Vysoká škola báňská – Technická univerzita Ostrava
Fakulta stavební

Algoritmizace inženýrských výpočtů

prof. Ing. Martin Krejsa, Ph.D.

Učební materiály

Ostrava, 2021

Učební materiály předmětu **Algoritmizace inženýrských výpočtů**

Studijní program: B3607 Stavební inženýrství

Učební obor: 3607R037 Konstrukce staveb

Klíčová slova:

Algoritmus, MATLAB, matice, vektor, funkce, polynom, algebraická nelineární rovnice, soustava lineárních rovnic, numerická integrace, diferenciální rovnice, iterace, interpolace, aproximace.

Předmluva

Učební materiály předmětu „Algoritmizace inženýrských výpočtů“ jsou určeny pro studenty učebního oboru „Konstrukce staveb“ (3607R037) studijního programu „Stavební inženýrství“ (B3607). Cílem je seznámit studenty se základními programátorskými technikami a tvorbou algoritmů pro řešení inženýrských úloh zaměřených na projekční činnost ve stavebnictví.

Pro aplikaci algoritmických postupů bylo zvoleno prostředí interaktivního výpočetního systému MATLAB, které umožňuje tvořit jednoduché výpočetní nástroje bez složitého programování uživatelského prostředí. MATLAB je uživatelsky příjemný nástroj pro implementaci matematických a numerických postupů, ve kterém lze kromě standardních programátorských nástrojů využít knihovnu, obsahující širokou škálu funkcí. Problémem není ani snadné zobrazení dosažených výsledků formou grafu.

Při volbě aplikačního prostředí bylo rovněž přihlédnuto k návaznosti na další předměty, vyučované na Katedře stavební mechaniky, které jsou rovněž zaměřeny na programování a tvorbu pokročilých výpočetních algoritmů.

V Ostravě, únor 2021 prof. Ing. Martin Krejsa, Ph.D.

Obsah

Kapitola 1

Matlab

Cíle

Kapitola je zaměřena na:

- seznámení s uživatelským prostředím systému **Matlab**,
- definici a správu proměnných,
- úvodní ukázku tvorby algoritmu s využitím **logického rozhodování**.

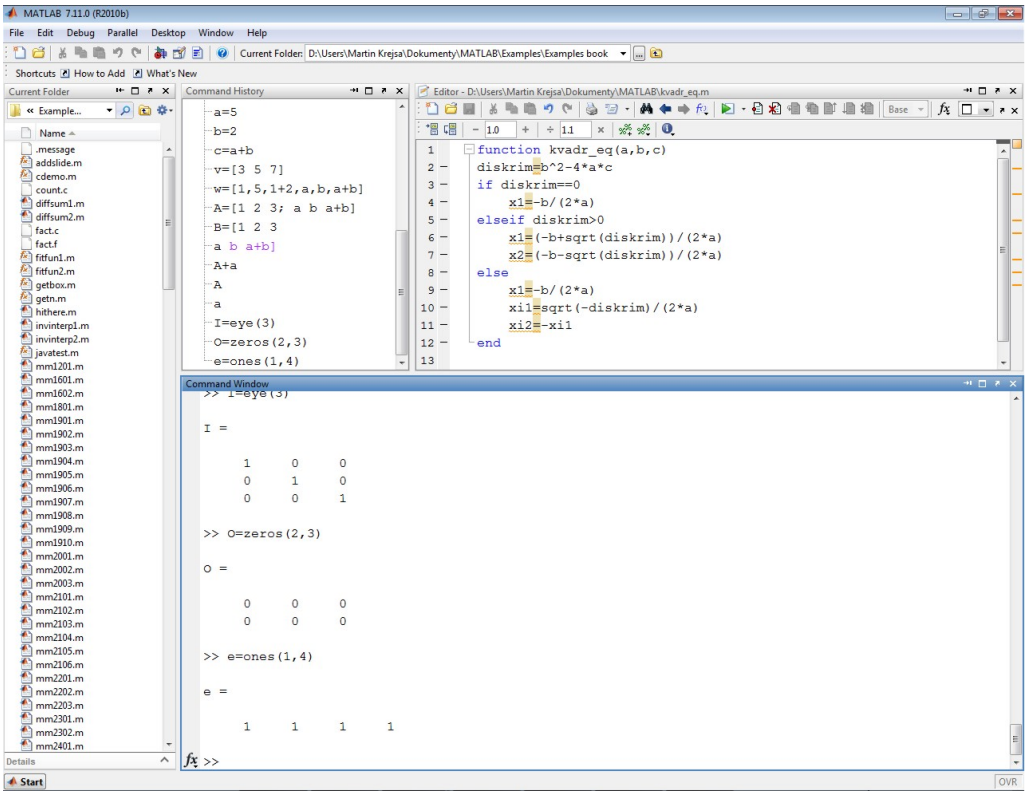
MATLAB [6] (viz obr. 1.1) je programové prostředí využívající skriptovací programovací jazyk pro vědeckotechnické numerické výpočty, modelování a návrhy algoritmů. Nástavbou systému MATLAB je Simulink — program pro simulaci a modelování dynamických systémů, který využívá algoritmy Matlabu pro numerické řešení především nelineárních diferenciálních rovnic.

Název MATLAB vznikl zkrácením slov MATrix LABoratory (volně přeloženo jako „maticová laboratoř“), což vychází ze skutečnosti, že klíčovou datovou strukturou při výpočtech v systému MATLAB jsou matice.

Pro zájemce:

MATLAB je komerční software, takže vítanou alternativou pro tvorbu algoritmů s využitím kompatibilních příkazů a skriptů je software nazývaný Octave, který je zdarma ke stažení např. z [2]. Program Octave vznikl jako open-source a existuje tedy v mnoha mutacích pro platformy Unix (Linux), Mac OS nebo MS Windows. Uživatelsky poněkud strohé prostředí programu obsahuje pouze příkazový řádek v textovém režimu pro zadání konkrétního příkazu a výpis výsledných hodnot.

Samotný systém MATLAB obsahuje řadu příkazů pro správu proměnných, základní operace algebry, výpočty s vektory i maticemi, a příkazy vyšší matematiky. Vzhledem k poněkud rozsáhlým možnostem je k dispozici rozsáhlá nápověda, se kterou lze pracovat s pomocí příkazů obsažených v tab. 1.1.



Obr. 1.1 Pracovní plocha programu MATLAB

Název proměnné	*Popis*
`help`	vyvolání obsahu nápovědy
`help příkaz`	vyvolání nápovědy, vztahující se ke konkrétnímu příkazu
`lookfor výraz`	výpis všech položek nápovědy, které obsahují hledaný výraz
`info`	informace o firmě MathWorks

Tab. 1.1 Přehled příkazů souvisejících s nápovědou

MATLAB si uchovává přesnou reprezentaci veškerých číselných hodnot, se kterými pracuje. Na uživateli však záleží, v jakém formátu jsou tyto hodnoty zobrazovány, k čemuž využívá příkaz `format` s příslušnou specifikací podle tab. 1.2.

1.1 Zadání proměnných

Základním typem proměnné v systému MATLAB je matice. Jednoduchá proměnná, obsahující jednu hodnotu, je interpretována jako matice typu $[1, 1]$. Proměnné se zadávají příkazy, které se definují v příkazovém řádku programu. Končí-li příkaz středníkem, výsledek příkazu se nezobrazí. Pokud je součástí zadání základní arit-

Příkaz	*Popis formátu*	*Zobrazení čísla π v daném formátu*
format short	pevná desetinná čárka, 5 zobrazených číslic	3.1416
format long	pevná desetinná čárka, 15 zobrazených číslic (typ proměnné double), resp. 7 (typ single)	3.141592653589793
format shorte	pohyblivá desetinná čárka, 5 zobrazených číslic	3.1416e+000
format longe	pohyblivá desetinná čárka, 15 zobrazených číslic (typ proměnné double), resp. 7 (typ single)	3.141592653589793e+000
format shorteng	inženýrský formát, 5 zobrazených číslic a exponent	3.1416e+000
format longeng	inženýrský formát, 16 zobrazených číslic a exponent	3.14159265358979e+000
format rat	vyjádření výsledku formou zlomku (implicitní nastavení)	355/113
format hex	zobrazení v šestnáctkové soustavě	400921fb54442d18

Tab. 1.2 Přehled možných typů formátů zobrazení číselných hodnot

metická operace, lze pro ně využít symboly z tab. 1.3. Desetinnou čárku lze zadat pomocí tečky.

Definice proměnné probíhá automaticky přiřazením její hodnoty pomocí rovnítka, např.:

```
a=5
b=2
c=a+b
d='Výsledek'
```

Operace	*Symbol*	*Příklad*
Součet	`+`	`4+11`, `a+b`
Rozdíl	`-`	`18-5`, `a-b`
Součin	`*`	`7.13*5`, `a*b`
Podíl ($\frac{1}{8}$)	`/` nebo `\`	`1/8` = `8\1`
Mocnina (2^8)	`^`	`2^8`

Tab. 1.3 Přehled symbolů základních aritmetických operací

Pro přiřazení hodnoty proměnným lze využít i předem nadefinovaných speciálních proměnných z tab. 1.4

Název proměnné	*Popis*
`ans`	proměnná, která obsahuje výsledek aritmetické operace
`pi`	Ludolfovo číslo π
`inf`	označení pro nekonečno ∞
`nargin`	počet vstupních parametrů dané funkce
`nargout`	počet výstupních parametrů dané funkce
`eps`	nejmenší použitelné číslo v daném formátu
`realmin`	absolutně nejmenší použitelné kladné reálné číslo
`realmax`	absolutně největší použitelné kladné reálné číslo

Tab. 1.4 Přehled předem nadefinovaných proměnných

K dispozici je rovněž množství elementárních funkcí s jedním vstupním parametrem. Přehled nejzákladnějších je uveden v tab. 1.5.

