumu spočítat přímo:

Kód - Wolfram Alpha 1.2: suma členů geometrické posloupnosti

```
Sum[q^k,{k,0,n}]
```

Příklad 1.1.3 Dokažte, že pro Fibonacciho posloupnost danou rekurentně pomocí

$$F_n := F_{n-1} + F_{n-2}, \qquad F_1 = 1, F_2 = 1,$$

platí Moivre-Binetova formule

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]. \tag{1.9}$$

Řešení: K přímému ověření pro deset prvních členů formule (1.9) využijeme počítač:

Kód - Mathematica 1.3: Fibonacciho posloupnost.

```
F[n_]=((1+Sqrt[5])^n-(1-Sqrt[5])^n)/(2^n*Sqrt[5]);
Do[Print["n=",n,":", F[n],"=",Simplify[F[n]]],{n,10}];
```

a výsledkem je opravdu prvních deset členů Fibonacciho posloupnosti

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55.$$

Nyní postupujme pomocí matematické indukce. Indukční předpoklad $F_1 = 1, F_2 = 1$ je tedy splněn. Musíme zjistit, zda platí $F_{n+1} = F_n + F_{n-1}$ pro všechna $n \in \mathbb{N} : n \geq 2$, kde za modré členy dosazujeme z (1.9). Ověřujeme tedy rovnost

$$\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right] = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n} - \left(\frac{1-\sqrt{5}}{2}\right)^{n}\right] + \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}\right]. \quad (1.10)$$

Počítač ověří (1.10) přímo pomocí příkazů:

Kód - Mathematica 1.4: indukční krok důkazu Fibonacciho posloupnosti.

```
F[n_]=((1+Sqrt[5])^n-(1-Sqrt[5])^n)/(2^n*Sqrt[5]);
Simplify[F[n+1]==F[n]+F[n - 1]]
```

a výsledkem je hodnota 'true', protože rovnost platí. Tím je důkaz matematickou indukcí hotov.

Poznámka k příkladu

Rovnost (1.10) lze odvodit bez počítače. Pro zjednodušení algebraických úprav lze zavést pomocné proměnné $\varphi = \frac{1+\sqrt{5}}{2}, \psi = \frac{1-\sqrt{5}}{2}$ a využijte rovnosti $\varphi = 1 + \frac{1}{\varphi}, \psi = 1 + \frac{1}{\psi}$.

Úloha k samostatnému řešení

Dokažte matematickou indukcí, ze počet úhlopříček v konvexním n-úhelníku je

$$p(n) = n(n-3)/2.$$

Více informací naleznete na (odkaz zde).

Úloha k samostatnému řešení

Na konto v bance vložíme při otevření konta na začátku roku částku 10000 Kč a během každého roku opětovně vkládáme dalších 1000 Kč.

1. Kolik bude naspořená částka po 10 letech za předpokladu, že na konci každého roku banka přidá 2% z aktuální částky?
2. Kolik bude naspořená částka po 10 letech za předpokladu, že na konci každého roku banka nejdříve přidá 2% z aktuální částky a potom si strhne manipulační poplatek 100 Kč?
3. Kolik bude naspořená částka po 10 letech za předpokladu, že na konci každého roku si banka nejdříve strhne si manipulační poplatek 100 Kč a potom přidá 2% z aktuální částky?

Vyjádřete rekurentní formule popisující zůstatky na kontě ve třech rozdílných případech uvedených výše. Z nich potom přepočítejte stavy konta postupně po jednotlivých rocích nebo nechte systém Mathematica nebo WolframAlpha vyřešit rekurentní formule za Vás.

1.2 Faktoriály a kombinační čísla, kombinační úlohy

Příklad 1.2.4 Odhadněte velikost čísla 1000! pomocí odhadu

$$n^{\frac{n}{2}} \leq n! \leq \left(\tfrac{n+1}{2}\right)^n. \tag{1.11}$$

Řešení: Odhad (1.11) byl již znám Gaussovi a jeho důkaz lze najít např. v [2]. Přímý výpočet na jednoduchém kalkulátoru hlásí chybu, protože velikosti počítaných čísel přetečou maximální dovolenou hodnotu exponentů rovné 99. Problémem už často bývá (ověřte na kalkulátoru) i výpočet hodnoty $70! \approx 1.1978 \cdot 10^{100}$.
Výhodné je proto přepsat obě strany odhadu jako

$$n^{\frac{n}{2}} = 10^{\log\left(n^{\frac{n}{2}}\right)} = 10^{\frac{n}{2}\log n}, \qquad \left(\tfrac{n+1}{2}\right)^n = 10^{\log\left(\frac{n+1}{2}\right)^n} = 10^{n\log\left(\frac{n+1}{2}\right)}$$

pomocí funkce dekadického logaritmu $\log(\cdot)$. Tak převedeme (1.11) do tvaru

$$10^{\frac{n}{2}\log n} \leq n! \leq 10^{n\log\left(\frac{n+1}{2}\right)},$$

který dále odhadneme ve tvaru s celočíselnými exponenty jako

$$10^{\left\lfloor\frac{n}{2}\log n\right\rfloor} \leq n! \leq 10^{\left\lceil n\log\left(\frac{n+1}{2}\right)\right\rceil}, \tag{1.12}$$

kde $\lfloor\cdot\rfloor$ a $\lceil\cdot\rceil$ představují funkce dolní a horní celé části čísla. Velikosti exponentů v (1.12) již potom na kalkulátoru spočítat lze a získáváme velmi hrubý odhad

$$10^{1500} \leq 1000! \leq 10^{2700}.$$

Následující program tento oboustranný odhad vyhodnocuje:

Kód - Mathematica 1.5: výpočet faktoriálu.

```
n = 1000;
Print[10^N[(n/2)*Log[10,n]]]
Print[10^N[n*Log[10,(n+1)/2]]]
```

Modifikací hodnoty n v prvním řádku získáme odhady $n!$ pro libovolné $n \in \mathbb{N}$.

Poznámka k příkladu

Existují odhady přesnější, např. [2] odvozuje

$$e\left(\tfrac{n}{e}\right)^n \leq n! \leq en\left(\tfrac{n}{e}\right)^n, \tag{1.13}$$

kde e představuje Eulerovo číslo. Pomocí (1.13) lze odhadnout (vypočtěte) přesněji

$$10^{2566} \leq 1000! \leq 10^{2570}.$$

Pro úplnost uveďme ještě Stirlingův vzorec

$$n! \approx \sqrt{2\pi n}\left(\tfrac{n}{e}\right)^n, \tag{1.14}$$

který využívá i Ludolfovo číslo π. Dosazením do (1.14) získáváme

$$1000! \approx 4.02353729203 \cdot 10^{2567}$$

a tento odhad je velmi přesný, protože platí $1000! = 4.02387260077 \cdot 10^{2567}$.

Příklad 1.2.5 Obarvěte liché a sudé vrcholy Pascalova trojúhelníku dvěma různými barvami.

Řešení: Výsledný obrazec je známý jako tzv. Sierpinskiho trojúhelník a má fraktální strukturu. Pascalův trojúhelník obsahuje v n-té řádce na k-té pozici hodnotu kombinačního čísla

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}, \qquad n \in \mathbb{N}_0, k \in \mathbb{N}_0 : k \leq n.$$

Řádky n kreslíme shora dolů a pozice k v řádce zleva doprava. Uvažujeme i nultou řádku $n = 0$ obsahující špičku trojúhelníka pro $k = 0$. Kraje trojúhelníku mají vždy hodnotu 1 (tedy lichou) neboť $\binom{n}{0} = \binom{n}{n} = 1$. K určení parity (lichosti nebo sudosti) čísla $\binom{n}{k}$ stačí znát parity čísel v řádku výše protože platí

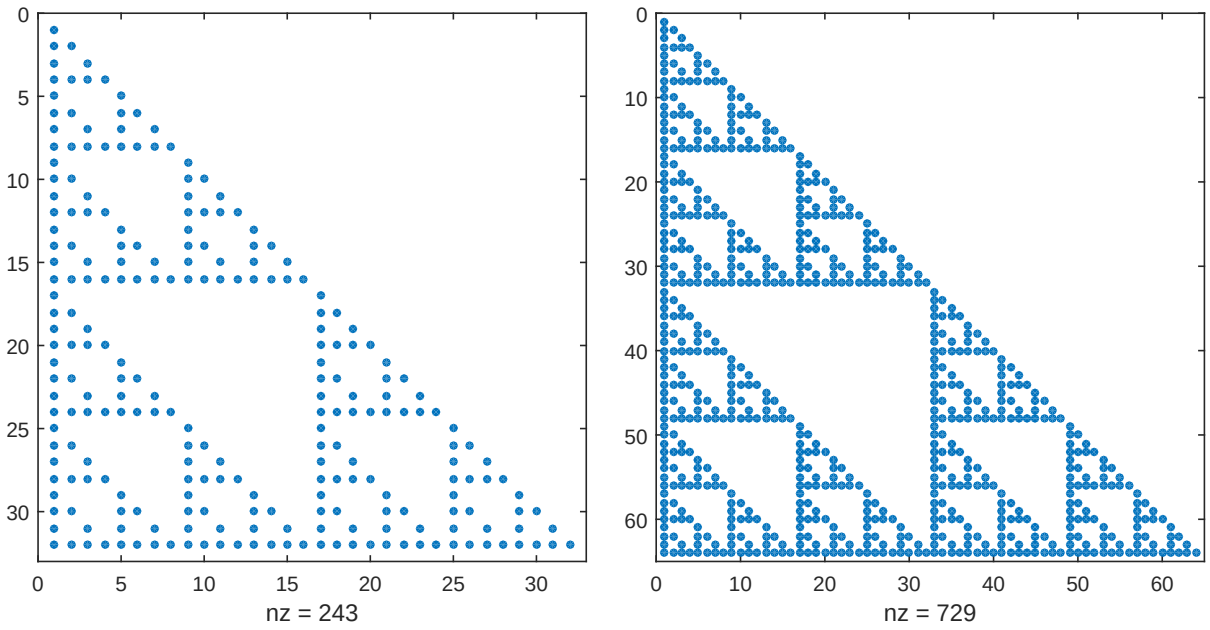
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

Počítat dovede vykreslit trojúhelník pomocí:

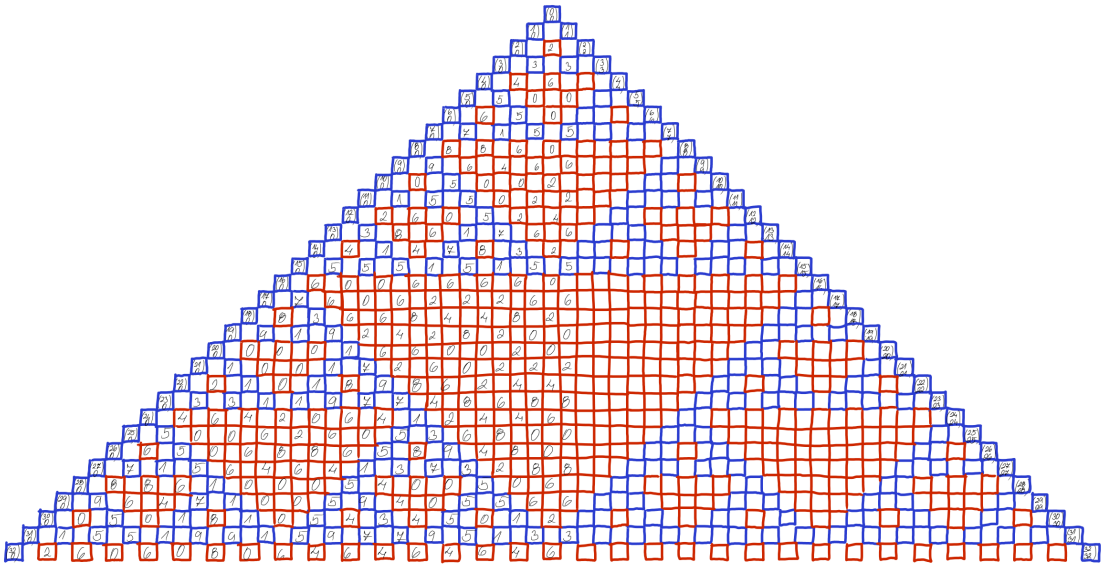
Kód - Matlab 1.6: Sierpinskiho trojúhelník.

```
n=32;                                   % pocet radek trojuhelnika
p=1;
A=zeros(n);
for i=1:n;
    A(i,1:i)=p;
    p=mod([p 0]+[0 p],2);
end;
spy(A)                                  % vykresleni
```

Trojúhelník je uložen v dolní trojúhelníkové části čtvercové matici A velikosti n a jeho obrázek odpovídá vizualizaci lichých parit čísel $\binom{n}{k}$ pomocí příkazu „spy“. Výsledná řešení jsou k dispozici na Obrázku 1.1 a jedno vlastní studentské řešení na Obrázku 1.2.



Obrázek 1.1: Sierpinskiho trojúhelník s 32 (vlevo) a 64 (vpravo) řádky. Hodnota nz uvádí počet modrých, zde tedy lichých prvků.



Obrázek 1.2: Sierpinskiho trojúhelník s 32 řádky od studentky Karolíny Wolfové.

Poznámka k příkladu

Příkaz „spy“ obecněji zobrazuje nenulové prvky tzv. řídké matice, tj. matice, která obsahuje velké množství nulových prvků. Takovými jsou např. matice sousednosti tzv. řídkých grafů nebo matice tuhosti a hmotnosti při řešení parciálních diferenciálních rovnic metodou konečných prvků.

Úloha k samostatnému řešení

Ukažte rovnost

$$\sum_{k=0}^{l} \binom{n}{k}\binom{m}{l-k} = \binom{n+m}{l}, \qquad 0 \le l \le m+n.$$

Nápověda: Rozepiště rovnost

$$(1+x)^n(1+x)^m = (1+x)^{m+n}, \qquad x \in \mathbb{R}$$

pomocí binomické věty.

Příklad 1.2.6 Kolika a jakými způsoby lze rozdělit 10 mincí do 4 skupin?

Řešení: Na řádce níže je uveden příklad takového rozdělení, ve formě grafické a v ekvivalentním součtovém tvaru:

$$oo\,|\,oooo\,|\,ooo\,|\,o \qquad \Longleftrightarrow \qquad 2+4+3+1=10.$$

Mince zobrazujeme symbolem o a oddělovače symbolem $|$. Zde jsme deset mincí rozdělili do čtyř skupin po 2, 4, 3 a 1 minci. Pokud by se levý nebo pravý oddělovač umístil úplně na kraj, znamená to, že levá nebo pravá skupina neobsahuje žádnou minci. Řádka níže ukazuje příklad takového rozdělení:

$$|\,oooooo\,|\,oooo\,| \qquad \Longleftrightarrow \qquad 0+6+4+0=10.$$

Uvědomme si, že obecně nám k rozhrnutí všech mincí do 4 skupin stačí jen 3 oddělovače. Máme tedy celkem nakreslených $10+4-1=13$ symbolů a 3 oddělovače můžeme umístit na jakoukoliv ze 13 pozic. Mince se potom mezi oddělovače automaticky poskládají. Počet všech možností lze tedy vyjádřit kombinačními čísly

$$\binom{10+4-1}{4-1} = \binom{13}{3} = \binom{13}{10},$$

která jsou rovna hodnotě 286. Pomocí programu níže vypíšeme všechna možná rozdělení mincí:

Kód - Matlab 1.7: Rozdělení mincí.

```
n=4;                                                    %pocet skupin
k=10;                                                   %pocet minci
items=nchoosek(1:n+k-1,n-1);
numb=size(items,1);
dist=diff([zeros(numb,1), items, (n+k)*ones(numb,1)],1,2)-1
```

Ta jsou uložena v matici `dist`, která je velikosti 286×4. Několik prvních řádků matice vypisujeme:

```
0     0     0    10
0     0     1     9
0     0     2     8
0     0     3     7
0     0     4     6
0     0     5     5
0     0     6     4
0     0     7     3
0     0     8     2
0     0     9     1
0     0    10     0
0     1     0     9
0     1     1     8
0     1     2     7
```

Řádky jsou automaticky uspořádány a nejdříve se zaplňují skupiny odzadu. To je dáno naší konkrétní implementací.

Poznámka k příkladu

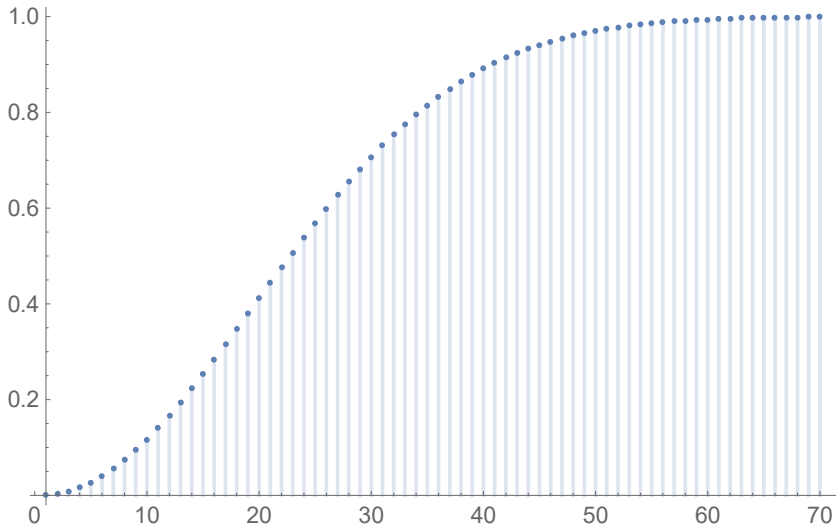
Z hlediska kombinatoriky se jedná o počet všech 10ti členných kombinací s opakováním ze 4 prvků označovaným ve středoškolské matematice jako $C'_{10}(4)$ a pro který obecně platí

$$C'_k(n) = \binom{n+k-1}{n-1} = \binom{n+k-1}{k}.$$

Příklad 1.2.7 Jaká je pravděpodobnost p jevu, že ve třídě o n-žácích mají alespoň dva žáci narozeniny ve stejný den.

Řešení: Jednodušší je spočítat pravděpodobnost opačného jevu $\tilde{p} = 1 - p$, tzn. jevu, že všichni uvažovaní žáci budou mít narozeniny v různý den. Ta je vyjádřena číslem (víte proč?)

$$\tilde{p} = \frac{365 \cdot 364 \cdots (365-n+1)}{365^n} = \frac{365!}{365^n (365-n)!}.$$



Obrázek 1.3: Závislost pravděpodobnosti p na počtu žáků n.

Zde jsme uvažovali nepřestupný rok s 365 dny. Proto je pravděpodobnost p rovna

$$p = 1 - \tilde{p} = 1 - \frac{365!}{365^n (365-n)!}.$$

Zkusme si na počítači vykreslit p jako funkci počtu žáků n pomocí:

Kód - Mathematica 1.8: narozeninový problém.

```
P[n_, m_] := 1 - m!/((m - n)!*m^n)
DiscretePlot[P[n, 365], {n, 70}]
```

Výsledný graf je ukázán na Obrázku 1.3. Je zřejmé (rozmyslete si proč), že platí

$$p(1) = 0, \quad p(366) = 1$$

a že p je rostoucí funkce v n (čím více je žáků, tím vyšší je pravděpodobnost, že někteří mají narozeniny ve stejný den). Konkrétní hodnoty pravděpodobností lze dodatečně vypočítat pomocí:

Kód - Mathematica 1.9: narozeninový problém - konkrétní hodnoty.

```
P[23, 365] // N
P[30, 365] // N
P[40, 365] // N
```

a získat hodnoty

$$p(23) \approx 0.507297, \quad p(30) \approx 0.706316, \quad p(40) \approx 0.891232.$$

Tedy již při počtu 23 žáků je pravděpodobnost asi 50%, při počtu 30 žáků asi 70% a při počtu 40 žáků přibližně 90%.

Poznámka k příkladu

Tato úloha je známá jako narozeninový problém. Při hodinách předmětu Diskrétní matematika na PřF JČU bývá přítomno kolem 30-60 studentů a zatím se vždy podařilo najít dokonce několik různých dvojic studentů, které mají narozeniny ve stejný den.

Úloha k samostatnému řešení

V komoře je rozbité světlo a naslepo si vybíráme ponožky různých 4 barev. Pokud si chceme být jisti, že vytáhneme alespoň dvě bílé ponožky, musíme jich z komory přinést 28. Abychom měli takovou jistotu pro šedé ponožky, musíme je přinést také 28, pro černé ponožky stačí 26 a pro modré ponožky 34. Kolik je celkem v komoře ponožek?
Poznámka: tento příklad je z matematické olympiády pro střední školy a jedno možné řešení vede na soustavu lineárních rovnic.

Úloha k samostatnému řešení

V této úloze hledáme optimální rozložení zápasů na turnaji např. v nohejbalu hraných na jednom hřišti. Jde o vlastní modifikaci tzv. Bergerových tabulek používaných pro rozlosování sportovních zápasů. Například pro 5 družstev doporučujeme hrát pořadí zápasů

$$2-5, \quad 3-4, \quad 1-2, \quad 5-3, \quad 4-1, \quad 3-2, \quad 5-4, \quad 1-3, \quad 4-2, \quad 5-1,$$

tedy je vidět, že se číslo žádného družstva neopakuje ve dvou následujících dvojicích zápasů. Více informací naleznete na webu (odkaz zde).
Vygenerujte vlastní tabulky pro případ 9 nebo 10 družstev první optimalizační úlohy (tzv. modré tabulky). Uveďte zároveň také minimální a maximální počet čekání na ostatní zápasy.
Poznámka: autorovi se zatím (snad) povedlo nalézt optimální vylosování pro 5-8 týmů, které již prakticky používají organizátoři několika nohejbalových turnajů na Plzeňsku.

1.3 Princip inkluze a exkluze

Příklad 1.3.8 Kolik zbude čísel mezi čísly 1 až 1000 po vyškrtání násobků prvních tří prvočísel?

Řešení: Označme A jako množinu všech uvažovaných čísel

$$A = \{1, 2, \ldots, 999, 1000\}$$

a A_2, A_3, A_5 jako množiny násobků jednotlivých prvočísel (ta jsou 2, 3, 5)

$$\begin{aligned} A_2 &= \{2, 4, \ldots, 998, 1000\}, \\ A_3 &= \{3, 6, \ldots, 996, 999\}, \\ A_5 &= \{5, 10, \ldots, 995, 1000\}. \end{aligned}$$

Protože množiny A_2, A_3, A_5 nemají prázdný průnik, velikost (známá také jako kardinalita) jejich sjednocení není rovna součtu velikostí jednotlivých množin a platí

$$|A_2 \cup A_3 \cup A_5| < |A_2| + |A_3| + |A_5|.$$

K výpočtu je třeba použít princip inkluze a exkluze, který říká

$$\begin{aligned} |A_2 \cup A_3 \cup A_5| = |A_2| + |A_3| + |A_5| \\ - |A_2 \cap A_3| - |A_2 \cap A_5| - |A_3 \cap A_5| \\ + |A_2 \cap A_3 \cap A_5|. \end{aligned} \tag{1.15}$$

Výrazů na pravé straně je v tomto případě 7, což odpovídá hodnotě $7 = 2^3 - 1$. Po dosazení do (1.15) získáme

$$|A_2 \cup A_3 \cup A_5| = 500 + 333 + 200 - 166 - 100 - 66 + 33 = 734$$

vyškrtaných čísel, protože

$$\begin{aligned}
|A_2| &= \lfloor * \rfloor \frac{1000}{2} = 500, \\
|A_3| &= \lfloor * \rfloor \frac{1000}{3} = 333, \\
|A_5| &= \lfloor * \rfloor \frac{1000}{5} = 200, \\
|A_2 \cap A_3| &= \lfloor * \rfloor \frac{1000}{6} = 166, \\
|A_2 \cap A_5| &= \lfloor * \rfloor \frac{1000}{10} = 100, \\
|A_3 \cap A_5| &= \lfloor * \rfloor \frac{1000}{15} = 66, \\
|A_2 \cap A_3 \cap A_5| &= \lfloor * \rfloor \frac{1000}{30} = 33.
\end{aligned}$$

Proto zbude jen

$$|A| - |A_2 \cup A_3 \cup A_5| = 1000 - 734 = 266$$

nevyškrtaných čísel. K vygenerování výsledků můžeme alternativně využít počítač:

Kód - Mathematica 1.10: prosívání prvočísel.

```
n = 1000;
np = 3;
P = Table[Prime[i], {i, np}];
S = Delete[Subsets[P], 1];
M = Apply[Times, S, {1}];
R = Table[Floor[n/M[[i]]], {i, Length[M]}]
L = Length /@ S;
suma = Sum[R[[i]]*(-1)^(L[[i]] + 1), {i, Length[M]}]
nv = n - suma
```

Poznámka k příkladu

Mezi nevyškrtanými čísly je stále číslo 1, které dodatečně také vyškrtneme a zůstane nám 265 nevyškrtaných čísel, která mohou, ale nemusí být dalšími prvočísly. Příkladem takového čísla je 77, které není násobkem 2, 3 a 5, ale je násobkem dalšího prvočísla 7. Spolu se třemi již vyškrtnutými prvočísly můžeme tedy odhadnout počet prvočísel shora jako

$$|A| - |A_2 \cup A_3 \cup A_5| + 2 = 268.$$

Lze ukázat, že první nevyškrtnuté číslo v uspořádané množině A je vždy prvočíslo, zde tedy číslo 7. Nám zde vlastně princip inkluze a exkluze prvočísla negeneruje, ale jenom je počítá.

Poznámka k příkladu

Počítačový kód výše nám dovoluje změnit hodnotu `np` vyjadřující počet vyškrtávaných prvočísel. Dále po číslech 2, 3, 5 vyškrtáváme další prvočísla

$$7, 11, 13, 17, 19, 23.$$

Připomínáme, že k výpočtu velikostí sjednocení obecně n množin v zobecnění formule (1.15) je nutno vyjádřit $2^n - 1$ průniků nejrůznějších množin, přičemž mnohé z nich představují prázdné množiny. Získáváme tím další odhady shora na celkový počet prvočísel:

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7| + 3 = 231,$$

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7 \cup A_{11}| + 4 = 211,$$

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7 \cup A_{11} \cup A_{13}| + 5 = 195,$$

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7 \cup A_{11} \cup A_{13} \cup A_{17}| + 6 = 185,$$

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7 \cup A_{11} \cup A_{13} \cup A_{17} \cup A_{19}| + 7 = 177,$$

$$|A| - |A_2 \cup A_3 \cup A_5 \cup A_7 \cup A_{11} \cup A_{13} \cup A_{17} \cup A_{19} \cup A_{23}| + 8 = 171.$$

Pokud budeme vyškrtávat násobky dalších prvočísel, počet nevyškrtaných čísel se bude ještě dále snižovat, až se úplně zastaví (zjistěte při zpracování jakého prvočísla) na hodnotě 168, které představuje celkový počet prvočísel mezi 1 až 1000. Praktické prosívání čísel je známé jako tzv. Eratosthenovo síto.

Úloha k samostatnému řešení

Kolik existuje přirozených čísel menších než $n = 1000$, která jsou s n nesoudělná?

Úloha k samostatnému řešení

Hledáme počet permutací bez pevného bodu z čísel $1, ..., n$ studovaných např. v úloze o šatnářce. Spočtěte počet permutací bez pevného bodu pro $n = 5$ (v ruce) nebo pro $n = 10$ (pomocí počítače). Srovnejte Vaše výsledky s odkazem zde.

Úloha k samostatnému řešení

Určete počet způsobů, jak lze obarvit políčka mřížky 3 × 3 červeně, žlutě a modře tak, že každá barva je použita právě třikrát a navíc žádný řádek ani sloupec není jednobarevný. (Obarvení lišící se pouze otočením považujeme za různá.) Ti, kteří použijí počítač, mohou najít i počet způsobů v případě mřížek 4 x 4 nebo 5 x 5 v případě 4 nebo 5 barev.
Poznámka: Úloha je převzata z Matematického korespondenčního semináře.

1.4 Základní grafové pojmy

Příklad 1.4.9 Zjistěte, zda je posloupnost

$$\{6, 6, 5, 4, 3, 3, 3, 3, 2, 1, 0\}$$

stupňovou posloupností nějakého grafu.

Řešení: Počet prvků posloupnosti představuje počet vrcholů grafu, v tomto případě je jich 11. Použijeme (opakovaně) větu Havel-Hakimi. K tomu je nutné posloupnost uspořádat od největšího čísla k nejmenšímu, tj.

$$\{6,6,5,4,3,3,3,3,2,1,0\}$$

vyjmout z ní největší číslo 6 a od 6 následujících čísel odečíst hodnotu 1. Tím získáme kratší posloupnost s 10 vrcholy tvaru $\{5,4,3,2,2,2,3,2,1,0\}$, kterou přeuspořádáme jako

$$\{5,4,3,3,2,2,2,2,1,0\}$$

a postup výše opakujeme. Dostáváme postupně kratší přeuspořádané posloupnosti ve tvaru

$$\{3,2,2,2,2,1,1,1,0\},$$

$$\{2,1,1,1,1,1,1,0\},$$

$$\{1,1,1,1,0,0,0\},$$

$$\{1,1,0,0,0,0\},$$

$$\{0,0,0,0,0\}.$$

Poslední posloupnost pěti nul odpovídá zřejmě grafu s pěti izolovanými vrcholy. Proto podle věty Havel-Hakimi původní posloupnost v zadání příkladu výše je stupňovou posloupností nějakého grafu. V počítači můžeme vygenerovat jednotlivé posloupnosti podle programu:

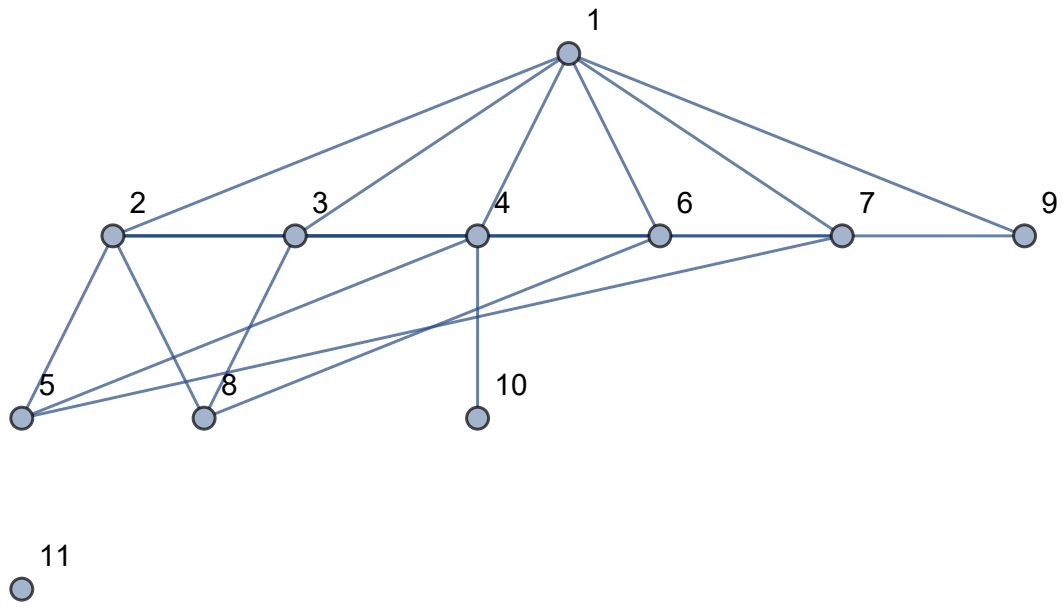
Kód - Matlab 1.11: Ověření stupňové posloupnosti.

```
%seq=[1, 2, 3, 4, 4];                          % jedna vstupni posloupnost
seq=[6, 6, 5, 4, 3, 3, 3, 2, 1, 0];            % druha vstupni posloupnost
while ~isempty(seq)
    seq=sort(seq,'descend');                   % serazeni
    fprintf('%d  ',seq); fprintf('\n',seq)     % vypis
    p=seq(1);
    if (p<0) || (numel(seq)<p)
        fprintf('Posloupnost neni grafova. \n'); return;
    end
    if (p==0) && (numel(seq)==1)
        fprintf('Posloupnost je grafova. \n'); return;
    end
    seq(1)=[];                                 % vymazani prvniho prvku
    seq(1:p)=seq(1:p)-1;                       % odecteni 1 od dalsi clenu
end
```

Věta sice nekonstruuje graf odpovídající původní stupňové posloupnosti (těch je také často více), ten ale dovedeme sestrojit/nakreslit na papír postupnou konstrukcí zpětným chodem od kratších posloupností k delším (rozmyslete si jak). Příklad jednoho takového grafu je uveden na Obrázku 1.4:

Poznámka k příkladu

Zakomentováním druhé řádky programu výše se zpracuje posloupnost $\{1,2,3,4,4\}$. Ta není stupňovou posloupností nějakého grafu, přestože součet jeho stupňů vrcholů má sudý.



Obrázek 1.4: Příklad grafu se stupňovou posloupností $\{6, 6, 5, 4, 3, 3, 3, 3, 2, 1, 0\}$. Čísla vrcholů odpovídají jejich pořadí v posloupnosti.

Příklad 1.4.10 Najděte všechny neizomorfní grafy na 4 vrcholech.

Řešení: Grafy budeme zpracovávat ve skupinách rozdělených podle počtu hran. Potom budou grafy v různých skupinách zaručeně neizomorfní a budeme muset ověřit, jestli jsou také neizomorfní v rámci jedné skupiny. Graf na 4 vrcholech obsahuje maximálně

$$\binom{4}{2} = \frac{4!}{2!2!} = 6$$

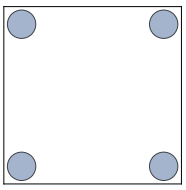
hran, tedy budeme muset hledat v 7 skupinách obsahujících 0-6 hran.
Ve skupinách podle 0, 1, 2, 3 hran získáváme postupně 1+1+2+3=7 vzájemně neizomorfních grafů, které jsou zobrazeny na Obrázku 1.5.
Další skupiny není nutno prohledávat, neboť počet neizomorfních grafů ve skupinách pro 4, 5, 6 hran odpovídá ze symetrie (proč?) počtu pro 2, 1, 0 hran, tj. získáváme dalších 2, 1, 1 neizomorfních grafů. Celkem tedy existuje

$$1 + 1 + 2 + 3 + 2 + 1 + 1 = 11$$

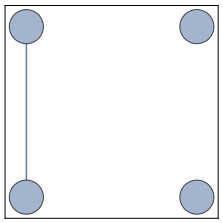
vzájemné neizomorfních grafů na 4 vrcholech. K vykreslení grafů jsme použili následující kód.

Kód - Mathematica 1.12: vykreslení neizomorfních grafů.

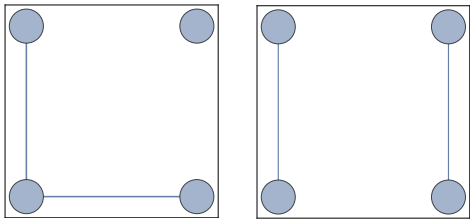
```
i = {1, 2, 3, 4};
v = Table[{Cos[Pi/4 + k*(Pi/2)], Sin[Pi/4 + k*Pi]}, {k, 1, 4}];
opts={VertexCoordinates->v, VertexSize->Medium, EdgeStyle->Thick, Frame->True
    };
Graph[i, {}, Evaluate[opts]]
Graph[i, {1 <-> 2}, Evaluate[opts]]
Graph[i, {1 <-> 2, 1 <-> 3}, Evaluate[opts]]
Graph[i, {1 <-> 2, 3 <-> 4}, Evaluate[opts]]
Graph[i, {1 <-> 2, 1 <-> 3, 3 <-> 4}, Evaluate[opts]]
Graph[i, {1 <-> 2, 1 <-> 3, 3 <-> 2}, Evaluate[opts]]
Graph[i, {1 <-> 2, 1 <-> 3, 1 <-> 4}, Evaluate[opts]]
```



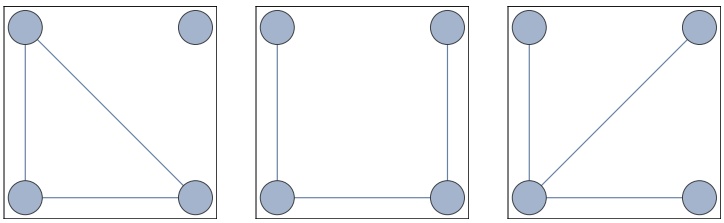
1 neizomorfní graf s 0 hranami



1 neizomorfní graf s 1 hranou



2 neizomorfní grafy s 2 hranami



3 neizomorfní grafy s 3 hranami

Obrázek 1.5: Vzájemně neizomorfní grafy na 4 vrcholech pro 0, 1, 2, 3 hran. Nejsou vykresleny další neizomorfní grafy pro 4, 5, 6 hran. S nimi máme celkem 11 neizomorfních grafů na 4 vrcholech.

Poznámka k příkladu

Program 1.12 neizomorfní grafy pouze vykresluje, ale neslouží k jejich generování. Pro dva grafy, u kterých rozhodujeme o jejich izomorfismu bychom měli obecně projít všechna vzájemné jednoznačná zobrazení vrcholů jednoho grafu do druhého, tedy permutace vrcholů. Pokud by některá permutace zároveň zachovala hrany jednotlivých grafů, tak jde o grafy izomorfní. Praktické ověření je tedy výpočetně náročné, pro grafy s menším počtem vrcholů realizovatelné. Příklad studentského řešení pro 4 vrcholy je uveden na Obrázku 1.6.