Název funkce	*Popis*
`abs(x)`	absolutní hodnota z x
`cos(x)`,`sin(x)`,`tan(x)`	goniometrické funkce (parametr x v radiánech)
`acos(x)`,`asin(x)`,`atan(x)`	inverzní goniometrické funkce
`exp(x)`	exponenciální funkce (e^x)
`log(x)`	přirozený logaritmus x
`log10(x)`	dekadický logaritmus x
`sqrt(x)`	druhá odmocnina x

Tab. 1.5 Přehled základních elementárních funkcí

1.2 Vektory a matice

K definici vektoru se používají hranaté závorky. Mezery nebo čárky oddělují prvky v řádku, např.:

```
u=[3 5 7]
v=[1,5,1+2,a,b,a+b]
w=[0,pi,2*pi]
```

K vytvoření vektorů lze využít i pokročilejších technik, popsaných v tab. 1.6

Definice matice se provádí podobným způsobem, jak je tomu u vektorů. Řádky se ale oddělují pomocí středníku nebo klávesou `<Enter>`, např:

```
A=[1 2 3; a b a+b]
B=[1 2 3
a b a+b]
```

Příkaz	*Popis*	*Výsledek*
`x=[1:5]`	vytvoří řádkový vektor `x`, začínající hodnotou 1, končící hodnotou 5, přičítá se hodnota 1	`x=[1,2,3,4,5]`
`x=[2:3:11]`	vytvoří řádkový vektor `x`, začínající hodnotou 2, končící hodnotou 11, přičítá se hodnota 3	`x=[2,5,8,11]`
`x=linspace(1,25,5)`	vytvoří řádkový vektor `x`, začínající hodnotou 1, končící hodnotou 25, vektor obsahuje 5 prvků	`x=[1,7,13,19,25]`

Tab. 1.6 Příkazy pro konstrukci vektorů

Poznámka 1.1. MATLAB rozlišuje malá a velká písmena v názvech proměnných. Příkaz

```
A+a
```

vygeneruje pro předchozí zadání matice **A** a proměnné **a**:

```
ans =
     6     7     8
    10     7    12
```

tzn., že ke každému prvku matice **A** se přičte obsah proměnné **a**, tedy hodnota 5.

K přímému generování matic nebo vektoru s předepsaným rozměrem lze využít některé ze standardních funkcí systému MATLAB, např.:

```
I=eye(3)
O=zeros(2,3)
e=ones(1,4)
```

V prvním případě je vygenerována čtvercová matice s hodnotami 1 na diagonále a 0 mimo diagonálu. Následuje vytvoření matice, resp. vektoru s hodnotou 0 resp. 1 ve všech jejích prvcích.

1.2.1 Přístup k maticím a vektorům

Po vytvoření vektoru nebo matice se lze k jednotlivým prvkům odkazovat. V případě předchozí matice A=[1 2 3; a b a+b] (konkrétní obsah je tedy [1 2 3; 5 2 7]) jsou na ukázku možné variace odkazů popsány v tab. 1.7.

Příkaz	*Popis*	*Výsledek*
`A(2,1)`	prvek z 2. řádku a 1. sloupce matice `A`	5
`A(2,[1 3])`	1. a 3. prvek z 2. řádku matice `A`	$5, 7$
`A(1,1:3)`	prvky 1 až 3 na 1. řádku matice `A`	$1, 2, 3$
`A(1,1:end)`	stejné jako předchozí	$1, 2, 3$
`A(1,:)`	stejné jako předchozí	$1, 2, 3$
`A(2,1:2:3)`	1. a 3. prvek z 2. řádku matice `A`, `a:b:c` definuje aritmetickou posloupnost s prvním prvkem rovným `a`, posledním rovným `c`, `b` je diference mezi sousedními prvky (pokud je rovno 1, nemusí se uvádět)	$5, 7$
`A(1:end,2:end)`	2. a 3. prvek z obou řádků (submatice)	$2, 3; 2, 7$

Tab. 1.7 Příkazy pro přístup k prvkům vektorů a matic

1.2.2 Maticové operace

S vektory i maticemi lze provádět maticové operace. Transponování matic se provádí s využitím operátoru ' (apostrof), např. příkaz `A'` transponuje původní matici `A`:

```
ans =
   1   5
   2   2
   3   7
```

K provedení operací mezi jednotlivými prvky matice nebo vektoru se musí umístit znak `.` (tečka) před operátor. Například pro dříve zadaný vektor `u=[3 5 7]` je výsledkem příkazu `u*u'`:

```
ans =  83
```

kdežto po zadání příkazu `u.*u` se zobrazí vektor:

```
ans =
    9   25   49
```

Podobně lze provádět i další operace s vektory a maticemi prvek po prvku, jak je pro obecně definované vektory $a = a_1, a_2, \ldots, a_n$; $b = b_1, b_2, \ldots, b_n$ a skalár c uvedeno v tab. 1.8.

K dispozici je také několik maticových funkcí. Výčet nejzákladnějších obsahuje tab. 1.9.

1.3 Správa proměnných

Pro správu zadaných proměnných, vektorů a matic je v prostředí systému MATLAB zavedeno několik příkazů:

Operace	*Příkaz*	*Výsledek*
skalární součet	`a+c`	$a_1 + c, a_2 + c, \ldots, a_n + c$
skalární součin	`a*c=a.*c`	$a_1 \cdot c, a_2 \cdot c, \ldots, a_n \cdot c$
vektorový součet	`a+b`	$a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n$
vektorový součin	`a.*b`	$a_1 \cdot b_1, a_2 \cdot b_2, \ldots, a_n \cdot b_n$
vektorové dělení zleva	`a./b`	$a_1/b_1, a_2/b_2, \ldots, a_n/b_n$
vektorové dělení zprava	`a.\b`	$a_1\backslash b_1, a_2\backslash b_2, \ldots, a_n\backslash b_n$
skalární mocnění	`a.^c`	$a_1^c, a_2^c, \ldots, a_n^c$
skalární mocnění	`c.^a`	$c^{a_1}, c^{a_2}, \ldots, c^{a_n}$
vektorové mocnění	`a.^b`	$a_1^{b_1}, a_2^{b_2}, \ldots, a_n^{b_n}$

Tab. 1.8 Operace s vektory a maticemi prvek po prvku

Název funkce	*Popis*
`det(A)`	determinant matice `A`
`inv(A)`	výpočet inverzní matice k `A`
`diag(A)`	výpis prvků na diagonále matice `A`

Tab. 1.9 Přehled základních maticových funkcí

- příkazy `who` nebo `whos` vypíší seznam všech aktuálně definovaných proměnných (druhý z nich i s jejich velikostmi),
- příkaz `size(proměnná)` vrátí rozměry dané proměnné,
- příkaz `clear(proměnná)` vymaže danou proměnnou z paměti,
- příkaz `clear` bez parametru vymaže všechny zadané proměnné z paměti.

1.4 Využití grafického výstupu

Základním nástrojem v grafickém režimu je příkaz `plot`. Syntaxe tohoto příkazu je `plot(x,y,options)`, kde `x` a `y` jsou souřadnice bodů, které se mají vykreslit. Parametr `options` definuje způsob, jakým se bude grafický výstup provádět. Obsahuje znaky pro definici barvy a stylu vykreslení bodů, případně jejich spojnice:

- barva bodů a jejich spojnice: **b** (modrá), **g** (zelená), **r** (červená), **c** (modrozelená), **m** (fialová), **y** (žlutá), **k** (černá), **w** (bílá);
- styl spojnice bodů: `-` (body budou spojeny plnou čarou), `:` (body budou spojeny tečkovanou čarou), `-.` (body budou spojeny čerchovanou čarou), `--` (body budou spojeny přerušovanou čarou);

- styl vykreslení bodů: `*` (body budou vykresleny jako hvězdičky), `.` (tečky), `x` (křížky), `+` (znaky plus), `o` (kolečka), `s` (čtverce), `d` (kosočtverce).

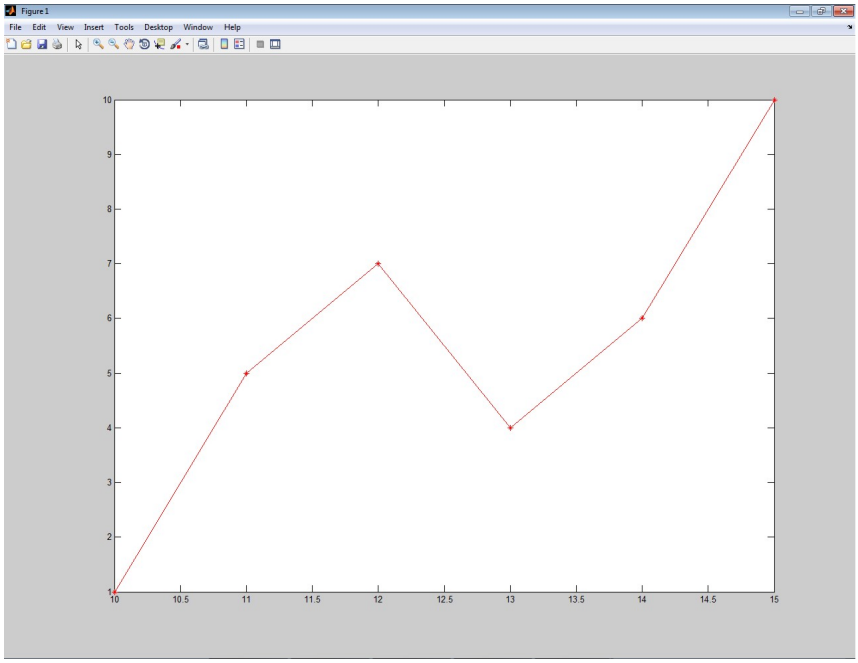
Příklad 1.2. Pro vykreslení spojnice pěti bodů o souřadnicích:

x	10	11	12	13	14	15
y	1	5	7	4	6	10

poslouží příkaz:

```
plot([10:15],[1,5,7,4,6,10],'r-*')
```

Výsledný grafický výstup je zobrazen na obr .1.2.



Obr. 1.2 Ukázka grafického výstupu z programu MATLAB

V souvislosti s grafickým prostředím je užitečný i příkaz `hold on`, který umožňuje grafický výstup více příkazů do jednoho okna (do původního stavu pak vše vrátí příkaz `hold off`).

1.4.1 Graf funkce

Při vytváření grafu funkce je nutné nejprve diskretizovat osu x. V získaných bodech je pak nutno určit odpovídající funkční hodnoty $f(x)$. Graf diskretizované funkce se pak zobrazí již známým příkazem `plot`.

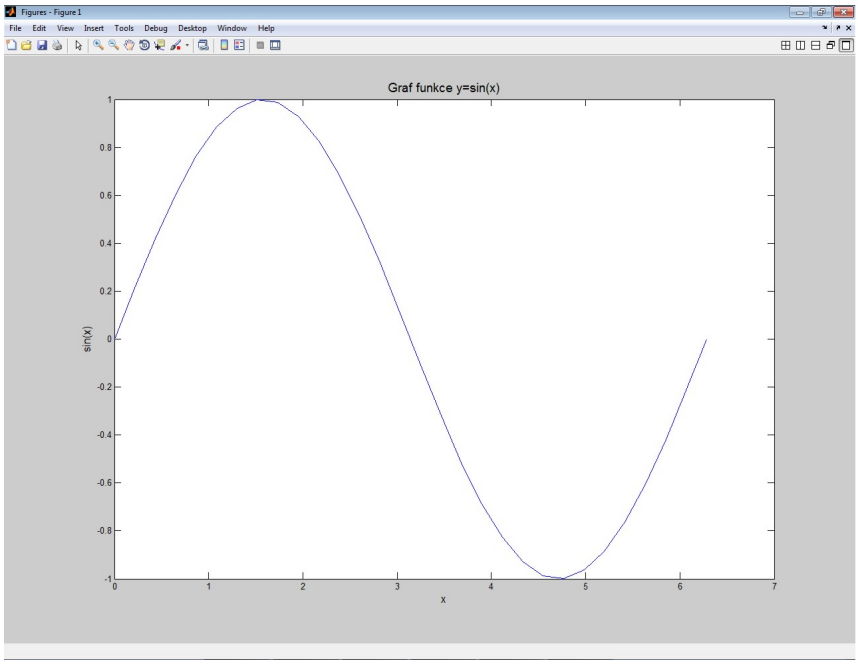
Příklad 1.3. Vykreslení grafu funkce sinus lze s využitím příkazu `plot` provést následující sekvencí příkazů:

```
x=linspace(0,2*pi,30);
y=sin(x);
plot(x,y,'b-')
```

Graf lze doplnit o nadpis, případně popisem os, následujícím doplněním:

```
title('Graf funkce y=sin(x)');
xlabel('x');
ylabel('sin(x)')
```

Výsledné zobrazení grafu funkce sinus je na obr. 1.3.



Obr. 1.3 Graf funkce sinus

1.5 Vytvoření skriptů

V programu MATLAB lze posloupnost příkazů zadávat jednoduše do příkazového řádku, systém interaktivně tyto povely postupně zpracovává. Pokud by se měl ale výpočet provádět opakovaně, musí se veškeré příkazy zadávat znovu, což je pracné a nevýhodné.

Řešením je uložení sledu matlabovských příkazů, tzv. `skriptů` do textového souboru s příponou `*.m` (tzv. m-soubor). S využitím příkazů, uložených v m-souboru, se výpočet spustí zadáním názvu souboru (bez přípony), přičemž se prohledává aktuální adresář a adresáře uvedené v seznamu systémové proměnné `path`.

Tímto způsobem lze vytvořit v samostatném souboru i speciální výpočetní funkci (tzv. „m-funkci“), kterou lze vyvolat z příkazového řádku zadáním názvu funkce, případně seznamem vstupních parametrů v závorce, oddělených čárkami. **Název souboru by měl být shodný s názvem funkce v jeho záhlaví.**

M-soubory mohou obsahovat i tzv. řídící příkazy, typické pro pokročilejší programovací platformy. Patří k nim zejména příkazy cyklu a logického rozhodování.

1.5.1 Příkazy cyklu

MATLAB umožnuje použití dvou typů programových cyklů:

- **for cyklus**, jehož syntaxe je

```
for i=počáteční hodnota:krok:koncová hodnota
    posloupnost příkazů
end
```

- **while cyklus** se syntaxí:

```
while logická podmínka
    posloupnost příkazů
end
```

Podrobnější výklad tvorby algoritmů s využitím obou typů cyklů je obsažen v následujících kapitolách.

1.5.2 Logické rozhodování

Syntaxe posloupnosti příkazů, které mají být rozděleny do tří bloků podle výsledků logického rozhodování, je následující:

```
if logická podmínka 1
    posloupnost příkazů 1
elseif logická podmínka 2
    posloupnost příkazů 2
else
    posloupnost příkazů 3
end
```

Při definování logických podmínek, jejichž výsledkem je buď „pravda“ nebo „nepravda“, je možno využít následující logické operátory: & (`and`), | (`or`), ~ (`not`). Relačními operátory mohou být: <, <=, >=, > a == (rovnost).

Příklad 1.4. K procvičení tvorby m-funkce poslouží řešení kvadratické rovnice, kterou lze s využitím logických operátorů vytvořit následujícím způsobem:

```
function kvadr_eq(a,b,c)
diskrim=b^2-4*a*c
if diskrim==0
    x1=-b/(2*a)
elseif diskrim>0
    x1=(-b+sqrt(diskrim))/(2*a)
    x2=(-b-sqrt(diskrim))/(2*a)
else
    x1=-b/(2*a)
    xi1=sqrt(-diskrim)/(2*a)
    xi2=-xi1
end
```

Funkce, kterou lze v příkazovém řádku programu MATLAB vyvolat se vstupními parametry zadáním např. `kvadr_eq(5,10,1)`, pak vrátí dva reálné kořeny kvadratické rovnice:

```
diskrim =
    80

x1 =
  -0.105572809000084

x2 =
  -1.894427190999916
```

Při vyvolání vytvořené funkce s parametry `kvadr_eq(10,5,1)` lze naopak získat výsledek:

```
diskrim =
   -15

x1 =
  -0.250000000000000

xi1 =
   0.193649167310371

xi2 =
  -0.193649167310371
```

Dosažené výsledky lze zkontrolovat speciální funkcí systému MATLAB s názvem `roots(c)`, kde `c` je vektor koeficientů polynomu seřazených sestupně podle mocnin x. Při obecném vyjádření polynomu n-tého stupně:

$$f(x) = c_n \cdot x^n + c_{n-1} \cdot x^{n-1} + \cdots + c_1 \cdot x + c_0 \,, \tag{1.1}$$

obsahuje vektor `c` prvky $c_n, c_{n-1}, \ldots, c_1, c_0$.

Pro první výpočet příkladu 1.4 je pak sled příkazů následující:

```
c=[5,10,1]
roots(c)
```

s výsledkem

```
ans =
  -1.894427190999916
  -0.105572809000084
```

Poznámka 1.5. Pro popis práce s programovým systémem MATLAB, resp. Octave, existuje množství publikací. Některé z nich jsou přeloženy do češtiny a v elektronické podobě jsou volně přístupné (např. [3, 13]).

Kapitola 2

Základy algoritmizace

Cíle

Kapitola by měla sloužit:

- k vysvětlení pojmu algoritmus,
- k položení základů pro tvorbu jednoduchých algoritmů.

Algoritmus je přesný návod či postup, kterým lze vyřešit daný typ úlohy. Pojem algoritmu se nejčastěji objevuje při programování, kdy se jím myslí teoretický princip řešení problému (oproti přesnému zápisu v konkrétním programovacím jazyce). Obecně se ale algoritmus může objevit v jakémkoli jiném vědeckém odvětví. Jako jistý druh algoritmu se může chápat i např. kuchyňský recept. V užším smyslu se slovem algoritmus rozumí pouze takové postupy, které splňují požadavky, označované vlastnostmi algoritmu.

2.1 Vlastnosti algoritmu

Následující výčet vlastností algoritmu je uveden např. v [14]:

- **Konečnost** (finitnost)

  Každý algoritmus musí skončit v konečném počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný. Postupy, které tuto podmínku nesplňují, se mohou nazývat výpočetní metody. Speciálním příkladem nekonečné výpočetní metody je reaktivní proces, který průběžně reaguje s okolním prostředím. Někteří autoři však mezi algoritmy zahrnují i takovéto postupy.

- **Obecnost** (hromadnost, masovost, univerzálnost)

  Algoritmus neřeší jeden konkrétní problém (např. „jak spočítat $3 \cdot 7$“), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel“), takže má širokou množinu možných vstupů.

- **Determinovanost**

  Každý krok algoritmu musí být jednoznačně a přesně definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat, takže pro stejné vstupy lze pokaždé získat stejné výsledky. Protože běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly pro zápis algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný význam. Vyjádření výpočetní metody v programovacím jazyce se nazývá *program*. Některé algoritmy ale determinované nejsou, např. pravděpodobnostní algoritmy s (pseudo)náhodným výběrem.

- **Výstup** (resultativnost)

  Algoritmus má alespoň jeden výstup, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší (algoritmus vede od zpracování hodnot k výstupu).

- **Elementárnost**

  Algoritmus se skládá z konečného počtu jednoduchých (elementárních) kroků.

V praxi jsou proto předmětem zájmu hlavně takové algoritmy, které jsou v nějakém smyslu kvalitní. Takové algoritmy splňují různá kritéria, měřená např. počtem kroků potřebných pro běh algoritmu, jednoduchost, efektivitu či eleganci. Problematikou efektivity algoritmů, tzn. metodami, jak z několika známých algoritmů řešících konkrétní problém vybrat ten nejlepší, se zabývají odvětví informatiky nazývané *algoritmická analýza* a *teorie složitosti*.

2.2 Elementární algoritmy

Následující výklad je zaměřen na několik elementárních algoritmů, které jsou základem mnoha výpočetních technik a postupů.

2.2.1 Záměna obsahu dvou proměnných

Za nejjednodušší algoritmus lze považovat postup popisující záměnu obsahu dvou proměnných. K této operaci je potřeba třetí, pomocné proměnné. Samotný algoritmus lze pro proměnné a a b i pro pomocnou proměnnou c popsat následovně:

```
c=a;
a=b;
b=c;
```

Byť se jedná o skutečně elementární algoritmus, lze jej společně s logickým rozhodováním (kap. 1.5.2) využít např. pro vzestupné setřídění vektoru d o 3 prvcích (c je opět pomocná proměnná):

```
function setrid(d)
  if length(d)==3
    if d(1)>d(2)
      c=d(1);
      d(1)=d(2);
      d(2)=c;
    end
    if d(2)>d(3)
      c=d(2);
      d(2)=d(3);
      d(3)=c;
    end
    if d(1)>d(2)
      c=d(1);
      d(1)=d(2);
      d(2)=c;
    end
    d
  end
end
```

Příklad 2.1. Proveďte vzestupné třídění vektoru `[8 24 2]` s využitím funkce pro třídění vektoru o třech prvcích, vytvořené v předchozím oddíle.