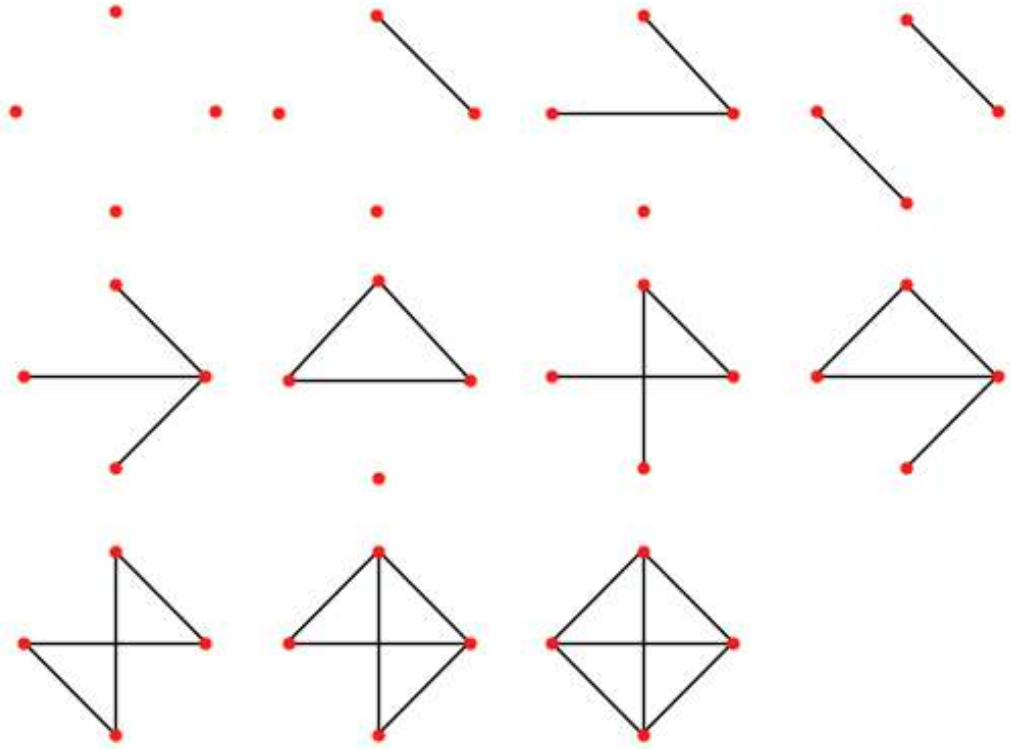
Poznámka k příkladu

Mathematica má jednotlivé neizomorfní grafy uložené a lze je vykreslit pomocí

Kód - Mathematica 1.13: vygenerování neizomorfních grafů.

```
1 GraphData /@ GraphData[4]
```

Grafický výstup zde pro úsporu místa neposkytujeme.



Obrázek 1.6: Neizomorfní grafy na 4 vrcholech, řešení studenta Filipa Nachtmana v jazyce Python.

Úloha k samostatnému řešení

1. Najděte sami ručně a vykreslete všechny neizomorfní grafy na 5 vrcholech ve skupinách podle hran. Kolik jich je?
2. Zkuste vytvořit vlastní počítačový program, který dovede najít všechny pro 5 a více vrcholů.

1.5 Prohledávání grafů

Příklad 1.5.11 Neorientovaný graf je dán výčtem hran:

$$1-2,\ 1-3,\ 3-4,\ 3-5,\ 5-6,\ 6-7,\ 5-8,\ 2-9,\ 6-10,\ 7-11,\ 5-12.$$

Rozhodněte, jestli je daný graf souvislý prohledáváním grafu

a) do šířky,

b) do hloubky.

Vypište přesné pořadí procházených vrcholů s tím, že začínáme hledání u vrcholu s číslem 1 a zpracováváme přednostně vrcholy s nižšími čísly.

Řešení: Nejnižší vrchol je 1 a ním jsou spojeny vrcholy $2, 3$. Můžeme si tedy vytvořit posloupnost navštívených vrcholů

$$\boxed{1}, 2, 3$$

a hrany $1-2$ a $1-3$ z výčtu hran výše vyškrtnout a dále je tedy v prohledávání již nevyužijeme. Pomocí zaobleného čtverce označujeme číslo vrcholu, ke kterému jsem již hledali sousední vrcholy.

Prohledávání do šířky: V tomto případě si vybereme další neoznačený vrchol, tj. vrchol číslo 2 a k němu hledáme jeho sousedy, kteří však v seznamu ještě nefigurují. Tím získáme další posloupnost

$$\boxed{1}, \boxed{2}, 3, 9$$

a hranu $2-9$ si můžeme také vyškrtnout. Postup opakujeme a získáme posloupnosti

$$\boxed{1}, \boxed{2}, \boxed{3}, 9, 4, 5$$

odstraněním hrany $3-5$,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, 4, 5$$

neodstraněním žádné hrany,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, 5$$

neodstraněním žádné hrany,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, 6, 8, 12$$

odstraněním hran $5-6, 5-8, 5-12$,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, \boxed{6}, 8, 12, 7, 10$$

odstraněním hran $6-7, 6-10$,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, \boxed{6}, \boxed{8}, 12, 7, 10$$

neodstraněním žádné hrany,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, \boxed{6}, \boxed{8}, \boxed{12}, 7, 10$$

neodstraněním žádné hrany,

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, \boxed{6}, \boxed{8}, \boxed{12}, \boxed{7}, 10, 11$$

odstraněním hrany $7-11$. Nyní již z původního výčtu hran žádné hrany nezůstaly a proto je graf souvislý a výsledkem prohledávání do šířky je posloupnost vrcholů

$$\boxed{1}, \boxed{2}, \boxed{3}, \boxed{9}, \boxed{4}, \boxed{5}, \boxed{6}, \boxed{8}, \boxed{12}, \boxed{7}, \boxed{10}, \boxed{11}.$$

Prohledávání do hloubky: To funguje trochu jinak a vždy expanduje prvního následníka každého vrcholu, pokud ho ještě nenavštívil. Pokud narazí na vrchol, který jíž nemá následníka, nebo všechny už byly navštíveny, vrací se k předchozímu vrcholu. To vede v tomto příkladě na posloupnost navštívených vrcholů

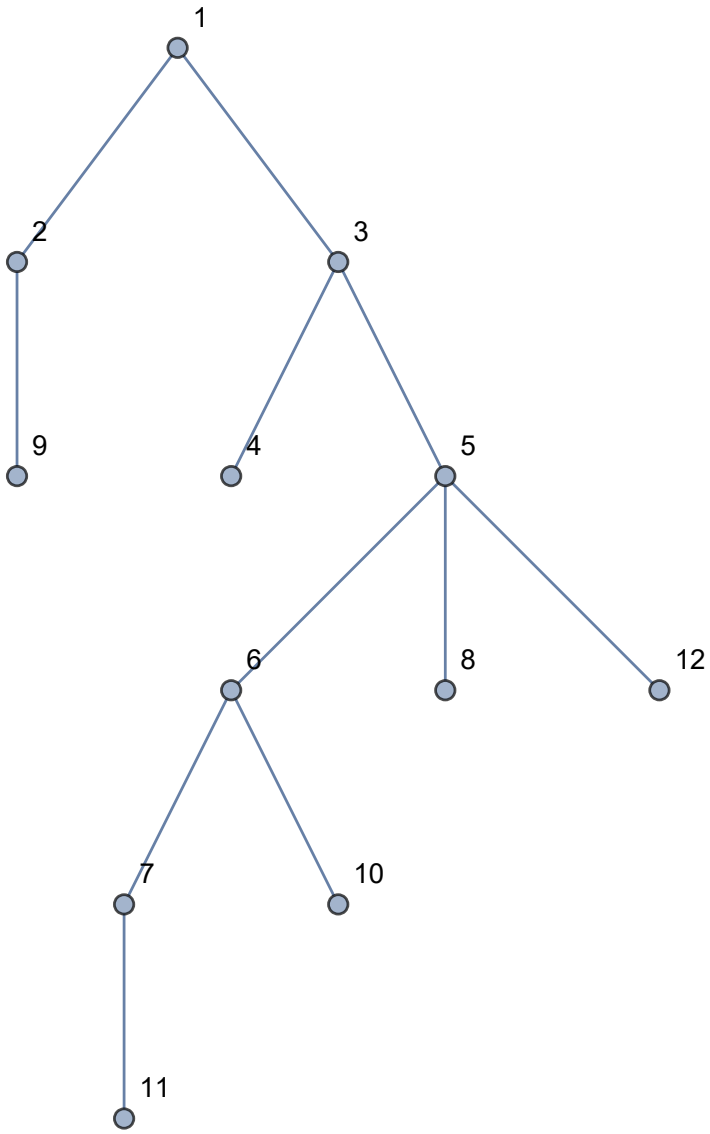
$$\boxed{1}, \boxed{2}, \boxed{9}, \boxed{3}, \boxed{4}, \boxed{5}, \boxed{6}, \boxed{7}, \boxed{11}, \boxed{10}, \boxed{8}, \boxed{12}.$$

Detailněji její sestavení však ukazovat nebudeme a čtenář si ho může provést sám. Počítačový systém má samozřejmě k dispozici oba prohledávací algoritmy:

Kód - Mathematica 1.14: prohledávání grafu.

```
ED ={1<->2,1<->3,3<->4,3<->5,5<->6,6<->7,5<->8,2<->9,6<->10,7<->11,5<->12}
G = Graph[ED, VertexLabels -> "Name"];
EdgeList[G]
BreadthFirstScan[G, 1, {"PrevisitVertex" -> (Print["Visiting ", #] &)}]
DepthFirstScan[G, 1, {"PrevisitVertex" -> (Print["Visiting ", #] &)}]
TreePlot[G, 1]
```

a obě výše zmíněné posloupnosti poskytne. Prohledávaný graf je vykreslen pomocí příkazu na poslední řádce a je ukázán na Obrázku 1.7.



Obrázek 1.7: Graf prohledávaný do šířky a hloubky.

Poznámka k příkladu

Graf, který jsme v příkladu výše prohledávali je strom a proto se žádný vrchol, který přidáváme do posloupností vrcholů v ní nebude opakovat. Pokud by tomu tak nebylo, například bychom ke grafu přidali další hranu 8-10 a vzniklý graf prohledávali, tak vrchol 10 nesmíme do posloupnosti přidat dvakrát.

Příklad 1.5.12 Neorientovaný graf je dán výčtem hran:

$$1-2,\ 1-3,\ 2-3,\ 1-4,\ 1-5.$$

Rozhodněte, kolik existuje sledů délky 3 z vrcholu 1 do vrcholu 1.

Řešení: Matice sousednosti daného grafu je

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Z teorie víme, že třetí mocnina matice sousednosti poskytuje veškerou informaci o počtu sledů délky 3. K jejímu sestavení potřebujeme druhou mocninu matice sousednosti

$$A^2 = \begin{pmatrix} 4 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Ta je zajímavá tím, že na diagonále obsahuje jednotlivé stupně vrcholů. Třetí mocnina matice sousednosti vychází

$$A^3 = \begin{pmatrix} 2 & 5 & 5 & 4 & 4 \\ 5 & 2 & 3 & 1 & 1 \\ 5 & 3 & 2 & 1 & 1 \\ 4 & 1 & 1 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Na pozici $(1,1)$ má hodnotu 2 a proto existují dva sledy délky 3 z vrcholu 1 do vrcholu 1. V grafu v Obrázku 1.8 je jednoduše vidět, že to jsou sledy

$$(1,2,3,1), \quad (1,3,2,1).$$

Graf a jednotlivé mocniny matice sousednosti jsme jednoduše získali pomocí kódu:

Kód - Mathematica 1.15: počítaní sledů.

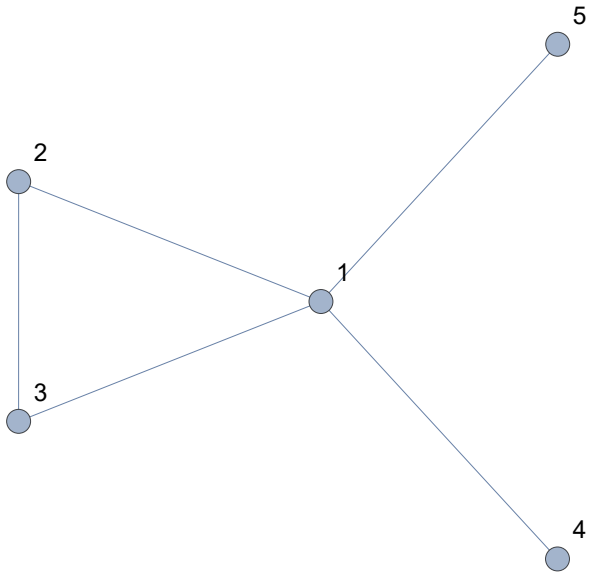
```
p = {{1, 2}, {1, 3}, {2, 3}, {1, 4}, {1, 5}}
B = SparseArray[p -> Table[2, {i, 1, Max[p]}], {Max[p], Max[p]}];
A := (B + Transpose[B])/2
Table[Print[Superscript["A", i], "=", MatrixForm[MatrixPower[A, i]]], {i,
   1, First[Dimensions[A]]}];
g = AdjacencyGraph[A, VertexSize -> Small, VertexLabels -> "Name",
  VertexLabelStyle -> Directive[15]]
```

Poznámka k příkladu

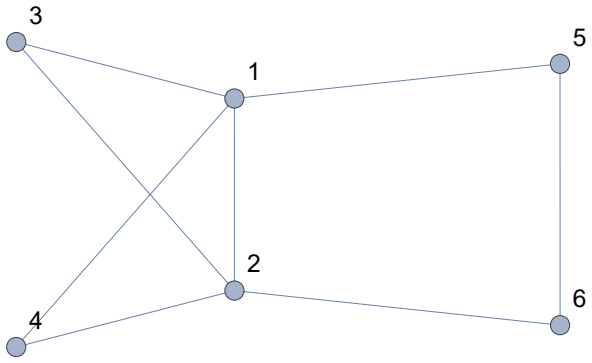
Lze ukázat, že součet diagonálních prvků (známý v lineární algebře jako stopa a označovaný symbolem tr) třetí mocniny matice dále vydělený číslem 6 udává počet trojúhelníků v daném grafu. Zde vychází

$$\frac{\operatorname{tr} A^3}{6} = 1$$

a proto má graf jeden trojúhelník. Podle Obrázku 1.8 je daný vrcholy $1,2,3$ a přesně na stejných pozicích na diagonále matice A^3 máme nenulová čísla.



Obrázek 1.8: Graf, u kterého počítáme sledy.



Obrázek 1.9: Graf, u kterého hledáme Eulerovský cyklus.

Příklad 1.5.13 Neorientovaný graf je dán na Obrázku 1.9. Najděte Eulerovský cyklus.

Řešení: Protože je graf souvislý a stupně všech vrcholů jsou sudé (mají buď stupně 2 nebo 4), je možné sestavit Eulerovský cyklus, tedy cyklus začínající v jednom vrcholů, procházející všechny hrany a končící ve stejném vrcholu. Příkladem je tah daný pořadí vrcholů

$$1 - 2 - 6 - 5 - 1 - 4 - 3 - 1.$$

Počítačový systém nalezne Eulerovský cyklus pomocí:

Kód - Mathematica 1.16: prohledávání grafu.

```
ED = { 1<->2 , 1<->3 , 1<->4 , 1<->5 , 2<->3 , 2<->4 , 2<->6 , 5<->6};
G = Graph[ED, VertexSize -> Small, VertexLabels -> "Name",
   VertexLabelStyle -> Directive[15]]
c = FindEulerianCycle[G]
Table[HighlightGraph[G, Part[First[c], 1 ;; i]], {i, Length[First[c]]}]
FindEulerianCycle[G,All]
```

a poslední příkaz vypíše všechny možné cykly. Těch je celkem 6 (zkuste je nalézt).

Příklad 1.5.14 Orientovaný graf je dán výčtem hran:

$$1 \to 2,\ 1 \to 4,\ 2 \to 3,\ 3 \to 1,\ 3 \to 4,\ 3 \to 5,\ 4 \to 3,\ 4 \to 5,\ 5 \to 1,\ 5 \to 2,\ 5 \to 4$$

a jejich váhy jsou

$$5,\ 2,\ 2,\ 3,\ 5,\ 7,\ 4,\ 1,\ 1,\ 3,\ 3.$$

Vypočtěte úplnou distanční matici grafu.

Řešení: K výpočtu použijeme Floydův–Warshallův algoritmus. K jeho použití sestavíme matici

$$M_0 = \begin{pmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & \infty & 0 & 5 & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & \infty & 3 & 0 \end{pmatrix},$$

která má na pozici hran jejich váhy, na diagonále nuly a všude jinde hodnotu ∞. Je prvotním přiblížením distanční matice a hodnoty ∞ říkají, že jsme zatím žádnou nejkratší cestu mezi odpovídajícími vrcholy nenašli. Nenulové hodnoty matice se mohou v průběhu algoritmu snižovat.
V první iteraci zjišťujeme, zda se hodnoty v matici M_0 nezlepší, půjdeme-li od vstupního vrcholu do výstupního vrcholu přes vrchol číslo 1. V matici M_0 tedy pracujeme jen s čísly v prvním sloupci a první řádce, zvýrazněnými modrou barvou. Na ní se původní hodnota ∞ vylepšila na hodnotu 8 (pozice v matici, na kterých dojde k vylepšení hodnoty, zvýrazňujeme červenou barvou). To je dáno tím, že z vrcholu 3 do vrcholu 2 sice neexistuje přímá hrana (na pozici (3,2) matice M_0 je hodnota ∞), ale existuje cesta přes vrchol 1, která má cenu $3 + 5 = 8$. Žádné další vylepšení zde není. Tím získáme matici

$$M_1 = \begin{pmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & 8 & 0 & 5 & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & \infty & 3 & 0 \end{pmatrix}$$

a ta se liší od matice M_0 jen na pozici $(3,2)$. Dále budeme vylepšovat hodnoty přes druhý a ostatní vrcholy. Získáváme posloupnost matic

$$M_2 = \begin{pmatrix} 0 & 5 & 7 & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & 8 & 0 & 5 & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{pmatrix},$$

$$M_3 = \begin{pmatrix} 0 & 5 & 7 & 2 & 14 \\ 5 & 0 & 2 & 7 & 9 \\ 3 & 8 & 0 & 5 & 7 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{pmatrix},$$

$$M_4 = \begin{pmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{pmatrix},$$

$$M_5 = \begin{pmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 2 & 4 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{pmatrix},$$

přičemž číslice označené červeně v jedné matici v další matici sníží hodnotu. Floydův–Warshallův algoritmus zaručuje, že poslední matice M_5 je zároveň úplnou distanční maticí našeho grafu. Počítačová realizace následuje:

Kód - Mathematica 1.17: sestavení úplné distanční matice.

```
b = Infinity;
M = {{0,5,b,2,b},{b,0,2,b,b},{3,b,0,b,7},{b,b,4,0,1},{1,3,b,b,0}};
s = First[Dimensions[M]];
P = Table[
    Which[(M[[i, j]] == 0) || (M[[i, j]] == Infinity), 0,
     M[[i, j]] > 0, i], {i, s}, {j, s}];
Print["i=", 0, ":  M=", MatrixForm[M], ", P=", MatrixForm[P]];
For[i = 1, i <= s, i++,
 For[j = 1, j <= s, j++,
  For[k = 1, k <= s, k++, n1 = M[[j, i]] + M[[i, k]];
   n2 = M[[j, k]];
   If[n1 < n2, P[[j, k]] = P[[i, k]]];
   If[n1 < n2, nMin = n1, nMin = n2];
  M[[j, k]] = nMin;]];
Print["i=", i, ":  M=", MatrixForm[M], ", P=", MatrixForm[P];]
```

Všimněme si, že z algoritmického hlediska představuje Floydův–Warshallův algoritmus trojitý cyklus a to konkrétně přes vstupní vrchol, přes výstupní vrchol a přes pomocný vrchol. Prakticky není nutné používat hodnoty ∞, ale stačí je nahradit dostatečně velkým číslem (promyslete si jakým).

Příklad 1.5.15 Pro graf uvedený na Obrázku 1.10 nalezněte minimální kostru.

Řešení: Postupujeme podle Jarníkova algoritmu, známého v zahraniční literatuře jako algoritmu Primova. Nalezneme hrany s nejnižšími váhami a ty přidáváme do kostry za podmínky, že nesmí dojít k zacyklení. Do výsledné kostry tedy určitě přidáme:

a) všech 7 hran s váhou 1:

$$1-2,\ 3-7,\ 5-6,\ 9-13,\ 14-15,\ 15-19,\ 16-20,$$

b) všechny 3 hrany s váhou 2:

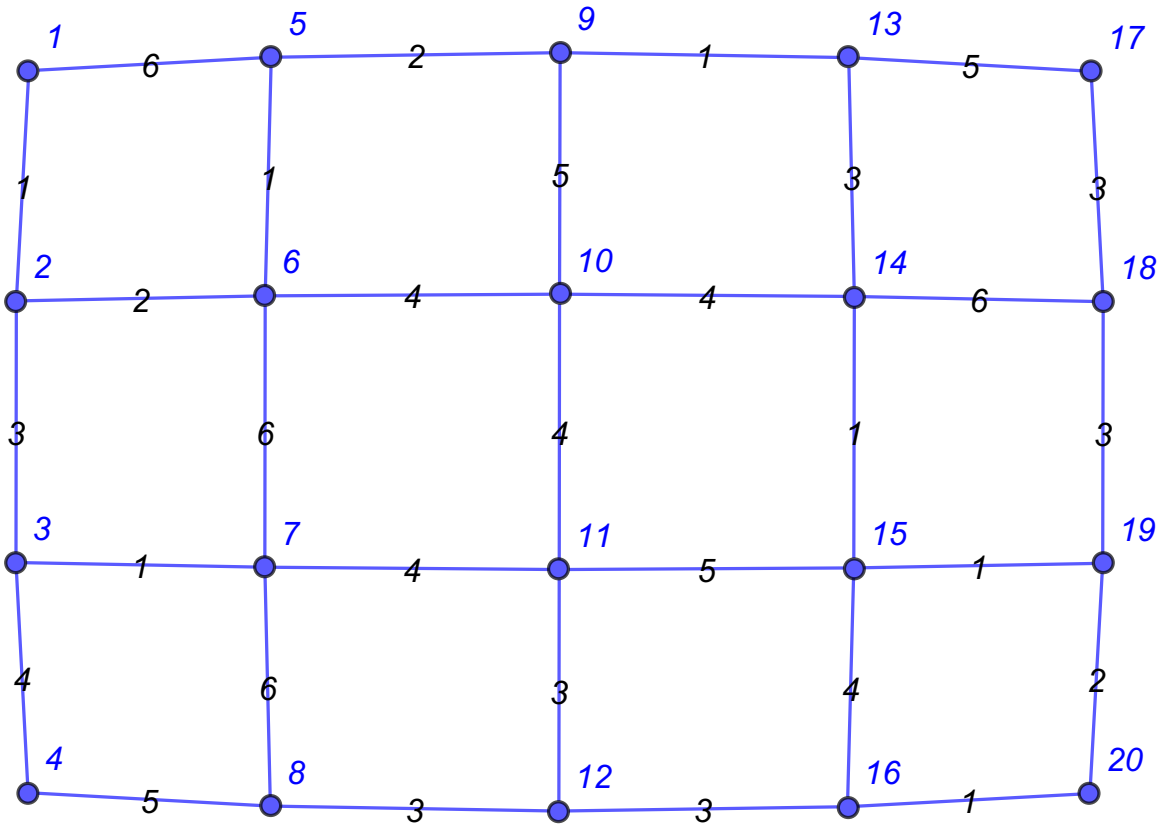
$$2-6,\ 5-9,\ 19-20,$$

c) všech 7 hran s váhou 3:

$$2-3,\ 8-12,\ 11-12,\ 12-16,\ 13-14,\ 17-18,\ 18-19.$$

Zatím jsme našli 17 hran. Víme, že celkový počet hran ve výsledné kostře vytvořené z grafu s 20 vrcholy musí být roven číslu 19. Pokusíme se tedy najít

d) další 2 hrany s váhou 4 vybrané z hran: $3-4,\ 6-10,\ 10-11,\ 10-14$.



Obrázek 1.10: Graf, ke kterému hledáme minimální kostru.

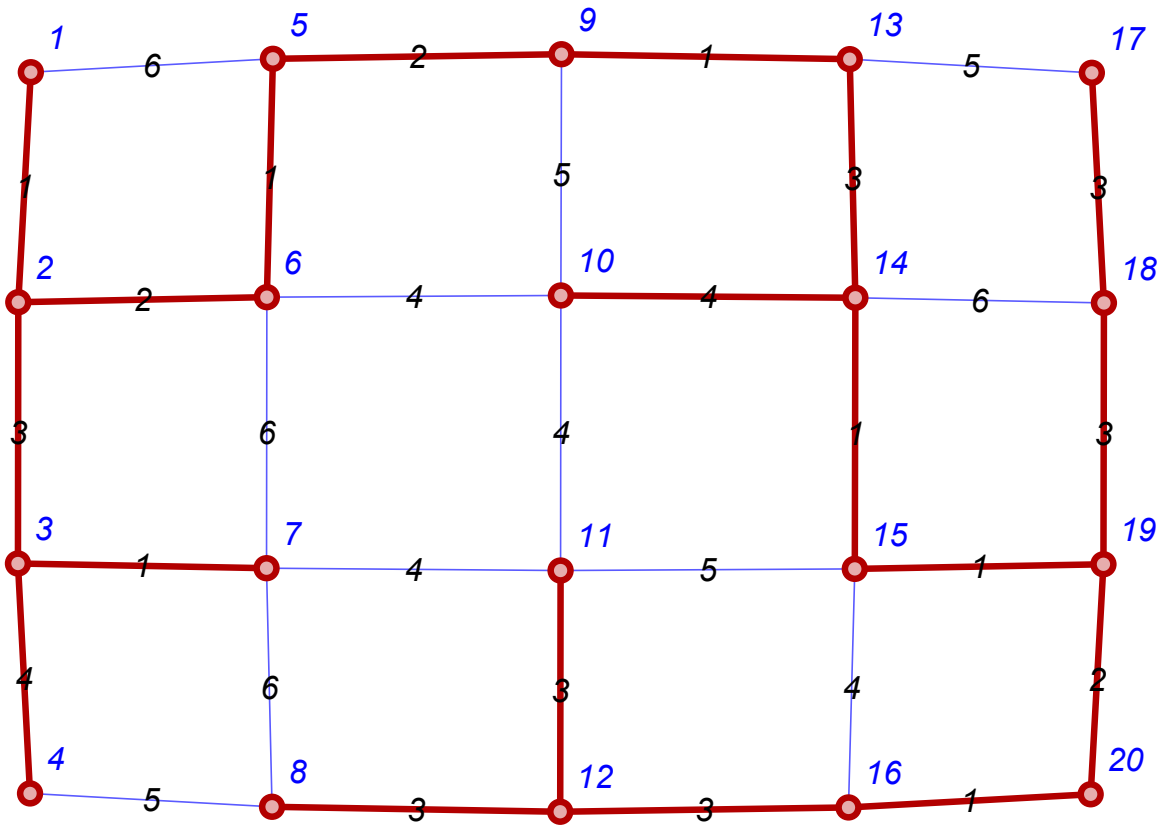
Z obrázku plyne, že hrana 3 - 4 bude součástí minimální kostry a potom zbývá libovolná z hran 6 - 10, 10 - 11, 10 - 14. Tím jsme vlastně ukázali, že existují hned 3 minimální kostry s výsledným součtem vah hran rovným číslu

$$7 + 2 \cdot 33 \cdot 7 + 4 \cdot 2 = 42.$$

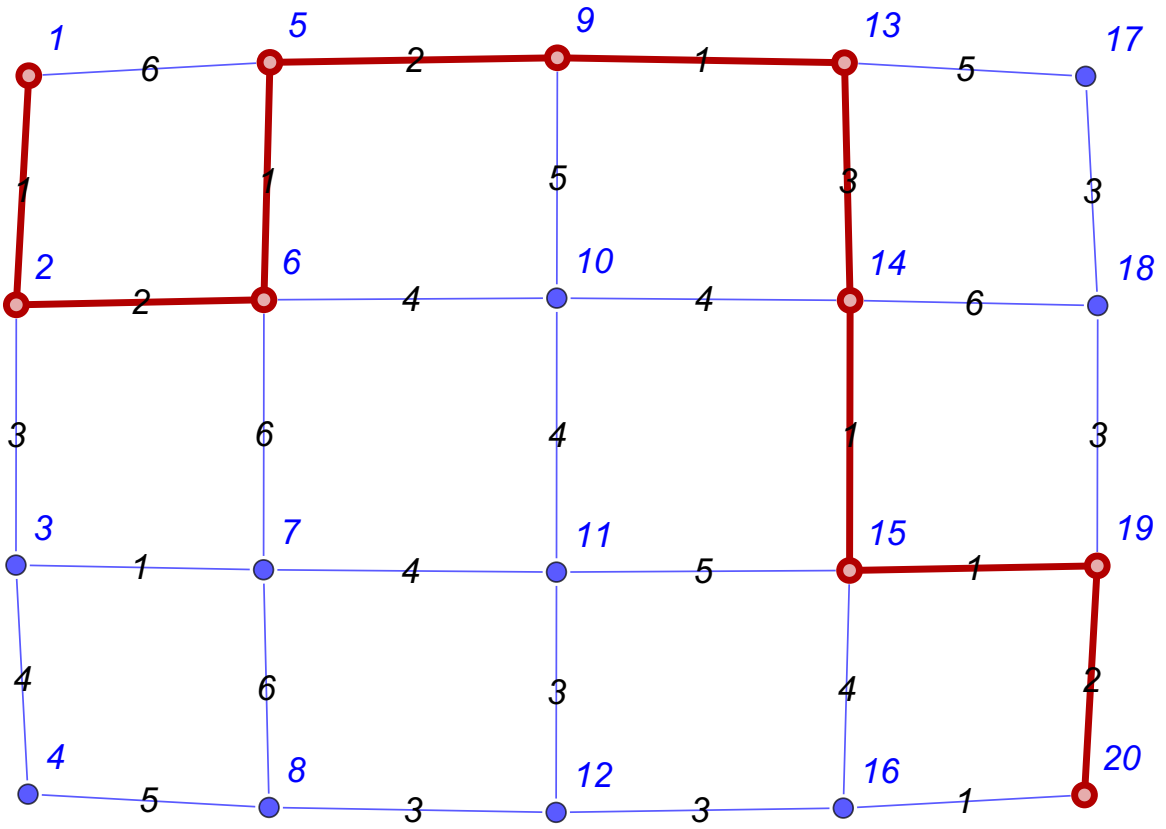
Kostra je zobrazena na Obrázku 1.11. Následující kód vygeneruje uvažovaný graf, vykreslí jeho minimální kostru a sečte váhy jejích hran:

Kód - Mathematica 1.18: nalezení minimální kostry.

```
G = GridGraph[{4, 5}];
w = {1, 6, 3, 2, 4, 1, 5, 1, 2, 6, 4, 6, 4, 2, 5, 5, 5, 3, 2, 5, 3, 6,
    1, 3, 3, 2, 6, 6, 4, 1, 3}
GG = Graph[EdgeList[G], EdgeWeight -> w, VertexLabels -> "Name",
   VertexStyle -> Directive[Opacity[0.65`], Blue],
   VertexLabelStyle -> Directive[Blue, Italic, 10],
   EdgeLabels -> "EdgeWeight",
   EdgeStyle -> Directive[Opacity[0.65`], Blue],
   EdgeLabelStyle -> Directive[Black, Italic, 10]]
k = FindSpanningTree[GG];
HighlightGraph[GG, k, GraphHighlightStyle -> "Thick"]
size[g_] := With[{edges = EdgeList[FindSpanningTree[{g, 1}]]},
  Total[PropertyValue[{g, #}, EdgeWeight] & /@ edges]]
size[GG]
```



Obrázek 1.11: Graf s vyznačenou minimální kostrou o délce 42.



Obrázek 1.12: Graf s vyznačenou minimální cestou o délce 14.

Poznámka k příkladu

Modifikací prvních dvou řádek výše uvedeného kódu do tvaru

Kód - Mathematica 1.19: nalezení minimální kostry - modifikace.

```
nx = 5; ny = 6; G = GridGraph[{nx, ny}];
w = RandomInteger[{1, 6}, EdgeCount[G]];
```

lze generovat náhodné mřížkové grafy libovolné velikosti k procvičování. Lze také najít minimální cestu pomocí tzv. Dijkstrova algoritmu vedoucí například z vrcholu 1 do vrcholu 20. Ta je uvedena na Obrázku 1.12.

Úloha k samostatnému řešení

Pro grafy na Obrázcích 1.14-1.15 nalezněte

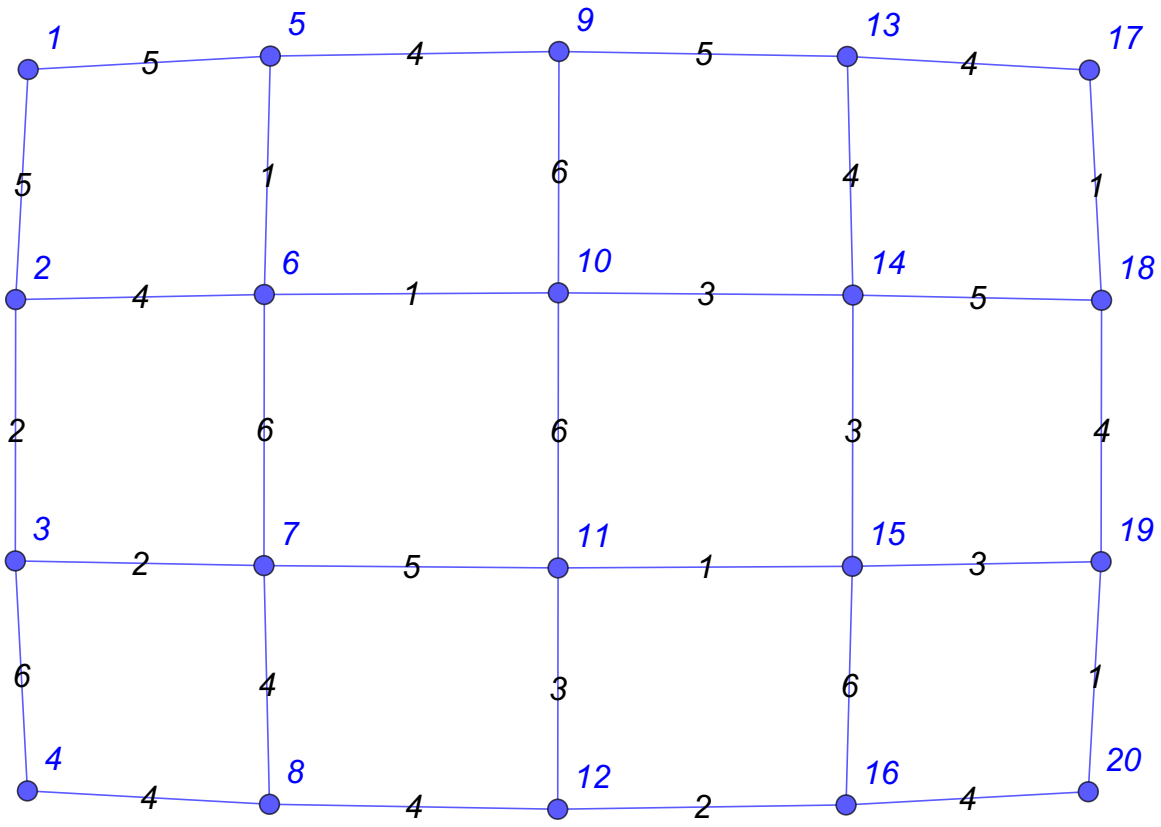
- minimální kostru a její délku,
- nejkratší cestu z vrcholu 1 do vrcholu 20 a její délku?

Úloha k samostatnému řešení

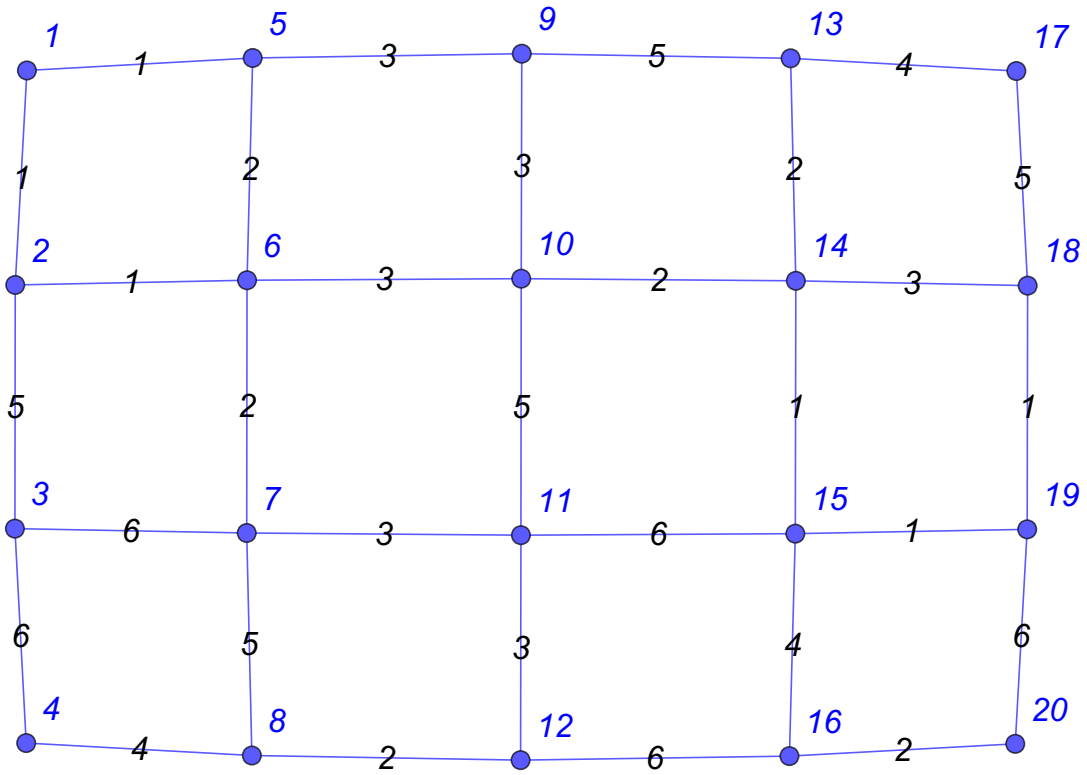
Pro graf definovaný distanční maticí uvedenou zde a zmenšený uvedený na Obrázku (1.13) najděte minimální kostru spojující uvedených 24 měst v Evropě. Udejte její výslednou délku.