Řešení. Nejprve je potřeba přiřadit vstupní hodnoty vektoru d:

```
d=[8 24 2]
```

Pak je možno vyvolat funkci `setrid`:

```
setrid(d)
```

Výsledkem je:

```
d =
     2     8    24
```

▲

Tento způsob třídění je samozřejmě velmi neefiktivní a v žádném případě jej nelze považovat za univerzální. Je však vhodný pro vysvětlení základů algoritmizace. Kvalitním algoritmům, které jsou zaměřeny na třídění vektorů, bude věnována jedna z následujících kapitol.

Kapitola 3

Výpočet hodnot funkcí

Cíle

Kapitola má za cíl demonstrovat:

- základní přístupy k tvorbě algoritmů pro výpočet hodnot funkcí,
- odlišnosti mezi algoritmy z hlediska jejich efektivity,
- využití cyklů **for** při tvorbě algoritmů,
- provádění tzv. „tabelace funkce“.

3.1 Výpočet hodnoty polynomu

Následující ukázka jednoho z nejzákladnějších algoritmů pro výpočet polynomu byla převzata z [12]. Spočívá v nalezení nejlepšího způsobu určení hodnoty polynomu:

$$f(x) = 2 \cdot x^4 + 3 \cdot x^3 - 3 \cdot x^2 + 5 \cdot x - 1 \tag{3.1}$$

pro konkrétně zadanou hodnotu x, např. $x = \frac{1}{2}$. Nejlepším způsobem výpočtu je chápán algoritmus s nejmenším počtem matematických operací.

Metoda 1:

První způsob vede k přímému určení požadované hodnoty:

$$f(x = \frac{1}{2}) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} + 3 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} - 3 \cdot \frac{1}{2} \cdot \frac{1}{2} + 5 \cdot \frac{1}{2} - 1 = \frac{5}{4}\,. \tag{3.2}$$

Při tomto výpočtu bylo provedeno 10 operací násobení a 4 pro součet/rozdíl.

Metoda 2:

Poněkud výhodnější řešení představuje postup, při němž jsou postupně určeny mocniny vstupního parametru x:

$$\frac{1}{2} \cdot \frac{1}{2} = \left(\frac{1}{2}\right)^2 \tag{3.3}$$

$$\left(\frac{1}{2}\right)^2 \cdot \frac{1}{2} = \left(\frac{1}{2}\right)^3 \tag{3.4}$$

$$\left(\frac{1}{2}\right)^3 \cdot \frac{1}{2} = \left(\frac{1}{2}\right)^4 \tag{3.5}$$

Nyní lze provést finální výpočet polynomu:

$$f(x = \frac{1}{2}) = 2 \cdot \left(\frac{1}{2}\right)^4 + 3 \cdot \left(\frac{1}{2}\right)^3 - 3 \cdot \left(\frac{1}{2}\right)^2 + 5 \cdot \left(\frac{1}{2}\right) - 1 = \frac{5}{4} \,. \tag{3.6}$$

Tento algoritmus je poněkud efektivnější než v prvním případě, neboť bylo provedeno celkem 7 operací násobení a stejný počet 4 operací pro součet/rozdíl.

Metoda 3:

Poslední způsob výpočtu, označovaný jako *Hornerova metoda*, předpokládá následující úpravu řešeného polynomu (3.1):

$$\begin{aligned} f(x) &= -1 + x \cdot (5 - 3 \cdot x + 3 \cdot x^2 + 2 \cdot x^3) = \\ &= -1 + x \cdot (5 + x \cdot (-3 + 3 \cdot x + 2 \cdot x^2)) = \\ &= -1 + x \cdot (5 + x \cdot (-3 + x \cdot (3 + 2 \cdot x))) \,. \end{aligned} \tag{3.7}$$

Sled výpočetních operací vypadá pro $x = \frac{1}{2}$ následovně (výpočet probíhá zevnitř směrem ven):

$$\frac{1}{2} \cdot 2 + 3 = 4 \tag{3.8}$$

$$\frac{1}{2} \cdot 4 - 3 = -1 \tag{3.9}$$

$$\frac{1}{2} \cdot (-1) + 5 = \frac{9}{2} \tag{3.10}$$

$$\frac{1}{2} \cdot \frac{9}{2} - 1 = \frac{5}{4} \,. \tag{3.11}$$

Celkem je pro určení požadované hodnoty polynomu potřeba pouze čtyř operací násobení i součtu/rozdílu. Jedná se tedy o nejvíce efektivní algoritmus, který lze pro obecně vyjádřený polynom (1.1) schématicky vyjádřit algoritmem 1.

Vstup : n, $\mathbf{c} = \{c_1, c_2, c_3, \ldots, c_n, c_{n+1}\}$, x
Výstup: $f(x)$

$y \leftarrow c_{n+1}$
for $i \leftarrow n, n-1, \ldots, 2, 1$ **do**
| $y \leftarrow y \cdot x + c_i$
end
$f(x) \leftarrow y$

Algoritmus 1: Hornerova metoda

Algoritmus 1 lze zapsat prostřednictvím m-souboru i v programu MATLAB:

```
function y=horner(d,c,x)
y=c(d+1);
for i=d:-1:1
  y=y*x+c(i);
end
```

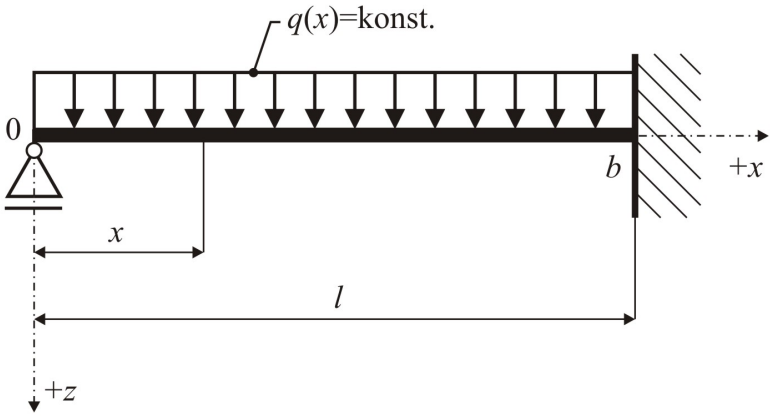
kde parametr d je stupeň zadávaného polynomu, c je vektor s $d+1$ konstantními koeficienty polynomu a x je hodnota, pro níž se polynom určuje. Funkce se pro polynom (3.1) a konkrétně zadanou hodnotu $x = \frac{1}{2}$ bude vyvolávat příkazem:

```
horner(4,[-1 5 -3 3 2],1/2)
```

Výstup z programu MATLAB pak bude:

```
ans =
    1.2500
```

Příklad 3.1. S využitím předchozího postupu určete průhyb v polovině rozpětí staticky neurčitého nosníku, jehož statické schéma je zobrazeno na obr. 3.1. Pro stanovení rovnic polynomů ohybové čáry a pootočení využijte metodu přímé integrace diferenciální rovnice 4. řádu $EI_y w_z(x)'''' = q_z(x)$, kde E je modul pružnosti v tahu a tlaku [MPa], I_y příslušný moment setrvačnosti [m^4] a q_z spojité silové zatížení, působící ve svislém směru [kN/m]. Konkrétní vstupní údaje jsou uvedeny v tabulce 3.1.



Obr. 3.1 Statické schéma řešeného staticky neurčitého nosníku

Spojité silové zatížení q_z :	4 kN/m
Rozpětí nosníku l :	6 m
Šířka obdélníkového průřezu b :	$0,02$ m
Výška obdélníkového průřezu h :	$0,15$ m
Moment setrvačnosti I_y :	$\frac{1}{12} \cdot 0,02 \cdot 0,15^3 = 5,625 \cdot 10^{-6}$ m^4
Modul pružnosti v tahu a tlaku E :	$2,1 \cdot 10^{11}$ Pa

Tab. 3.1 Vstupní údaje příkladu 3.1

Řešení. V diferenciální rovnici 4. řádu

$$EI_y w_z(x)'''' = q_z(x) \tag{3.12}$$

bude na pravé straně člen odpovídající rovnoměrnému spojitému zatížení, tedy $q(x) = q = \text{konst}$.

Vztah (3.12) pak lze postupně integrovat:

$$EI_y w_z(x)'''' = q_z \ , \tag{3.13}$$

$$EI_y w_z(x)''' = -V_z(x) = q_z \cdot x + C_1 \ , \tag{3.14}$$

$$EI_y w_z(x)'' = -M_y(x) = q_z \cdot \frac{x^2}{2} + C_1 \cdot x + C_2 \ , \tag{3.15}$$

$$EI_y w_z(x)' = q_z \cdot \frac{x^3}{6} + C_1 \cdot \frac{x^2}{2} + C_2 \cdot x + C_3 \ , \tag{3.16}$$

$$EI_y w_z(x) = q_z \cdot \frac{x^4}{24} + C_1 \cdot \frac{x^3}{6} + C_2 \cdot \frac{x^2}{2} + C_3 \cdot x + C_4 \ . \tag{3.17}$$

Integrační konstanty $C_1, \ldots, C_4$ se určí z okrajových podmínek:

a) Pro $x = 0$ je $M_y(x) = 0$ a platí tedy:

$$M_y(x = 0) = -q \cdot \frac{0^2}{2} - C_1 \cdot 0 - C_2 = 0 \ , \tag{3.18}$$

z čehož vyplývá, že $C_2 = 0$,

b) Pro $x = 0$ je $w_z(x) = 0$ a platí tedy:

$$w_z(x = 0) = \frac{1}{EI_y} \cdot \left(q_z \cdot \frac{0^4}{24} + C_1 \cdot \frac{0^3}{6} + 0 \cdot \frac{0^2}{2} + C_3 \cdot 0 + C_4 \right) = 0 \ , \tag{3.19}$$

z čehož plyne, že $C_4 = 0$,

c) Pro $x = l$ je $w_z'(x) = 0$ a platí tedy:

$$w_z'(x = l) = \frac{1}{EI_y} \cdot \left(q_z \cdot \frac{l^3}{6} + C_1 \cdot \frac{l^2}{2} + 0 \cdot l + C_3 \right) = 0 \ , \tag{3.20}$$

d) Pro $x = l$ je $w_z(x) = 0$ a platí tedy:

$$w_z(x = l) = \frac{1}{EI_y} \cdot \left(q_z \cdot \frac{l^4}{24} + C_1 \cdot \frac{l^3}{6} + 0 \cdot \frac{l^2}{2} + C_3 \cdot l + 0 \right) = 0 \ . \tag{3.21}$$

Poslední dvě rovnice představují soustavu dvou lineárních rovnic o dvou neznámých integračních konstantách C_1 a C_2. Řešením soustavy lze získat:

$$C_1 = -\frac{3}{8} q_z l \ , \tag{3.22}$$

a

$$C_3 = \frac{1}{48} q_z l^3 \ . \tag{3.23}$$

Dosazením výsledných hodnot integračních konstant $C_1, \ldots, C_4$ do vztahů 3.14 až 3.17 je možno získat výsledné rovnice pro dvojici statických veličin (posouvající sílu V_z a ohybový moment M_y) i pro obě deformační veličiny - průhyb w_z a pootočení $w_z^{'} = \varphi_y$:

$$V_z(x) = -\left(q_z x - \frac{3}{8} q_z l\right) = \frac{3}{8} q_z l - q_z x \ , \tag{3.24}$$

$$M_y(x) = -\left(q_z \frac{x^2}{2} - \frac{3}{8} q_z l x + 0\right) = \frac{3}{8} q_z l x - q_z \frac{x^2}{2} \ , \tag{3.25}$$

$$\begin{aligned} w_z(x)^{'} = \varphi_y(x) &= \frac{1}{EI_y} \cdot \left(q_z \frac{x^3}{6} - \frac{3}{8} q_z l \frac{x^2}{2} + 0 \cdot x + \frac{1}{48} q_z l^3\right) = \\ &= \frac{1}{EI_y} \cdot \left(q_z \frac{x^3}{6} - \frac{3}{16} q_z l x^2 + \frac{1}{48} q_z l^3\right) \ , \end{aligned} \tag{3.26}$$

$$\begin{aligned} w_z(x) &= \frac{1}{EI_y} \cdot \left(q_z \frac{x^4}{24} - \frac{3}{8} q_z l \frac{x^3}{6} + 0 \cdot \frac{x^2}{2} + \frac{1}{48} q_z l^3 x + 0\right) = \\ &= \frac{1}{2EI_y} \cdot \left(q_z \frac{x^4}{12} - \frac{3}{24} q_z l x^3 + \frac{1}{24} q_z l^3 x\right) \ . \end{aligned} \tag{3.27}$$

Při využití programu MATLAB i vytvořené m-funkce `horner` pro stanovení požadovaného průhybu bude sled příkazů následující:

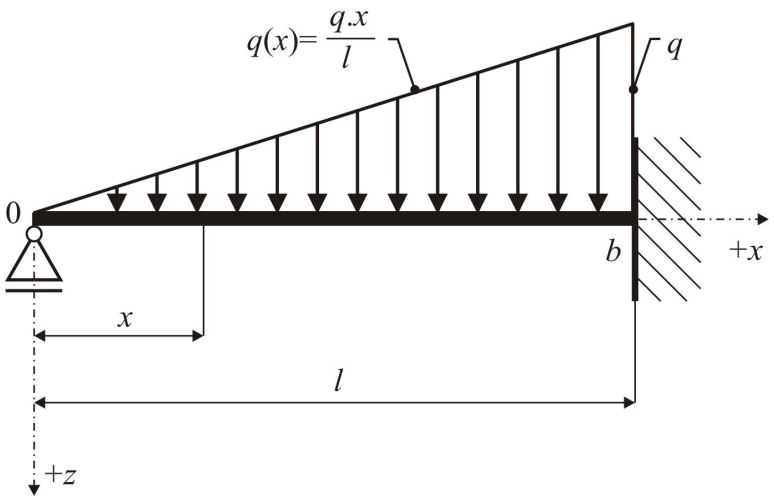
```
qz=4000;
l=6;
b=0.02;
h=0.15;
Iy=1/12*b*h^3;
E=2.1*10^11;
c=[0 1/(48*E*Iy)*qz*l^3 0 -3/(48*E*Iy)*qz*l qz/(24*E*Iy)];
horner(4,c,l/2)*1000
```

Výsledný průhyb v milimetrech pak v polovině rozpětí vychází:

```
ans =
  22.857142857142865
```

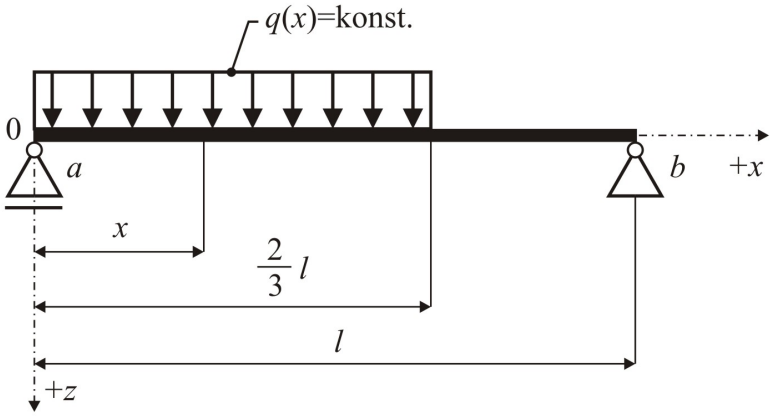
▲

Příklad 3.2. Výše vysvětleným postupem určete průhyb v polovině rozpětí staticky neurčitého nosníku, který je schématicky zobrazen na obr. 3.2.



Obr. 3.2 Statické schéma řešeného staticky neurčitého nosníku

Příklad 3.3. Stanovte průhyb v polovině rozpětí staticky určitého nosníku, jehož schéma je zobrazeno na obr. 3.3. Pro stanovení rovnice ohybové čáry ve zkoumaném intervalu $< 0; \frac{2}{3}\, l >$ využijte Clebschovu metodu.



Obr. 3.3 Statické schéma řešeného staticky určitého nosníku

3.1.1 Tabelace řešené funkce

Výpis hodnot funkce s parametrem x lze nejlépe provést s využitím cyklu `for` a funkcemi pro výpis na obrazovku v předepsaném formátu `disp` a `sprintf`.

Poznámka 3.4. Funkce `disp(x)` zobrazí obsah proměnné x typu `text`.

Poznámka 3.5. Funkce `sprintf` převede data do textového řetězce v požadovaném formátu pomocí tzv. „konverzních specifikátorů“, které začínají znakem %. Běžnou konverzi je možno provést specifikátorem `%f` pro převod numerické hodnoty do tvaru s pevnou desetinnou čárkou, specifikátorem `%e` pro vyjádření číselné hodnoty v exponenciálním tvaru, příp. specifikátorem `%g` pro automatickou volbu. Mezi znak `%` a specifikátor `f`, `e` nebo `g` lze navíc vložit počet znaků požadované šířky formátu, případně tečku a počet požadovaných znaků za desetinnou čárkou. Parametr `\n` zajistí přechod na nový řádek.

Příklad 3.6. Proveďte tabelizaci funkce ohybové čáry nosníku z příkladu 3.1. Výsledné průhyby určete v průřezech s roztečí 10 cm.

Řešení. Sled příkazů pro výpis výsledných průhybů ve sledovaných bodech x by mohl vypadat následovně:

```
format long
disp('  x [mm]    w(x) [mm]')
disp('______________________')
for x=0:.1:l
disp(sprintf('%8.1f %12.4f',x*1000,horner(4,c,x)*1000))
end
```

Výpis pak bude mít následující vzhled:

```
  x [mm]    w(x) [mm]
----------------------
     0.0       0.0000
   100.0       1.5226
   200.0       3.0377
   300.0       4.5383
   400.0       6.0176
     ...         ...
  5700.0       0.6297
  5800.0       0.2881
  5900.0       0.0741
  6000.0       0.0000
```

▲

Příklad 3.7. Výpis z předchozího příkladu doplňte i o hodnotu posouvající síly V_z, ohybového momentu M_y a pootočení φ_y.

3.1.2 Vykreslení grafu řešené funkce

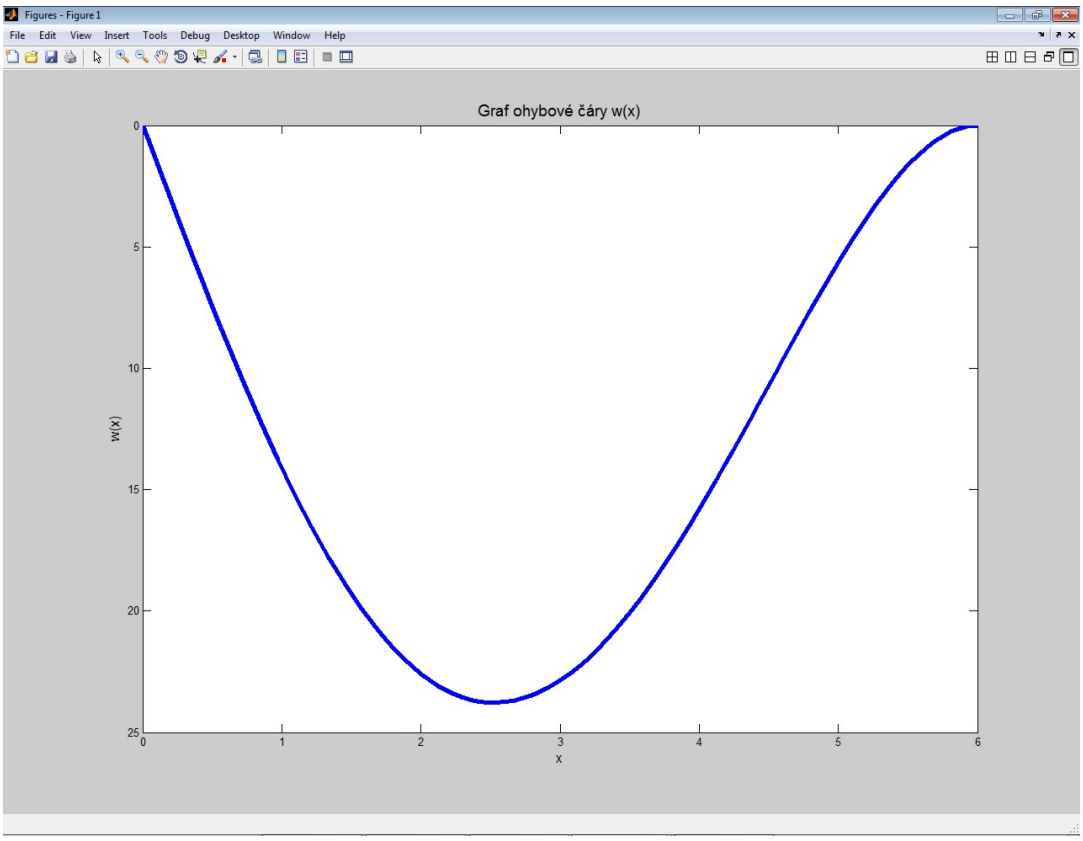
Graf řešené funkce lze vykreslit s využitím postupu, popsaném v kap. 1.4.1.