	Barc.	Běleh.	Berlín	Brusel	Bukur.	Budap.	Kodaň	Dublin	Hamb.	Istan.	Kiev	Londýn	Madrid	Miláno	Moskva	Mnichov	Paříž	Praha	Řím	Petroh.	Sofia	Stockh.	Vídeň	Varš.
Barcelona	0	1528	1497	1062	1968	1498	1757	1469	1471	2230	2391	1137	504	725	3006	1054	831	1353	856	2813	1745	2276	1347	1862
Bělehrad	1528	0	999	1372	447	316	1327	2145	1229	809	976	1688	2026	885	1710	773	1445	738	721	1797	329	1620	489	826
Berlín	1497	999	0	651	1293	689	354	1315	254	1735	1204	929	1867	840	1607	501	876	280	1181	1319	1318	810	523	516
Brusel	1062	1372	651	0	1769	1131	766	773	489	2178	1836	318	1314	696	2253	601	261	721	1171	1903	1697	1280	914	1159
Bukurešť	1968	447	1293	1769	0	639	1571	2534	1544	445	744	2088	2469	1331	1497	1186	1869	1076	1137	1740	296	1742	855	946
Budapešť	1498	316	689	1131	639	0	1011	1894	927	1064	894	1450	1975	788	1565	563	1247	443	811	1556	629	1316	216	545
Kodaň	1757	1327	354	766	1571	1011	0	1238	287	2017	1326	955	2071	1157	1558	838	1025	633	1529	1143	1635	521	868	667
Dublin	1469	2145	1315	773	2534	1894	1238	0	1073	2950	2513	462	1449	1413	2792	1374	776	1465	1882	2314	2471	1626	1680	1823
Hamburg	1471	1229	254	489	1544	927	287	1073	0	1983	1440	720	1785	900	1779	610	744	492	1307	1414	1554	809	742	750
Istanbul	2230	809	1735	2178	445	1064	2017	2950	1983	0	1052	2496	2734	1669	1753	1582	2253	1507	1373	2099	502	2171	1273	1386
Kiev	2391	976	1204	1836	744	894	1326	2513	1440	1052	0	2131	2859	1672	756	1391	2022	1138	1673	1051	1020	1265	1052	690
Londýn	1137	1688	929	318	2088	1450	955	462	720	2496	2131	0	1263	957	2498	916	340	1034	1431	2093	2012	1431	1233	1445
Madrid	504	2026	1867	1314	2469	1975	2071	1449	1785	2734	2859	1263	0	1187	3437	1484	1053	1773	1360	3183	2250	2591	1807	2288
Miláno	725	885	840	696	1331	788	1157	1413	900	1669	1672	957	1187	0	2283	348	641	646	476	2122	1166	1650	623	1143
Moskva	3006	1710	1607	2253	1497	1565	1558	2792	1779	1753	756	2498	3437	2283	0	1957	2484	1664	2374	632	1777	1227	1669	1149
Mnichov	1054	773	501	601	1186	563	838	1374	610	1582	1391	916	1484	348	1957	0	685	300	698	1773	1096	1311	354	809
Paříž	831	1445	876	261	1869	1247	1025	776	744	2253	2022	340	1053	641	2484	685	0	885	1105	2157	1758	1541	1033	1365
Praha	1353	738	280	721	1076	443	633	1465	492	1507	1138	1034	1773	646	1664	300	885	0	922	1476	1064	1052	250	514
Řím	856	721	1181	1171	1137	811	1529	1882	1307	1373	1673	1431	1360	476	2374	698	1105	922	0	2339	894	1974	763	1316
Petrohrad	2813	1797	1319	1903	1740	1556	1143	2314	1414	2099	1051	2093	3183	2122	632	1773	2157	1476	2339	0	1969	688	1577	1023
Sofia	1745	329	1318	1697	296	629	1635	2471	1554	502	1020	2012	2250	1166	1777	1096	1758	1064	894	1969	0	1884	817	1076
Stockholm	2276	1620	810	1280	1742	1316	521	1626	809	2171	1265	1431	2591	1650	1227	1311	1541	1052	1974	688	1884	0	1241	808
Vídeň	1347	489	523	914	855	216	868	1680	742	1273	1052	1233	1807	623	1669	354	1033	250	763	1577	817	1241	0	557
Varšava	1862	826	516	1159	946	545	667	1823	750	1386	690	1445	2288	1143	1149	809	1365	514	1316	1023	1076	808	557	0

Obrázek 1.13: Vzdálenosti 24 měst v Evropě.



Obrázek 1.14: řešení - délka minimální kostry je 52, délka minimální cesty je 17 a prochází vrcholy 1 - 5 - 6 - 10 - 14 - 15 - 19 - 20.



Obrázek 1.15: řešení - délka minimální kostry je 45, délka minimální cesty je 14 a prochází vrcholy 1 - 2 - 6 - 10 - 14 - 15 - 20.

KAPITOLA

2

NUMERICKÁ MATEMATIKA 1

2.1 Vliv zaokrouhlování, konvergence

Příklad 2.1.16 Vypočtete přibližnou hodnotu čísla π z Machinových řad různé délky. Jak souvisí chyba aproximace s délkou použité řady? Jaké nejpřesnější aproximace jste schopni dosáhnout?

Řešení:
Přibližné hodnoty čísla π určíme pomocí Machinova vzorce

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}, \tag{2.1}$$

ve kterém funkci arctan nahradíme Taylorovou řadou v okolí bodu 0,

$$\arctan x \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} + \dots \qquad \text{pro } |x| < 1$$

a za argument x dosazujeme hodnoty $\frac{1}{5}$ nebo $\frac{1}{239}$. K praktickému výpočtu použijeme vzorec (2.1) vynásobený hodnotou 4, tedy

$$\pi = 16\arctan\frac{1}{5} - 4\arctan\frac{1}{239}. \tag{2.2}$$

Pro různý počet k nenulových členů Taylorovy řady dostáváme různé aproximace, např.

$$k=1: \quad \pi_1 \approx 16\cdot\frac{1}{5} - 4\cdot\frac{1}{239} \approx 3.183263598326360,$$
$$k=2: \quad \pi_2 \approx 16\cdot(\frac{1}{5} - \frac{1}{3\cdot 5^3}) - 4\cdot(\frac{1}{239} - \frac{1}{3\cdot 239^3}) \approx 3.140597029326060,$$
$$k=3: \quad \pi_3 \approx 16\cdot(\frac{1}{5} - \frac{1}{3\cdot 5^3} + \frac{1}{7\cdot 5^5}) - 4\cdot(\frac{1}{239} - \frac{1}{3\cdot 239^3} + \frac{1}{5\cdot 239^5}) \approx 3.141621029325034.$$

Přepočítání pro různá k můžeme zautomatizovat pomocí Kódu 2.1:

Kód - Matlab 2.1: Aproximace čísla π.

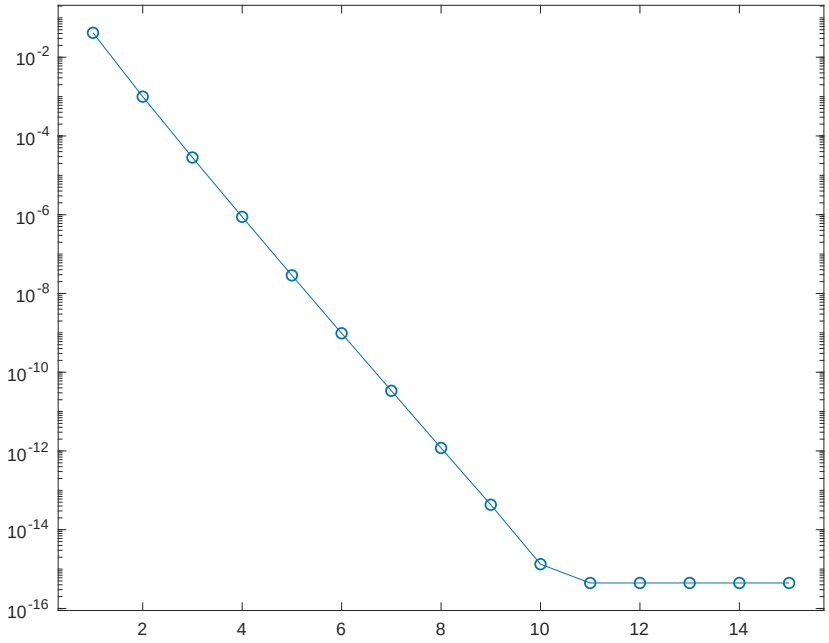
```
n=15;                                          % maximalni pocet clenu
p=[];
for k=1:n
    p=[(-1)^(k-1)*(1/(2*k-1)), 0, p];          % koeficienty polynomu
    pi_k(k)=16*polyval(p,1/5)-4*polyval(p,1/239);  % aproximace pi
end
e_k=pi_k-pi;                                   % absolutni chyba
format long;
fprintf('  %2.0f:  %1.16f  %2.2d\n', [(1:n)' pi_k' abs(e_k)'].')
semilogy(abs(e_k),'o-');                       % graf v log skale
```

Matlabovská funkce „polyval“ vypočítává hodnotu polynomu specifikovaného vektorem koeficientů v daném bodě. Tím získáme výpis u kterého je v prvním sloupci je uveden počet členů Taylorovy řady k, v prostředním aproximace π_k a v pravém velikost absolutní chyby, zde hodnoty $\pi_k - \pi$ (v absolutní hodnotě):

```
 1:  3.1832635983263602  4.17e-02
 2:  3.1405970293260608  9.96e-04
 3:  3.1416210293250346  2.84e-05
 4:  3.1415917721821778  8.81e-07
 5:  3.1415926824043998  2.88e-08
 6:  3.1415926526153091  9.74e-10
```

```
 7:  3.1415926536235550  3.38e-11
 8:  3.1415926535886025  1.19e-12
 9:  3.1415926535898362  4.31e-14
10:  3.1415926535897918  1.33e-15
11:  3.1415926535897936  4.44e-16
12:  3.1415926535897936  4.44e-16
13:  3.1415926535897936  4.44e-16
14:  3.1415926535897936  4.44e-16
15:  3.1415926535897936  4.44e-16
```

Je zřejmé, že posledních pět aproximací je totožných a vykazuje prakticky nulovou absolutní chybu. To je dáno omezením systému Matlab, který standardně pracuje ve formátu double s přesností na 16 desetinných míst. Generování dalších aproximací nemá v Matlabu smysl, což také ukazuje Obrázek 2.1.



Obrázek 2.1: Absolutní chyby aproximace čísla π v systému Matlab.

Kód - Mathematica 2.2: Aproximace čísla π.

```
k = 10;
s[x_] = Normal[Series[ArcTan[x], {x, 0, 2*k}]]
piAprox = 16*s[1/5] - 4*s[1/239]
error = N[Abs[piAprox - Pi], 3]
N[piAprox, 3]
```

Mathematica oproti Matlabu dovede uchovávat jednotlivé aproximace ve zlomcích a proto Kód 2.2 přepočte poslední řádky výpočtu v Matlabu jako:

```
10:  3.1415926535897917  1.77e-15
11:  3.1415926535897933  5.63e-17
12:  3.1415926535897932  2.07e-18
13:  3.1415926535897932  7.67e-20
14:  3.1415926535897932  2.86e-21
15:  3.1415926535897932  1.07e-22
```

Výsledky vypočtené pomocí Machinova vzorce lze dále ještě zpřesňovat. Cenou je vyšší paměťová náročnost. Např. aproximaci pro $k = 10$ ukládá Mathematica jako zlomek

$$\frac{899286197155536297279342607251940330683169516449531712999212996 56}{286251687063232837596301958915406579332976668481878471374511718 75}.$$

Poznámka k příkladu

Odvození Machinova vzorce (2.1) je založené na opakovaném použití rovnosti

$$\arctan x \pm \arctan y = \arctan\left(\frac{x+y}{1 \mp xy}\right)$$

pro vhodně zvolené parametry x, y. Počáteční volbou $x = y = 1/5$ dostáváme rovnost

$$2\arctan\frac{1}{5} = \arctan\frac{5}{12}.$$

Vynásobením dvěma a přepočtem dále z ní získáme rovnosti

$$4\arctan\frac{1}{5} = 2\arctan\frac{5}{12} = \arctan\frac{120}{119}$$

a úplně na závěr obdržíme

$$4\arctan\frac{1}{5} - \arctan 1 = \arctan\frac{1}{239},$$

což představuje vzorec (2.1), uvážíme-li, že $\arctan 1 = \frac{\pi}{4}$. Volbou jiných počátečních hodnot x, y získáme další formule (odvoďte), např.

$$\frac{\pi}{4} = 4\arctan\frac{1}{2} - \arctan\frac{31}{17}, \tag{2.3}$$

$$\frac{\pi}{4} = 4\arctan\frac{1}{3} - \arctan\frac{17}{31}, \tag{2.4}$$

$$\frac{\pi}{4} = 4\arctan\frac{1}{4} - \arctan\frac{79}{401}. \tag{2.5}$$

Pomocí druhého a třetího vzorce výše je také možno přepočítat číslo π jednoduchou modifikací Kódů 2.1 a 2.2. Pomocí prvního vzorce ale nikoliv, protože $\frac{31}{17} > 1$ a odpovídající Taylorova řada diverguje.

Příklad 2.1.17 Ukažte, že posloupnosti generované pomocí rekurentních formulí

$$a_{k+1} = 1 + \frac{1}{a_k}, \quad a_1 = 1, \tag{2.6}$$

$$b_{k+1} = \sqrt{1 + b_k}, \quad b_1 = 1 \tag{2.7}$$

konvergují ke stejnému číslu a porovnejte rychlost jejich konvergence.

Řešení:
Několik prvních členů posloupností lze jednoduše vygenerovat pomocí:

Kód - Mathematica 2.3: Rekurentní formule

```
n=20;
ak=RecurrenceTable[{a[k+1]==1+1/a[k],a[1]==1},a,{k,n}];
bk=RecurrenceTable[{b[k+1]==Sqrt[1+b[k]],b[1]==1},b,{k,n}];
N[ak]
N[bk]
```

a získat posloupnosti

$$a_k = \{1, 2, 1.5, 1.6667, 1.6, 1.625, 1.61538, 1.61905, 1.61765, 1.61818, \dots\},$$
$$b_k = \{1, 1.4142, 1.5538, 1.5981, 1.6118, 1.6161, 1.6174, 1.6179, 1.6180, 1.6180, \dots\}.$$

Z praktických důvodů zaokrouhlujeme hodnoty na 4 desetinná místa. Zdá se, že obě posloupnosti konvergují ke stejnému číslu. Za předpokladu, že opravdu konvergují (to zde neukazujeme), musí pro jejich limity, které označíme jako a, b po dosazení do (2.6) a (2.7) platit

$$a = 1 + \frac{1}{a}, \qquad b = \sqrt{1 + b}. \tag{2.8}$$

Řešením obou rovnic výše získáme možná řešení

$$a_{1,2} = \frac{1 \pm \sqrt{5}}{2}, \qquad b = \frac{1 + \sqrt{5}}{2}.$$

Řešení a_2 je záporné a nevyhovuje rovnice pro b (ověřte). Protože posloupnost $(a_k)_{k=1}^{\infty}$ obsahuje pouze kladná čísla, konvergují obě posloupnosti nutně ke stejné kladné hodnotě

$$\varphi := \frac{1 + \sqrt{5}}{2} \approx 1.6180.$$

Pomocí dodatečného kódu:

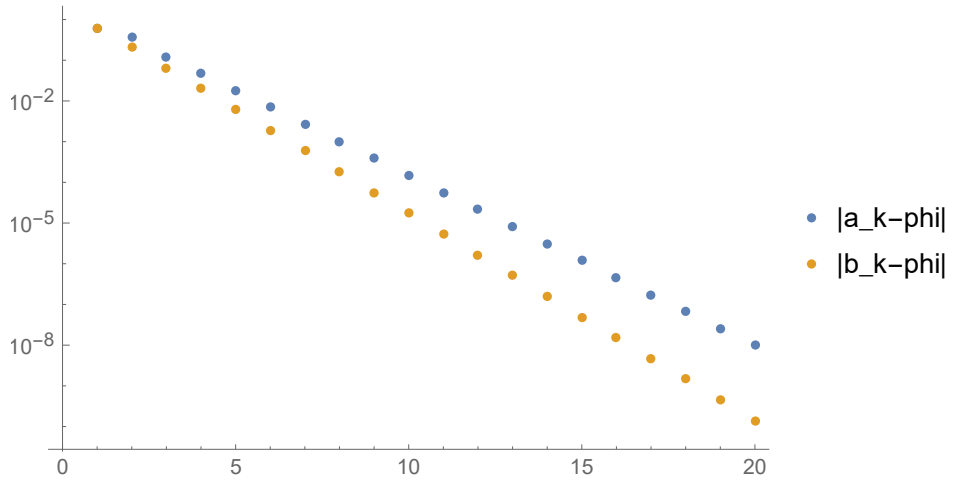
Kód - Mathematica 2.4: Rekurentní formule - dodatek

```
phi = (1 + Sqrt[5])/2;
eak = Abs[ak - phi];
ebk = Abs[bk - phi];
ListLogPlot[{eak, ebk}, PlotLegends -> {"|a_k-phi|", "|b_k-phi|"}]
```

můžeme vypočítat členy posloupností absolutních chyb $(e_k^a)_{k=1}^{\infty}, (b_k^a)_{k=1}^{\infty}$, kde

$$e_k^a := a_k - \varphi, \qquad e_k^b := b_k - \varphi, \qquad k = 1, 2, \dots$$

a ty zobrazit v absolutní hodnotě a poté v logaritmické škále na Obrázku 2.2.



Obrázek 2.2: Srovnání rychlosti konvergence posloupností $(a_k)_{k=1}^{\infty}$, $(b_k)_{k=1}^{\infty}$.

Obrázek naznačuje, že posloupnost $(a_k)_{k=1}^{\infty}$ konverguje k hodnotě φ pomaleji než $(b_k)_{k=1}^{\infty}$. Přesněji můžeme pomocí dodatečného kódu:

Kód - Mathematica 2.5: Rekurentní formule - rychlost

```
N[Table[Abs[eak[[k+1]]/eak[[k]]],{k,1,n-1}]]
N[Table[Abs[ebk[[k+1]]/ebk[[k]]],{k,1,n-1}]]
```

určit poměry za sebou jdoucích členů posloupností absolutních chyb

$$\left(\frac{|e^a_{k+1}|}{|e^a_k|}\right)_{k=1}^{\infty} = \{0.6180, 0.3090, 0.4120, 0.3708, 0.3863, 0.3803, 0.3826, 0.3817, 0.3821, \dots\},$$
$$\left(\frac{|e^b_{k+1}|}{|e^b_k|}\right)_{k=1}^{\infty} = \{0.3298, 0.3153, 0.3109, 0.3096, 0.3092, 0.3091, 0.3090, 0.3090, 0.3090, \dots\}.$$

Posloupnosti výše naznačují konvergenci a asymptoticky (pro dostatečně velká k) opravdu platí

$$|e^a_{k+1}| \approx 0.3820\,|e^a_k|, \qquad |e^b_{k+1}| \approx 0.3090\,|e^b_k|.$$

To odpovídá tzv. lineární konvergenci obou původních posloupností $(b_k)_{k=1}^{\infty}$, $(b_k)_{k=1}^{\infty}$.

Poznámka k příkladu

Četná použití např. v umění a biologii má číslo

$$\varphi := \frac{1+\sqrt{5}}{2} \approx 1.6180339887498948482$$

udávající hodnotu tzv. zlatého řezu. Také se objevuje v Moivre-Binetově formuli (viz Příklad 1.1.3). Je to číslo iracionální a lze jej podle formule (2.6) vyjádřit ekvivalentně pomocí tzv. nekonečného řetězového zlomku

$$\varphi = 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+\dots}}}, \tag{2.9}$$

nebo podle formule (2.7) pomocí nekonečné-krát vnořených druhých odmocnin

$$\varphi = \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\dots}}}}. \tag{2.10}$$

Mathematica umí s řetězovými zlomky pracovat přímo. Proto můžeme první deset členů posloupnost $(a_n)_{n=1}^{\infty}$ získat alternativně pomocí příkazu:

Kód - Mathematica 2.6: řetězový zlomek

```
Do[Print[FromContinuedFraction[ConstantArray[1, n]]],{n,10}]
```

Úloha k samostatnému řešení

Vypočtěte přibližně hodnotu nekonečné řady (tzv. Basilejský problém)

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

pomocí konečných součtů s 10, 1000 nebo 1.000.000 prvními členy. Jaká je absolutní chyba výpočtů v jednotlivých případech?

Úloha k samostatnému řešení

Vypočtěte hodnoty binomických koeficientů

$$\binom{100}{2}, \binom{100}{50}, \binom{999}{3}, \binom{999}{5}, \binom{999}{7}$$

ve zvoleném výpočetním systému. Uvažujte výpočet z definiční formule

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!},$$

její modifikace

$$\binom{n}{k} = \frac{(k+1)}{1}\frac{(k+2)}{2}\cdots\frac{n}{(n-k)}$$

nebo přímo pomocí určené funkce daného systému. Vysvětlete možné problémy počítačové realizace.

2.2 Řešení nelineárních rovnic

Příklad 2.2.18 Pomocí metody bisekce najděte řešení rovnice

$$\frac{x}{3} = \sin x \tag{2.11}$$

na intervalu $I = [1,3]$ s přesností na 5 desetinných míst.

Řešení: Rovnici si přepíšeme do tvaru rovnice ve tvaru $f(x) = 0$, ve které volíme

$$f(x) = \frac{x}{3} - \sin x.$$

Dále definujeme podle zadání úlohy body $a = 1, b = 3$ intervalu $I = [a,b]$. Protože je funkce f spojitá na I a platí zároveň nerovnost

$$f(a)\, f(b) < 0 \tag{2.12}$$

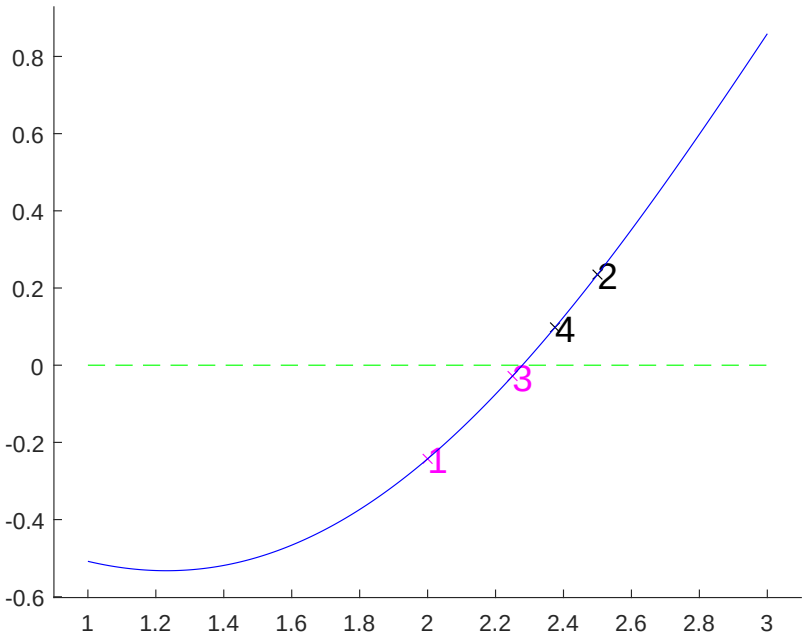
(konkrétně zde platí $f(a) \approx -0.508 < 0$ a $f(b) \approx 0.859 > 0$), musí mít funkce f v intervalu I alespoň jeden nulový bod (kořen), který hledáme. Vypočteme středový bod intervalu I, tj. bod

$$x_s = \frac{a+b}{2} = 2.$$

Protože $f(x_s) < 0$, nahradíme levý bod a intervalu I hodnotou x_s (v opačném případě bychom nahradili pravý bod b hodnotou x_s) a zúžíme původní interval I na

$$I = [2,3].$$

Pro nový poloviční interval I (od toho jde zde o metodu půlení intervalu - bisekci) potom zase platí podmínka (2.12) a proto můžeme postup výše opakovat až do libovolné přesnosti.



Obrázek 2.3: Hledání řešení rovnice $\frac{x}{3} = \sin x$ pomocí metody bisekce. Zvýrazněné jsou středové body x_s prvních čtyř iterací intervalů I, které slouží jako aproximace řešení.

Půlení intervalů lze jednoduše realizovat například pomocí kódu:

Kód - Matlab 2.7: Bisekce.

```
format long;
epsilon=1e-5;                                   % tolerance
f=@(x) x/3-sin(x);                              % funkce pro hledani korene
a=1; b=3;                                       % levy a pravy bod intervalu
%odhad=ceil((log((b-a)/epsilon))/log(2))        % odhad poctu iteraci
figure; hold on;
xx=linspace(a,b,1000);                          % body pro graf funkce
plot(xx,f(xx),'b-',xx,0*xx,'g--');              % vykresleni funkce a osy x
str='  %2.0f: a=%2.6f, b=%2.6f, f(a)=%2.2d, f(b)=%2.2d \n';
col{1}='m'; col{2}='k';                         % format vystupu
k=0;
while(b-a>epsilon)
    fa=f(a); fb=f(b);                           % funkcni hodnoty
    signa=sign(fa); signb=sign(fb);             % znamenka
    xs=(a+b)/2;                                 % stredovy bod
    fprintf(str,k,a,b,fa,fb);                   % vypis iteraci
    signxs=sign(f(xs));                         % znamenko v bode x
    if (signa==signxs)
        a=xs; cas=1;                            % pripad 1
    else
        b=xs; cas=2;                            % pripad 2
    end
    if k<=4                                     % vykresleni iteraci
        plot(xs,f(xs),'kx');
        text(xs,f(xs),num2str(k),'Color',col{cas},'FontSize',16);
    end
    k=k+1;                                      % cislo iterace
end
hold off;
str2='  %2.0f: a=%2.6f, b=%2.6f -> xs=%2.6f, f(xs)=%2.2d \n';
fprintf(str2,k,a,b,(a+b)/2,f((a+b)/2));         % vypis finalnich hodnot
```

a získáváme výpis, ve kterém v prvním sloupci máme čísla jednotlivých iterací, v druhém a třetím hodnoty a, b, ve čtvrtém a pátém hodnoty $f(a), f(b)$.

```
  0: a=1.000000, b=3.000000, f(a)=-5.08e-01, f(b)=8.59e-01
  1: a=2.000000, b=3.000000, f(a)=-2.43e-01, f(b)=8.59e-01
  2: a=2.000000, b=2.500000, f(a)=-2.43e-01, f(b)=2.35e-01
  3: a=2.250000, b=2.500000, f(a)=-2.81e-02, f(b)=2.35e-01
  4: a=2.250000, b=2.375000, f(a)=-2.81e-02, f(b)=9.80e-02
  5: a=2.250000, b=2.312500, f(a)=-2.81e-02, f(b)=3.35e-02
  6: a=2.250000, b=2.281250, f(a)=-2.81e-02, f(b)=2.35e-03
  7: a=2.265625, b=2.281250, f(a)=-1.30e-02, f(b)=2.35e-03
  8: a=2.273438, b=2.281250, f(a)=-5.33e-03, f(b)=2.35e-03
  9: a=2.277344, b=2.281250, f(a)=-1.49e-03, f(b)=2.35e-03
 10: a=2.277344, b=2.279297, f(a)=-1.49e-03, f(b)=4.27e-04
 11: a=2.278320, b=2.279297, f(a)=-5.33e-04, f(b)=4.27e-04
 12: a=2.278809, b=2.279297, f(a)=-5.32e-05, f(b)=4.27e-04
 13: a=2.278809, b=2.279053, f(a)=-5.32e-05, f(b)=1.87e-04
 14: a=2.278809, b=2.278931, f(a)=-5.32e-05, f(b)=6.69e-05
 15: a=2.278809, b=2.278870, f(a)=-5.32e-05, f(b)=6.86e-06
 16: a=2.278839, b=2.278870, f(a)=-2.32e-05, f(b)=6.86e-06
 17: a=2.278854, b=2.278870, f(a)=-8.15e-06, f(b)=6.86e-06
 18: a=2.278862, b=2.278870 -> xs=2.278866, f(xs)=3.10e-06
```

První čtyři aproximace řešení rovnice jsou zobrazeny s funkcí f na Obrázku 2.3.

Rozborem hodnot v tabulce zjistíme v 18-té iteraci je vzdálenost čísel a, b opravdu menší než 10^{-5} a jejich hodnoty se liší až na pátém desetiném místě. Můžeme tedy zaručit že

$$x \approx 2.2788 \tag{2.13}$$

je řešením rovnice (2.11) přesným na 4 desetinná místa. Není zatím jasné, jestli přesným číslem na pátém desetinném místě je hodnota 6 nebo 7. K tomu je třeba iterovat dále a přepočtem pro např. `epsilon=1e-6` získáme dodatečné iterace:

```
18: a=2.278862, b=2.278870, f(a)=-6.50e-07, f(b)=6.86e-06
19: a=2.278862, b=2.278866, f(a)=-6.50e-07, f(b)=3.10e-06
20: a=2.278862, b=2.278864, f(a)=-6.50e-07, f(b)=1.23e-06
21: a=2.278862, b=2.278863 -> xs=2.278862, f(xs)=-1.81e-07
```

Z iterace 19 je zřejmé, že správnou číslicí na pátém desetinném místě je 6 a tedy aproximace

$$x \approx 2.27886 \tag{2.14}$$

je řešením rovnice (2.11) přesným na 5 desetinných míst.

Poznámka k příkladu

Počet iterací k dosažení obecné přesnosti 10^{-d} je možné určit před provedením samotného výpočtu z jednoduché úvahy: délka intervalu I_k v iteraci k je dána

- v nulté iteraci hodnotou $I_0 = b - a$,
- v první iteraci hodnotou $I_1 = (b - a)/2$,
- ve druhé iteraci hodnotou $I_2 = (b - a)/4$ atd.

Délky intervalů tedy tvoří prvky geometrické posloupnosti s kvocientem $1/2$ ve tvaru

$$|I_k| = (b - a)\,(1/2)^k$$

a my hledáme nejmenší celočíselný index k_0, pro který platí: $|I_{k_0}| \leq 10^{-d}$. Zlogaritmováním získáme nerovnici

$$k_0 \geq d \log_2 10 + \log_2(b - a). \tag{2.15}$$

V našem příkladě je $b - a = 2$ a $d = 5$ a vyjde $k_0 \approx 17.6$, tedy je nutné volit 18 iterací. Zvýšení přesnosti např. na 10^{-15} vede (ukažte) na hodnotu $k_0 \approx 50.8$ a tedy je nutné volit 51 iterací. Potom dospějeme ještě k přesnějšímu řešení

$$x \approx 2.278862660075828. \tag{2.16}$$

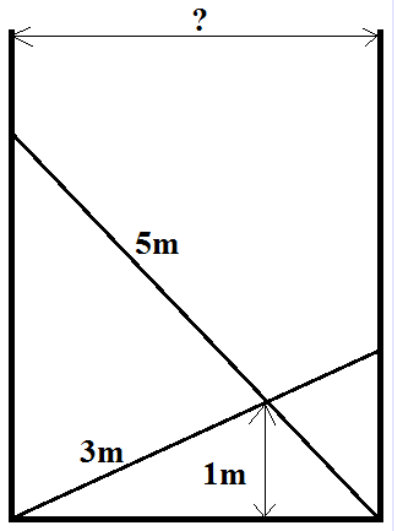
Matlab samozřejmě disponuje vlastním řešičem nelineárních rovnic a kód

Kód - Matlab 2.8: Přesné řesení nelineární rovnice.

```
1 sym x;
2 vpasolve(sin(x)-x/3==0, x, [1  3])
```

nadefinuje symbolickou proměnnou x a rovnici (2.11) vyřeší. K úspěšnému spuštění kódu je třeba mít nainstalovaný symbolický balíček (angl. Symbolic Math Toolbox) Matlabu.

Příklad 2.2.19 Nalezněte šířku studny podle Obrázku 2.4. Odvozte rovnici, která šířku popisuje a vyřešte jí pomocí metody bisekce.



Obrázek 2.4: Úloha o studni: najděte šířku studny.

Řešení: Označme si délky jednotlivých úseček a výšku průsečíku jako

$$d_1 = 3, \quad d_2 = 5, \quad v = 1$$

a neznámou šířku studny jako ℓ. Přímky, na kterých jednotlivé úsečky leží, můžeme parametrizovat jako

$$p_1 : y(x) = \frac{\sqrt{d_1^2 - \ell^2}}{\ell} x, \qquad p_2 : y(x) = -\frac{\sqrt{d_2^2 - \ell^2}}{\ell} x + \sqrt{d_2^2 - \ell^2},$$

tedy platí, že bod $[0,0]$ leží na přímce p_1 a bod $[0,\ell]$ na přímce p_2. Přímky se protínají v bodě

$$[x_p, y_p] = \left[\frac{\ell\sqrt{d_2^2 - \ell^2}}{\sqrt{d_1^2 - \ell^2} + \sqrt{d_2^2 - \ell^2}}, v \right]$$

a dosazením x_p do p_1 (nebo shodně do p_2) získáváme nelineární rovnici ve tvaru

$$f(\ell) := \frac{\sqrt{d_1^2 - \ell^2}\sqrt{d_2^2 - \ell^2}}{\sqrt{d_1^2 - \ell^2} + \sqrt{d_2^2 - \ell^2}} - v = 0 \tag{2.17}$$

pro hledanou neznámou ℓ. Z geometrie úlohy budeme hledat řešení na intervalu

$$I = (a,b) = (0, \min\{d_1, d_2\}),$$

protože záporná hodnota šířky studny je nepřípustná a zároveň musí vyjít menší než menší z délek d_1, d_2. Pro naše konkrétní parametry musíme řešit nelineární rovnici

$$f(\ell) = \frac{\sqrt{9 - \ell^2}\sqrt{25 - \ell^2}}{\sqrt{9 - \ell^2} + \sqrt{25 - \ell^2}} - 1 = 0 \tag{2.18}$$

pro neznámou šířku studny ℓ na otevřeném intervalu (0,3). Tím jsme odvodili nelineární rovnici šířky studny.
Použitím metody bisekce lze dopočítat přímo použitím hodnot $a = 0, b = 3$ hodnotu (proveďte)

$$\ell \approx 2.69810,$$

která představuje hledanou šířku studny.

Příklad 2.2.20 Pomocí Newtonovy metody pro rovnici

$$f(x) = x^2 - 10 = 0$$

vypočtěte přibližně hodnotu kořenu $\bar{x} = \sqrt{10}$. Jako počáteční iteraci zvolte $x_0 = 4$.