Příklad 3.8. Vykreslete graf ohybové čáry nosníku z příkladu 3.1.

Řešení. Sled příkazů, s využitím cyklu **for**, odvozeného vektoru c z příkladu 3.1 a m-funkce `horner` může být následující:

```
x=linspace(0,l,100);
for i=1:100, y(i)=horner(4,c,x(i))*1000; end
plot(x,y,'b-');
title('Graf ohybové čáry w(x)');
xlabel('x');
ylabel('w(x)');
```

Výsledný graf ohybové čáry je pak zobrazen na obr. 3.4.



Obr. 3.4 Ohybová čára staticky neurčitého nosníku z příkladu 3.1

▲

Příklady k procvičení

1. Obdobně sestrojte graf ohybové čáry staticky neurčitém nosníku, který je schématicky zobrazen na obr. 3.2.

2. Vykreslete graf ohybové čáry staticky určitého nosníku, jehož schéma je zobrazeno na obr. 3.3.

3.1.3 Určení extrému diskretizované funkce

Zjednodušený výpočet největší hodnoty funkce v předem definovaném intervalu lze provést ve třech krocích nejprve diskretizací osy x, stanovením hodnot funkce pro všechna x v požadovaném rozsahu a algoritmem 2 pro stanovení největšího čísla ve vektoru.

Vstup : n, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: *maximum*

$maximum \leftarrow b_1$
for $i \leftarrow 2, 3, \ldots, n-1, n$ **do**
  if $b_i > maximum$ **then**
    $maximum \leftarrow b_i$
  end
end

Algoritmus 2: Algoritmus pro stanovení největšího čísla ve vektoru

Při programování uvedeného algoritmu v systému MATLAB je možno vytvořit m-funkci `maximum` s následující posloupností příkazů:

```
function m=maximum(b)
n=length(b)
m=b(1);
if n>1
  for i=2:n
    if b(i)>m
      m=b(i);
    end
  end
end
```

Poznámka 3.9. Funkce `length` vrací rozměr vektoru, obsaženého ve vstupním parametru.

Příklad 3.10. Určete hodnotu největšího průhybu nosníku z příkladu 3.1 s využitím tabelizovaných hodnot ohybové čáry z příkladu 3.6.

Řešení. Nejprve je nutno vytvořit vektor $b = b_1, b_2, \ldots, b_n$ s hodnotami průhybu v sledovaných průřezech ($b_i = w_z(x_i)$) o souřadnicích x_i s roztečí 10 cm (n tedy bude při rozpětí $l = 6$ m rovno hodnotě 61):

```
i=0;
for x=0:.1:l
  i=i+1;
  b(i)=horner(4,c,x)*1000;
end
```

Nyní lze vyvolat funkci pro hledání největšího čísla ve vektoru b. Po zadání příkazu `maximum(b)` bude výpis m-funkce a výsledek největšího průhybu v mm následující:

```
n =
    61

ans =
   23.7654
```

▲

Poznámka 3.11. Tento způsob výpočtu největší hodnoty funkce je pouze přibližný, neboť je zatížen chybou danou diskretizací osy x. Způsob výpočtu, který povede k přesnému řešení, bude ukázán v následující kapitole.

Příklady k procvičení

1. S využitím m-funkce `maximum` určete i největší průhyb na staticky neurčitém nosníku, který je schématicky zobrazen na obr. 3.2.

2. Stanovte největší průhyb na staticky určitém nosníku, jehož schéma je zobrazeno na obr. 3.3.

Kapitola 4

Řešení nelineárních algebraických rovnic

Cíle

Kapitola umožňuje bližší seznámení:

- s principem iteračních metod,
- se základními algoritmy pro stanovení kořenů nelineárních algebraických rovnic,
- s využitím cyklu typu **while** při tvorbě algoritmů.

4.1 Iterace

Slovo iterace může být použito ve stejném významu jako opakování (lat. *iteretur* – opakovat). V matematice se iterací označuje řešení problému postupným opakováním, které vede k přiblížení se k žádoucímu výsledku.

4.1.1 Taylorova řada

Jednoduchý iterační výpočet je možno demonstrovat na Taylorově řadě, tedy zvláštní mocninné řadě, pojmenované po anglickém matematikovi Brooku Taylorovi, který ji publikoval v roce 1712 (metoda aproximace funkce mocninnou řadou byla objevena již roku 1671 Jamesem Gregorym).

Za určitých předpokladů o funkci $f(x)$ v okolí bodu a lze tuto funkci vyjádřit (rozvinout) jako mocninnou řadu. Toto vyjádření funkce prostřednictvím *Taylorovy řady* se označuje jako *Taylorův rozvoj*:

$$f(x) = f(a) + \frac{f(a)'}{1!} \cdot (x-a) + \frac{f(a)''}{2!} \cdot (x-a)^2 + \frac{f(a)^{(3)}}{3!} \cdot (x-a)^3 + \cdots = \\ = \sum_{k=0}^{\infty} \frac{f(a)^{(k)}}{k!} \cdot (x-a)^k \,. \tag{4.1}$$

Pro přibližné vyjádření hodnot funkce není nutné vyjadřovat všechny členy Taylorovy řady, členy s vyššími derivacemi lze zanedbat. Tímto způsobem se získá tzv. Taylorův polynom, kterým lze aproximovat hodnoty funkce, která má v daném bodě derivaci, pomocí polynomu, jehož koeficienty závisí na derivacích funkce v tomto bodě.

Příklad 4.1. Stanovte s využitím prvních deseti členů Taylorova rozvoje hodnotu funkce e^x pro $x = 1$.

Řešení. Vztah pro Taylorův rozvoj, kterým lze stanovit hodnotu funkce e^x, má následující tvar:

$$f(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{(n)}}{n!} \,. \tag{4.2}$$

Vztah (4.2) je platný pro $x \in < -\infty, \infty >$. Algoritmicky lze daný výpočet vyjádřit pomocí algoritmu 3.

Vstup : n, x
Výstup: $f(x)$

$y \leftarrow 1$
for $i \leftarrow 1, 2, 3, \ldots, n-1$ **do**
  $y \leftarrow y + \frac{x^i}{i!}$
end
$f(x) \leftarrow y$

Algoritmus 3: Stanovení hodnoty funkce e^x s využitím Taylorova rozvoje

Skript programu MATLAB pak může obsahovat následující instrukce:

```
function y=ecko(n,x)
if n<2
  error('Počet členů Taylorova rozvoje n musí být nejméně 2!')
end
y=1;
for i=1:n-1
  y=y+(x^i)/factorial(i);
end
```

Po vyvolání této m-funkce:

```
ecko(10,1)
```

lze získat výsledek:

```
ans =
   2.718281525573192
```

Průběh postupného zpřesňování výsledku s přibývajícím počtem členů Taylorova rozvoje lze zobrazit pomocí následující tabulky:

```
   i     f(xi)          f(xi)-e^x
-----------------------------------
   1   1.00000000   -1.71828183e+000
   2   2.00000000   -7.18281828e-001
   3   2.50000000   -2.18281828e-001
   4   2.66666667   -5.16151618e-002
   5   2.70833333   -9.94849513e-003
   6   2.71666667   -1.61516179e-003
   7   2.71805556   -2.26272903e-004
   8   2.71825397   -2.78602051e-005
   9   2.71827877   -3.05861778e-006
  10   2.71828153   -3.02885853e-007
```

kde poslední sloupec představuje rozdíl dílčího výsledku s přesnou hodnotou e^x, vyvolanou např. s využitím funkce `exp(x)`.

▲

Poznámka 4.2. V m-souboru s názvem `ecko` byla využita funkce `error`, která zobrazí chybovou zprávu a ukončí činnost funkce.

Poznámka 4.3. Pro výpočet hodnoty faktoriálu čísla n byla použita funkce systému MATLAB s názvem `factorial`.

4.1.2 Zastavovací podmínka cyklu

V příkladu 4.1 byl při výpočtu aproximace hodnoty funkce s využitím Taylorova rozvoje zadán přesný počet požadovaných členů rozvoje (tím pádem bylo použito cyklu `for`). V praxi se ale nejčastěji používá tzv. „zastavovací podmínka cyklu“, která může nabývat tvaru:

$$|x_k - x_{k-1}| < \varepsilon \,, \tag{4.3}$$

kde x_{k-1}, x_k jsou členy Taylorova rozvoje na $(k-1)$. a k. místě a ε dostatečně malé číslo, označované jako tolerance nepřesnosti výsledku.

4.1.3 Rekurentní vzorec

Rekurentní vzorec určuje členy posloupnosti pomocí znalosti jednoho nebo více předcházejících prvků. Součástí každého rekurentního vzorce musí být zadání prvního, případně několika prvních členů posloupnosti. Nevýhodou zadání pomocí rekurentního vzorce je skutečnost, že libovolný prvek posloupnosti lze určit jen tehdy, pokud jsou známy členy předcházející. První prvek dané posloupnosti, který bývá nazýván „nultou aproximací“, je potřeba vždy odhadnou.

Příklad 4.4. S využitím tzv. „Heronova vzorce“ stanovte hodnotu funkce $f(x) = \sqrt{x}$ pro $x = 2$ s tolerancí nepřesnosti výsledku $\varepsilon = 0,001$.

Řešení. Obecný Heronův vzorec pro určení hodnoty funkce $f(x) = \sqrt{x}$ je rekurentní a nabývá tvar:

$$f(x) = y_k = \frac{1}{2}\left(y_{k-1} + \frac{x}{y_{k-1}}\right) , \tag{4.4}$$

kde y_{k-1}, y_k jsou opět $(k-1)$. a k. členy řady.