Řešení: Newtonova metoda generuje posloupnost iterací $(x_k)_{k=1}^{\infty}$ které (za určitých podmínek) konvergují k hodnotě $\bar{x}$. Iterace se vypočítávají podle rekurentní formule

$$x_k := x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}, \qquad k = 1, 2, \ldots \tag{2.19}$$

kde x_0 je daná počáteční iterace. Protože $f'(x) = 2x$, získáváme po úpravě výrazu výše

$$x_k = \frac{1}{2}(x_{k-1} + \frac{10}{x_{k-1}}), \qquad k = 1, 2, \ldots . \tag{2.20}$$

Několik prvních iterací generujeme jednoduše pomocí kódu:

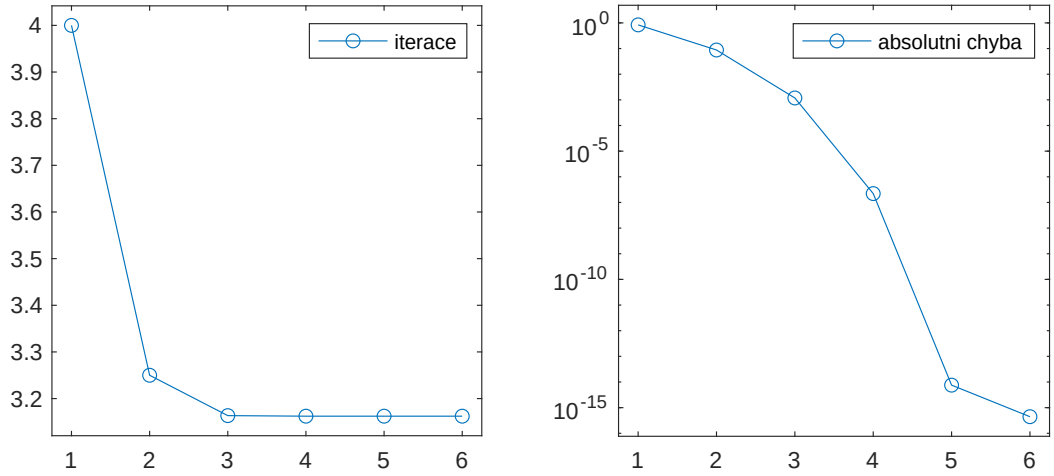
Kód - Matlab 2.9: Babylónská metoda výpočtu druhé odmocniny.

```
d=10;                                   % hodnota, kterou odmocnujeme
x0=4;                                   % pocatecni iterace
n=5;                                    % pocet iteraci
x=zeros(n+1,1); x(1)=x0;                % vektor iteraci
for k=1:n
  x(k+1)=(x(k)+d/x(k))/2;               % iteracni formule
end
error=abs(x-sqrt(d));                   % absolutni chyba
fprintf('    %.15f    %2.2d \n',[x,error]')   % vypis
```

a získáme posloupnost 6 iterací (včetně hodnoty x_0) čísla $\sqrt{10}$ uvedenou v levém sloupci a velikost jejich absolutní chyb v pravém sloupci:

```
4.000000000000000   8.38e-01
3.250000000000000   8.77e-02
3.163461538461538   1.18e-03
3.162277881692775   2.22e-07
3.162277660168387   7.55e-15
3.162277660168379   4.44e-16
```

Výsledné iterace a jejich absolutní chyby jsou zobrazeny níže:



Obrázek 2.5: Výpočet druhé odmocniny čísla 10.

Kód - Mathematica 2.10: Newtonova metoda.

```
d=10;
f[y_] = y^2 - d;
Df[y_] = f'[y];
x0 = 4;
n = 7;
x = Table[x0, {i, n + 2}];
Table[x[[i + 1]] = x[[i]] - f[x[[i]]]/Df[x[[i]]], {i, n + 1}];
error = Table[x[[i]] - x[[n + 2]], {i, n + 1}];
N[TableForm[Transpose[{Drop[x, -1], Abs[error]}]], 15]
```

Systém Mathematica počítá přesněji ve zlomcích a Kód 2.10 poskytuje následující výpis pro prvních 8 iterací:

```
{4.00000000000000, 0.837722339831621},
{3.25000000000000, 0.0877223398316207},
{3.16346153846154, 0.00118387829315913},
{3.16227788169278, 2.21524395977798*10^-7},
{3.16227766016839, 7.75913121004078*10^-15},
{3.16227766016838, 9.51910673324984*10^-30},
{3.16227766016838, 1.43272354196402*10^-59},
{3.16227766016838, 3.24559853417270*10^-119}
```

Prvních šest iterací v levém sloupci odpovídá zlomkům

$$\frac{4}{1}, \frac{13}{4}, \frac{329}{104}, \frac{216401}{68432}, \frac{93658779041}{29617506464}, \frac{17543933782901678712641}{5547878987314330442048},$$

další zlomky neuvádíme kvůli úspoře místa. V pravém sloupci je vidět, že chyby od šesté iterace (ta odpovídá šesté řádce) počítaje nejsou ve skutečnosti nulové.

Poznámka k příkladu

Dodatečný kód

Kód - Mathematica 2.11: Newtonova metoda - poměr chyb.

```
N[Table[error[[k + 1]]/error[[k]]^2, {k, 1, n}]]
```

počítá posloupnost poměrů absolutních chyb iterací ve tvaru

$$\frac{|e_{k+1}|}{|e_k|^2} = \frac{|x_{k+1} - \bar{x}|}{|x_k - \bar{x}|^2}, \tag{2.21}$$

ve které si všimněme druhých mocnin ve jmenovatelích zlomků. Ty tvoří v našem příkladě pro $k = 1, \ldots, 7$ posloupnost čísel

$$0.125,\ 0.153846,\ 0.158055,\ 0.158114,\ 0.158114, 0.158114,\ 0.158114,$$

která se zdá konvergovat. Proto usuzujeme, že Newtonova metoda konverguje kvadraticky, což je také, za určitých předpokladů, známý teoretický výsledek.
Nahrazením hodnoty 10 obecným číslem $d > 0$ ve vzorci (2.20) a v obou kódech získáváme známou Babylónskou (neboli Hérónovu) metodu výpočtu druhé odmocniny čísla d ve tvaru

$$x_k = \frac{1}{2}(x_{k-1} + \frac{d}{x_{k-1}}), \qquad k = 1, 2, \ldots . \tag{2.22}$$

Tu lze zobecnit i pro výpočty odmocnin vyšších řádů.

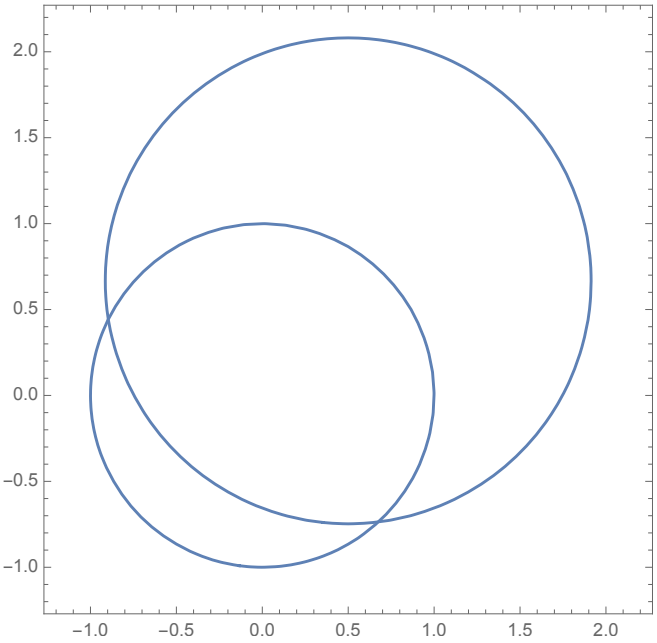
Příklad 2.2.21 Vyřešte pomocí Newtonovy metody soustavu nelineárních rovnic

$$f_1 : x^2 + y^2 - 1 = 0,$$
$$f_2 : (x - 1/2)^2 + (y - 2/3)^2 - 2 = 0.$$

Jako počáteční iteraci zvolte vektor

$$(x_0, y_0) = (1, -1).$$

Řešení: Cílem je nalezení jednoho ze dvou průsečíků dvou kružnic na obrázku níže.



Obrázek 2.6: Kružnice a jejich průsečíky.

Je třeba použít iterační formuli ve tvaru

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1}{\partial x}(x_{k-1}, y_{k-1}) & \frac{\partial f_1}{\partial y}(x_{k-1}, y_{k-1}) \\ \frac{\partial f_2}{\partial x}(x_{k-1}, y_{k-1}) & \frac{\partial f_2}{\partial y}(x_{k-1}, y_{k-1}) \end{pmatrix}^{-1} \begin{pmatrix} f_1(x_{k-1}, y_{k-1}) \\ f_2(x_{k-1}, y_{k-1}) \end{pmatrix}.$$

Tu také ekvivalentně můžeme přepsat do tvaru

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \end{pmatrix} + \begin{pmatrix} \Delta x_{k-1} \\ \Delta y_{k-1} \end{pmatrix}, \tag{2.23}$$

kde poslední vektor je řešením soustavy lineárních rovnic

$$\begin{pmatrix} \frac{\partial f_1}{\partial x}(x_{k-1}, y_{k-1}) & \frac{\partial f_1}{\partial y}(x_{k-1}, y_{k-1}) \\ \frac{\partial f_2}{\partial x}(x_{k-1}, y_{k-1}) & \frac{\partial f_2}{\partial y}(x_{k-1}, y_{k-1}) \end{pmatrix} \begin{pmatrix} \Delta x_{k-1} \\ \Delta y_{k-1} \end{pmatrix} = - \begin{pmatrix} f_1(x_{k-1}, y_{k-1}) \\ f_2(x_{k-1}, y_{k-1}) \end{pmatrix}.$$

Ta má v našem příkladě tvar

$$\begin{pmatrix} 2x_{k-1} & 2y_{k-1} \\ 2(x_{k-1} - \frac{1}{2}) & 2(y_{k-1} - \frac{2}{3}) \end{pmatrix} \begin{pmatrix} \Delta x_{k-1} \\ \Delta y_{k-1} \end{pmatrix} = \begin{pmatrix} 1 - x_{k-1}^2 - y_{k-1}^2 \\ 2 - (x_{k-1} - \frac{1}{2})^2 - (y_{k-1} - \frac{2}{3})^2 \end{pmatrix}.$$

V každé iteraci je tedy nutné (tzv. Jacobiho) matici soustavy a pravou stranu přepočítat.

V první iteraci (pro k=1) dostáváme

$$\begin{pmatrix} 2 & -2 \\ 1 & -\frac{10}{3} \end{pmatrix} \begin{pmatrix} \Delta x_0 \\ \Delta y_0 \end{pmatrix} = \begin{pmatrix} -1 \\ -\frac{37}{36} \end{pmatrix}.$$

a jejím vyřešením a dosazením do (2.23) iteraci

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} + \begin{pmatrix} -\frac{23}{84} \\ \frac{19}{84} \end{pmatrix} = \begin{pmatrix} \frac{61}{84} \\ -\frac{65}{84} \end{pmatrix} \approx \begin{pmatrix} 0.72619047619047619048 \\ -0.77380952380952380952 \end{pmatrix}.$$

Tuto a další iterace lze vygenerovat pomocí kódu níže.

Kód - Mathematica 2.12: Newtonova metoda pro průnik kružnic.

```
f[x_, y_] = {x^2 + y^2 - 1, (x - 1/2)^2 + (y - 2/3)^2 - 2};
{x0, y0} = {1, -1};

Df[x_, y_] = Grad[f[x, y], {x, y}] ;
i = 0; Print[i, ": ", N[{x0, y0}, 20]]
Do[
 Delta = -Inverse[Df[x0, y0]] . f[x0, y0];
 {x1, y1} = {x0, y0} + Delta ;
 {x0, y0} = {x1, y1};
 Print[i, ": ", N[{x0, y0}, 20]], {i, 1, 6}
 ]
```

Získáváme

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \approx \begin{pmatrix} 0.67792059876342336479 \\ -0.73760711573923419026 \end{pmatrix},$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \approx \begin{pmatrix} 0.67644203539645503210 \\ -0.73649819321400794074 \end{pmatrix},$$

$$\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} \approx \begin{pmatrix} 0.67644064549768834253 \\ -0.73649715078993292357 \end{pmatrix},$$

$$\begin{pmatrix} x_5 \\ y_5 \end{pmatrix} \approx \begin{pmatrix} 0.67644064549646013886 \\ -0.73649715078901177081 \end{pmatrix}$$

a tedy poslední uvedená iterace popisuje souřadnice hledaného průsečíku. Lze ukázat, že další šestá iterace je identická páté iteraci uvedené výše, tedy všechny zobrazené cifry jsou stejné.

Poznámka k příkladu

Změnou počáteční iterace na např.

$$(x_0, y_0) = (-1, 1)$$

lze dokonvergovat k druhému průsečíku (ověřte).

Příklad 2.2.22 Vykreslete Newtonův fraktál pro funkci

$$f(z) = z^3 - 1$$

Řešení: Newtonův fraktál vykresluje počty iterací Newtonovy metody pro rovnici

$$f(z) = 0.$$

Newtonova metoda je dána rekurentní formulí (2.19), která v případě dané funkce má tvar

$$z_k := z_{k-1} - \frac{z^3 - 1}{3z^2}, \qquad k = 1, 2, \ldots \tag{2.24}$$

pro nějaké počáteční $z_0 \in \mathbb{C}$. Například v případě $z_0 = 1 + i$ získáváme posloupnost iterací

$$\{1 + i,\ 0.66 + i\,0.5,\ 0.57 - i\,0.12,\ 1.24 + i\,0.31,\ 1 + i\,0.11,\ 0.98,\ 1\},$$

ve které jsme pro přehlednost zapsali reálné a imaginární složky na pouhé dvě platné číslice za desetinnou čárkou. Uvedená posloupnost zřejmě konverguje prvnímu z kořenů funkce $f(z)$, tedy hodnot

$$\bar{z}_1 = 1, \quad \bar{z}_2 = e^{(2i\pi)/3} \approx -0.5 + i\,0.86, \quad \bar{z}_3 = e^{-(2i\pi)/3} \approx -0.5 - i\,0.86.$$

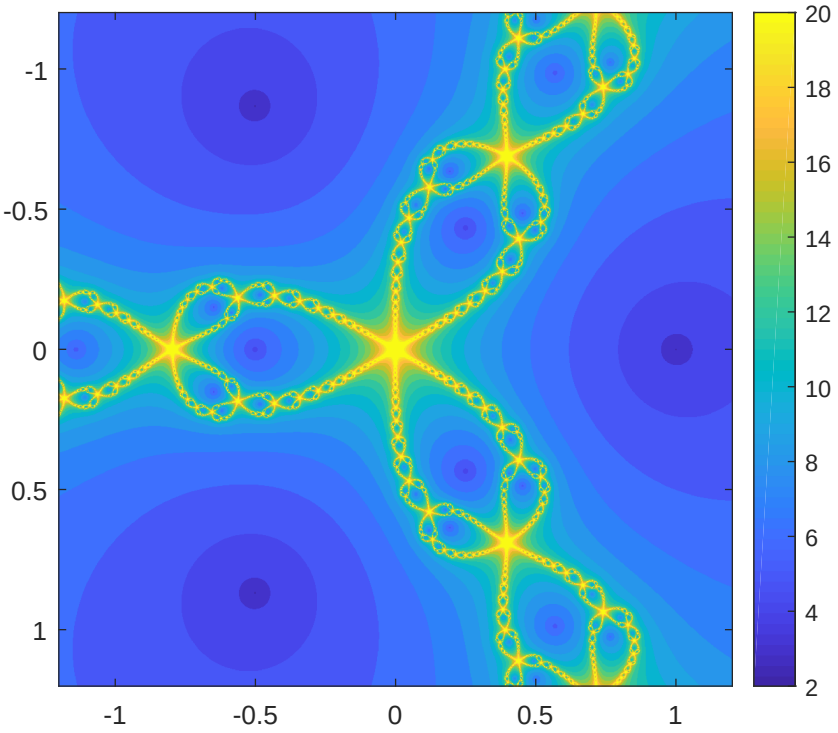
Ke druhým dvěma kořenům lze dokonvergovat (ukažte) například z počátečních hodnot $z_0 = i$ a $z_0 = -i$.
Následující program je modifikací kódu uvedeném v [4] a realizuje Newtonovu metodu pro jeden milion počátečních bodů rozmístěných rovnoměrně ve čtvercové oblasti:

```
xMin=-1.2; xMax=1.2;                    % obdelnikova oblast
yMin=-1.2; yMax=1.2;
np=1000;                                % pocet bodu na osach
niter = 20;                             % pocet iteraci
tol = 1e-5;                             % tolerance convergence
f= @(z) z.^3 -1;                        % funkce
Df=@(z) 3*z.^2;                         % a jeji derivace

xx=linspace(xMin,xMax,np);              % diskretizace osy x
yy=linspace(yMin,yMax,np);              % diskretizace osy y
[x,y] = meshgrid(xx,yy);                % matice bodu x, y
zV = x(:) + 1i*y(:);                    % vektor bodu z
select = 1:numel(x);                    % vybrane indexy
niters = niter*ones(numel(x),1);        % pocty iteraci

for iter = 1:niter
    z0V = zV(select);
    zV(select)=z0V-f(z0V)./Df(z0V);     % Newtonovy iterace
    differ = abs(z0V - zV(select));     % rozdily iteraci
    converged = differ < tol;           % konverguje?
    niters(select(converged)) = iter;   % prirad pocet iteraci
    diverged = isnan(differ);           % diverguje?
    niters(select(diverged)) = niter+1; % nastav maximalni iter.
    select(converged | diverged) = [];  % zredukuj indexy
end
niters = reshape(niters,size(x));       % preved na matici
z = reshape(zV,size(x));                % preved na matici
figure;
imagesc(niters,'XData',[xMin,xMax],...
                'YData',[yMin,yMax]);   % vykresli fraktal
axis image; colorbar;
```

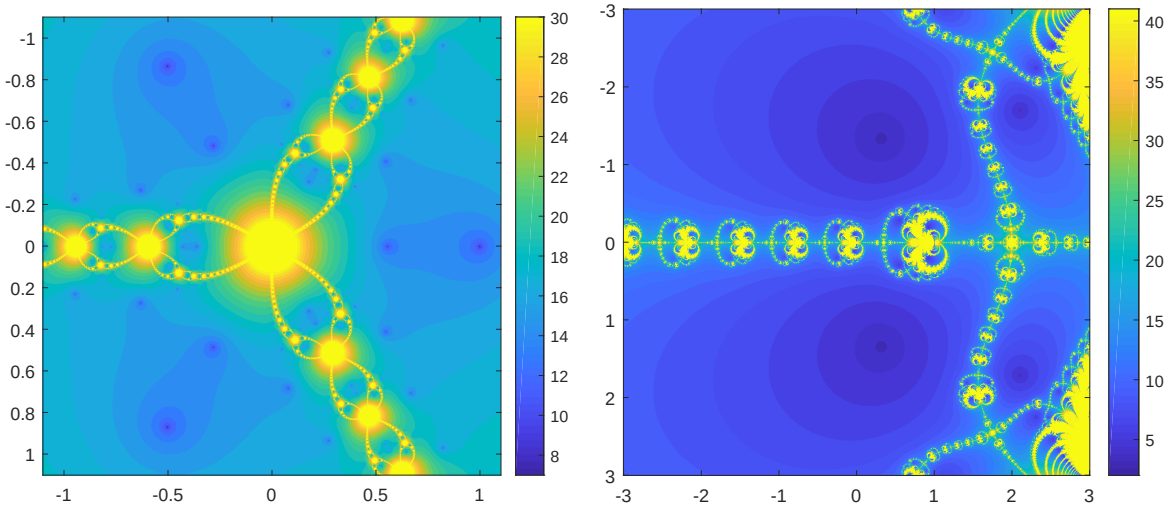
Výsledkem je Newtonův fraktál zobrazený na Obrázku 2.7. Modré zóny označují nízký počet iterací k dosažení konvergence, oproti jim žluté naopak vysoký počet iterací nebo dokonce divergenci metody, například z důvodu dělení nulou.

Poznámka k příkladu

Modifikací programu lze jednoduše získat další zajímavé fraktály, viz Obrázek 2.8.



Obrázek 2.7: Newtonův fraktál pro funkci $f(z) = z^3 - 1$.



Obrázek 2.8: Newtonovy fraktály pro funkce $f(z) = z^6 - 2z^3 + 1$ (vlevo) a $f(z) = z\,e^{-z} - 1$ (vpravo).

Úloha k samostatnému řešení

Pomocí metody bisekce najděte řešení rovnice

$$2\sin(4x) = 3\sin(2x) \tag{2.25}$$

na intervalu $I = [0.1, 0.5]$ s přesností na 5 desetinných míst. Potom ukažte pomocí trigonometrických identit, že přesné řešení je rovno $x = \arctan \frac{1}{\sqrt{7}}$.

Úloha k samostatnému řešení

Vypočtěte hodnotu $\sqrt[3]{10}$ pomocí vhodné Newtonovy metody. Jako počáteční iteraci zvolte $x_0 = 3$. Napište iterační formuli a vypište prvních pět iterací. Určete absolutní chyby jednotlivých iterací.

Úloha k samostatnému řešení

Vypočtěte řešení rovnice

$$x^x = 5$$

pomocí Newtonovy metody. Jako počáteční iteraci zvolte $x_0 = 3$. Napište iterační formuli a vypište prvních pět iterací. Určete absolutní chyby jednotlivých iterací.

Poznámka: Ve výpočtu Newtonovy metody potřebujete znát derivaci funkce

$$f(x) = x^x - 5.$$

Jeden student ji při zkoušce stanovil jako

$$f'(x) = (x-1)x^x,$$

což není správně! Ukažte, že i s takto špatně určenou derivací Newtonova metoda přesto konverguje, ale pomaleji.

2.3 LU rozklad matice

Příklad 2.3.23 Najděte LU rozklad matice

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 3 & 3 \\ 2 & 5 & 6 & 6 \\ 4 & 7 & 9 & 10 \end{pmatrix}$$

a ten využijte k řešení systému rovnic $Ax = b$, kde pravá strana je

$$b = (1,1,1,1)^T.$$

Řešení: Rozklad předpokládá nalezení dolní trojúhelníkové matice L a horní trojúhelníkové matice U tak, že

$$A = L\,U.$$

Sestavení rozkladu je ekvivalentní Gaussově eliminaci bez prohazování řádek.
Formálně napíšeme

$$A = L_1\,U_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 3 & 3 \\ 2 & 5 & 6 & 6 \\ 4 & 7 & 9 & 10 \end{pmatrix},$$

kde tedy $L_1 = I$ (identická matice) a $U_1 = A$.
Nyní vynásobíme celý první řádek matice U_1 multiplikátorem -2 a přičteme k druhému řádku, potom multiplikátorem -2 a přičteme k třetímu řádku a nakonec multiplikátorem -4 a přičteme k čtvrtému řádku. Ověříme, že platí

$$A = L_2\,U_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 4 & 4 \\ 0 & 3 & 5 & 6 \end{pmatrix}$$

a všimněme si, že se opačné (proč?) hodnoty multiplikátorů objevily v prvním sloupci matice L_2 pod diagonálou.
Dále vynásobíme druhý řádek matice U_2 multiplikátorem -3 a přičteme k třetímu řádku a multiplikátorem -3 a přičteme ke čtvrtému řádku. Tím jsme získali rozklad (ověřte)

$$A = L_3\,U_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 4 & 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 2 & 3 \end{pmatrix}.$$

Zbývá vynásobit třetí řádek matice U_3 hodnotou -2 a přičíst k čtvrtému řádku. Získáváme (zase ověřte)

$$A = L_4\,U_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

což představuje požadovaný rozklad matice A.
V počítači můžeme všechny postupné rozklady vygenerovat pomocí kódu:

Kód - Matlab 2.13: LU rozklad matice.

```
format rat
A=[1 1 1 1; 2 3 3 3; 2 5 6 6; 4 7 9 10];          % matice m x n
[m,n]=size(A);                                    % velikost
L = eye(m); U = A;                                % inicializace L, U
for j=1:m-1
    ii=j+1:m;                                     % indexy radek
    L(ii,j)=U(ii,j)/U(j,j)                        % multiplikatory do L
    jj=j:n;                                       % indexy sloupcu
    U(ii,jj)=U(ii,jj)-L(ii,j)*U(j,jj)             % prepocet U
    fprintf('---\n')
end
```

Díky sestavenému rozkladu můžeme nyní řešení soustavy rovnic

$$LUx = b$$

převedením na postupné řešení dvou jednodušších soustav rovnic:

1. nejprve vyřešíme pro pomocný vektor y soustavu

$$Ly = b \qquad \text{(dopředná substituce)},$$

2. poté vyřešíme pro vektor řešení x soustavy

$$Ux = y \qquad \text{(zpětná substituce)}.$$

Pro náš konkrétní příklad tedy nejdříve řešíme soustavu dopředné substituce

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

a složky vektoru $y = (y_1, y_2, y_3, y_4)^T$ hledáme v rostoucím pořadí indexů postupně z 1. až 4. rovnice. Tím získáme vektor

$$y = (1, -1, 2, -4)^T,$$

který poté vložíme jako pravou stranu v soustavě zpětné substituce

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 2 \\ -4 \end{pmatrix}$$

a složky vektoru $x = (x_1, x_2, x_3, x_4)^T$ hledáme v klesajícím pořadí indexů postupně z 4. až 1. rovnice. Získáme vektor

$$x = (2, -3, 6, -4)^T,$$

který představuje řešení rovnice $Ax = b$. Programově lze obě substituce napsat takto:

Kód - Matlab 2.14: LU rozklad matice: dopředná a zpětná substituce.

```
b=ones(n,1);                                      %vektor prave strany

y=zeros(n,1);                                     % inicializace
for j=1:n                                         % dopredna substituce
    jj=1:(j-1);                                   % indexy sloupcu
    y(j)=b(j)-dot(L(j,jj),y(jj));
```

```
7  end
8
9  x=zeros(n,1);                                    % inicializace
10 for i=n:-1:1                                     % zpetna substituce
11     x(i)=y(i);
12     ii=n:-1:(i+1);
13     for j=n:-1:(i+1)
14         x(i)=x(i)-U(i,j)*x(j);
15     end
16     x(i)=x(i)/U(i,i);
17 end
```

Poznámka k příkladu

Obecně neplatí, že lze postupem výše LU rozklad sestavit pro libovolnou regulární matici. Např. pro matici

$$A = \begin{pmatrix} 0 & 3 \\ 5 & 2 \end{pmatrix}$$

je zřejmé, že první řádek můžeme vynásobit jakýmkoliv multiplikátorem a přičíst k druhému řádku, ale hodnoty 5 v druhé řádce a prvním sloupci se nezbavíme a Kód 2.13 narazí na dělení nulou. Klasický rozklad zde nefunguje, ale je možné implementovat obecnější rozklad s tzv. pivotací ve tvaru

$$PA = LU,$$

který navíc pracuje s permutační maticí P. Přenásobením A zleva permutační maticí

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

prohazujeme oba řádky matice A a pro matici

$$PA = \begin{pmatrix} 5 & 2 \\ 0 & 3 \end{pmatrix}$$

už LU rozklad máme již hotový, protože PA je v horním trojúhelníkovém tvaru. Tedy platí $L = I$ a $U = PA$. V praxi tím numericky ošetříme možné dělení nejen nulou, ale i malými čísly a dosáhneme vyšší stability výpočtu. O technice pivotace se lze dočíst v literatuře a je běžně implementována ve všech matematických systémech. Například v Matlabu dosáhneme rozkladu pomocí příkazu:

```
[L,U,P]=lu([0 3; 5 2])
```

Na závěr upozorněme, že LU rozklad lze uvažovat i pro obdelníkové matice a Kód 2.13 funguje a např. po modifikaci matice A v 2. řádce za matici

```
A=[1 2 -3 1; 2 5 0 7; -1 3 2 0];
```

najde její rozklad ve tvaru

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & -3 & 1 \\ 0 & 1 & 6 & 5 \\ 0 & 0 & -31 & -24 \end{pmatrix}.$$

Úloha k samostatnému řešení

Najděte řešení $x \in \mathbb{R}^n$ rovnice $A_n x_n = b$ pomocí LU-rozkladu pro matici $A_n \in \mathbb{R}^{n\times n}$ a vektor $b \in \mathbb{R}^n$ ve tvaru

$$A_n := (n+1)^2 \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

Nalezněte hodnoty největší složky vektoru x_n pro jednotlivé případy $n \in \{10, 100, 1000\}$. Poznámka: nevypsané prvky matice A jsou vždy nulové!

Úloha k samostatnému řešení

Najděte inverzní matice k maticím L_4, U_4 z příkladu 2.3.23 přímým výpočtem z definice, tedy nalezením koeficientů v rovnicích.

$$L_4\, L_4^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} c_{1,1} & 0 & 0 & 0 \\ c_{2,1} & c_{2,2} & 0 & 0 \\ c_{3,1} & c_{3,2} & c_{3,3} & 0 \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$U_4\, U_4^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} d_{1,1} & d_{1,2} & d_{1,3} & d_{1,4} \\ 0 & d_{2,2} & d_{2,3} & d_{2,4} \\ 0 & 0 & d_{3,3} & d_{3,4} \\ 0 & 0 & 0 & d_{4,4} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Tím si připomeneme známý fakt, že inverze dolní nebo horní trojúhelníkové matice má stejný trojúhelníkový tvar. Potom pomocí nich vypočtěte inverzní matici $A^{-1} = U_4^{-1} L_4^{-1}$.

Úloha k samostatnému řešení

Ukažte, že přepisování matice U v LU-rozkladu vyžaduje pro matici $n \times n$ celkem

$$\sum_{k=2}^{n} k(k-1) = \frac{1}{3}(n-1)n(n+1)$$

součtů a stejný počet součinů různých čísel.

Úloha k samostatnému řešení

Najděte horní trojúhelníkovou matici U tak, aby platilo

$$\begin{pmatrix} 16 & 12 & -4 & -20 \\ 12 & 18 & 6 & -21 \\ -4 & 6 & 19 & -4 \\ -20 & -21 & -4 & 34 \end{pmatrix} = \begin{pmatrix} u_{1,1} & 0 & 0 & 0 \\ u_{1,2} & u_{2,2} & 0 & 0 \\ u_{1,3} & u_{2,3} & u_{3,3} & 0 \\ u_{1,4} & u_{2,4} & u_{3,4} & u_{4,4} \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{pmatrix}.$$

Jde o příklad Choleského rozkladu $A = U^T U$ symetrické pozitivně definitní matice A.

2.4 Interpolace

Příklad 2.4.24 Pro Rungeho funkci

$$f(x) = \frac{1}{1+25x^2}, \quad x \in I = [-1,1]$$

uvažujme interpolační polynom $p_n(x)$ stupně $n \in \mathbb{N}_0$, který prochází rovnoměrně rozloženými body na intervalu I. Sestrojte konkrétní interpolační polynomy

$$p_2(x),\ p_4(x),\ p_6(x),\ p_8(x)$$

a spočtěte jejich odchylky od funkce f.

Řešení:
Konstrukci sestavení jednotlivých polynomů si ukážeme nejdříve pro případ polynomu

$$p_2(x) = a_0 + a_1 x + a_2 x^2,$$

jehož 3 koeficienty $a_0, a_1, a_2 \in \mathbb{R}$ musíme najít. Polynom $p_2(x)$ musí ve 3 uzlových bodech $x_0, x_1, x_2 \in I$ dosahovat hodnot funkce $f(x)$, tedy splňovat podmínky

$$\begin{aligned} a_0 + a_1 x_0 + a_2 x_0^2 &= f(x_0), \\ a_0 + a_1 x_1 + a_2 x_1^2 &= f(x_1), \\ a_0 + a_1 x_2 + a_2 x_2^2 &= f(x_2). \end{aligned}$$

Tento systém rovnice představuje soustavu lineárních rovnic ve tvaru

$$\begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \end{pmatrix}.$$

Volba konkrétních rovnoměrně rozložených uzlových bodů (pro indexy 0, 1, 2) a v nich odpovídajících hodnot funkce f je uvedena v tabulce:

x_i	-1	0	1
$f(x_i)$	1/26	1	1/26

Tabulka 2.1: Hodnoty uzlů $i = 0, 1, 2$ pro výpočet $p_2(x)$.

Proto získáváme konkrétní systém rovnic

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1/26 \\ 1 \\ 1/26 \end{pmatrix}.$$

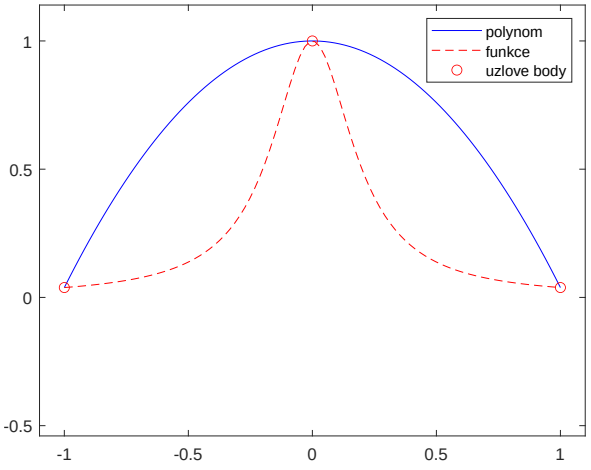
Jeho řešením je $a_0 = 1, a_1 = 0, a_2 = -25/26$ a tedy tvar interpolačního polynomu je

$$p_2(x) = 1 - \frac{25}{26}x^2. \tag{2.26}$$

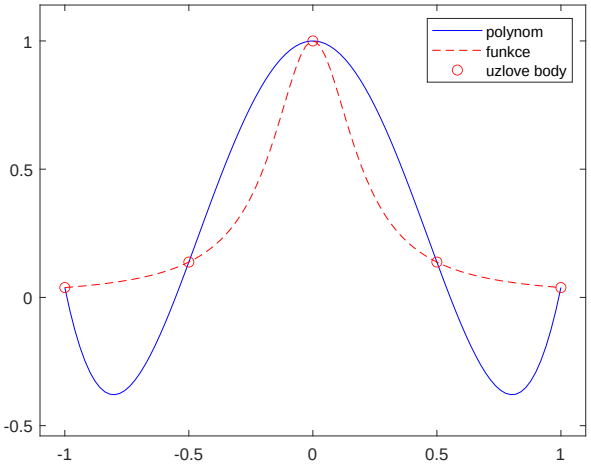
Funkce f a její interpolační polynom p_2 jsou vykresleny vlevo na Obrázku 2.9 (a).
Jejich vzájemnou vzdálenost neboli metriku lze vyjádřit v takzvané max-normě pomocí výrazu

$$\max_{x\in[-1,1]} |f(x) - p_2(x)| = \sqrt{1/676}(27 - 2\sqrt{26}) \approx 0.6462292. \tag{2.27}$$

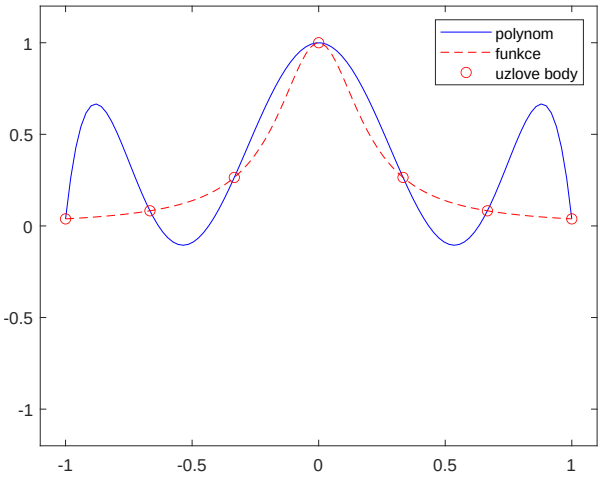
Obrázek a přibližnou hodnotu vzdálenosti (2.27) lze získat pomocí kódu:



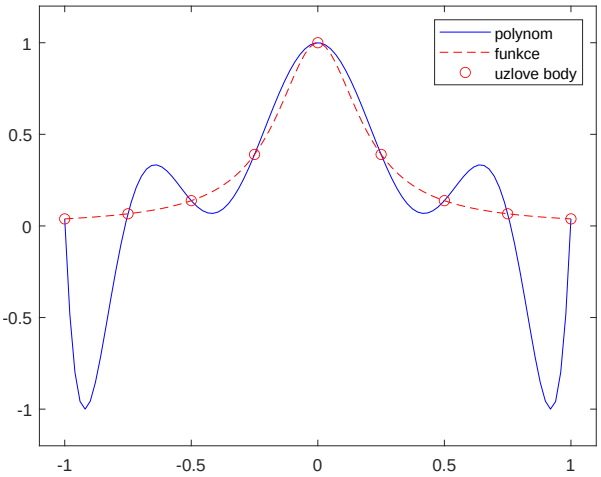
(a) Interpolace polynomem $p_2(x)$.



(b) Interpolace polynomem $p_4(x)$.

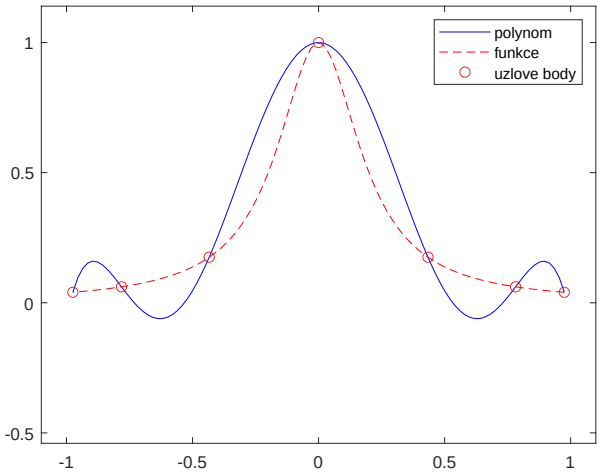


(c) Interpolace polynomem $p_6(x)$.

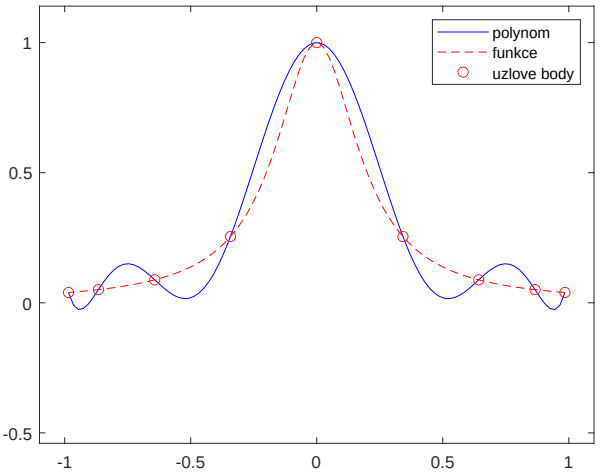


(d) Interpolace polynomem $p_8(x)$.