Vztah (4.4) je platný pro $x > 0$. Výpočetní postup pro danou úlohu lze vyjádřit pomocí algoritmu 4.

Vstup : x, ε
Výstup: $f(x)$

$y_1 \leftarrow x$
$y_2 \leftarrow \frac{1}{2} \cdot \left(y_1 + \frac{x}{y_1}\right)$
while $|y_1 - y_2| \geqq \varepsilon$ **do**
  $y_1 \leftarrow y_2$
  $y_2 \leftarrow \frac{1}{2} \cdot \left(y_1 + \frac{x}{y_1}\right)$
end
$f(x) \leftarrow y_2$

Algoritmus 4: Stanovení hodnoty funkce $f(x) = \sqrt{x}$ s využitím Heronova vzorce

Skript programu MATLAB pak může obsahovat následující instrukce:

```
function y=odmocnina(x,tol)
if x<=0
  error('Není splněna podmínka x>0!')
end
y1=x; y2=1/2*(y1+x/y1);
while abs(y1-y2)>tol
  y1=y2;
  y2=1/2*(y1+x/y1);
end
y=y2;
```

Po vyvolání této m-funkce, např.:

```
odmocnina(2,0.001)
```

lze získat odpovídající výsledek:

```
ans =
   1.414213562374690
```

Průběh postupného zpřesňování výsledku je zobrazen v následující tabulce:

```
   i      f(xi)          f(xi)-x^0.5
-----------------------------------
   1   2.00000000   5.85786438e-001
   2   1.50000000   8.57864376e-002
   3   1.41666667   2.45310429e-003
   4   1.41421569   2.12390141e-006
   5   1.41421356   1.59472435e-012
```

kde poslední sloupec představuje rozdíl dílčího výsledku s přesnou hodnotou $\sqrt{x}$, vyvolanou např. s využitím funkce programu MATLAB s názvem `sqrt(x)`.

Odchylku od přesného řešení lze rovněž zobrazit např. vyvoláním příkazu:

```
odmocnina(2,0.001)^2
```

který zobrazí:

```
ans =
   2.000000000004511
```

▲

Příklad 4.5. Ověřte platnost Taylorova rozvoje pro výpočet hodnoty funkce $\sin(x)$, jenž he uváděn ve tvaru:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{(2n+1)}}{(2n+1)!}\,, \tag{4.5}$$

s požadovanou tolerancí nepřesnosti výsledku $\varepsilon = 0,001$. Vztah (4.5) je platný pro $x \in < -\infty, \infty >$.

Příklad 4.6. Ověřte platnost Taylorova rozvoje pro výpočet hodnoty funkce $\cos(x)$, který nabývá tvar:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} (-1)^n \cdot \frac{x^{2n}}{(2n)!}\,. \tag{4.6}$$

s požadovanou tolerancí nepřesnosti výsledku $\varepsilon = 0,001$. Vztah (4.6) je platný pro $x \in < -\infty, \infty >$.

4.2 Iterační metody řešení nelineárních algebraických rovnic

Principů, popsaných v kap. 4.1, lze využít například při řešení nelineárních algebraických rovnic. Jednotlivé metody, popsané v následujícím textu, se liší zejména v rychlosti konvergence a univerzálnosti použití.

Konvergence je pojem označující sbíhání, sbíhavost, sbližování, popř. vývoj, který vede ke sblížení. O daných vlastnostech, které se sbližují, lze tvrdit, že konvergují. Objekty, procesy, vlastnosti apod., které se účastní konvergence, se označují jako konvergentní (výjimečně též jako konvergenční), např. konvergentní řada v matematice. V matematice je konvergence úzce spojena s pojmem limita. Opakem konvergence je **divergence**.

4.2.1 Prostá iterace

Řešenou rovnici:

$$f(x) = 0 \,, \tag{4.7}$$

je nutno nejprve upravit na tvar:

$$x = g(x) \,, \tag{4.8}$$

což lze většinou provést několika způsoby. Předpokladem výpočtu je skutečnost, že existuje interval $< a_0, b_0 >$, který spadá do definičního oboru i oboru spojitosti funkcí $f(x)$ a $g(x)$, jenž rovněž obsahuje společný kořen obou rovnic. Z tohoto intervalu je nutno zvolit i hodnotu nulté aproximace.

Poznámka 4.7. Nevýhodou této metody je v případě nevhodně zvolené funkce $g(x)$ konvergence řešení ke kořenu, který neleží v intervalu $< a_0, b_0 >$. Vyřešený kořen pak není společný pro obě rovnice, nýbrž je kořenem pouze rovnice $g(x)$.

Algoritmicky lze výpočet pro k iteračních kroků vyjádřit postupem algoritmu 5.

Vstup : x_0, k
Výstup: $f(x)$

for $i \leftarrow 1, 2, 3, \ldots, k$ **do**
  $x_i \leftarrow g(x_{i-1})$
end

$f(x) \leftarrow x_k$

Algoritmus 5: Stanovení kořene rovnice $f(x)$ s využitím metody prosté iterace

Zápis ve formě m-funkce pak vypadá následovně:

```
function xc=prosta_it(g,x0,k)
x(1)=x0;
for i=1:k
  x(i+1)=g(x(i));
end
x'
xc=x(k+1);
```

V seznamu parametrů m-funkce `prosta_it` je obsažena i řešená funkce g. Tu lze definovat prostřednictvím proměnné pomocí funkce programu MATLAB s názvem `inline`.

Příklad 4.8. Metodou prosté iterace stanovte aproximaci kořene rovnice:

$$f(x) = \left(\frac{x}{2}\right)^2 - \sin(x) = 0\ . \tag{4.9}$$

Vstupní parametry zvolte: počet iteračních cyklů $k = 10$, nultá aproximace $x_0 = 1,5$.

Řešení. Rovnici (4.9) lze upravit na tvar:

$$x = 2\sqrt{\sin(x)}\ . \tag{4.10}$$

Do programu MATLAB pak lze výraz (4.10) zadat příkazem:

```
g=inline('2*sqrt(sin(x))')
```

Nyní lze spustit výpočet povelem:

```
xc=prosta_it(g,1.5,10)
```

Seznam prvních deseti iterací je následující (desátou, poslední iteraci lze považovat za vyřešený kořen nelineární rovnice):

```
1.500000000000000
1.997493415863046
1.908232350897023
1.942788324690179
1.930393907098011
1.934981663979237
1.933302091730397
1.933919512286077
1.933692884811449
1.933776115524553
1.933745554580009
```

Po dosazení vypočteného kořene 1.933745554580009 do původní rovnice (4.9) vychází:

```
ans =
    -1.085057641792009e-005
```

což dostatečně vypovídá o nepřesnosti daného řešení. ▲

Příklad 4.9. Proveďte výpočet kořene nelineární rovnice z příkladu 4.8 s využitím metody prosté iterace s nultou aproximací $x_0 = 2,0$ a $k = 20$ iteračními cykly. Zobrazte dosaženou přesnost řešení.

Příklad 4.10. Upravte popsaný algoritmus metody prosté iterace tak, aby byl cyklus zakončen podmínkou (4.3). Příklad 4.8 pak vyřešte s parametrem $\varepsilon = 0,05$, vyjadřujícím toleranci nepřesnosti výsledku.

4.2.2 Metoda bisekce (půlení intervalů)

Touto metodou lze přibližně (se zadanou tolerancí ε) vyřešit kořen reálné nelineární rovnice $f(x) = 0$, která je spojitá pro $x \in < a_0; b_0 >$. Také se předpokládá, že platí: $f(a_0) \cdot f(b_0) < 0$, tj. sign $f(a_0) = -$ sign $f(b_0)$.

Postup výpočtu kořene nelineární rovnice $f(x) = 0$ metodou bisekce je vyjádřen algoritmem 6.

Po n výpočetních krocích bude mít zkoumaný interval s hledaným kořenem rovnice $f(x) = 0$ šířku:

$$b_n - a_n = \frac{1}{2}(b_{n-1} - a_{n-1}) = \frac{1}{2^2}(b_{n-2} - a_{n-2}) = \ \ldots \ = \frac{1}{2^n}(b_0 - a_0) \ . \tag{4.11}$$

Pro odhad chyby (nepřesnosti), kterou je zatížen výsledek, pak platí:

$$|a_n - \alpha| \leqq \frac{b_0 - a_0}{2^n} \ , \tag{4.12}$$

resp.

$$|b_n - \alpha| \leqq \frac{b_0 - a_0}{2^n} \ , \tag{4.13}$$

kde α představuje přesnou hodnotu kořene rovnice $f(x) = 0$.

Poznámka 4.11. Metoda bisekce konverguje velice pomalu (v [7] se uvádí zlepšení přesnosti o 1 desetinné místo po 3,3 výpočetních krocích, neboť $10^{-1} \approx 2^{-3,3}$). Na rychlost konvergence však nemá vliv řešená funkce $f(x)$. Výhodou je jednoduchost aplikace i možnost přesného odhadu počtu kroků, potřebných k dosažení požadované přesnosti.

Vstup : $\varepsilon > 0$, a_0, b_0
Výstup: x_c

$a \leftarrow a_0$
$b \leftarrow b_0$
while $|a - b| \geqq \varepsilon$ **do**
  $c \leftarrow \frac{a+b}{2}$
  if $f(c) = 0$ **then**
    $x_c \leftarrow c$ `/* c je výsledný kořen rovnice */`
    konec;
  end
  if $\text{sign}(f_c) \cdot \text{sign}(f_a) < 0$ **then**
    $b \leftarrow c$ `/* nové hranice intervalu jsou` $< a, c >$ `*/`
  else
    $a \leftarrow c$ `/* nové hranice intervalu jsou` $< c, b >$ `*/`
  end
end
$x_c \leftarrow \frac{a+b}{2}$

Algoritmus 6: Algoritmus metody bisekce (půlení intervalů)

Příklad 4.12. Metodou bisekce stanovte aproximaci kořene rovnice:

$$f(x) = x^3 + x - 1 \,. \tag{4.14}$$

Vstupní parametry zvolte: $\varepsilon = 0,001$, $a_0 = 0$ a $b_0 = 1$.