Obrázek 2.9: Funkce $f(x)$ a její interpolační polynomy $p_2(x), p_4(x), p_6(x), p_8(x)$ v případě rovnoměrně rozložených uzlových bodů.



(a) Interpolace polynomem $p_6(x)$.



(b) Interpolace polynomem $p_8(x)$.

Obrázek 2.10: Funkce $f(x)$ a její interpolační polynomy $p_6(x), p_8(x)$ v případě nerovnoměrně rozložených uzlových bodů generovaných pomocí kořenů Čebyševových polynomů.

Kód - Matlab 2.15: Polynomiální interpolace.

```
1  format rational
2  d=2;                                          % stupen polynomu
3  a=-1; b=1;                                    % interval
4  f=@(x)1./(1+25*x.^2);                         % Rungeho funkce
5  n=d+1;                                        % pocet uzlu
6  x=linspace(a,b,n);                            % rovnomerne body
7  %x=(a+b)/2+(b-a)*cos((2*(1:n)-1)*pi/2/n)/2;   % Chebysevovy body
8  y=f(x);                                       % funkce v bodech
9  M = bsxfun(@power,x',d:-1:0);                 % matice
10 p=transpose(M\y')                             % koeficienty polynomu
11 xx=linspace(x(1),x(end));
12 yy=polyval(p,xx);                             % vykreslovani
13 plot(xx,yy,'b',xx,f(xx),'r--',x,y,'ro');
14 legend('polynom','funkce','uzlove body')
15 fprintf('odchylka=%d\n',norm(f(xx)-yy,'inf')) % vypocet odchylky
```

Pozorný čtenář si všimne, že hodnotu (2.27) nehledáme analyticky, ale počítáme jen z diskrétních bodů 'xx' generovaných na 10. řádce pomocí funkce „linspace“, která generuje ve výchozím nastavení 100 rovnoměrně rozložených diskrétních bodů na intervalu $[a, b]$. Takové nahrazení není úplně přesné, poskytuje totiž jen spodní odhad hodnoty maxima (2.27), ten pro naše účely zcela dostačuje.
Modifikací kódu na hodnoty d=4, d=6, nebo d=8 získáme další hledané polynomy:

$$p_4(x) = 1 - \frac{3225}{754}x^2 + \frac{1250}{377}x^4,$$
$$p_6(x) = 1 - \frac{773}{88}x^2 + \frac{985}{47}x^4 - \frac{2824}{125}x^6,$$
$$p_8(x) = 1 - \frac{2601}{197}x^2 + \frac{7855}{128}x^4 - \frac{20563}{200}x^6 + \frac{5530}{103}x^8.$$

Jejich odchylky od funkce $f(x)$ jsou dané hodnotami

$$\max_{x\in[-1,1]} |f(x) - p_4(x)| \approx 0.4382729,$$
$$\max_{x\in[-1,1]} |f(x) - p_6(x)| \approx 0.6164016,$$
$$\max_{x\in[-1,1]} |f(x) - p_8(x)| \approx 1.045078.$$

Na Obrázku 2.9 (b), (c), (d) jsou pak zobrazeny výsledné polynomy $p_4(x), p_6(x), p_8(x)$ a je u nich zřejmá velká odchylka od funkce $f(x)$ na zejména na krajích intervalu $[-1, 1]$.

Poznámka k příkladu

Pro Rungeho funkci f jsme právě ukázali, že zvyšování stupně polynomu nevede ke snižování odchylky v případě rovnoměrného rozložení uzlových bodů. Obecně lze k dané spojité funkci na omezeném uzavřeném intervalu najít libovolně přesnou polynomiální aproximaci, což říká známá Weierstrassova aproximační věta. Jak tedy kvalitu našich interpolačních polynomů vylepšit? Řešením je například volba nerovnoměrně rozložených uzlových bodů ve tvaru kořenů Čebyševových polynomů

$$T_n(x) = \cos(n \arccos(x)), \qquad x \in [-1, 1]. \tag{2.28}$$

Ty detailněji nepopisujeme, pro praktické použití stačí v Kódu 2.15 odkomentovat řádku 7. Přepočítané polynomy $p_6(x), p_8(x)$ jsou pak ukázány na Obrázku 2.10.

Poznámka k příkladu

Koeficienty aproximačního polynomu $p_2(x)$ nemusíme hledat jen řešením soustavy lineárních rovnic. V příkladu je hledáme z bodů uvedených v Tabulce 2.1, připomínáme, že tedy hodnot

x_i	-1	0	1
$f(x_i)$	1/26	1	1/26

.

Polynom (2.26) lze alternativně vyjádřit v Lagrangeově tvaru

$$L_2(x) = \frac{1}{26}\,\frac{(x-0)(x-1)}{(-1-0)(-1-1)} + 1\,\frac{(x-(-1))(x-1)}{(0-(-1))(0-1)} + \frac{1}{26}\,\frac{(x-(-1))(x-0)}{(1-(-1))(1-0)}$$
$$= \frac{1}{26}\frac{x^2-x}{2} + 1\frac{x^2-1}{-1} + \frac{1}{26}\frac{x^2+x}{2},$$

nebo Newtonově tvaru

$$N_2(x) = \frac{1}{26} + \frac{\frac{1}{26}-1}{-1-0}(x-(-1)) + \left(\frac{\frac{1}{26}-1}{(-1-0)(-1-1)} - \frac{1-\frac{1}{26}}{(0-1)(-1-1)}\right)(x-(-1))(x-0)$$
$$= \frac{1}{26} + \frac{25}{26}\,(x+1) - \frac{25}{26}\,(x+1)x$$

interpolačního polynomu, přičemž modře zvýrazněné hodnoty jsou vzaté přímo z tabulky. Úpravou obou tvarů obdržíme hledaný polynom $p_2(x) = 1 - \frac{25}{26}x^2$ z příkladu.

Úloha k samostatnému řešení

Pro Rungeho funkci vypočtěte interpolační polynom $p_4(x)$ procházející body

$$x \in \{-1, -1/2, 0, 1/2, 1\}$$

pomocí Lagrangeova a Newtonova tvaru.

Úloha k samostatnému řešení

Pro funkci

$$f(x) = \cos(\pi x), \quad x \in [-1, 1]$$

volte rovnoměrně rozložené uzlové body a pro odpovídající interpolační polynomy stupňů 2, 4, 6, 8, 10, 12 přepočtěte jejich hodnoty vzdáleností od funkce f.

Úloha k samostatnému řešení

Dokažte rekurentní formuli pro Čebyševovy polynomy ve tvaru

$$T_{n+2}(x) = 2x\,T_{n+1}(x) - T_n(x), \qquad x \in [-1, 1], n \in \mathbb{N}_0$$

zavedením substituce $x = \cos\theta$ do definice (2.28) a vhodnou úpravou výrazů

$$T_{n+2}(x) = \cos((n+2)\,\theta), \quad T_n(x) = \cos(n\,\theta).$$

Vygenerujte prvních pět Čebyševových polynomů a nalezněte jejich kořeny.

Úloha k samostatnému řešení

Pro Rungeho funkci na intervalue $[-1,1]$ volte nerovnoměrně rozložené uzlové body ve tvaru kořenů Čebyševových polynomů. Pro odpovídající interpolační polynomy stupňů 2, 4, 6, 8, 10, 12 přepočtěte jejich hodnoty vzdáleností od Rungeho funkce.

Příklad 2.4.25 Nalezněte vhodnou interpolaci Rungeho funkce

$$f(x) = \frac{1}{1+25x^2}, \quad x \in I = [-1,1]$$

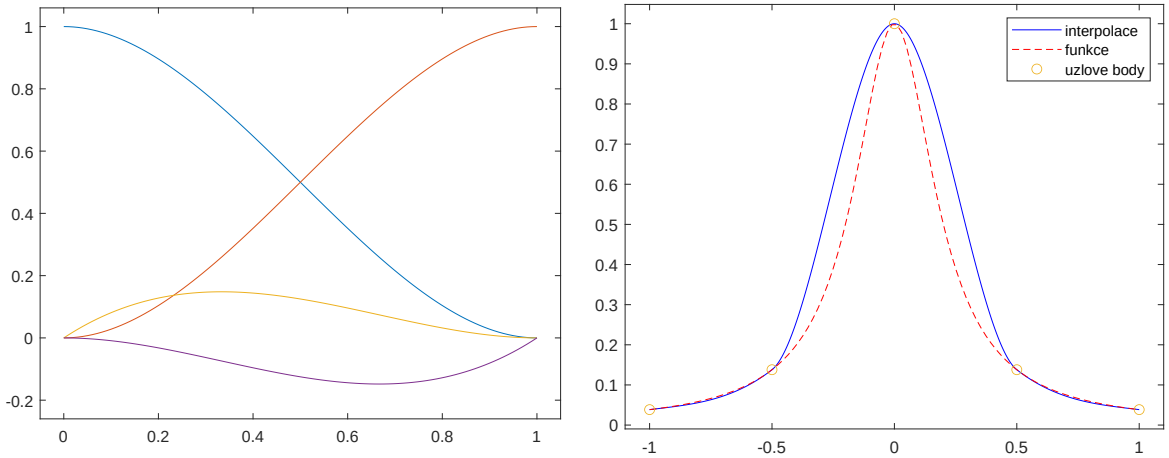
danou hodnotami funkce a její první derivace podle tabulky:

x	-1	-1/2	0	1/2	1
$f(x)$	1/26	4/29	1	4/29	1/26
$f'(x)$	25/338	400/841	0	-400/841	-25/338

Řešení: Kubické polynomy

$$\varphi_1(t) = 1 - 3t^2 + 2t^3, \qquad \varphi_2(t) = 3t^2 - 2t^3,$$
$$\varphi_3(t) = t - 2t^2 + t^3, \qquad \varphi_4(t) = -t^2 + t^3;$$

a jsou ukázány pro $t \in [0,1]$ na Obrázku 2.11 (vlevo).



(a) Grafy polynomů $\varphi_1, \varphi_2, \varphi_3, \varphi_4$.

(b) Interpolace po částech kubickým polynomem.

Obrázek 2.11: Kubické polynomy na intervalu $[0,1]$ (vlevo) a výsledná interpolace (vpravo), ta má s původní funkcí stejné hodnoty funkce a první derivace v uzlových bodech.

Mají zajímavou vlastnost: pro libovolné $k \in \{1,2,3,4\}$ je vždy jen jedna z hodnot

$$\varphi_k(0),\ \varphi_k(1),\ \varphi_k'(0),\ \varphi_k'(1),$$

rovná jedné a ostatní nulové. Jejich lineární kombinací ve tvaru

$$\varphi(t) = \sum_{k=1}^{4} c_k \varphi_k\left(\frac{x-a}{b-a}\right)$$

získáme opět kubický polynom $\varphi : [a,b] \to \mathbb{R}$, který splňuje podmínky (ověřte)

$$c_1 = \varphi(a), \quad c_2 = \varphi(b), \quad c_3 = \varphi'(a)(b-a), \quad c_4 = \varphi'(b)(b-a).$$

Proto lze funkci danou hodnotami v tabulce interpolovat na 4 intervalech

$$[a^i, b^i], \qquad i = 1, \ldots, 4$$

pomocí po částech kubické funkce s koeficienty

$$c_1^i,\ c_2^i,\ c_3^i,\ c_4^i, \quad i = 1, \ldots, 4$$

uvedenými v tabulce níže:

i	a^i	b^i	c_1^i	c_2^i	c_3^i	c_4^i
1	-1	-1/2	1/26	4/29	25/676	200/841
2	-1/2	0	4/29	1	200/841	0
3	0	1/2	1	4/29	0	-200/841
4	1/2	1	4/29	1/26	-200/841	-25/676

K vypočtení a vykreslení výsledné po částech kubické funkce zobrazené na Obrázku 2.11 (vpravo) slouží program níže.

Kód - Matlab 2.16: Po částech kubická interpolace.

```
n=5; a=-1; b=1;
f=@(x)(1./(1+25*x.^2));          % funkce
Df=@(x)-50*x./(1+25*x.^2).^2;    % a jeji derivace
x=linspace(a,b,n); y=f(x); Dy=Df(x);

p=[2,-3,0,1; -2,3,0,0; 1,-2,1,0; 1,-1,0,0];
phi1=@(t)polyval(p(1,:),t); phi2=@(t)polyval(p(2,:),t);
phi3=@(t)polyval(p(3,:),t); phi4=@(t)polyval(p(4,:),t);
phi=@(t)[phi1(t); phi2(t); phi3(t); phi4(t)];

xx_ref=linspace(0,1); xx=[]; phiphi=[];
for i=1:numel(x)-1
    c=[y(i) y(i+1) Dy(i)*(x(i+1)-x(i)) Dy(i+1)*(x(i+1)-x(i))]
    phiphi=[phiphi c*phi(xx_ref)];
    xx_i=((x(i+1)-x(i))*xx_ref+x(i)); xx=[xx xx_i];
end
figure; xx_ab=linspace(a,b);
plot(xx,phiphi,'b-',xx_ab,f(xx_ab),'r--',x,y,'o')
legend('interpolace','funkce','uzlove body')
```

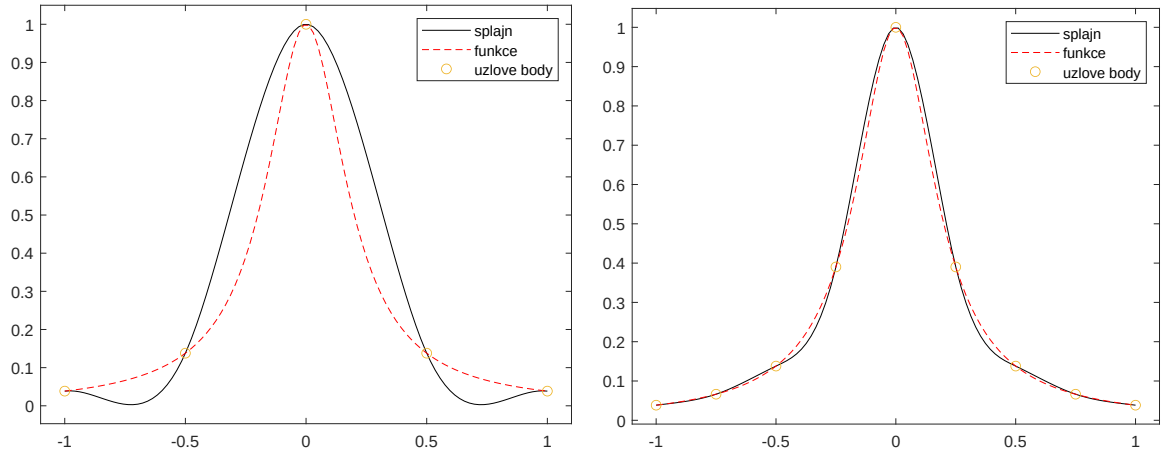
Připomeňme, že výsledná interpolační funkce má s Rungeho funkcí shodné funkční hodnoty a hodnoty první derivace v uzlových bodech. Jedná se tedy o funkci se spojitou první derivací na celém intervalu $[-1,1]$.

Poznámka k příkladu

K danému příkladu je možné také sestavit tzv. kubický Hermitův splajn. Ten využívá jen hodnoty původní funkce v interpolovaných bodech. Hodnoty první derivace funkce nepotřebuje dopředu znát a dopočítává je tak [1], aby výsledný splajn měl dokonce druhou derivaci spojitou ve vnitřních uzlových bodech.
Dopočet vyžaduje řešit systém lineárních rovnic pro koeficienty

$$c_1^i,\ c_2^i,\ c_3^i,\ c_4^i, \qquad i = 1, \ldots, n,$$

kde n představuje počet jednotlivých podintervalů. Aby byl systém lineárních rovnic jednoznačně určen, volí se dodatečné podmínky v hraničních uzlových bodech, například hodnoty prvních derivací. Pomocí dodatečného kódu



(a) $n = 5$. (b) $n = 9$.

Obrázek 2.12: Kubický Hermitův splajn v případě 5 a 9 uzlových bodů.

Kód - Matlab 2.17: Kubický Hermitův splajn.

```
figure;
s = spline(x,[Dy(1) y Dy(end)],xx_ab);
plot(xx_ab,s,'k-',xx_ab,f(xx_ab),'r--',x,y,'o')
legend('spline','funkce','uzlove body')
```

můžeme výsledný splajn vypočítat a vizualizovat, viz Obrázek 2.12.

Úloha k samostatnému řešení

Nalezněte polynom p_3 který splňuje podmínky:

$$p(2) = 1, \quad p(5) = 2, \quad p'(2) = 1, \quad p'(5) = -1.$$

Úloha k samostatnému řešení

Pro Rungeho funkci najděte kubický Hermitův splajn, který prochází 3, 4, 5, 10, 20 rovnoměrně rozloženými body na intervalu $[-1, 1]$ a spočtěte jejich odchylky od funkce f.

2.5 Aproximace metodou nejmenších čtverců

Příklad 2.5.26 Pro funkci

$$f(x) = \cos(\pi x)$$

nalezněte kvadratický polynom

$$p_2(x) = a_0 + a_1 x + a_2 x^2,$$

který funkci $f(x)$ aproximuje na intervalu $I = [-1, 1]$ ve smyslu nejmenších čtverců v 5 rovnoměrně rozložených bodech.

Řešení: Označme 5 rovnoměrně rozložených bodů jako

$$x_1 = -1, \quad x_2 = -1/2, \quad x_3 = 0, \quad x_4 = 1/2, \quad x_5 = 1.$$

Soustava lineárních rovnic interpolačního typu ve tvaru

$$\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{pmatrix}$$

nemá řešení (vysvětlete). Proto přecházíme k systému normálních rovnic

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \end{pmatrix} \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \end{pmatrix} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{pmatrix}.$$

Ten dále upravíme do tvaru

$$\begin{pmatrix} 5 & \sum_{i=1}^5 x_i & \sum_{i=1}^5 x_i^2 \\ \sum_{i=1}^5 x_i & \sum_{i=1}^5 x_i^2 & \sum_{i=1}^5 x_i^3 \\ \sum_{i=1}^5 x_i^2 & \sum_{i=1}^5 x_i^3 & \sum_{i=1}^5 x_i^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^5 f(x_i) \\ \sum_{i=1}^5 x_i f(x_i) \\ \sum_{i=1}^5 x_i^2 f(x_i) \end{pmatrix}, \tag{2.29}$$

který je v našem příkladě dán jako

$$\begin{pmatrix} 5 & 0 & 2.5 \\ 0 & 2.5 & 0 \\ 2.5 & 0 & 2.125 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ -2 \end{pmatrix}.$$

Jeho vyřešením obdržíme koeficienty

$$a_0 \approx 0.6571, \quad a_1 = 0, \quad a_2 \approx -1.7143$$

a tedy aproximační polynom ve tvaru

$$p_2(x) \approx 0.6571 - 1.7143 x^2. \tag{2.30}$$

Koeficienty polynomu lze vypočítat a výsledky vizualizovat pomocí Kódu 2.18. Na Obrázku 2.13 jsou zobrazeny původní funkce a její aproximační polynomy v případě 5 (vlevo) a 20 (vpravo) uzlových bodů.

Kód - Matlab 2.18: Metoda nejmenších čtverců.

```
1  M_fun=@(t,deg)power(t,0:deg);                    % matice (obecna)
2  d=2;                                             % stupen polynomu
3  n=5;                                             % pocet uzlu
4  a=-1; b=1;                                       % interval
5  f=@(t)cos(pi*t);                                 % funkce
6  x=linspace(a,b,n)'; y=f(x);                      % uzlove body
7  M=M_fun(x,d);                                    % matice (konkretni)
8  coefs=(M'*M)\(M'*y)                              % reseni norm. rov.
9  xx=a+(b-a)*linspace(0,1)';                       % vykreslovani
10 MM = M_fun(xx,d); yy=MM*coefs;
11 plot(xx,yy,'b',xx,f(xx),'r--',x,y,'ro');
12 legend('aproximace','funkce','uzlove body')
13 fprintf('odchylka=%d\n',norm(f(xx)-yy,'inf'))   % vypocet odchylky
```

Poznámka k příkladu

Zvyšováním počtu uzlových bodů lze přepočítat koeficienty polynomu $p_2(x)$ a získáváme:

$$\begin{aligned} p_2(x) &\approx 0.7221 - 2.0957x^2 \quad \text{(pro 20 bodů)} \\ p_2(x) &\approx 0.7407 - 2.1849x^2 \quad \text{(pro 40 bodů)} \\ p_2(x) &\approx 0.7502 - 2.2317x^2 \quad \text{(pro 80 bodů)} \\ p_2(x) &\approx 0.7550 - 2.2555x^2 \quad \text{(pro 160 bodů)} \end{aligned}$$

Všimneme si, že koeficienty polynomů se zvyšováním počtu bodů mírně mění. Přechodem od sum k integrálům lze popsat systém normálních rovnic pro počet uzlových bodů blížící se k nekonečnu ve tvaru (rozmyslete si proč)

$$\begin{pmatrix} \int_{-1}^{1} 1\,\mathrm{d}x & \int_{-1}^{1} x\,\mathrm{d}x & \int_{-1}^{1} x^2\,\mathrm{d}x \\ \int_{-1}^{1} x\,\mathrm{d}x & \int_{-1}^{2} x^2\,\mathrm{d}x & \int_{-1}^{1} x^3\,\mathrm{d}x \\ \int_{-1}^{1} x^2\,\mathrm{d}x & \int_{-1}^{1} x^3\,\mathrm{d}x & \int_{-1}^{1} x^4\,\mathrm{d}x \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \int_{-1}^{1} \cos(\pi x)\,\mathrm{d}x \\ \int_{-1}^{1} x\cos(\pi x)\,\mathrm{d}x \\ \int_{-1}^{1} x^2\cos(\pi x)\,\mathrm{d}x \end{pmatrix}. \tag{2.31}$$

Po výpočtu integrálů získáváme soustavu

$$\begin{pmatrix} 2 & 0 & 2/3 \\ 0 & 2/3 & 0 \\ 2/3 & 0 & 2/5 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -\frac{4}{\pi^2} \end{pmatrix}$$

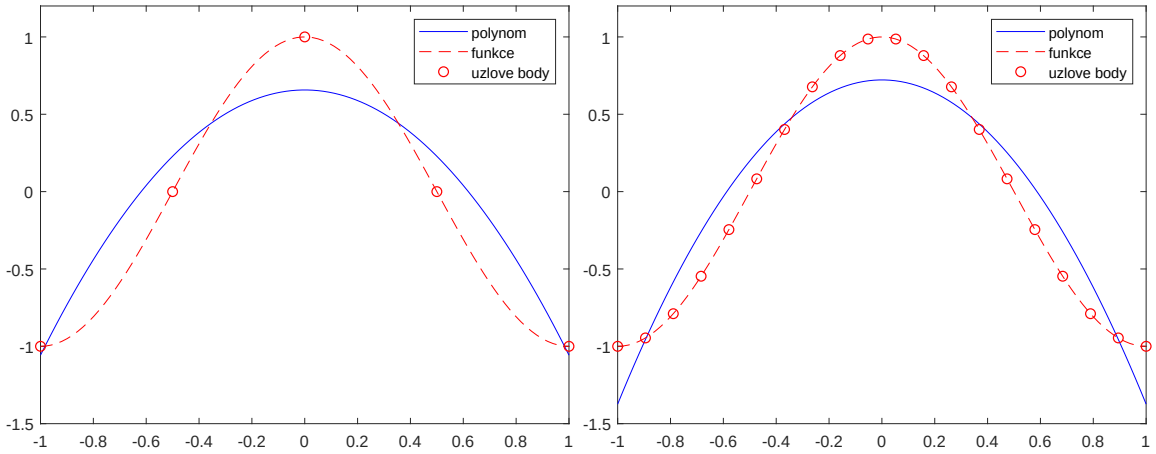
a jejím řešením koeficienty polynomu

$$p_2(x) \approx 0.7599 - 2.2797x^2. \tag{2.32}$$

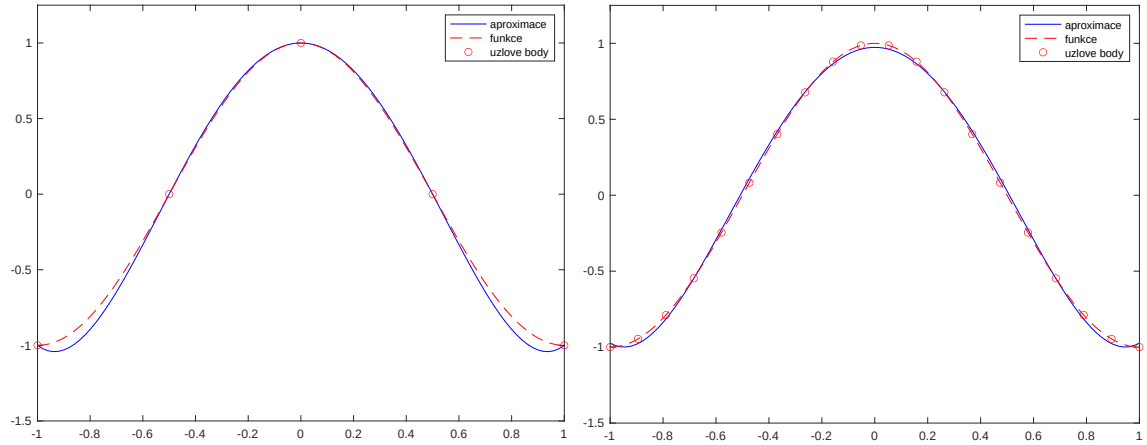
Obdobně je možné v 2. řádce kódu zvýšit také stupeň aproximačního polynomu, například na hodnotu 4 a získáváme

$$\begin{aligned} p_4(x) &\approx 1.0000 - 4.6667x^2 + 2.6667x^4 \quad \text{(pro 5 bodů)} \\ p_4(x) &\approx 0.9736 - 4.3982x^2 + 2.4509x^4 \quad \text{(pro 20 bodů)} \\ p_4(x) &\approx 0.9756 - 4.4265x^2 + 2.4928x^4 \quad \text{(pro 40 bodů)} \\ p_4(x) &\approx 0.9769 - 4.4442x^2 + 2.5189x^4 \quad \text{(pro 80 bodů)} \\ p_4(x) &\approx 0.9776 - 4.4538x^2 + 2.5332x^4 \quad \text{(pro 160 bodů)} \end{aligned}$$

V případě 5 uzlových bodů by stačilo použít přímo metodu polynomiální interpolace. Obdobně jako v případě polynomu 2. stupně bychom mohli sestavit systém normálních rovnic odpovídající (2.31). Ten by obsahoval matici velikosti 5×5. Aproximace polynomem 4. stupně dává kvalitativně lepší výsledky (srovnejte Obrázky 2.13 a 2.14).



Obrázek 2.13: Funkce $f(x)$ a její aproximační polynomy $p_2(x)$ vypočtené pomocí metody nejmenších čtverců za použití 5 uzlových bodů (vlevo) a 20 uzlových bodů (vpravo).



Obrázek 2.14: Funkce $f(x)$ a její aproximační polynomy $p_4(x)$ vypočtené pomocí metody nejmenších čtverců za použití 5 uzlových bodů (vlevo) a 20 uzlových bodů (vpravo).

Příklad 2.5.27 Pro diskrétní body dané tabulkou níže nalezněte regresní přímky

$$y = a + bx, \qquad x = c + dy,$$

kde $a, b, c, d \in \mathbb{R}$ jsou neznámé koeficienty. Poté najděte jejich průsečík.

x	100	50	80	40	50	30	95	25	50	75
y	70	50	80	60	60	55	50	50	55	80

Řešení: Soustavy normálních rovnic pro naši úlohu jsou ve tvaru

$$\begin{pmatrix} 10 & \sum_{i=1}^{10} x_i \\ \sum_{i=1}^{10} x_i & \sum_{i=1}^{10} x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{10} y_i \\ \sum_{i=1}^{10} x_i y_i \end{pmatrix}, \tag{2.33}$$

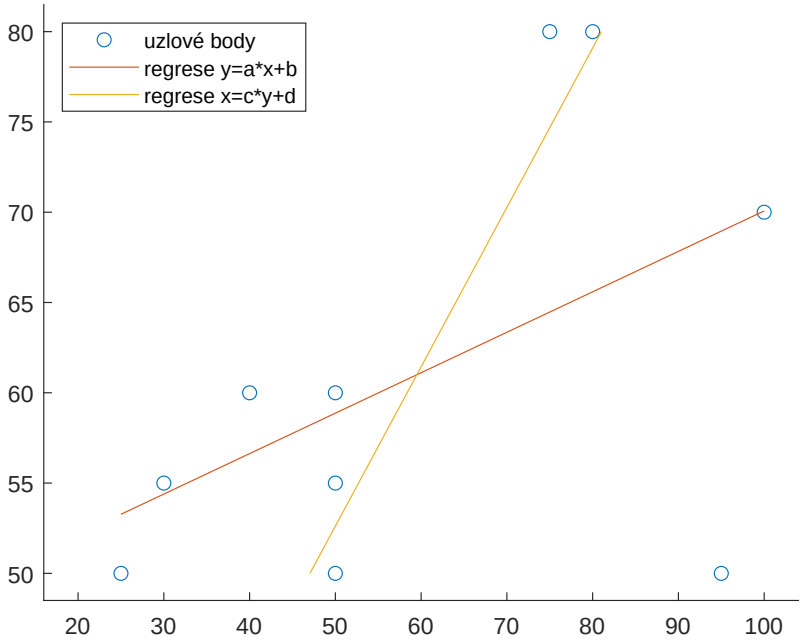
$$\begin{pmatrix} 10 & \sum_{i=1}^{10} y_i \\ \sum_{i=1}^{10} y_i & \sum_{i=1}^{10} y_i^2 \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{10} x_i \\ \sum_{i=1}^{10} y_i x_i \end{pmatrix} \tag{2.34}$$

a po dosazení konkrétních hodnot máme

$$\begin{pmatrix} 10 & 595 \\ 595 & 41675 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 610 \\ 37700 \end{pmatrix}$$
$$\begin{pmatrix} 10 & 610 \\ 610 & 38450 \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} 595 \\ 37700 \end{pmatrix}$$

a jejich vyřešením získáváme regresní přímky

$$y = \frac{119610}{2509} + \frac{562}{2509}\, x \approx 47.6724 + 0.223994\, x, \tag{2.35}$$
$$x = -\frac{2385}{248} + \frac{281}{248}\, y \approx -9.61694 + 1.13306\, y. \tag{2.36}$$



Obrázek 2.15: Regresní přímky.

Průsečíkem regresních přímek uvedených výše je bod

$$(\bar{x}, \bar{y}) = (61, 59.5),$$

jehož souřadnice přesně odpovídají aritmetickému průměru z hodnot x a y (ukažte).

Poznámka k příkladu

Hodnoty vektorů x a y v tabulce odpovídají procentuálnímu výsledku dvou testů vybraných studentů z předmětu Numerická matematika 1. Kladné směrnice obou regresních přímek lze interpretovat tak, že pokud má student více bodů z prvního testu, lze očekávat vyšší bodovou úspěšnost i při druhém testu.

Úloha k samostatnému řešení

Najděte koeficienty $a, b \in \mathbb{R}$ tak, aby funkce

$$g(x) = a\cos x + b\sin x$$

co nejlépe aproximovala funkci

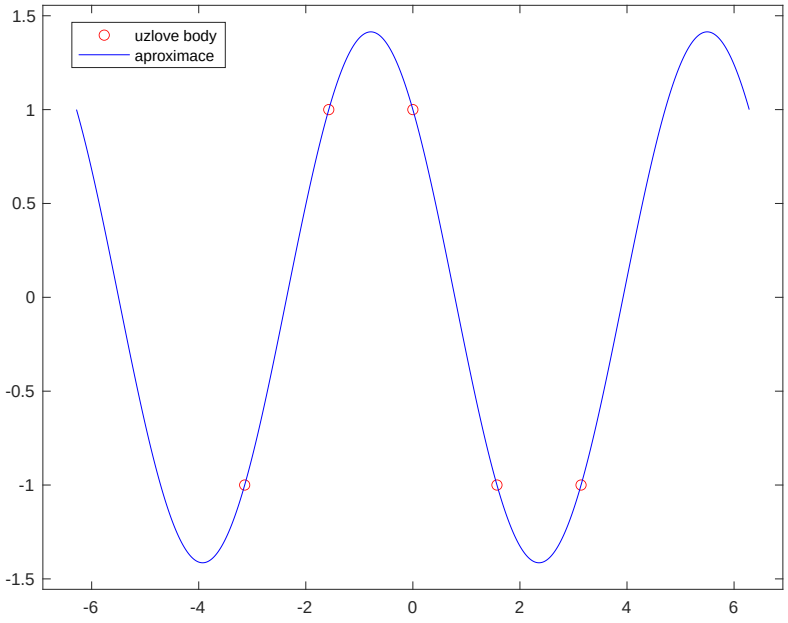
$$f(x) = \sqrt{2}\cos\left(x + \pi/4\right)$$

v bodech

$$x \in \{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$$

ve smyslu metody nejmenších čtverců. Napište odpovídající systém normálních rovnic, vypočtěte koeficienty a, b. Poté vypočtěte hodnoty Vámi nalezené funkce g ve všech pěti bodech x .

Nápověda: Pro sestavení soustavy normálních rovnic není nutné používat kalkulačku, stačí znalost základních hodnot trigonometrických funcí.



Obrázek 2.16: Řešení: $g(x) = \cos x - \sin x$. Platí známá identita $g(x) = f(x), x \in \mathbb{R}$.

Úloha k samostatnému řešení

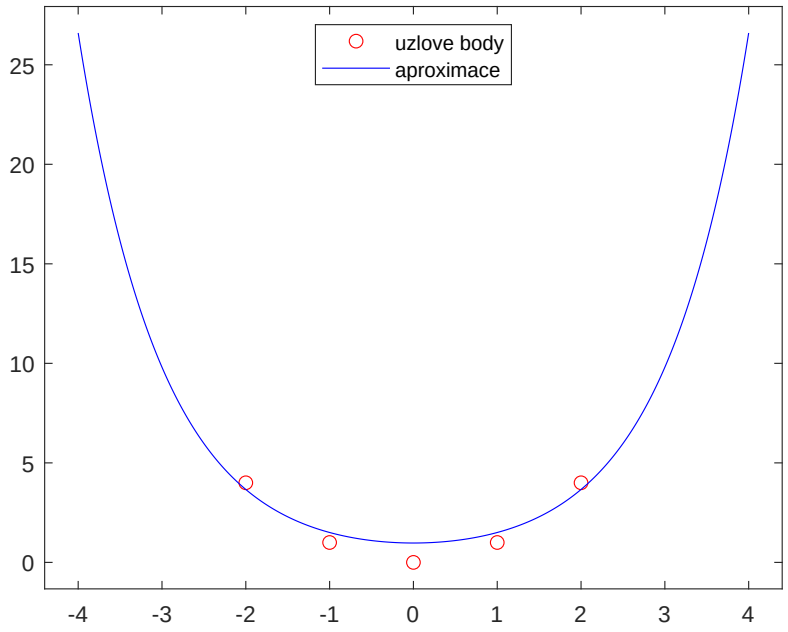
Najděte koeficienty $a, b, c \in \mathbb{R}$ tak, aby funkce

$$y(x) = a\, e^{x} + b + c\, e^{-x}$$

co nejlépe aproximovala 5 bodů daných tabulkou

x	-2	-1	0	1	2
y	4	1	0	1	4

ve smyslu metody nejmenších čtverců. Napište také odpovídající systém normálních rovnic a vypočtěte hodnoty nalezené funkce ve všech bodech x uvedených v tabulce .



Obrázek 2.17: Řešení: $y = 0.48\, e^{x} + 0.48\, e^{-x} = 0.96 \cosh x$.

Úloha k samostatnému řešení

Pomocí metody nejmenších čtverců za použití 100 uzlových bodů proložte kubickým polynomem funkci

$$f(x) = e^{2x}$$

na intervalu $[-1, 1]$. Napište systém normálních rovnic a najděte koeficienty výsledného polynomu.

2.6 Výpočet derivace

Příklad 2.6.28 Uvažujme obecnou kvadratickou funkci

$$f(x) = a_1 + a_2 x + a_3 x^2, \qquad a_1, a_2, a_3 \in \mathbb{R}$$

a její hodnoty v bodech

$$x \in \{-h, 0, h\}, \qquad h > 0.$$

Odvoďte formule pro přesné hodnoty derivací

$$f'(0) = ?, \quad f''(0) = ?$$

ve formě lineárních kombinací hodnot

$$f(-h),\ f(0),\ f(h).$$

Řešení: Hledáme tedy koeficienty $w_1, w_2, w_3 \in \mathbb{R}$ lineární kombinace

$$w_1 f(-h) + w_2 f(0) + w_3 f(h),$$

které se rovnají jedné z hodnot $f'(0) = a_2$ nebo nebo $f''(0) = 2a_3$. Musíme tedy splnit buď podmínku

$$w_1 (a_1 - h a_2 + h^2 a_3) + w_2 a_1 + w_3 (a_1 + h a_2 + h^2 a_3) = a_2 \tag{2.37}$$

nebo podmínku

$$w_1 (a_1 - h a_2 + h^2 a_3) + w_2 a_1 + w_3 (a_1 + h a_2 + h^2 a_3) = 2 a_3 \tag{2.38}$$

a to pro libovolné $a_1, a_2, a_3 \in \mathbb{R}, h > 0$. V případě (2.37) obdržíme systém rovnic

$$w_1 + w_2 + w_3 = 0, \quad h(w_3 - w_1) = 1, \quad h^2(w_1 + w_3) = 0$$

s řešením

$$w_1 = -\frac{1}{2h}, \quad w_2 = 0, \quad w_3 = \frac{1}{2h}$$

a v případě (2.38) systém rovnic

$$w_1 + w_2 + w_3 = 0, \quad h(w_3 - w_1) = 0, \quad h^2(w_1 + w_3) = 2$$

s řešením

$$w_1 = \frac{1}{h^2}, \quad w_2 = -\frac{2}{h^2}, \quad w_3 = \frac{1}{h^2}.$$

Výpočty lze ověřit pomocí kódu níže:

Kód - Mathematica 2.19: Odvození diferenčních formulí.

```
order=2;x0=0;
xx={x0-h,x0,x0+h}
(*xx={x0-2h,x0-h,x0,x0+h,x0+2h}*)
n=First[Dimensions[xx]];
p[x_]=Sum[Indexed[a,i]*x^(i-1),{i,1,n}]
pDerivative[x_]=D[p[x],{x,order}]
expr=Sum[Indexed[w,i]*p[xx[[i]]],{i,1,n}]
eqL=Table[Coefficient[Collect[expr,Indexed[a,i],Simplify],Indexed[a,i]],{i
    ,1,n}];
eqR=Table[Coefficient[Collect[pDerivative[x0],Indexed[a,i],Simplify],
    Indexed[a,i]],{i,1,n}];
eq=Table[eqL[[i]]==eqR[[i]],{i,1,n}]
variables=Table[Indexed[w,{i}],{i,1,n}];
Solve[eq,variables]
```

Odvodili jsme tedy vzorce

$$f'(0) = \frac{f(h) - f(-h)}{2h}, \tag{2.39}$$

$$f''(0) = \frac{f(h) - 2f(0) + f(-h)}{h^2}, \tag{2.40}$$

které platí přesně pro libovolnou kvadratickou funkci a libovolný krok h.

Poznámka k příkladu

Modifikací hodnoty x0=0 na libovolnou hodnotu v Kódu 2.19 se ověří, že obecné formule

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}, \tag{2.41}$$

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2.42}$$

platí v libovolném bodě $x \in \mathbb{R}$ a jsou přesné pro libovolný kvadratický polynom. My jsme si volbou x0=0 pouze zkrátili výrazy při odvozování. Modifikací kódu získáme další formule pro první derivace, například

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}, \tag{2.43}$$

$$f'(x) = \frac{f(x-2h) - 4f(x-h) + 3f(x)}{2h} \tag{2.44}$$

počítané z jednostranných diskrétních hodnot a platné pro libovolný kvadratický polynom. Také je možno získat vzorce využívající více diskrétních bodů, například

$$f'(x) = \frac{f(x-2h) - 8f(x+h) + 8f(x+h) - f(x+2h)}{12h}, \tag{2.45}$$

$$f''(x) = \frac{-f(x-2h) + 16f(x+h) - 30f(x) + 16f(x+h) - f(x+2h)}{12h^2}, \tag{2.46}$$

$$f'''(x) = \frac{-f(x-2h) + 2f(x+h) - 2f(x+h) + f(x+2h)}{2h^3}, \tag{2.47}$$

$$f''''(x) = \frac{f(x-2h) - 4f(x+h) + 6f(x) - 4f(x+h) + f(x+2h)}{h^4} \tag{2.48}$$

a ty platí pro libovolný polynom čtvrtého stupně. Je zřejmé, že dalším modifikacím kódu se meze nekladou.

Příklad 2.6.29 Vypočtěte přibližně první a druhou derivaci funkce

$$f(x) = \arctan(x)$$

na intervalu $[a, b] = [-4, 4]$ a určete chybu její aproximace.

Řešení: Interval $[a, b]$ rozdělíme pomocí 101 rovnoměrně rozdělených bodů na 100 stejně dlouhých podintervalů. V každém z nich aplikujeme formule na přibližné derivace

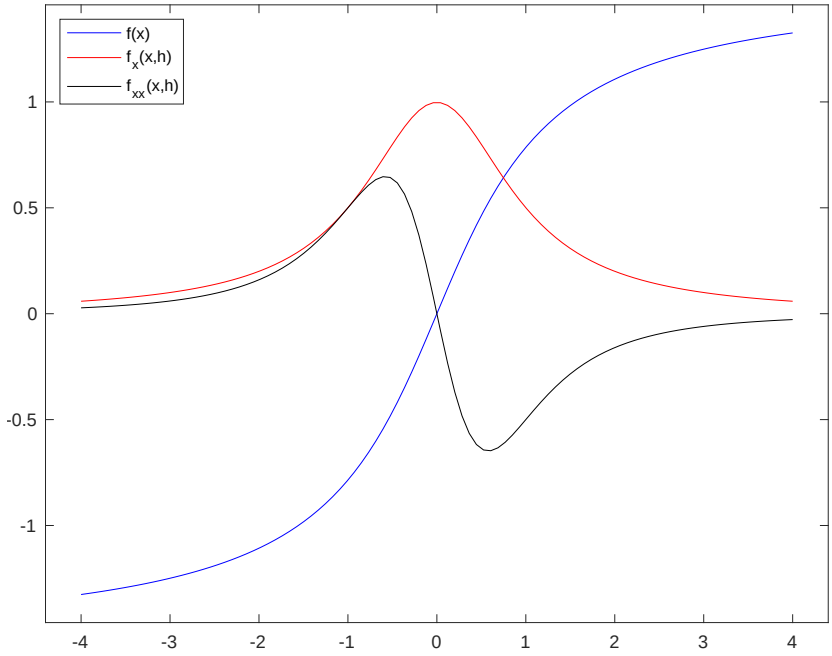
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{2.49}$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}, \tag{2.50}$$

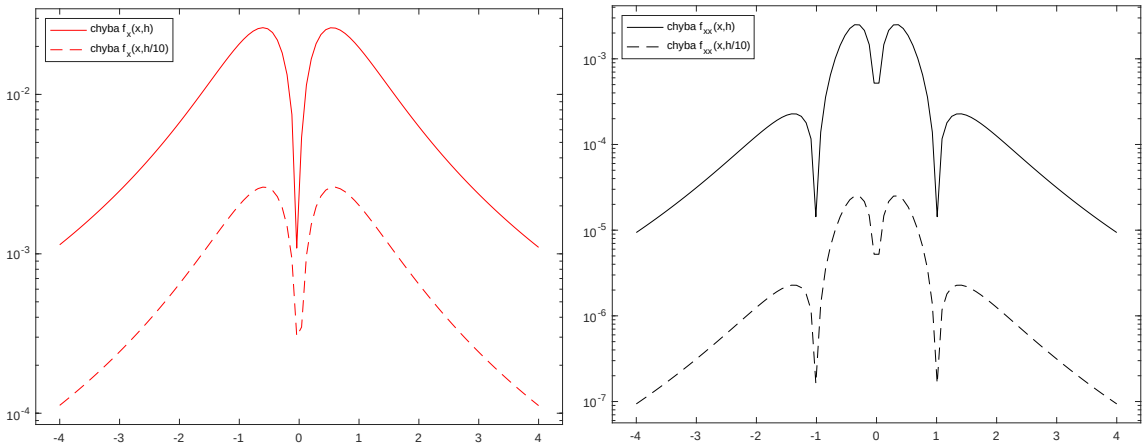
ve kterých volíme parametr $h > 0$. První formule je známá jako dopředná poměrná diference. Pokud je h rovné vzdálenosti dvou sousedních bodů, využíváme v formulích přímo hodnoty funkce v bodech rozdělení. V našem příkladě jde o hodnotu

$$h = \frac{b-a}{100} = 0.08. \tag{2.51}$$

Pro výpočty derivací v krajních bodech je třeba dodatečných hodnot funkce v bodech mimo interval $[a, b]$ (jakých). Kód 2.20 hodnoty vypočte a vizualizuje. Výsledek je ukázán na Obrázku (2.18).



Obrázek 2.18: Grafy funkcí f (modrá) a f' (červená) a f'' (černá) vypočtených numericky.



Obrázek 2.19: Absolutní chyba výpočtu f' (vlevo) a f'' (vpravo) v případě $h = 0.08$ (plná čára) a $h = 0.008$ (přerušovaná čára).

Srovnání s přesnými hodnotami obou derivací,

$$f'(x) = \frac{1}{1+x^2}, \qquad f''(x) = \frac{-2x}{(1+x^2)^2}$$

udávají aproximační chyby. Dodatečný Kód 2.21 aproximační chyby vizualizuje. Chyby aproximací v případě dvou voleb $h = 0.08$ a $h = 0.008$ jsou ukázány na Obrázku 2.19.

Obrázek potvrzuje, že absolutní chyba aproximace první derivace je přímo úměrná hodnotě h a absolutní chyba aproximace druhé derivace hodnotě h^2 (proč?).

Kód - Matlab 2.20: Výpočet první a druhé derivace.

```
close all
a=-4; b=4;                                  % interval
f=@(x) atan(x);                             % funkce
x=linspace(a,b,101);                        % uzlove body
fx=@(x,h)(f(x+h)-f(x))/h;                   % dopredna diference
%fx=@(x,h)(f(x+h)-f(x-h))/h/2;              % centralni diference
fxx=@(x,h)(f(x+h)-2*f(x)+f(x-h))/h^2;       % druha diference

h=(b-a)/(numel(x)-1);                       % krok diskretizace
figure; plot(x,f(x),'b-',x,fx(x,h),'r-',x,fxx(x,h),'k-')
legend('f(x)','f_x(x,h)','f_{xx}(x,h)',Location='northwest')
```

Kód - Matlab 2.21: Výpočet první a druhé derivace - vizualizace chyby.

```
Df=@(x)1./(1+x.^2);
DDf=@(x)-2*x./(1+x.^2).^2;
error_fx=@(x,h)abs(fx(x,h)-Df(x));
error_fxx=@(x,h)abs(fxx(x,h)-DDf(x));
figure;
semilogy(x,error_fx(x,h),'r-',x,error_fx(x,h/10),'r--')
legend('chyba f_x(x,h)','chyba f_x(x,h/10)',Location='northwest')
figure;
semilogy(x,error_fxx(x,h),'k-',x,error_fxx(x,h/1e1),'k--')
legend('chyba f_{xx}(x,h)','chyba f_{xx}(x,h/10)',Location='northwest')
```

Úloha k samostatnému řešení

Použitím tzv. centrální diference

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{2.52}$$

ukažte, že lze chybu aproximace první derivace v předchozím příkladě vylepšit.

Úloha k samostatnému řešení

Použitím Vámi zvolených formulí vypočtěte hodnoty první a druhé derivace funkce

$$f(x) = \sqrt{x}$$

v bodě $x = 1$ v případě $h = 0.1, h = 0.01, h = 0.001$. Vypočítejte absolutní chyby získaných aproximací a vysvětlete řád konvergence zvolené formule.

2.7 Výpočet integrálu

Příklad 2.7.30 Vypočtěte určitý integrál

$$I = \int_0^1 x^3 \, \mathrm{d}x$$

pomocí rovnoběžníkového pravidla a určete řád jeho integrační chyby.

Řešení: Pro jednoduchost předpokládejme rovnoměrné rozdělení intervalu $(0, 1)$ na 4 stejně dlouhé podintervaly o délce $h = 1/4$ s 5 uzlovými body

$$0,\ 1/4,\ 1/2,\ 3/4,\ 1.$$

Aplikace rovnoběžníkového pravidla vyžaduje znalost hodnot integrandu $f(x)$ ve 4 středových bodech jednotlivých podintervalů, tedy v bodech

$$1/8,\ 3/8,\ 5/8,\ 7/8$$

a vede na hodnotu

$$I_{1/4} = (1/4)[(1/8)^3 + (3/8)^3 + (5/8)^3 + (7/8)^3] = \frac{2^5 - 1}{2^7} = 0.2421875.$$

Faktor $1/4$ zde představuje velikost libovolného podintervalu. Přepočtem pro 8 podintervalů dostáváme potom hodnotu

$$I_{1/8} = \frac{2^7 - 1}{2^9} = 0.248046875.$$

Rovnoběžníkové pravidlo odpovídá nahrazení funkce f funkcí po částech konstantní a její integrací, viz Obrázek 2.20. Protože je přesná hodnota integrálu rovna $I = 1/4$, činí absolutní chyby obou výpočtů

$$e_{1/4} = |I_{1/4} - I| = \frac{1}{2^7} = 7.8125 \cdot 10^{-3},$$
$$e_{1/8} = |I_{1/8} - I| = \frac{1}{2^9} = 1.953125 \cdot 10^{-3}.$$

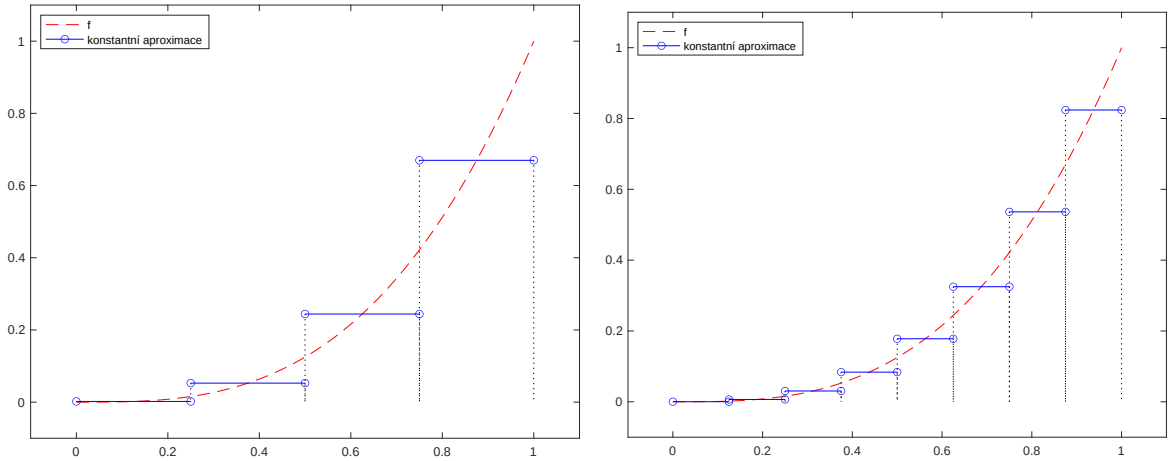
Rovnoběžníkové pravidlo vykazuje kvadratickou integrační chybu. Při zdvojnásobení počtu podintervalů se chyba výpočtu sníží 4-krát. Přesnost řádu p chyby aproximace lze také vypočítat z výrazu

$$p = \log_2 \frac{e_{1/4}}{e_{1/8}} = 2.$$

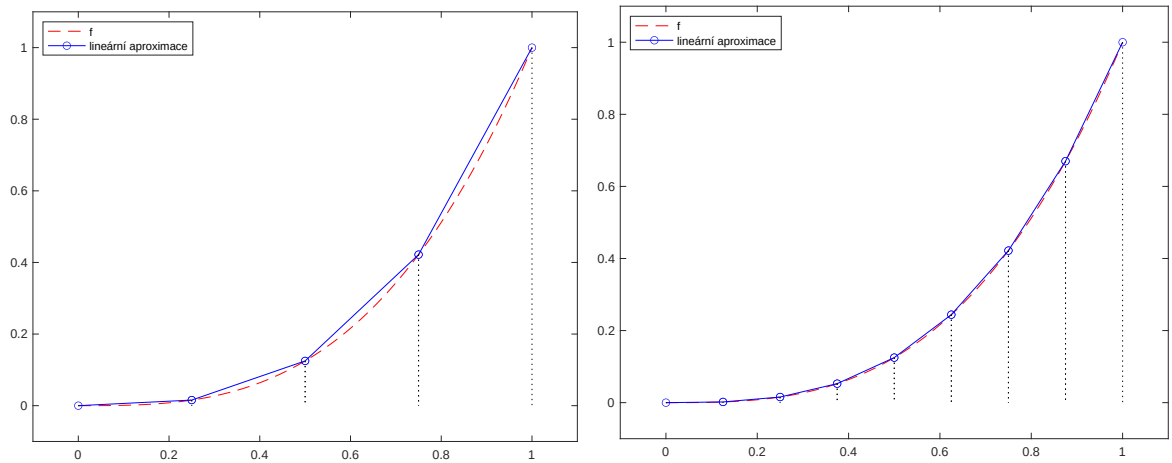
Při větším počtů podintervalů je praktické využít počítač:

Kód - Matlab 2.22: Výpočet integrálu rovnoběžníkovým pravidlem.

```
format long
a=0; b=1;                                % integracni meze
f=@(x) x.^3;                             % integrand
n=16;                                    % pocet podintervalu
x=linspace(a,b,n+1);
h=(b-a)/n;                               % delka podintervalu
I=1/4;                                   % presna hodnota

xM=(x(1:end-1)+x(2:end))/2;              % stredove body
Ih=h*sum(f(xM))                          % priblizna hodnota
eh=abs(Ih-I)                             % chyba
```



Obrázek 2.20: Funkce f a její po částech konstantní aproximace s naznačenou plochou pod ní na 4 podintervalech (vlevo) a 8 podintervalech (vpravo).



Obrázek 2.21: Funkce f a její po částech lineární aproximace s naznačenou plochou pod ní na 4 podintervalech (vlevo) a 8 podintervalech (vpravo).

Získáváme tím přesnější hodnoty integrálu a odpovídající absolutní chyby, např.

$$I_{1/16} = \frac{2^9 - 1}{2^{11}} = 0.24951171875, \qquad e_{1/16} = \frac{1}{2^{11}} = 4.8828125 \cdot 10^{-4},$$
$$I_{1/32} = \frac{2^{11} - 1}{2^{13}} = 0.2498779296875, \qquad e_{1/32} = \frac{1}{2^{13}} = 1.220703125 \cdot 10^{-4},$$
$$I_{1/64} = \frac{2^{13} - 1}{2^{15}} = 0.249969482421875, \qquad e_{1/64} = \frac{1}{2^{15}} = 3.0517578125 \cdot 10^{-5},$$

které opět potvrzují kvadratickou konvergenci rovnoběžníkové metody.

Poznámka k příkladu

Další integračním (kvadraturním) pravidlem je lichoběžníkové pravidlo, které funkci f nahrazuje funkcí po částech lineární a přímo využívá hodnoty funkce v krajích podintervalů, viz Obrázek 2.21. To je např. v Matlabu realizováno přímo a dodatečný kód

Kód - Matlab 2.23: Výpočet integrálu lichoběžníkovým pravidlem.

```
1 Ih_2=trapz(x,f(x))                    % priblizna hodnota
2 eh_2=abs(Ih_2-I)                      % chyba
```

je realizuje. Lichoběžníkové pravidlo má kvadratický řád chyby ($p = 2$) stejně jako rovnoběžníkové pravidlo. Vyššího řádu chyby lze docílit až implementací Simpsonova pravidla. To funkci f nahrazuje funkcí po částech kvadratickou a implementujeme jej jako

Kód - Matlab 2.24: Výpočet integrálu Simpsonových pravidlem.

```
so=f(x(1))+f(x(end));                    % vnejsi body - suma
sx=sum(f(x(2:end-1)));                   % vnitrni body - suma
sm=sum(f(xM));                           % stredove body - suma
Ih_3=(h/2/3)*(so+2*sx+4*sm)              % priblizna hodnota
eh_3=abs(Ih_3-I)                         % chyba
```

Lze ukázat že Simpsonovo pravidlo je dokonce čtvrtého řádu ($p = 2$) a proto pro libovolný počet podintervalů dává v našem příkladě přesnou hodnotu integrálu (ověřte).

Příklad 2.7.31 Rombergovou metodou vypočtěte přibližnou hodnotu integrálu

$$I = \int_{-1}^{1} \frac{\sin x}{x} dx.$$

Řešení: Integrand funkce $\frac{\sin x}{x}$ lze spojitě dodefinoval hodnotou 1 v bodě $x = 0$ a formálně ho nahradit funkci sinc x (lat. sinus cardinalis). Tím se vyhneme počítačovému dělení nulou. Rombergova metoda přepočítává hodnotu integrálu z posloupnosti přibližných hodnot

$$I_h,\ I_{h/2},\ I_{h/4},\ I_{h/8}, \dots \tag{2.53}$$

spočtených pomocí lichoběžníkového pravidla na systému vnořených podintervalů. Volme pro jednoduchost $h = 1$, což odpovídá rozdělení $[-1, 1]$ na dva podintervaly $[-1, 0], [0, 1]$. Dostáváme

$$\begin{aligned}
I_h &= \frac{1}{2}\left(\operatorname{sinc}(-1) + 2\operatorname{sinc}(0) + \operatorname{sinc}(1)\right) = 1 + \operatorname{sinc}(1) \approx 1.841470984807897,\\
I_{h/2} &= \frac{1}{4}\left(\operatorname{sinc}(-1) + 2\operatorname{sinc}(-\tfrac{1}{2}) + 2\operatorname{sinc}(0) + 2\operatorname{sinc}(\tfrac{1}{2}) + \operatorname{sinc}(1)\right)\\
&= \frac{1}{2}\left(1 + 2\operatorname{sinc}(\tfrac{1}{2}) + \operatorname{sinc}(1)\right) \approx 1.879586569612354,\\
I_{h/4} &= \frac{1}{8}\left(\operatorname{sinc}(-1) + 2\operatorname{sinc}(-\tfrac{3}{4}) + 2\operatorname{sinc}(-\tfrac{1}{2}) + 2\operatorname{sinc}(-\tfrac{1}{4}) + 2\operatorname{sinc}(0)\right.\\
&\quad \left. + 2\operatorname{sinc}(\tfrac{1}{4}) + 2\operatorname{sinc}(\tfrac{1}{2}) + 2\operatorname{sinc}(\tfrac{3}{4}) + \operatorname{sinc}(1)\right)\\
&= \frac{1}{4}\left(1 + 2\operatorname{sinc}\frac{1}{4} + 2\operatorname{sinc}\frac{1}{2} + 2\operatorname{sinc}\frac{3}{4} + \operatorname{sinc}(1)\right) \approx 1.889027043330779,\\
I_{h/8} &= \frac{1}{16}\left(\operatorname{sinc}(-1) + 2\operatorname{sinc}(-\tfrac{7}{8}) + 2\operatorname{sinc}(-\tfrac{6}{8}) + \dots\right) \approx 1.891381727165402.
\end{aligned}$$

Získané modré hodnoty (více jich pro jednoduchost neuvažujeme) zapíšeme do prvního sloupce matice I uvedené níže :

```
1.841470984807897                0                0                0
1.879586569612354   1.892291764547174                0                0
1.889027043330779   1.892173867903588   1.892166008127349                0
1.891381727165402   1.892166621776943   1.892166138701834   1.892166140774445
```

Nenulové prvky ve druhém až čtvrtém sloupci přepočítáme z předchozích sloupců rekurentně pomocí formule

$$I(j,k) = I(j,k-1) + \frac{I(j,k-1) - I(j-1,k-1)}{2^{p(k-1)} - 1}, \qquad j \in \{2,3,4\}, 2 \le k \le j, \tag{2.54}$$

příčemž volíme hodnotu $p = 2$, která odpovídá teoretickému řádu chyby lichoběžníkového pravidla. Např. k vygenerování druhého a třetího diagonálního prvku matice využijeme tedy přepočet:

$$I(2,2) = I(2,1) + \frac{I(2,1) - I(1,1)}{4-1} = I_{h/2} + \frac{I_{h/2} - I_h}{3}, \tag{2.55}$$

$$I(3,3) = I(3,2) + \frac{I(3,2) - I(2,2)}{16-1} = \dots. \tag{2.56}$$

Celou matici jsme sestavili pomocí:

Kód - Matlab 2.25: Výpočet integrálu Rombergových pravidlem.

```
1  format long
2  a = -1; b = 1;
3  f = @(x) sin(x)./x;
4  n=2;                                    % pocet podintervalu
5  jmax=4;                                 % pocet integralu
6  p=2;                                    % rad integr. pravidla
7  I=zeros(jmax);
8  for j=1:jmax
9      nb=n*2^(j-1)+1;
10     x=linspace(a,b,nb); y=f(x);         % uzly a hodnoty funce v nich
11     y(isnan(y))=1;                      % korekce hodnoty pro x=0
12     I(j,1)=trapz(x,y);                  % lichobeznikove pravidlo
13     for k=2:j
14         I(j,k)=I(j,k-1)+(I(j,k-1)-I(j-1,k-1))/(2^(p*(k-1))-1);
15     end
16 end
17 I
18 Iexact=1.892166140734366;              % presna hodnota integralu
19 for j=1:jmax
20     for k=1:j
21         e(j,k)=abs(I(j,k)-Iexact);
22     end
23 end
24 e
25 e(1:end-1,:)./e(2:end,:)
```

Dodatečným využitím přesné hodnoty integrálu vypočte blok příkazů na řádkách 18-24 vypočte a vypíše odpovídající absolutní chyby všech přibližných hodnot integrálu:

```
0.050695155926469                   0                   0                   0
0.012579571122012   0.000125623812808                   0                   0
0.003139097403587   0.000007727169222   0.000000132607017                   0
0.000784413568964   0.000000481042577   0.000000002032532   0.000000000040079
```

Je vidět, že chyba na pozici posledního diagonálního prvku je nejmenší. Úplně poslední příkaz vypisuje poměry hodnot chyb na pozicích výše a hodnot chyb na pozicích níže:

```
4.029959005340308                   0                 NaN                 NaN
4.007384768513065  16.257417069773798                   0                 NaN
4.001839753656848  16.063378973127048  65.242265979091329                   0
```

Interpretace těchto poměrů je poměrně jasná: lichoběžníkové pravidlo, které používáme k výpočtu hodnot chyb v prvním sloupci je řádu 2 a dává tedy zhruba hodnotu poměrů chyb blízkou číslu 4. Ve druhém sloupci je poměr chyb zhruba roven 16, tedy odpovídá řádu 4 konvergence. Nakonec ve 3. sloupci je poměr o trochu větší než 64 a měl by vyjadřovat konvergenci řádu 6. Hodnoty NaN (angl. Not a Number) odpovídají dělení nulou a neměli bychom jim věnovat žádnou pozornost.

Poznámka k příkladu

Lze ukázat, že výpočty ve druhém sloupci odpovídají přesně Simpsonovu pravidlu a ve třetím tzv. Booleho pravidlu.

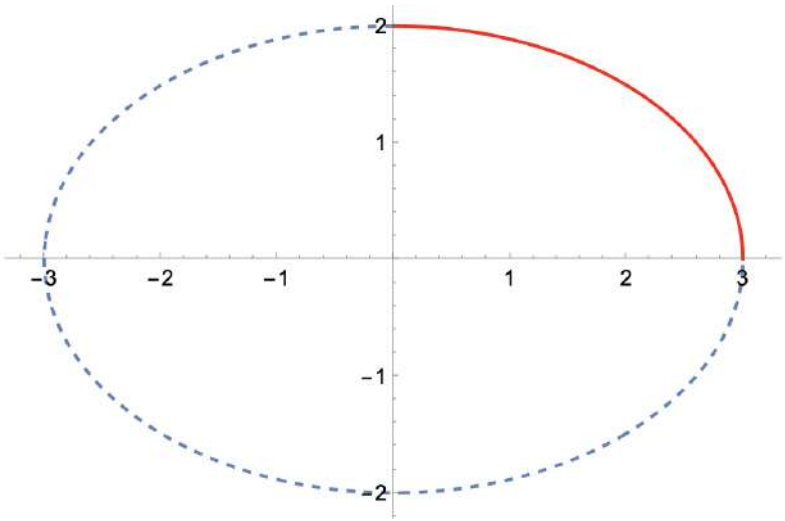
Úloha k samostatnému řešení

Spočtěte přibližně integrál

$$I = a \int_0^{\pi/2} \sqrt{1 - e^2 \cos^2 t}\, \mathrm{d}t$$

za pomoci lichoběžníkového a Simpsonova pravidla pro 4, 40 a 400 rovnoměrně rozdělených intervalů a hodnoty

$$a = 3, \quad b = 2, \quad e = \frac{\sqrt{a^2 - b^2}}{a}.$$



Obrázek 2.22: Elipsa s vyznačenou čtvrtinou, jejíž délku počítáme.

Nápověda: Pro elipsu s hlavní poloosou a a vedlejší poloosou b popisuje integrál I čtvrtinu délky jejího oblouku, viz Obrázek 2.22. Hodnota e pak představuje její tzv. excentricitu.

Výsledek pro kontrolu: hodnota integrálu vychází

$$I = 3 \int_0^{\pi/2} \sqrt{1 - (5/9) \cos^2 t}\, \mathrm{d}t \approx 3.966359897322647.$$

Úloha k samostatnému řešení

Vypočtěte určitý integrál

$$I = \int_{-1}^{1} \frac{1}{x^2 + 1}\, \mathrm{d}x$$

pro funkci pomocí lichoběžníkového a Simpsonova pravidla s 16, 32, 64 rovnoměrně rozloženými podintervaly a odhadněte řády integračních chyb obou metod.

KAPITOLA

3

NUMERICKÁ MATEMATIKA 2

3.1 Diskretizace Poissonovy rovnice

Příklad 3.1.32 Diskretizujte okrajovou úlohu pro obyčejnou diferenciální rovnici

$$-u''(x) = x^2, \quad u(0) = 0,\, u(1) = 0$$

za použití rovnoměrného rozdělení intervalu $(0,1)$, tu převeďte na soustavu rovnic a tu vyřešte.

Řešení:
Rozdělíme interval $(0,1)$ na $n+2$ rovnoměrně rozložených uzlů pro $n \in \mathbb{N}_0$ ve tvaru

$$x_i := i\,h, \qquad i = 0, \ldots, n+1,$$

kde $h := \frac{1}{n+1}$ představuje délku kroku rozdělení intervalu. Druhou derivaci v libovolném vnitřním bodě nahradíme druhou centrální diferencí

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}, \qquad i = 1, \ldots, n.$$

Nyní lze již přepsat diferenciální rovnici na soustavu lineárních rovnic ve tvaru

$$A_{n,h}\mathbf{u} = \mathbf{b}$$

s třídiagonální maticí (prázdná místa v matici představují nulové prvky)

$$A_{n,h} := \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{3.1}$$

sloupcovým vektorem pravé strany

$$\mathbf{b} = (x_1^2, x_2^2, \ldots, x_{n-1}^2, x_n^2)^T \in \mathbb{R}^{n\times 1} \tag{3.2}$$

a sloupcovým vektorem neznámých

$$\mathbf{u} = (u_1, u_2, \ldots, u_{n-1}, u_n)^T \in \mathbb{R}^{n\times 1}. \tag{3.3}$$

Vektor neznámých aproximuje hodnoty řešení u ve vnitřních uzlech, tj.

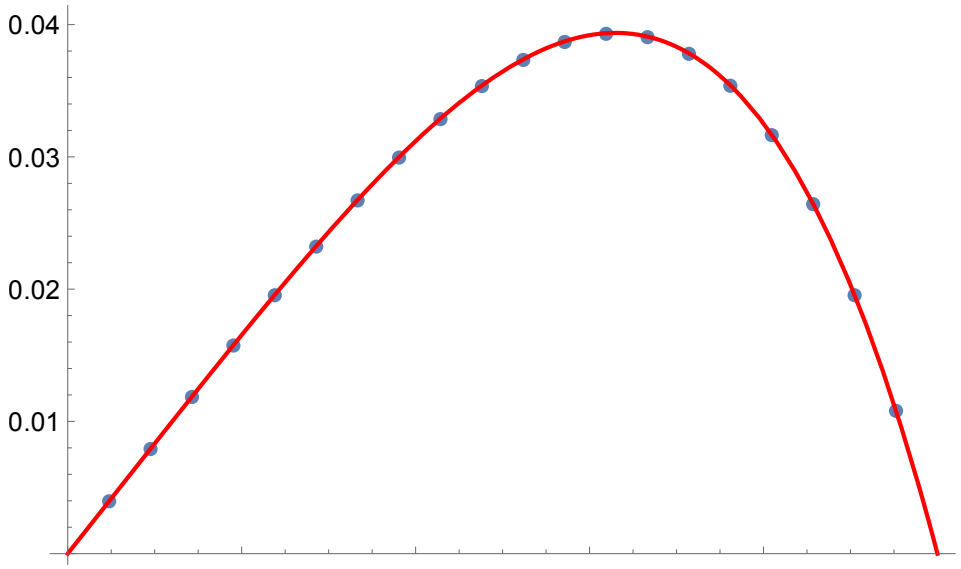
$$u_i \approx u(x_i), \qquad i = 1, \ldots, n,$$

kde u představuje představuje přesné řešení okrajové úlohy.
Výpočet pomocí kódu níže poskytuje vektor řešení u pro 20 vnitřních uzlů.

Kód - Mathematica 3.1: Řešení okrajové úlohy.

```
n = 20;
h = 1/(n + 1);
A = (1/h^2)*SparseArray[{{i_,i_}:>2,{i_,j_}:> -1 /; Abs[i-j]==1}, {n,n}];
x = Table[(i*h), {i, 1, n}];
b = Table[(x[[i]])^2, {i, 1, n}];
u = LinearSolve[A, b];
t = Table[{x[[i]], u[[i]]}, {i, 1, n}];
p1 = ListPlot[t, PlotMarkers -> Automatic, PlotMarkers -> Automatic];
uExact[y_] = (y - y^4)/12;
p2 = Plot[uExact[x], {x, 0, 1}, PlotStyle -> Red];
Show[p1, p2, PlotRange -> All]
e = N[Max[Abs[u - uExact[x]]]]
```

Obrázek 3.1 zobrazuje hodnoty přibližného vektoru řešení **u** (modré body) společně s grafem přesného řešením (3.4) okrajové úlohy (červená křivka).



Obrázek 3.1: Přesné řešení okrajové úlohy u (červená křivka) a jeho vypočtená diskrétní aproximace (modré body).

To je možné pomocí dvojnásobné integrace získat (odvoďte) a platí

$$u(x) = \frac{x - x^4}{12}, \qquad x \in [0, 1]. \tag{3.4}$$

Přestože se zdá, že složky vektoru **u** leží přesně na grafu přesného řešení, není tomu tak. Maximální svislá odchylka jednoho (přesněji nespecifikovaného) modrého bodu od červené křivky je zhruba rovna hodnotám

- $4.7 \cdot 10^{-5}$ (pro n=20),
- $5.2 \cdot 10^{-7}$ (pro n=200),
- $5.2 \cdot 10^{-9}$ (pro n=2000).

Zvýšením počtu uzlů tedy dosahujeme vyšší přesnosti.

Poznámka k příkladu

Tento způsob získání přibližného řešení okrajové úlohy je známý jako metoda konečných diferencí (angl. finite difference method).

Úloha k samostatnému řešení

Vyřešte okrajové úlohy pro obyčejné diferenciální rovnice

1. $-u''(x) = 1, \quad u(0) = 0, u(1) = 0,$
2. $-u''(x) = x\,e^x, \quad u(0) = 0, u(1) = 0,$
3. $-u''(x) = x^2, \quad u(0) = 0, u(1) = 1$

za použití rovnoměrného rozdělení intervalu $(0, 1)$ a

a) 10 podintervalů,

b) 100 podintervalů,

c) 1000 podintervalů.

Porovnejte maximální hodnotu vypočtené funkce u na intervalu $(0, 1)$ s přesnou hodnotou vypočtenou analyticky. Jak vypočtená hodnota maxima konverguje přesné hodnotě maxima vzhledem k délce podintervalů h?

Poznámka: řešení 3. úlohy vyžaduje určitou modifikaci kódu s ohledem na nehomogenní Dirichletovu podmínku. Prakticky stačí modifikovat pravou stranu **b** soustavy lineárních rovnic. Podaří se Vám to?

3.2 Iterační maticové metody

Příklad 3.2.33 Najděte řešení rovnice $Ax = b$, kde

$$A = 5^2 \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix},$$

pomocí Jacobiho a Gauss-Seidelovy iterační metody a počátečního vektoru ve tvaru

$$x_0 = (0,0,0,0)^T.$$

Porovnejte rychlost konvergence obou metod.

Řešení:
Obě iterační metody předpokládají aditivní rozklad matice

$$A = L + D + U,$$

kde D je diagonální matice a L a U jsou dolní a horní trojúhelníkové matice bez diagonálních částí. Tedy máme

$$L = 5^2 \begin{pmatrix} 0 & & & \\ -1 & 0 & & \\ & -1 & 0 & \\ & & -1 & 0 \end{pmatrix}, \quad D = 5^2 \begin{pmatrix} 2 & & & \\ & 2 & & \\ & & 2 & \\ & & & 2 \end{pmatrix}, \quad U = 5^2 \begin{pmatrix} 0 & -1 & & \\ & 0 & -1 & \\ & & 0 & -1 \\ & & & 0 \end{pmatrix}.$$

Jacobiova metoda generuje posloupnost aproximací vektorů $\{u_k\}_1^\infty$ z iterační formule

$$x_k = -D^{-1}(L+U)x_{k-1} + D^{-1}b, \qquad k \in \{1,2,\dots\} \tag{3.5}$$

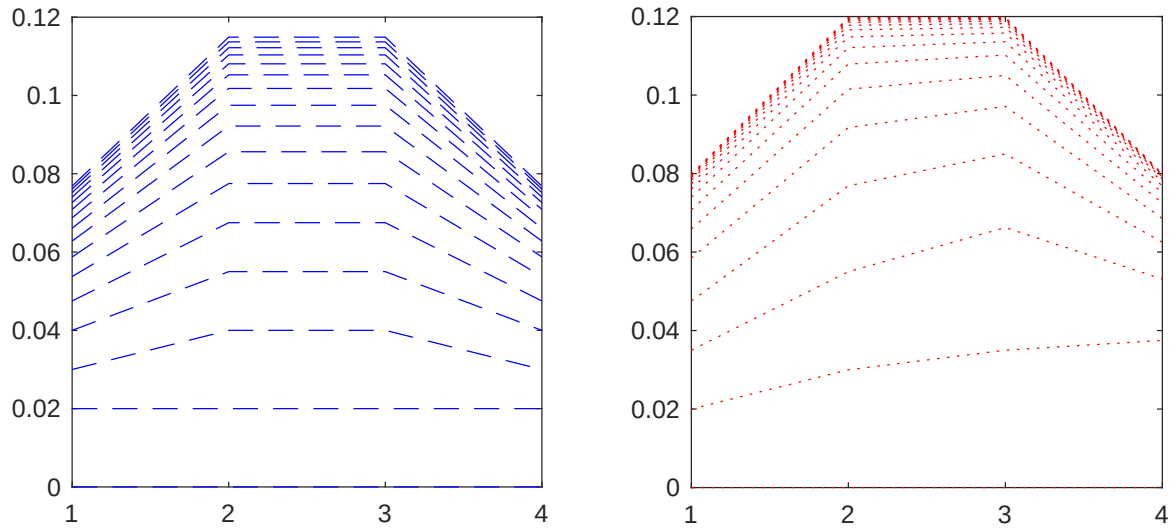
a Gauss-Seidelova metoda z formule

$$x_k = -(L+D)^{-1}U\,x_{k-1} + (L+D)^{-1}b, \qquad k \in \{1,2,\dots\}. \tag{3.6}$$

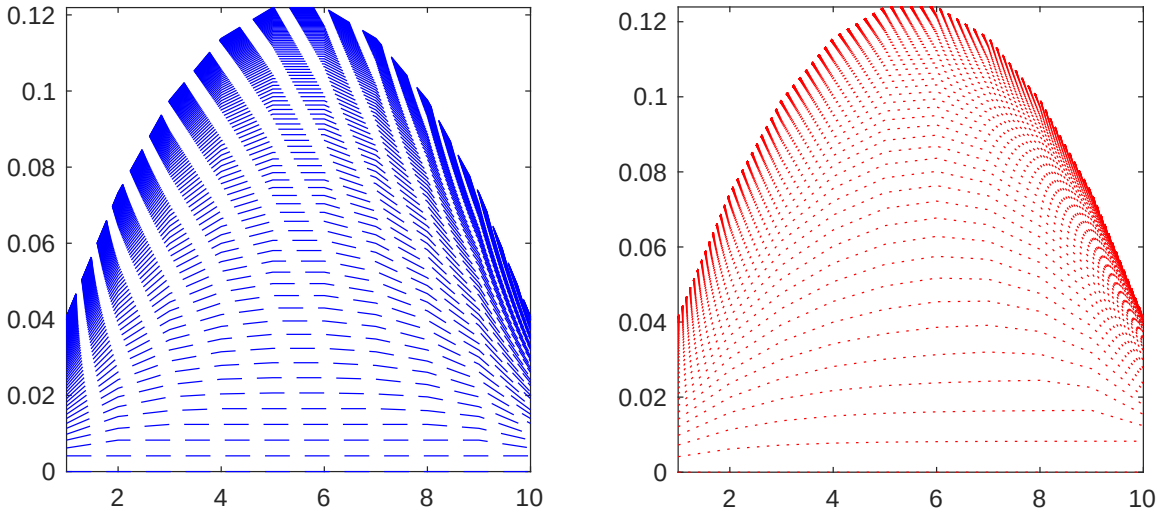
V první iteraci obou metod použijeme stejný počáteční vektor x_0. Jednotlivé iterace jsou zobrazeny na Obrázku 3.2 jako lomené čáry. K jejich vygenerování slouží Kód 3.2.

Kód - Matlab 3.2: Iterační maticové metody.

```
n=4; iters=15;                          % velikost matice a pocet iteraci
A =(n+1)^2*gallery('tridiag',n,-1,2,-1);    % sestaveni ridke matice
b=ones(n,1);
D=diag(diag(A)); L=tril(A,-1); U=triu(A,1);    % rozklad A=L+D+U
C_J=-D\(L+U); d_J=D\b;                  % Jacobi
C_GS=-(L+D)\U; d_GS=(L+D)\b;            % Gauss-Seidel
x_J=zeros(n,iters+1);  x_GS=x_J;        % iterace
for k=1:iters
    x_J(:,k+1)=C_J*x_J(:,k)+d_J;
    x_GS(:,k+1)=C_GS*x_GS(:,k)+d_GS;
end
figure;
subplot(1,2,1); plot(x_J,'b--'); axis square;
subplot(1,2,2); plot(x_GS,'r:'); axis square;
rho_Jac=max(abs(eigs(C_J)))             % spektralni polomer
rho_GS=max(abs(eigs(C_GS)))             % spektralni polomer
```



Obrázek 3.2: Iterace Jacobiho metody (vlevo) a Gauss-Seidelovy metody (vpravo) pro $n = 4$.



Obrázek 3.3: Iterace Jacobiho metody (vlevo) a Gauss-Seidelovy metody (vpravo) pro $n = 10$.

Pozorujeme, že obě metod konvergují, přičemž iterace Gauss-Seidelovy metody o něco rychleji. Zahušťují se totiž více kolem vektoru přesného řešení

$$x = (0.08, 0.12, 0.12, 0.08)^T.$$

Poslední (zde navolené jako 15-té) iterace představují vektor

$$x = (0.0768, 0.1149, 0.1149, 0.0768)^T$$

pro Jacobiho metodu a vector

$$x = (0.0798, 0.1197, 0.1198, 0.0799)^T$$

pro Gauss-Seidlovu metodu. O rychlosti konvergence rozhodují vlastnosti příslušných iteračních matic

$$C_J := -D^{-1}(L + U), \qquad C_{GS} := -(L + D)^{-1}U$$

zvýrazněných modře ve vzorcích (3.5)-(3.6). Nás zajímá spektrální poloměr, který je definován pro matici $C \in \mathbb{R}^{n\times n}$ jako její největší vlastní číslo, tj.

$$\rho(C) = \max\{|\lambda(C)|\}. \tag{3.7}$$

Pro naše matice konkrétně vychází

$$\rho(C_J) \approx 0.8090, \qquad \rho(C_{GS}) \approx 0.6545,$$

tedy je splněna podmínka $1 > \rho(C_{Jac}) > \rho(C_{GS})$. Protože jsou obě hodnoty spektrálních poloměrů ostře menší než jedna, musí obě metody v tomto případě zaručeně konvergovat dokonce pro libovolný počáteční vektor x_0.

Poznámka k příkladu

Modifikací první řádky kódu výše do tvaru `n=10; iters=100;` získáváme prvních 100 iterací pro soustavu 10 rovnic, které jsou ukázány na Obrázku 3.3. Odpovídající spektrální poloměry jsou

$$\rho(C_J) \approx 0.9595, \qquad \rho(C_{GS}) \approx 0.9206$$

a obě metody opět konvergují, avšak ve srovnání s menší maticí v příkladu ještě pomaleji, protože oba spektrální poloměry jsou blíže hodnotě 1.
Poznamenejme, že program výše je vhodný pouze pro demonstraci obou metod. V praktických výpočtech se s maticemi C_J a C_{GS} vůbec nepracuje, protože jejich přímý výpočet je příliš nákladný. Namísto toho se používají přepočty aproximačních vektorů po jednotlivých komponentách . Odpovídající algoritmy lze jednoduše najít na internetu (zkuste si je sami implementovat).

Příklad 3.2.34 Nalezněte řešení Příkladu 3.2.33 pro nulovou počáteční iteraci x_0 pomocí metody největšího spádu (the steepest descent method).

Řešení: Řešení rovnice $Ax = b$ se symetrickou a pozitivně definitní maticí $A \in \mathbb{R}^{n\times n}$ hledáme řešení z ekvivalentní (proč?) minimalizační úlohy

$$E(x) = \min_{y\in\mathbb{R}^n} E(y), \tag{3.8}$$

kde výraz

$$E(y) = \frac{1}{2}y^T Ay - y^T b, \qquad y \in \mathbb{R}^n \tag{3.9}$$

představuje tzv. energetický kvadratický funkcionál v (sloupcovém) vektoru y. Jeho gradient vychází (ukažte)

$$\nabla E(y) = Ay - b, \qquad y \in \mathbb{R}^n. \tag{3.10}$$

Iterační metoda tvaru

$$x_k = x_{k-1} - \alpha_{k-1}(Ax_{k-1} - b), \qquad k \in \{1, 2, \dots\} \tag{3.11}$$

představuje nejjednodušší gradientní metodu s krokem $\alpha_{k-1} \in \mathbb{R}$. Hodnota kroku se volí tak, že

$$\alpha_{k-1} = \arg\min_{\alpha\in\mathbb{R}} E(x_{k-1} - \alpha(Ax_{k-1} - b)), \tag{3.12}$$

tedy abychom dosáhli co nejměnší hodnoty energie. Lze odvodit (proveďte) formuli

$$\alpha_{k-1} = \frac{(Ax_{k-1} - b)^T(Ax_{k-1} - b)}{(Ax_{k-1} - b)^T A(Ax_{k-1} - b)}, \qquad k \in \{1, 2, \dots\}. \tag{3.13}$$

K vygenerování jedotlivých iterací slouží Kód 3.3.

Kód - Matlab 3.3: Iterační maticové metody, metoda nejmenšího spádu

```
n=4; iters=10;                          % velikost matice a pocet iteraci
A =(n+1)^2*gallery('tridiag',n,-1,2,-1);    % sestaveni ridke matice
b=ones(n,1);
x_SD=zeros(n,iters+1);                  % iterace
r=b-A*x_SD(:,1);                        % reziduum
for k=1:iters
    Ar=A*r;
    gamma=dot(r,r)/dot(r,Ar);
    x_SD(:,k+1)=x_SD(:,k)+gamma*r;
    r=r-gamma*Ar;                       % reziduum
    norm(r)
end
figure;
plot(x_SD,'k:'); axis square; axis tight;
```

Iterace včetně počáteční uvádíme níže jako vektory v řádcích výpisu

```
         0         0         0         0
    0.0800    0.0800    0.0800    0.0800
    0.0640    0.0960    0.0960    0.0640
    0.0800    0.1120    0.1120    0.0800
    0.0768    0.1152    0.1152    0.0768
    0.0800    0.1184    0.1184    0.0800
    0.0794    0.1190    0.1190    0.0794
    0.0800    0.1197    0.1197    0.0800
    0.0799    0.1198    0.1198    0.0799
    0.0800    0.1199    0.1199    0.0800
    0.0800    0.1200    0.1200    0.0800
```

Je zřejmé, 10 iterací metody největšího spádu poskytuje lepší aproximaci přesného řešení než Jacobiho a Gauss Seidlova metoda s 15 iteracemi.

Poznámka k příkladu

Další metodou, která by poskytla ještě rychlejší konvergenci je metoda sdružených gradientů (angl. conjugate gradient method). To, že pro úlohy s větším počtem neznámých dochází ke zpomalení konvergence, je známý fakt a existují tzv. techniky předpodmínění (např. multigrid nebo metodu rozdělení oblasti), které konvergenci urychlují.

Metoda největšího spádu je velmi populární při strojovém učení, kde se používá pro minimalizaci obecných nelineárních funkci. Potom je nutné šikovně zvolit krok metody, který se v angličtině označuje jako learning rate.

Úloha k samostatnému řešení

Uvažujme matici $A = \begin{pmatrix} -4 & 3 & 0 & 0 \\ 4 & 20 & 13 & 1 \\ 0 & 0 & 2 & 1 \\ 3 & -6 & 7 & 30 \end{pmatrix}$.

1. Najděte inverzní matici A^{-1} a vypočítejte čísla podmíněnosti

$$\kappa(A) = ||A||||A^{-1}||$$

pro řádkovou, sloupcovou a Frobeniovu normu.

2. Uvažujte nulovou počateční iteraci $x^0 = (0,0,0,0)^T$ a vypočtěte prvních pět iterací Jacobiho and Gauss-Seidelovy iterační metody pro rovnici $Ax = b$, kde $b = (1,1,1,1)^T$. Jak konvergují obě metody?

Úloha k samostatnému řešení

Za pomocí Gelfandovy formule (pro dostatečně vysokou mocninu) pro odhad spektrálního poloměru určete co možná nejpřesněji spektrální poloměry

$$\rho(C_{Jac}), \quad \rho(C_{GS})$$

obou iteračních matic v Příkladě 3.2.33 pro $n = 10$. Uvažujte případy a) řádkové, b) sloupcové, c) Frobeniovy, d) spektrální maticové normy.

3.3 Vlastních čísla matic

Příklad 3.3.35 Pomocí mocninné metody vypočtěme největší vlastní číslo

$$\lambda_{max}(A),$$

kde matice $A := A_{h,n}$ je definována v (3.1) pro případ $h = 1$ a $n = 5$.

Řešení:
V mocninné metodě sestrojujeme posloupnost normalizovaných sloupcových vektorů rekurentně ve tvaru

$$u_k = \frac{Au_{k-1}}{||Au_{k-1}||}, \qquad k = 1, 2, \ldots$$

pro počáteční sloupcový vektor $u_0 \in \mathbb{R}^n$. Pokud označíme $\tilde{u}_k = Au_{k-1}$, lze přeformulovat rekurenci jednodušeji jako

$$u_k = \frac{\tilde{u}_k}{||\tilde{u}_k||}, \qquad k = 1, 2, \ldots$$

Posloupnost takových vektorů konverguje za určitých předpokladů k vlastnímu vektoru u_{max} odpovídajícímu vlastnímu číslu $\lambda_{max}(A)$, tedy platí

$$Au_k \approx \lambda_{max}(A)u_k$$

pro dostatečně velké hodnoty k. Posloupnost odpovídajících tzv. Rayleighových podílů

$$\lambda_k = \frac{u_k^T Au_k}{u_k^T u_k}, \qquad k = 1, 2, \ldots$$

potom konverguje k vlastnímu číslu $\lambda_{max}(A)$. Protože vektory u_k normalizujeme, lze vyjádřit jednodušeji

$$\lambda_{k-1} = u_{k-1}^T \tilde{u}_k, \qquad k = 1, 2, \ldots$$

Iterativní proces demonstrujeme pomocí Kódu 3.4, ve kterém volíme počáteční vektor

$$u_0 = (1, 0, 0, 0, 0)^T.$$

Prvních 8 iterací vychází postupně jako

```
iter=1, lambda=2.00000000, u=(0.894427 -0.447214 0.000000 0.000000 0.000000)
iter=2, lambda=2.80000000, u=(0.771517 -0.617213 0.154303 0.000000 0.000000)
iter=3, lambda=3.14285714, u=(0.675926 -0.675926 0.289683 -0.048280 0.000000)
iter=4, lambda=3.33333333, u=(0.602340 -0.688389 0.387219 -0.114731 0.014341)
iter=5, lambda=3.45454545, u=(0.544428 -0.680535 0.453690 -0.181476 0.041245)
iter=6, lambda=3.53814749, u=(0.497820 -0.663760 0.497820 -0.241367 0.074267)
iter=7, lambda=3.59789719, u=(0.459735 -0.643629 0.526606 -0.292237 0.108022)
iter=8, lambda=3.64059937, u=(0.428391 -0.623114 0.545137 -0.334113 0.139302)
```

a poslední dvě iterace jako

```
iter=33, lambda=3.73204893, u=(0.289318 -0.500643 0.577350 -0.499357 0.288032)
iter=34, lambda=3.73204960, u=(0.289192 -0.500517 0.577350 -0.499483 0.288158)
```

Kód se zastaví, protože rozdíl posledních dvou vypočtených hodnot λ je menší než nastavená hodnota tolerance 'tol=1e-6'. Hodnota z poslední iterace

$$\lambda \approx 3.73204960 \tag{3.14}$$

přibližuje přesnou hodnotu největšího vlastního čísla

$$\lambda_{max} \approx 3.73205081 \tag{3.15}$$

vypočtenou nezávisle pomocí příkazu Matlabu 'eigs(A,1)'.

Kód - Matlab 3.4: Mocninná metoda.

```
n=5 ;                                          % velikost matice
h=1/(n+1); h=1;                                % parametr diskretizace
A =(1/h^2)*gallery('tridiag',n,-1,2,-1);       % sestaveni ridke matice
u=zeros(n,1); u(1)=1;                          % normaliz. pocatecni vektor
lambda_old=-inf;                               % horni odhad max. lambda
iterMax=1e5;                                   % maximalni pocet iteraci
tol=1e-6;                                      % tolerance
for iter=1:iterMax
    u_old=u;
    if 1
        u=A*u;                                 % mocninna metoda
    else
        u=A\u;                                 % inverzni mocninna metoda
    end
    lambda=dot(u_old,u);                       % Rayleighuv podil
    u=u/norm(u);                               % normalizace vektoru
    g=sprintf('%f ', u(1:min(8,n)));
    fprintf('iter=%d, lambda=%2.8f, u=(%s\b)\n', iter, lambda, g)    %
    vypis
    crit=abs(lambda-lambda_old);
    %crit=acos(dot(u_old,u));
    if crit<tol
        break
    end
    lambda_old=lambda;                         % aktualizace
end
```

Poznámka k příkladu

Matice A v našem příkladu je symetrická a pozitivně definitní, má tedy všechna vlastní čísla kladná reálná. Proto mocninná metoda konverguje k největšímu z nich, které je dominantní. Nadefinujeme-li v 2. řádce v programu výše matici A pomocí

```
A=randi(10,n,n)-randi(10,n,n);
```

stává se nesymetrickou maticí s náhodnými celočíselnými prvky. Potom můžeme ověřit, že mocninná metoda konverguje k dominantnímu vlastnímu číslu, tj. číslu největšímu v absolutní hodnotě (pokud existuje jediné). Pokud je však dominantní vlastní číslo komplexní (tedy s ním do páru je dominantní i číslo komplexně sdružené), může mocninná metoda oscilovat. Příkladem je matice

$$A = \begin{pmatrix} -1 & 1 & 5 & 9 & 7 \\ 5 & 1 & -5 & 5 & -3 \\ 4 & -2 & -2 & -4 & 4 \\ -5 & 0 & 3 & -6 & 2 \\ -3 & -7 & 3 & -7 & 6 \end{pmatrix}. \tag{3.16}$$

Přepnutím v 10. řádce realizujeme inverzní mocninou metodu. Ta hledá největší vlastní číslo matice A^{-1}, které v našem příkladě odpovídá převrácené hodnotě nejmenšího vlastního čísla matice A,

$$\lambda_{max}(A^{-1}) = \frac{1}{\lambda_{min}(A)}.$$

Na to je třeba dát při interpretaci výsledků pozor! My konkrétně získáváme

```
iter=8, lambda=3.73205073, u=(0.288698 0.500023 0.577350 0.499977 0.288652)
iter=9, lambda=3.73205080, u=(0.288681 0.500006 0.577350 0.499994 0.288669)
```

a proto obdržíme přibližnou hodnotu

$$\lambda_{min}(A) \approx 1/3.73205080 = 0.26794919,$$

která velmi dobře aproximuje přesnou hodnotu

$$\lambda_{min}(A) \approx 0.267949192431122.$$

Příklad 3.3.36 Pomocí mocninné metody vypočtěme největší a nejmenší vlastní čísla

$$\lambda_{max}(A), \lambda_{min}(A),$$

kde matice $A := A_{h,n}$ je definována v (3.1) pro případ

$$n = 50, h = 1/(n+1).$$

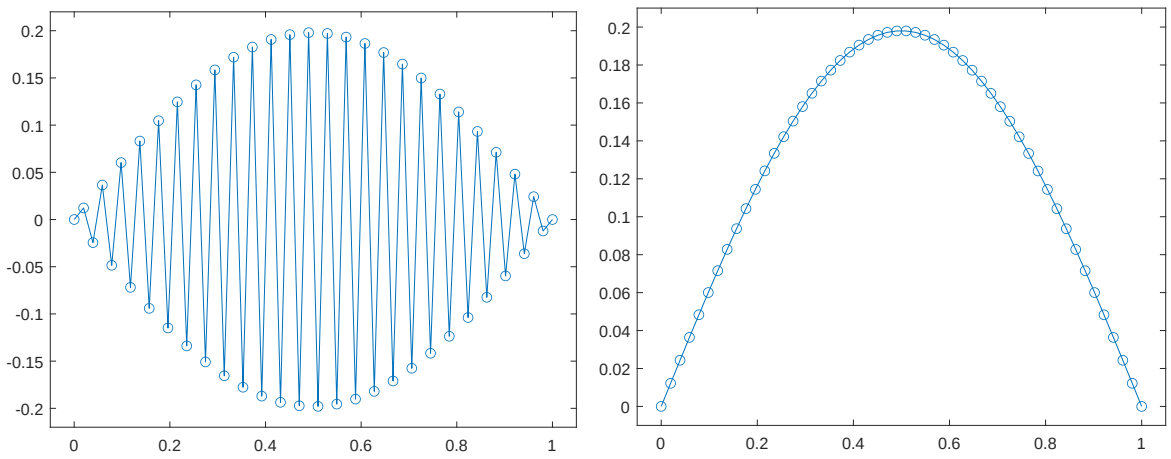
Odpovídající vlastní vektory zobrazte.

Řešení: Připoměňme si, že fakticky aproximujeme numericky nejmenší a největší vlastní číslo spojité úlohy

$$-u''(x) = \lambda u, \qquad u(0) = u(1) = 0 \tag{3.17}$$

o které víme, že má nekonečně mnoho vlastních čísel a odpovídajících vlastních funkcí ve tvaru (odvoďte)

$$\lambda_k = k\pi^2, \quad u_k(x) = \sin(k\pi x), \qquad k \in \mathbb{N}. \tag{3.18}$$



Obrázek 3.4: Vlastní funkce odpovídající největšímu a nejmenšímu vlastnímu číslu.

Změnou parametrů n, h v Kódu 3.4 dostaneme poslední iteraci ve tvaru

```
iter=2357, lambda=10394.13334123, u=(0.012250 -0.024453 0.036561 -0.048529)
```

kde vypisujeme jen první čtyři složky odpovídajícího vlastního vektoru. Dále přepnutím mocninné methody na inverzní mocninou metodu máme

```
iter=7, lambda=0.10135321, u=(0.012194 0.024341 0.036397 0.048314)
```

Proto získáváme přibližné hodnoty

$$\lambda_{max}(A) \approx 10394.1333, \quad \lambda_{min}(A) \approx \frac{1}{0.10135321} = 9.8665.$$

Odpovídající vlastní vektory je možné vizualizovat pomocí dodatečného příkazu

```
plot(linspace(0,1,n+2),[0; u; 0],'o-')
```

ke kterém přidáváme krajní hodnoty 0, abychom zohlednili i okrajovou podmínku. Výsledek je zobrazen na Obrázku 3.4.

Příklad 3.3.37 Pro sloupce matice

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{pmatrix}$$

proveďte Gram-Schmidtův proces ortogonalizace a získejte z něho QR rozklad matice A.

Řešení:
Jednotlivé sloupce matice A označíme jako vektory

$$a_1 = \begin{pmatrix} 2 \\ -1 \\ 0 \\ 0 \end{pmatrix}, \quad a_2 = \begin{pmatrix} -1 \\ 2 \\ -1 \\ 0 \end{pmatrix}, \quad a_3 = \begin{pmatrix} 0 \\ -1 \\ 2 \\ -1 \end{pmatrix}, \quad a_4 = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 2 \end{pmatrix}.$$

Gram-Schmidtův proces vytvoří ortogonální bázi sloupcových vektorů q_1, q_2, q_3, q_4 konstruovaných z vektorů a_1, a_2, a_3, a_4 podle schématu

$$\begin{aligned} q_1 &:= a_1, \\ q_2 &:= a_2 - r_{1,2} \cdot q_1, \\ q_3 &:= a_3 - r_{1,3} \cdot q_1 - r_{2,3} \cdot q_2, \\ q_4 &:= a_4 - r_{1,4} \cdot q_1 - r_{2,4} \cdot q_2 - r_{3,4} \cdot q_3 \end{aligned}$$

ve kterém pomocí podílů skalárních součinů definujeme koeficienty

$$r_{k,l} := \frac{a_l^T q_k}{q_k^T q_k}, \qquad k \in \{1,2,3\}, l \in \{k+1, \ldots, 4\}.$$

Gram-Schmidtův proces je ekvivalentní (ověřte) maticovému QR rozkladu

$$\begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 \end{pmatrix} \begin{pmatrix} 1 & r_{1,2} & r_{1,3} & r_{1,4} \\ & 1 & r_{2,3} & r_{2,4} \\ & & 1 & r_{3,4} \\ & & & 1 \end{pmatrix}, \tag{3.19}$$

kde Q je matice složená z vektorů $q_1, \ldots, q_4$ a R je horní trojúhelníková matice složená z koeficientů $r_{k,l}$. Pomocí kódu

Kód - Mathematica 3.5: QR rozklad matice.

```
A = {{2,-1,0,0},{-1,2,-1,0},{0,-1,2,-1},{0,0,-1,2}};
dim1 = Dimensions[A][[1]]; dim2 = Dimensions[A][[2]];
Q = Table[Table[0, dim2], dim1];
R = Table[Table[0, dim2], dim2];
For[i=1,i<= dim2,i++, ui = A[[All, i]]; si = Table[0, dim1];
 For[j=1,j<= i-1,j++, sksj = A[[All, i]].Q[[All, j]];
  sksj2 = Q[[All, j]].Q[[All, j]];
  R[[j,i]] = sksj/sksj2;
  si += sksj*Q[[All, j]]/sksj2];
  Q[[All,i]] = ui - si;
  R[[i,i]] = (Q[[All,i]].A[[All,i]])/(Q[[All,i]].Q[[All,i]])]
Q // MatrixForm
R // MatrixForm
```

získáváme (spočtěte ručně)

$$Q = \begin{pmatrix} 2 & 3/5 & 2/7 & 1/6 \\ -1 & 6/5 & 4/7 & 1/3 \\ 0 & -1 & 6/7 & 1/2 \\ 0 & 0 & -1 & 2/3 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & -4/5 & 1/5 & 0 \\ & 1 & -8/7 & 5/14 \\ & & 1 & -4/3 \\ & & & 1 \end{pmatrix},$$

přičemž platí $A = QR$. Ve výpočtech ještě pokračujme dále. Matice Q výše je opravdu ortogonální, neboť

$$D = Q^T Q = \begin{pmatrix} 5 & & & \\ & \frac{14}{5} & & \\ & & \frac{15}{7} & \\ & & & \frac{5}{6} \end{pmatrix}$$

je diagonální matice. Přeškálováním pomocí transformací

$$\tilde{Q} := QD^{-1/2}, \quad \tilde{R} := D^{1/2}R$$

získáme matice

$$\tilde{Q} = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{3}{\sqrt{70}} & \frac{2}{\sqrt{105}} & \frac{1}{\sqrt{30}} \\ -\frac{1}{\sqrt{5}} & 3\sqrt{\frac{2}{35}} & \frac{4}{\sqrt{105}} & \sqrt{\frac{2}{15}} \\ 0 & -\sqrt{\frac{5}{14}} & 2\sqrt{\frac{3}{35}} & \sqrt{\frac{3}{10}} \\ 0 & 0 & -\sqrt{\frac{7}{15}} & 2\sqrt{\frac{2}{15}} \end{pmatrix}, \qquad \tilde{R} = \begin{pmatrix} \sqrt{5} & -\frac{4}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ & \sqrt{\frac{14}{5}} & -8\sqrt{\frac{2}{35}} & \sqrt{\frac{5}{14}} \\ & & \sqrt{\frac{15}{7}} & -4\sqrt{\frac{5}{21}} \\ & & & \sqrt{\frac{5}{6}} \end{pmatrix}$$

a platí $A = \tilde{Q}\tilde{R}$, přičemž matice $\tilde{Q}$ je ortonormální tedy splňuje rovnost $Q^T Q = I$, kde I je identická matice. Transformace lze jednoduše realizovat dodatečným kódem:

Kód - Mathematica 3.6: QR rozklad matice - normalizace.

```
S = Transpose[Q].Q;
Qtilde = Q.MatrixPower[S, -1/2];
Rtilde = MatrixPower[S, 1/2].R;
Qtilde // MatrixForm
Rtilde // MatrixForm
```

Poznámka k příkladu

V praxi se nerealizuje QR rozklad pomocí Gram-Schmidtova rozkladu popsaného výše, ale numericky stabilnějšími metodami, např. tzv. Givensovými rotacemi nebo Householderovými reflexemi.

Příklad 3.3.38 Diagonalizací pomocí QR metody vypočtěte všechna vlastní čísla matice

$$A = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{pmatrix}.$$

Řešení:
Metoda diagonalizace převádí matici $A_0 := A$ postupně na posloupnost podobných matic

$$A_1, A_2, \ldots, A_k, A_{k+1}, \ldots$$

které mají stejná vlastní čísla. Pro dané $k \in \mathbb{N}$ sestrojíme QR rozklad matice

$$A_k = Q_k R_k$$

a další matici A_{k+1} definujeme jako součin matic Q_k a R_k v opačném pořadí. tj.

$$A_{k+1} := R_k Q_k.$$

Matice A_{k+1} a A_k jsou podobné, protože platí

$$Q_k^T A_k Q_k = Q_k^T Q_k R_k Q_k = R_k Q_k = A_{k+1}.$$

Proto mají stejná vlastní čísla (matematickou indukcí ukážeme, že jsou shodná s původní maticí A). Jednotlivé matice vygenerujeme pomocí kódu

Kód - Matlab 3.7: QR metoda.

```
n = 4;
A = full(gallery('tridiag',n,-1,2,-1));      % sestaveni ridke matice
tol = 1e-4;                                  % tolerance
Asim=A; V = eye(n); i=0;
while norm(tril(Asim,-1))>tol
    i = i+1
    [Q,R] = qr_own(Asim);
    Asim = R*Q
    V = V*Q;
end
D = diag(diag(Asim))
check = norm(A*V-V*D)                         % kontrola vl. cisel a vekt.
```

který poskytuje matice A_1, A_2, A_3 oddělených čarou:

```
    2.8000   -0.7483    0.0000   -0.0000
   -0.7483    2.3429   -0.8748         0
         0   -0.8748    2.1905    0.6236
         0         0    0.6236    0.6667
    ------------------------------------
    3.1429   -0.5803   -0.0000    0.0000
   -0.5803    2.6349   -0.7007   -0.0000
         0   -0.7007    1.8222   -0.1477
         0         0   -0.1477    0.4000
    ------------------------------------
    3.3333   -0.4689   -0.0000   -0.0000
   -0.4689    2.7585   -0.4270    0.0000
         0   -0.4270    1.5249    0.0361
         0         0    0.0361    0.3832
```

Matice konverguje podle teorie k diagonální matici (pro nesymetrickou matici A obecněji k horní trojúhelníkové matici) a čísla na diagonále aproximují všechna vlastní čísla. Poslední matice A_{30} vygenerovaná s ohledem na zastavovací kritérium v 5. řádce vychází

```
  3.6180   -0.0001    0.0000   -0.0000
 -0.0001    2.6180   -0.0000    0.0000
       0   -0.0000    1.3820    0.0000
       0         0   -0.0000    0.3820
```

a uložená v proměnné `Asim`. To znamená, že čísla z diagonály matice A_{30}

```
  3.6180    2.6180    1.3820    0.3820
```

představují přibližná vlastní čísla matice A, která se shodují s přesnými vlastními čísly ve všech uvedených cifrách. Zároveň lze ověřit, že matice V, která obsahuje součiny jednotlivých ortogonálních matic je rovna

```
  0.3718    0.6015    0.6015    0.3717
 -0.6015   -0.3717    0.3717    0.6015
  0.6015   -0.3718   -0.3717    0.6015
 -0.3717    0.6015   -0.6015    0.3717
```

a obsahuje ve sloupcích jednotlivé vlastní vektory. To platí ale jen pro původní symetrickou matici A (experimentujte s kódem).

Příklad 3.3.39 Pro nesymetrickou matici

$$A = \begin{pmatrix} 2 & -3 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{pmatrix}$$

nalezněte pomocí QR metody nebo její vhodné modifikace všechna vlastní čísla a vektory.

Řešení:
Oproti příkladu 3.3.38 se A liší jen prvkem na pozici (1, 2). Kód 3.7 poskytuje (ověřte) po 25. iteracích matici A_{25} podobnou k A ve tvaru

```
  4.0743   -0.4211   -0.9275    1.5554
 -0.0001    2.8350   -0.3067    0.6246
       0   -0.0000    1.1650    0.2430
       0         0   -0.0000   -0.0743
```

a matici V ve tvaru

```
  0.7646    0.4066    0.3388   -0.3677
 -0.5287   -0.0059    0.3723   -0.7628
  0.3320   -0.6243   -0.5201   -0.4791
 -0.1600    0.6670   -0.6901   -0.2310
```

Protože je A_{25} horní trojúhelníková, představuje její diagonála

```
  4.0743    2.8350    1.1650   -0.0743
```

aproximace všech vlastních čísel matice A. Pozor, V ale nyní není maticí vlastních vektorů matice A. Tu lze ale dopočítat. Pomocí dodatečného kôdu

Kód - Matlab 3.8: QR metoda, nesymetrické rozšíření.

```
Y = zeros(n,n); Y(1,1) = 1;
for i=2:n
    A11 = Asim(1:i-1,1:i-1);
    a = Asim(1:i-1,i);
    y = -( A11-Asim(i,i)*eye(i-1) ) \ a;
    Y(1:i-1,i) = y;
    Y(i,i) = 1;
    if i~=n
        Y(i+1,i) = 0;
    end
end
check = norm(Asim*Y-Y*D)                    % kontrola vl. cisel a vekt.
W=V*Y
check = norm(A*W-W*D)                       % kontrola vl. cisel a vekt.
```

sestavíme novou matici Y ve tvaru

```
    1.0000    0.3398    0.3454   -0.4427
         0    1.0000    0.1837   -0.2354
         0         0    1.0000   -0.1961
         0         0         0    1.0000
```

a ta představuje matici vlastních vektorů horní trojúhelníkové matice `Asim`. Pokud tuto matici vynásobíme zleva maticí V, získáme novou matici $W = V\,Y$ ve tvaru

```
    0.7646    0.6664    0.6775   -0.8684
   -0.5287   -0.1855    0.1886   -0.6004
    0.3320   -0.5115   -0.5201   -0.3771
   -0.1600    0.6126   -0.6228   -0.1818
```

která je již správnou maticí vlastích čísel původní nesymetrické matice A.

Poznámka k příkladu

O tom, jak k dané horní trojúhelkové matici sestavit matici odpovídajících vlastních čísel, tedy jak funguje Kód 3.8, se dozvíme například v knize [5], sekce 5.8.2.

Úloha k samostatnému řešení

Mocninou metodou spočtěte největší vlastní číslo $\lambda_{max}(A)$ kde matice $A := A_{h,n}$ je definována v (3.1) v případě $n = 1000, h = 1$. Uvažujme počáteční sloupcový vektor $u_0 = (1,0,\ldots,0)^T$. Zobrazte graf závislosti chyby aproximace $\lambda_{max}(A)$ vypočtené pomocí

$$10, 100, 1000, 10000$$

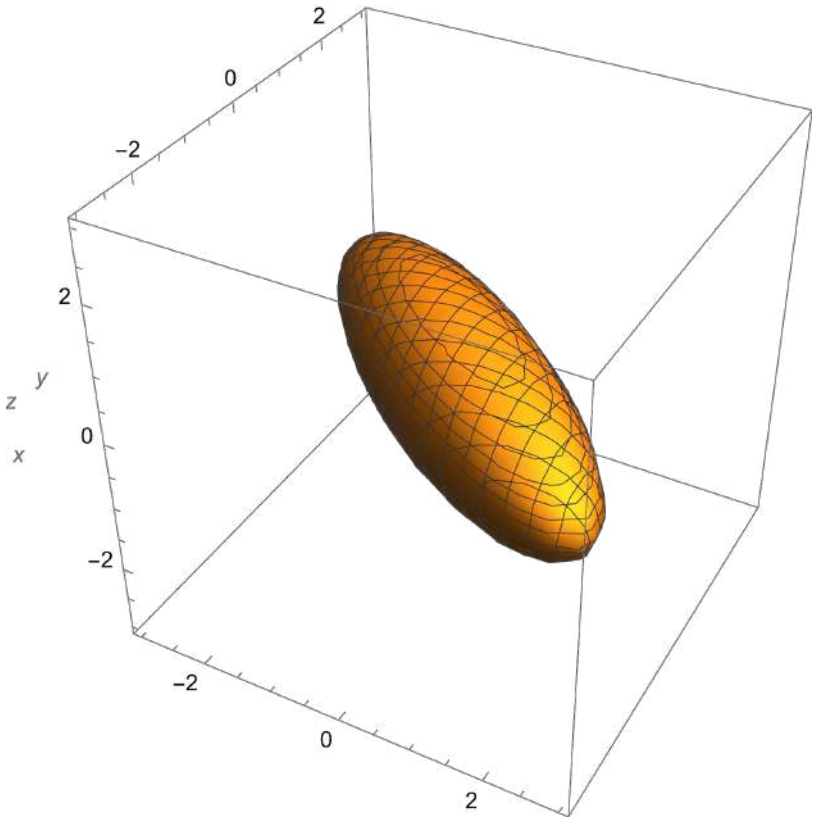
iterací v logaritmické škále. Přesnou hodnotu největšího vlastního čísla získejte nezávisle na výpočtu, např. pomocí příkazu „eigs(A,1)“ v Matlabu.

Úloha k samostatnému řešení

Elipsoid je popsán rovnicí

$$2x^2 + 3y^2 + 4z^2 + 4xy = 5.$$

a zobrazen na Obrázku 3.5. Vypočtěte numericky vlastní čísla a vlastní vektory odpovídající kvadratické formy a z nich určete směry a velikosti hlavních os elipsoidu.



Obrázek 3.5: Elipsoid z úlohy.

3.4 Singulární čísla matic

Příklad 3.4.40 Sestavte singulární rozklad matice

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \end{pmatrix}.$$

Řešení: Singulární rozklad hledáme ve tvaru

$$A = U\,\Sigma\,V^T, \tag{3.20}$$

kde matice U, V jsou čtvercové ortonormální a Σ je obdélníková matice s nezápornými prvky pouze na hlavní diagonále uspořádanými od největšího k nejmenšímu. Platí

$$AA^T = U\,\Sigma\,V^T V\,\Sigma^T\,U^T = U\,(\Sigma\,\Sigma^T)\,U^T,$$
$$A^TA = V\,\Sigma^T\,U^T U\,\Sigma^T\,V^T = V\,(\Sigma^T\,\Sigma)\,V^T,$$

což je ekvivalentní dvěma úlohám na vlastní čísla psaným v maticovém tvaru jako

$$(AA^T)\,U = U\,(\Sigma\,\Sigma^T),$$
$$(A^TA)\,V = V\,(\Sigma^T\,\Sigma).$$

Sloupcové vektory matic U a V tedy představují vlastní vektory matic AA^T a A^TA a čísla na diagonálách matic $\Sigma\,\Sigma^T$ a $\Sigma^T\,\Sigma$ jejich vlastní čísla. V našem příkladě platí

$$AA^T = \begin{pmatrix} 30 & 60 \\ 60 & 120 \end{pmatrix}, \qquad A^TA = \begin{pmatrix} 5 & 10 & 15 & 20 \\ 10 & 20 & 30 & 40 \\ 15 & 30 & 45 & 60 \\ 20 & 40 & 60 & 80 \end{pmatrix}.$$

Řešení úloh na vlastní čísla poskytuje matice

$$U \approx \begin{pmatrix} 0.4472 & -0.8944 \\ 0.8944 & 0.4472 \end{pmatrix}, \qquad \Sigma\,\Sigma^T = \begin{pmatrix} 150 & 0 \\ 0 & 0 \end{pmatrix},$$

$$V \approx \begin{pmatrix} 0.3586 & -0.8944 & 0.1952 & 0.1826 \\ 0.7171 & 0.4472 & 0.3904 & 0.3651 \\ -0.5976 & 0 & 0.5855 & 0.5477 \\ 0 & 0 & -0.6831 & 0.7303 \end{pmatrix}, \qquad \Sigma^T\,\Sigma = \begin{pmatrix} 150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Protože výsledná nezáporná matice Σ musí mít stejnou velikost jako A, je nutné aby platilo

$$\Sigma \approx \begin{pmatrix} 12.2474 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Všechny matice je možné vypočítat pomocí následujícího kódu:

Kód - Matlab 3.9: Singulární rozklad.

```
format short
A=[1 2 3 4; 2 4 6 8]

AAt=A*A'
AtA=A'*A

[U,SigmaSigmat] = eig(AAt);                    % uloha na vlastni cisla
```

```
[d1, ind1] = sort(diag(SigmaSigmat),'descend'); % serazeni
U = U(:, ind1)
SigmaSigmat=diag(d1)

[V,SigmatSigma] = eig(AtA);                     % uloha na vlastni cisla
[d2, ind2] = sort(diag(SigmatSigma),'descend'); % serazeni
V = V(:, ind2)
SigmatSigma=diag(d2)

if numel(d1)<=numel(d2)                         % orezani Sigma
   d=d1;
else
   d=d2;
end
Sigma=full(sparse(1:numel(d),1:numel(d),sqrt(d),size(A,1),size(A,2)))

check=norm(U*S*V'-A)                            % kontrola rozkladu
```

Všimněme si, že kód využívá dvakrát řešič vlastních vektorů a vlastních čísel „eig“. Poslední řádka kódu kontroluje správnost singulárního rozkladu.

Poznámka k příkladu

Připomeňme, že jsme matici $A \in \mathbb{R}^{2\times 4}$ v našem příkladu rozložili pomocí singulárního rozkladu (3.20) na matice $U \in \mathbb{R}^{2\times 2}, \Sigma \in \mathbb{R}^{2\times 4}, V \in \mathbb{R}^{4\times 4}$. Protože matice Σ obsahuje jen jedno nenulové singulární číslo, lze původní matice oříznout a získat

$$\tilde{U} \approx \begin{pmatrix} 0.4472 \\ 0.8944 \end{pmatrix}, \qquad \tilde{\Sigma} \approx \begin{pmatrix} 12.2474 \end{pmatrix}, \qquad \tilde{V} \approx \begin{pmatrix} 0.3586 \\ 0.7171 \\ -0.5976 \\ 0 \end{pmatrix}.$$

Ověřme, že platí $A = \tilde{U}\tilde{\Sigma}\tilde{V}^T$. Takovému rozkladu říkáme tzv. ekonomický singulární rozklad. Matlab samozřejmě disponuje vlastním řešičem a příkazy

```
[U2,Sigma2,V2] = svd(A)
[U3,Sigma3,V3] = svd(A,"econ")
```

vygenerují příslušné singulární rozklady (ověřte).

Úloha k samostatnému řešení

Sestavte pro matici

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \end{pmatrix}$$

její pseudoinverzní (Moore–Penrose inverzní) matici $A^\dagger$. Dále vyřešte s její pomocí soustavy rovnic $Ax = b$, kde

$$b = \begin{pmatrix} 10 \\ 20 \end{pmatrix} \qquad \text{nebo} \qquad b = \begin{pmatrix} 11 \\ 20 \end{pmatrix}.$$

Srovnejte získaná řešení s obecnými řešení soustav lineárních rovnic z lineární algebry.

Poznámka k příkladu

Zajímavou úlohou je komprese fotografií pomocí singulárního rozkladu. Uvažujme černobílou fotografii vánočního trhu z Obrázku 3.6 (nahoře). Fotografie je uložena jako matice o velikosti 1530×2040 a má tedy 1530 singulárních čísel. Zanedbáním velkého množství malých singulárních čísel a odpovídajících levých a pravých singulárních vektorů lze dosáhnout uspokojivé aproximace matice a tím komprimovat původní fotografii, viz dolní část Obrázku 3.6.

Obrázek 3.6: Původní fotografie (nahoře) reprezentovaná pomocí 1530 singulárních čísel a komprimovaná (dole) za použití 80 největších singulárních čísel.

3.5 Řešení diferenciálních rovnic

Příklad 3.5.41 Vyřešte diferenciální rovnici s počáteční podmínkou

$$y(x)y'(x) = 2x^4, \qquad y(0) = 1$$

přibližně pomocí Eulerovy explicitní metody. K jaké hodnotě konverguje metoda v bodě $x_{max} = 2$? Porovnejte s analytickým řešením a odhadněte řád konvergence metody srovnáním chyb pro 10, 100, 1000 nebo 10000 rovnoměrných kroků.

Řešení: Hledáme aproximaci funkce $y = y(x)$ na intervalu $[0, 2]$. Diferenciální rovnici je vhodné přepsat do tvaru

$$y'(x) = \frac{2x^4}{y} =: f(x, y),$$

kde derivace hledané funkce stojí samostatně na levé straně. Přesné řešení diferenciální rovnice je dáno (odvoďte pomocí separace proměnných) jako

$$y(x) = \sqrt{\frac{4x^5}{5} + 1}$$

a tedy platí $y(2) = \frac{\sqrt{665}}{5} \approx 5.157518783291051$. Eulerova explicitní metoda generuje posloupnost aproximací řešení podle rekurentní formule

$$\begin{aligned} x_{i+1} &= x_i + h, \\ y_{i+1} &= y_i + h\frac{2x_i^4}{y_i} \end{aligned} \tag{3.21}$$

pro $i = 0, 1, \ldots, n-1$, kde n představuje počet kroků metody. Délka kroku $h := 2/n$ odpovídá rozdělení intervalu $[0, 2]$ na n stejně dlouhých podintervalů. Hodnoty

$$x_0 = 0, \quad y_0 = 1$$

odpovídají počáteční podmínce úlohy. Posloupnost $\{y_k\}$ necháme vygenerovat ve všech čtyřech případech pomocí Kódu 3.10, který realizuje Eulerovy metody pro různá n a poskytuje výpis:

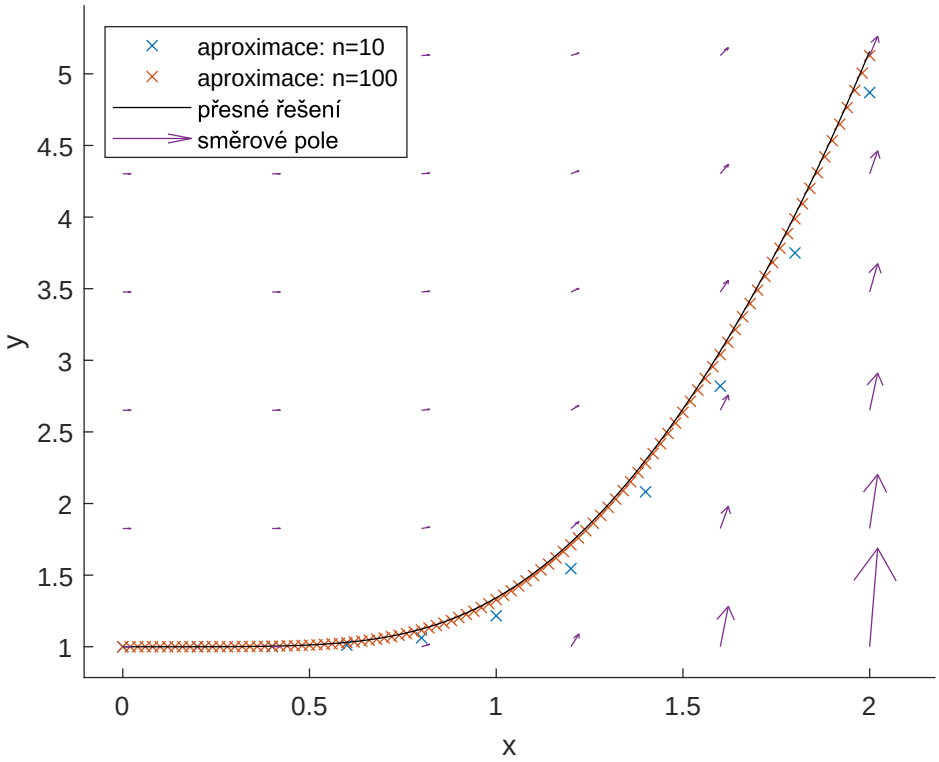
```
h=2.00e-01,  y_h(2)=4.869488,  e(2)=2.8803e-01
h=2.00e-02,  y_h(2)=5.127952,  e(2)=2.9567e-02
h=2.00e-03,  y_h(2)=5.154556,  e(2)=2.9628e-03
h=2.00e-04,  y_h(2)=5.157222,  e(2)=2.9634e-04
```

Kromě délky kroku h a přibližné hodnoty $y_h(2)$ jsou zobrazeny v posledním sloupci absolutní chyby $e(2) = |y_h(2) - y(2)|$. Ty se postupně snižují přibližně 10 krát. Protože délka kroku h se v uvažovaných případech také 10 krát snižuje, platí zhruba odhad $e(2) \approx h$. Numericky jsme tedy demonstrovali první řád konvergence Eulerovy explicitní metody. Rozšířením kódu je možné také vykreslit jednotlivé aproximace řešení, graf přesného řešení a směrového pole odpovídající diferenciální rovnici, viz Obrázek 3.7.

Poznámka k příkladu

Implementací Heunovy metody druhého řádu docílíme hodnot

```
h=2.00e-01,  y_h(2)=5.205293,  e(2)=4.7775e-02
h=2.00e-02,  y_h(2)=5.157938,  e(2)=4.1890e-04
h=2.00e-03,  y_h(2)=5.157523,  e(2)=4.1415e-06
h=2.00e-04,  y_h(2)=5.157519,  e(2)=4.1369e-08
```



Obrázek 3.7: Aproximace získané pomocí Eulerovy metody.

Absolutní chyba v druhém sloupci se postupně zhruba 100 krát snižuje, tedy $e(2) \approx h^2$. V praxi jsou populární Runge Kutovy metody čtvrtého řádu, které dosahují hodnot:

```
h=2.00e-01,  y_h(2)=5.157766,  e(2)=2.4677e-04
h=2.00e-02,  y_h(2)=5.157519,  e(2)=2.2689e-08
h=2.00e-03,  y_h(2)=5.157519,  e(2)=2.2515e-12
h=2.00e-04,  y_h(2)=5.157519,  e(2)=9.7700e-15
```

Zde se chyba snižuje zhruba 10000 krát a platí $e(2) \approx h^4$. Jen v posledním sloupci nám formát čísla 'double' v MATLABu nestačí konvergenci zachytit, bylo by zapotřebí více desetinných míst.

Kód - Matlab 3.10: Řešení diferenciální rovnice.

```
clearvars; close all;
f=@(x,y) (2*x.^4)./(y);                          % f
x0=0; y0=1;                                      % pocatecni podminka
xmax=2;                                          % konecna hodnota
all_n=[10 100 1000 10000];                       % pocty kroku
yExact=@(x)sqrt(((4*x.^5)/5)+1);                 % presne reseni
solver=@euler;                                   % volba resice

for j=1:numel(all_n)                             % cyklus pro ruzna n
   xy=solver(f,x0,xmax,y0,all_n(j));              % resic
   all_xy{j}=xy;
   h=xy(2,1)-xy(1,1);                            % delka kroku
   fprintf('   h=%2.2d,   y_h(%d)=%f', h, xmax, xy(end,2))   % vypis
   if exist('yExact','var')
       e=abs(xy(end,2)-yExact(xmax));
       fprintf(',   e(%d)=%2.4d \n', xmax, e);
   else
```

```
        fprintf('\n');
    end
end

function xy=euler(f,x0,xMax,y0,n)          % Eulerova metoda
    x=linspace(x0,xMax,n+1)'; h=x(2)-x(1);
    y=zeros(n+1,1); y(1)=y0;
    for i=1:n
        y(i+1)=y(i)+h*f(x(i),y(i));
    end
    xy=[x, y];
end

function xy=heun(f,x0,xMax,y0,n)     % Heunova metoda
    x=linspace(x0,xMax,n+1)';
    h=x(2)-x(1);
    y=zeros(n+1,1); y(1)=y0;
    for i=1:n
        k1=f(x(i),y(i));
        k2=f(x(i+1),y(i)+h.*k1);
        y(i+1)=y(i)+h*(1/2)*(k1+k2);
    end
    xy=[x, y];
end

function xy=rk4(f,x0,xMax,y0,n)     % RK4 metoda
    x=linspace(x0,xMax,n+1)';
    h=x(2)-x(1);
    y=zeros(n+1,1); y(1)=y0;
    for i=1:n
        k1=h*f(x(i),y(i));
        k2=h*f(x(i)+(1/2)*h,y(i)+(1/2)*k1);
        k3=h*f(x(i)+(1/2)*h,y(i)+(1/2)*k2);
        k4=h*f(x(i)+h,y(i)+k3);
        y(i+1)=y(i)+(1/6)*(k1+2*k2+2*k3+k4);
    end
    xy=[x, y];
end
```

Příklad 3.5.42 Určete oblast stability Heunovy metody dané předpisem

$$\begin{aligned} x_{i+1} &= x_i + h, \\ k_1 &= f(x_i, y_i), \\ k_2 &= f(x_{i+1}, y_i + hk_1), \\ y_{i+1} &= y_i + \frac{h}{2}(k_1 + k_2) \end{aligned} \tag{3.22}$$

pro $i = 0, 1, \ldots, n-1$.

Řešení: Stabilita se vyšetřuje pro řešení $y(x) = e^{\lambda x}$ diferenciální rovnice

$$y'(x) = \lambda y = f(y)$$

splňující počáteční podmínku $y(0) = 1$. Dosazením do (3.22) získáváme postupně

$$k_1 = \lambda y_i,$$

z toho dále

$$k_2 = \lambda(y_i + hk_1) = \lambda y_i + h\lambda^2 y_i$$

a na závěr

$$y_{i+1} = y_i + \frac{h}{2}(\lambda y_i + \lambda y_i + h\lambda^2 y_i) = \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right) y_i. \tag{3.23}$$

Dále uvažujeme je případy $\lambda < 0$ pro které platí $\lim_{x\to\infty} e^{\lambda x} = 0$, tedy přesné řešení diferenciální rovnice konverguje do nuly. Aby Heunova metoda generovala posloupnost aproximací $(y_i)_{i=1}^{\infty}$, která také konverguje do nuly, je nutné aby geometrická posloupnost (3.23) splňovala podmínku na její kvocient ve tvaru

$$|1 + h\lambda + \frac{(h\lambda)^2}{2}| < 1.$$

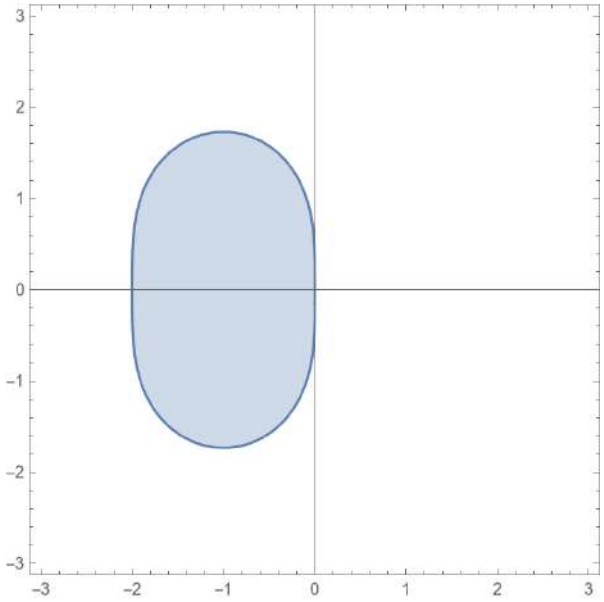
Zavedeme-li substituci $z = h\lambda$, tak dostaneme podmínku stability ve tvaru

$$|1 + z + \frac{z^2}{2}| < 1 \tag{3.24}$$

a jejím řešení jsou čísla $z \in (-2, 0)$. To ovšem znamená, že $h\lambda \in (-2, 0)$ neboli

$$h < \frac{2}{-\lambda}.$$

Tím jsme ukázali, že Heunova metoda je podmínečně stabilní a krok její diskretizace musí splňovat podmínku výše.



Obrázek 3.8: Oblast stability Heunovy metody.

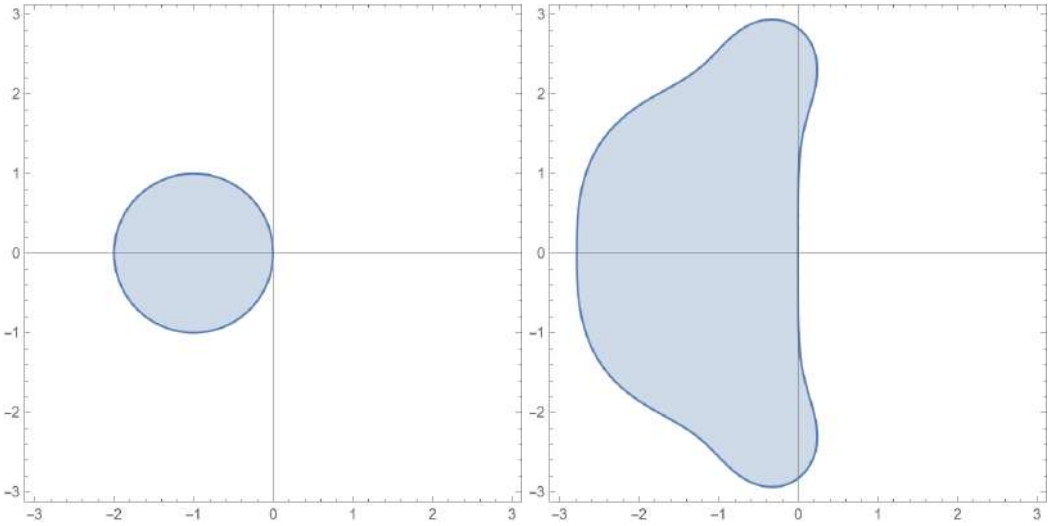
Obecněji lze uvažovat i diferenciální rovnice v komplexním oboru, pro které $\lambda \in \mathbb{C}$ a podmínku (3.24) řešíme pro $z \in \mathbb{C}$. Kód níže množinu řešení v komplexní rovině zobrazuje a výsledek je uveden na Obrázku 3.8.

Kód - Mathematica 3.11: Stabilita Heunovy metody.

```
d = 2;
q[z_] = Normal[Series[Exp[z], {z, 0, d}]]
ComplexRegionPlot[Abs[q[z]] <= 1, {z, -3 - 3 I, 3 + 3 I},
 Axes -> True, AxesOrigin -> {0, 0}]
```

Poznámka k příkladu

Lze ukázat, že kvocienty geometrických posloupností vygenerované pomocí Eulerovy metody a Runge-Kutovy metody 4. řádu jsou přímo dány Taylorovými polynomy stupně 1 a 4 pro funkce e^z v okolí bodu $z = 0$ (ukažte). Proto je možné modifikací Kódu 3.23 získat oblasti jejich stability uvedené na Obrázku 3.9.



Obrázek 3.9: Oblasti stability pro Eulerovu metodu (vlevo) and Runge-Kutovu metodu 4. řádu (vpravo).

Úloha k samostatnému řešení

Vyřešte numericky diferenciální rovnici s počáteční podmínkou

$$y'(x) = \sin(y + x), \qquad y(0) = 1$$

na intervalu (0,10) za použití $10, 100, 1000$ rovnoměrně rozložených kroků. Srovnejte s řešením ve WolframAlpha.

Úloha k samostatnému řešení

Vyřešte numericky diferenciální rovnici nelineárního kyvadla s počáteční podmínkou

$$\varphi''(t) + \frac{g}{l} \sin \varphi(t) = 0, \qquad \varphi'(0) = 1,\ \varphi(0) = \pi/2$$

na intervalu $(0, 20)$. Volte $l = 1$ (délka kyvadla) a $g = 9.81$ (gravitační zrychlení).
Nápověda: převeďte rovnici druhého řádu pomocí substituce $\omega(t) = \varphi'(t)$ na soustavu rovnic prvního řádu pro neznámé funkce $\varphi = \varphi(t), \omega = \omega(t)$ představující úhel a úhlovou rychlost kyvadla a aplikujte vhodnou numerickou metodu.

Příklad 3.5.43 Pomocí metody střely vyřešte okrajovou úlohu pro diferenciální rovnici

$$y''(x) = \frac{3\,y^2(x)}{2}, \qquad y(0) = 4, \quad y(1) = 1$$

na intervalu $(0, 1)$.

Řešení: Metoda střely (angl. shooting method) spočívá v nahrazení okrajové podmínky na jednom okraji podmínkou na derivaci hledané funkce na okraji druhém. Namísto okrajových podmínek

$$y(0) = 4, \quad y(1) = 1$$

předpokládáme počáteční podmínky

$$y(0) = 4, \quad y'(0) = \mu$$

a hodnotu $\mu \in \mathbb{R}$ hledáme tak, aby výsledné řešení diferenciální rovnice splnilo nahrazenou podmínku $y(1) = 1$. Vhodné hodnoty parametru μ hledáme za pomoci následujícího kódu:

Kód - Mathematica 3.12: Metoda střely.

```
(* vypocet rovnice y''=(3/2)y^2, y(0)=4, y(1)=1 na intervalu (0,1) *)
f[y_, x_] = (3/2) (y)^2;
eq = {y''[x] == f[y[x], x]};
mu1 = -4; mu2 = -8; mu3 = -16; mu4 = -35.85;
s1 = NDSolve[{eq, y[0] == 4, y'[0] == mu1}, y, {x, 0, 1}];
s2 = NDSolve[{eq, y[0] == 4, y'[0] == mu2}, y, {x, 0, 1}];
s3 = NDSolve[{eq, y[0] == 4, y'[0] == mu3}, y, {x, 0, 1}];
s4 = NDSolve[{eq, y[0] == 4, y'[0] == mu4}, y, {x, 0, 1}];
p1 = Plot[y[x] /. s1, {x, 0, 1}, PlotStyle -> {Orange}];
p2 = Plot[y[x] /. s2, {x, 0, 1}, PlotStyle -> {Blue}];
p3 = Plot[y[x] /. s3, {x, 0, 1}, PlotStyle -> {Green}];
p4 = Plot[y[x] /. s4, {x, 0, 1}, PlotStyle -> {Red}];
Show[p1, p2, p3, p4, PlotRange -> {{0, 1}, {-10, 8}},
 AxesOrigin -> Automatic]
y[1] /. s1
y[1] /. s2
y[1] /. s3
y[1] /. s4
```

Kód využívá opakovaně numerický řešič diferenciálních rovnic „NDSolve“ systému Mathematica. Řešení pro různé hodnoty parametrů μ jsou ukázána na Obrázku 3.10. Všechna představují konvexní funkce (jak to zdůvodníte?). Modifikací kódu ověříme, že pro kladné hodnoty μ řešení $y_\mu = y_\mu(x)$ roste a dosahuje v bodě $x = 0$ velkých hodnot. Pro hodnotu

$$\mu = -8$$

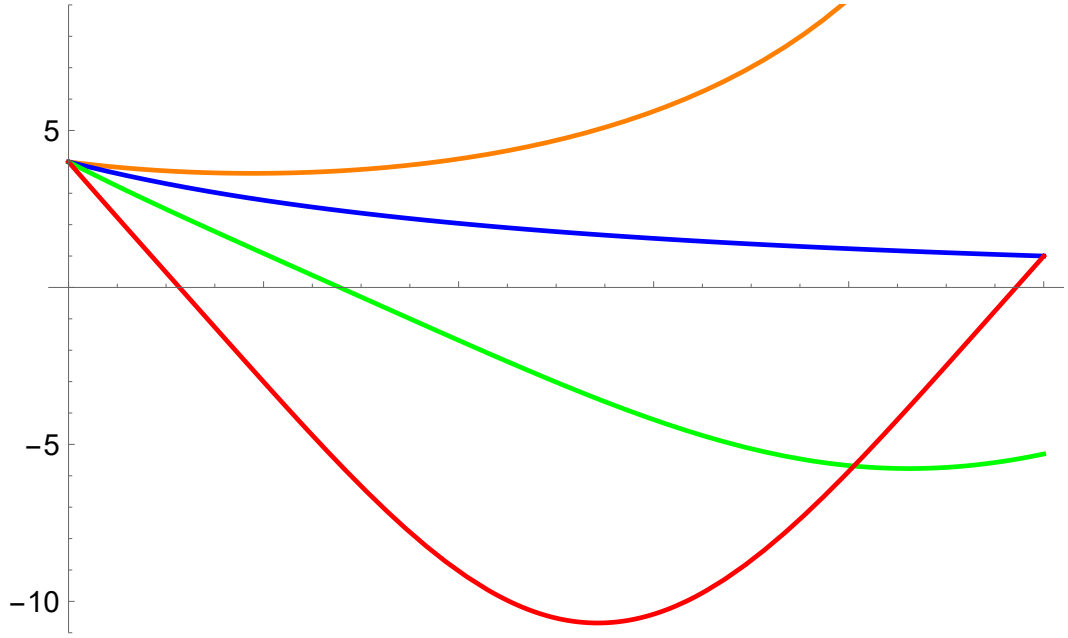
splňuje podmínku $y(1) = 1$ přesně. Lze navíc ukázat (ověřte), že funkce

$$y_{\mu=-8}(x) = 4/(1+x)^2$$

a přestavuje odpovídající řešení okrajové úlohy. Druhé přibližné řešení získáme pro volbu

$$\mu \approx -35.85,$$

jeho předpis však neznáme.



Obrázek 3.10: Přibližná řešení získaná metodou střely pro $\mu \in \{-4, -8, -16, -35.85\}$ (oranžová, modrá, zelená, červená křivka).

Poznámka k příkladu

Zavedením nové funkce $z(x) = y'(x)$ převedeme rovnici druhého řádu v příkladu na soustavu dvou rovnic prvního řádu ve tvaru

$$\begin{aligned} y'(x) &= z(x), \\ z'(x) &= \frac{3\,y^2(x)}{2}, \end{aligned}$$

splňujících počáteční podmínky $y(0) = 4$ a $z(0) = \mu$. K nalezení přibližného řešení na intervalu $x \in [0, 1]$ můžeme využít vhodnou numerickou metodu, například Eulerovu explicitní metodu (rozepište ji).

Příklad 3.5.44 Sestavte matice hmotnosti M a tuhosti K,

$$\begin{aligned} M &= \int_a^b u(x)v(x)\,\mathrm{d}x, \\ K &= \int_a^b u'(x)v'(x)\,\mathrm{d}x. \end{aligned}$$

pro po částech afinní a globálně spojité funkce $u, v : [a, b] \to \mathbb{R}$ splňující okrajové podmínky

$$u(a) = u(b) = 0, \qquad v(a) = v(b) = 0. \tag{3.25}$$

Řešení:
Předpokládejme, že u je po částech afinní a definovaná v $n \in \mathbb{N}$ uzlových bodech ve tvaru

$$x_i = a + ih, \qquad i \in \{1, \ldots, n\}, \qquad h := \frac{b-a}{n+1}$$

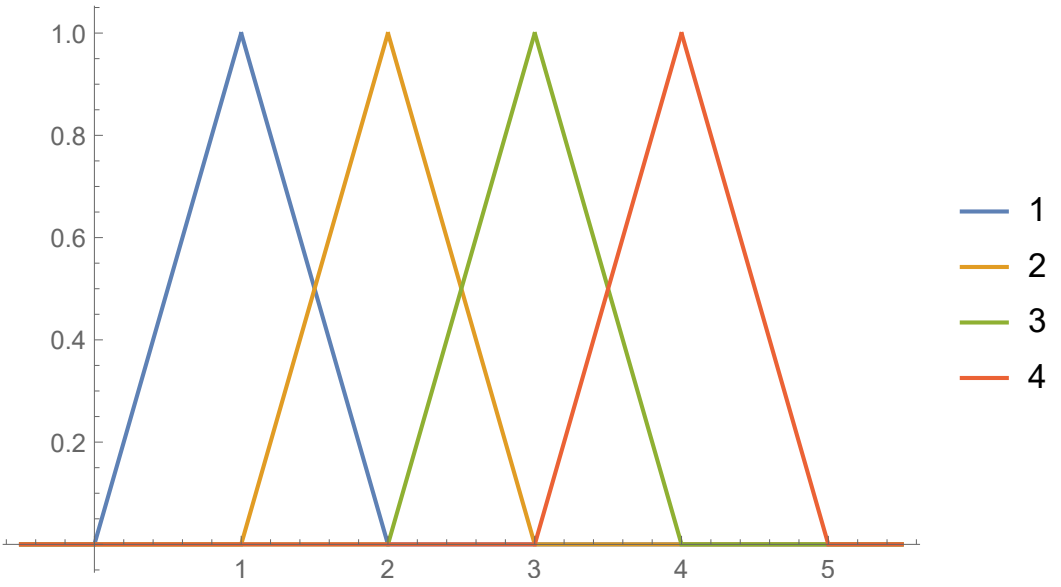
a parametr $h > 0$ zde přestavuje délku diskretizačního kroku. Funkce u, v lze reprezentovat jako lineární kombinace

$$u(x) = \sum_{i=1}^{n} u_i \, \varphi_i(x), \quad v(x) = \sum_{j=1}^{n} v_j \, \varphi_j(x),$$

kde koeficienty $u_i = u(x_i), v_i = u(x_i), i \in \{1, \ldots, n\}$ představují hodnoty funkcí v uzlových bodech. Funkce $\varphi_i, i \in \{1, \ldots, n\}$ jsou známé jako bázové (někdy také tvarové) funkce ve tvaru

$$\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h} & x \in [x_{i-1}, x_i], \\ \frac{x_{i+1} - x}{h} & x \in [x_i, x_{i+1}], \\ 0 & \text{jinde.} \end{cases}$$

Příklad bázových funkcí je uveden Obrázku 3.11. Bázové funkce splňují vlastnost



Obrázek 3.11: 4 bázové funkce pro interval $[a, b] = [0, 5]$.

$$\varphi_i(x_j) = \begin{cases} 1 & \text{pro } i = j \\ 0 & \text{pro } i \neq j. \end{cases}$$

Složky matic hmotnosti $M \in R^{n,n}$ a tuhosti $K \in R^{n,n}$ jsou definovány jako integrály

$$M_{i,j} = \int_a^b \varphi_i(x) \varphi_j(x) \, \mathrm{d}x,$$
$$K_{i,j} = \int_a^b \varphi_i'(x) \varphi_j'(x) \, \mathrm{d}x.$$

a vypočteme je pro případ z Obrázku 3.11 dodatečně pomocí

Kód - Mathematica 3.13: Matice hmotnosti a tuhosti - 1D.

```
a = 0; b = 5; n = 4;
h = (b - a)/(n + 1);
Phi[x_, k_] = Piecewise[{{(x - (a + (k - 1) h))/h,
     a + (k - 1) h <= x < a + k h}, {-((x - (a + (k + 1) h))/h),
     a + k h <= x < a + (k + 1) h}}, 0];
Plot[Evaluate@Table[Phi[x, k], {k, 1, n}], {x, a - h/2, b + h/2},
```

```
 PlotLegends -> Table[i, {i, 1, n}]]
M=Table[Integrate[Phi[x, k]*Phi[x, l], {x, a, b}], {k, n}, {l, n}] //
    MatrixForm
K=Table[Integrate[D[Phi[x, k],x]*D[Phi[x, l],x], {x, a, b}], {k, n}, {l, n
    }] // MatrixForm
```

Získáváme matice

$$M = \begin{pmatrix} \frac{2}{3} & \frac{1}{6} & & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & \\ & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & & \frac{1}{6} & \frac{2}{3} \end{pmatrix}, \qquad K = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{pmatrix}.$$

Pro obecný případ n bázových funkcí mají obě matice velikost $n \times n$ a lze ukázat (ověřte analyticky)

$$M = h \begin{pmatrix} \frac{2}{3} & \frac{1}{6} & & & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & & & \frac{1}{6} & \frac{2}{3} \end{pmatrix}, \qquad K = \frac{1}{h} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{3.26}$$

Prázdné prvky matic jsou automaticky nuly.

Poznámka k příkladu

Obě matice jsou třídiagonální a symetrické. Lze navíc ukázat, že také pozitivně definitní.

Matice tuhosti a hmotnosti lze také obě uvažovat pro funkce u, v, které nesplňují homogenni podmínky (3.25). Potom mají obě matice velikost $(n+2) \times (n+2)$ a vychází

$$M = h \begin{pmatrix} \frac{1}{3} & \frac{1}{6} & & & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & & & \frac{1}{6} & \frac{1}{3} \end{pmatrix}, \qquad K = \frac{1}{h} \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}. \tag{3.27}$$

Modré prvky na krajích obou matic jsou navíc oproti maticím (3.26). Tyto matice jsou opět třídiagonální a symetrické. Lze navíc ukázat, že M je pozitivně definitní, ale K jenom pozitivně semidefinitní (součet všech řádek nebo sloupců představuje nulový vektor, proto je singulární - vysvětlete).

KAPITOLA

4

KDE ČÍST DÁLE?

Seznam uvedený zde by mohl být velmi rozsáhlý, zmiňujeme však jen některé tituly v češtině.

V diskrétní matematice to jsou:

- kniha autorů J. Matouška a J. Nešetřila [2],
- neúplná skripta O. Pangráce (odkaz zde),
- skripta P. Kováře (odkaz zde),
- sbírka příkladů M. Kubesy (odkaz zde),
- sbírka příkladů P. Kováře (odkaz zde).

V numerické matematice to jsou:

- skripta R. Kučery a Z. Morávkové [1] (odkaz zde),
- skripta S. Míky a M. Brandnera [3],
- skripta V. Vondráka a L. Pospíšila (odkaz zde).

SEZNAM POUŽITÉ LITERATURY

[1] KUČERA, Radek a MORÁVKOVÁ, Zuzana. *Numerické metody*. VŠB - Technická univerzita Ostrava, 2016, ISBN 978-80-248-3893-9.

[2] MATOUŠEK, Jiří a NEŠETŘIL, Jaroslav. *Kapitoly z diskrétní matematiky*. Karolinum, 2007.

[3] MÍKA, Stanislav a BRANDNER, Marek. *Numerické metody I*. Plzeň, Západočeská univerzita, 2000, ISBN 80-7082-619-3.

[4] Počítačový kód k výpočtu Newtonova fraktálu, dostupný na: https://www.computervisionblog.com/2009/07/simple-newtons-method-fractal-code-in.html, Copyright (C) 2011 by Tomasz Malisiewicz.

[5] QUARTERONI, Alfio a SACCO, Riccardo a SALERI, Fausto. *Numerical Mathematics*, Springer 2006.