Řešení. Funkci pro výpočet kořene nelineární rovnice metodou bisekce lze naprogramovat prostřednictvím m-souboru následovně:

```
function xc=bisect(f,a,b,tol)
if sign(f(a))*sign(f(b))>=0
  error('Není splněna podmínka f(a)*f(b)<0!')
end
fa=f(a);
fb=f(b);
while(b-a)>tol
  c=(a+b)/2;
  fc=f(c);
  if fc==0
```

```
    xc=c
    break
  end
  if sign(fc)*sign(fa)<0  %nové hranice intervalu jsou <a,c>
    b=c;
    fb=fc;
  else                    %nové hranice intervalu jsou <c,b>
    a=c;
    fa=fc;
  end
end
xc=(a+b)/2;
```

V seznamu parametrů je obsažena i řešená funkce f. Tu lze definovat prostřednictvím proměnné pomocí funkce programu MATLAB s názvem `inline`:

```
f=inline('x^3+x-1')
```

Nyní lze spustit výpočet zadáním příkazu:

```
xc=bisect(f,0,1,0.001)
```

Výsledek je pak:

```
xc =
    0.6821
```

Na demonstraci práce funkce bisect je možno zobrazit jak se mění obsah proměnných a, b a c během iterace. Sloupce označené $f(a)$, $f(b)$ a $f(c)$ pak obsahují hodnotu -1 nebo $+1$ podle toho, jestli je příslušná hodnota funkce $f(a)$, $f(b)$ a $f(c)$ záporná či kladná.

```
  i     a    f(a)    c    f(c)    b    f(b)
-----------------------------------------
  0   0.0000  -1   0.5000  -1   1.0000   1
  1   0.5000  -1   0.7500   1   1.0000   1
  2   0.5000  -1   0.6250  -1   0.7500   1
  3   0.6250  -1   0.6875   1   0.7500   1
  4   0.6250  -1   0.6563  -1   0.6875   1
  5   0.6563  -1   0.6719  -1   0.6875   1
  6   0.6719  -1   0.6797  -1   0.6875   1
  7   0.6797  -1   0.6836   1   0.6875   1
  8   0.6797  -1   0.6816  -1   0.6836   1
  9   0.6816  -1   0.6826   1   0.6836   1
 10   0.6816  -1   0.6821  -1   0.6826   1
```

▲

Příklad 4.13. Metodou bisekce stanovte aproximaci kořene rovnice (4.9) z příkladu 4.8. Vstupní parametry zvolte: $\varepsilon = 0,05$, $a_0 = 1,5$ a $b_0 = 2,0$.

Příklad 4.14. Pro nosník z příkladu 3.1 stanovte největší průhyb na konstrukci s využitím metody bisekce.

Řešení. V průřezu s největším průhybem platí: $w_z(x)' = \varphi_y(x) = 0$ a pro jeho určení je tedy nutno vyřešit kořeny polynomu 3. stupně. Vztáhne-li se odvozená rovnice pro pootočení 3.26 ke vztahu 1.1 s obecným vyjádřením polynomu n-tého stupně, vektor `c` s prvky $[c_n, c_{n-1}, \ldots, c_1, c_0]$ pak nabývá tvar:

$$\mathbf{c} = \left[\frac{q_z}{6}\,,\quad -\frac{3}{16}\,q_z l\,,\quad 0\,,\quad \frac{1}{48}\,q_z l^3 \right]\,. \tag{4.15}$$

Výpočet kořene polynomu metodou bisekce lze provést s nepatrně upravenou m-funkcí z příkladu 4.12:

```
function xc=bisekce(a,b,d,tol)
if sign(horner(3,d,a))*sign(horner(3,d,b))>=0
  error('Není splněna podmínka f(a)*f(b)<0!')
end
fa=horner(3,d,a);
fb=horner(3,d,b);
while b-a>tol
  c=(a+b)/2;
  fc=horner(3,d,c);
  if fc==0
    xc=c
    break
  end
  if sign(fc)*sign(fa)<0  %nové hranice intervalu jsou <a,c>
    b=c;
    fb=fc;
  else                    %nové hranice intervalu jsou <c,b>
    a=c;
    fa=fc;
  end
end
xc=(a+b)/2
```

Největší průhyb se pak získá dosazením výsledného kořene polynomu pootočení z intervalu $(0; l)$ do rovnice ohybové čáry 3.27.

Celá posloupnost příkazů, potřebných pro určení průřezu s nulovým pootočením i největšího průhybu na nosníku, může být následující:

```
format long
qz=4000;
l=6;
b=0.02;
h=0.15;
Iy=1/12*b*h^3;
E=2.1*10^11;
c=[0 1/(48*E*Iy)*qz*l^3 0 -3/(48*E*Iy)*qz*l qz/(24*E*Iy)];
fi=[1/(48*E*Iy)*qz*l^3 0 -3/(16*E*Iy)*qz*l qz/(6*E*Iy)];
xmax=bisekce(0,l-0.00001,fi,0.00000000000001)
poot=horner(3,fi,xmax)
wmax=horner(4,c,xmax)*1000
```

Pro dané zadání pak vychází největší průhyb v průřezu se souřadnicí:

```
xmax =
   2.529210992451759
```

Pootočení je v tomto řezu:

```
poot =
   1.734723475976807e-017
```

a samotný maximální průhyb pak v milimetrech vychází:

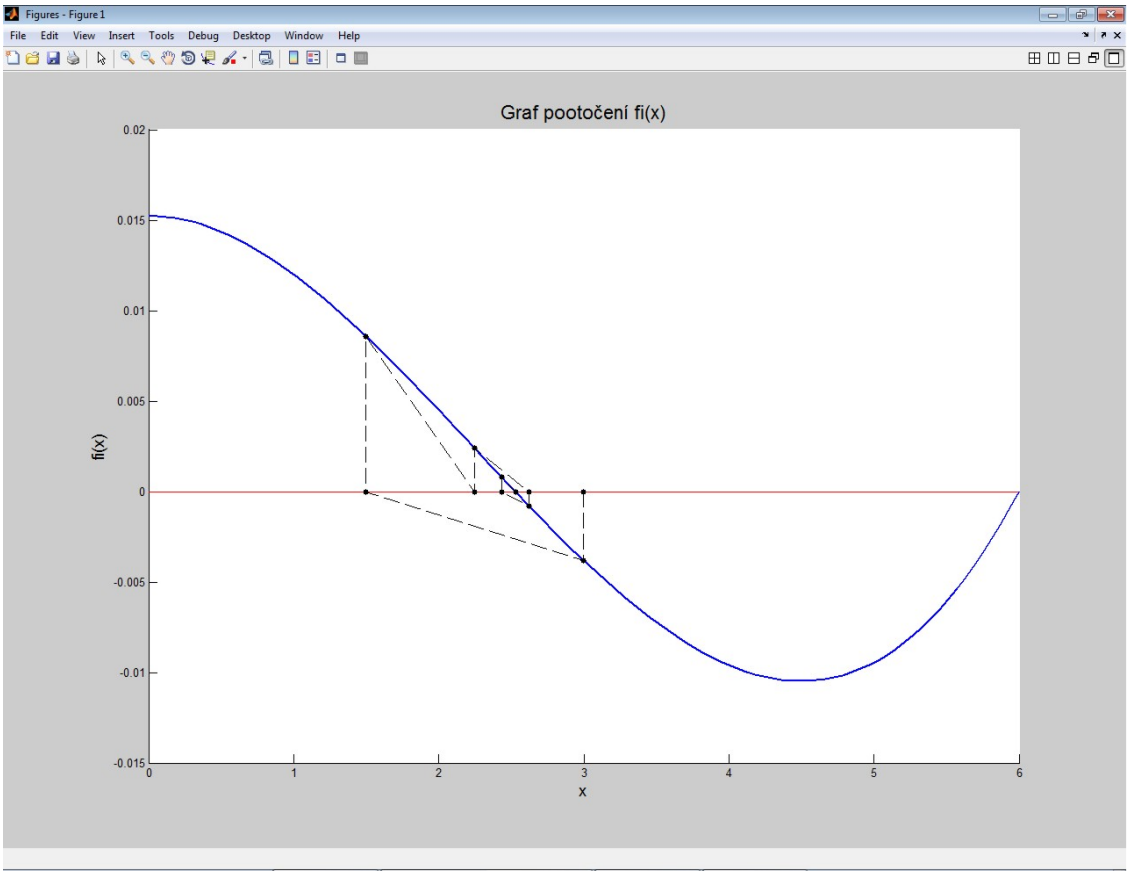
```
wmax =
   23.769036533008371
```

▲

Poznámka 4.15. Pro znázornění principu fungování algoritmu metody bisekce byl na obr. 4.1 vytvořen graf pootočení nosníku z předchozího příkladu posloupností příkazů:

```
x=linspace(0,l,100);
plot([0,l],[0,0],'r-')
for i=1:100, y(i)=horner(3,fi,x(i)); end
plot(x,y,'b-');
title('Graf pootočení fi(x)');
xlabel('x');
ylabel('fi(x)');
```

V grafu je vyznačena cesta iteračního výpočtu od středu intervalu směrem k řešenému kořenu rovnice pootočení.



Obr. 4.1 Graf pootočení staticky neurčitého nosníku z příkladu 3.1 a iterace k průřezu s největším průhybem metodou bisekce

Poznámka 4.16. V grafu na obr. 4.1 lze pozorovat, že hraniční hodnota pootočení v bodě b je nulová. Pokud by tedy byly při vyvolání funkce bisekce zvoleny meze $a_0 = 0$ a $b_0 = l$, nebyla by splněna požadovaná podmínka pro řešení touto metodou: $f(a_0) \cdot f(b_0) < 0$. Z tohoto důvodu byla hodnota vstupního parametru $b_0 = l$ nepatrně upravena (na hodnotu $b_0 = l - 0.00001$):

```
xmax=bisekce(0,l-0.00001,fi,0.00000000000001,1000)
```

4.2.3 Metoda regula falsi

Touto metodou lze přibližně (se zadanou tolerancí ε) rovněž vyřešit kořen reálné nelineární rovnice $f(x) = 0$, která je spojitá pro $x \in < a; b >$. Předpoklad: $f(a_0) \cdot f(b_0) < < 0$, tj. sign $f(a_0) = -$ sign $f(b_0)$ zůstává stále v platnosti, neboť odpovídá skutečnosti, že v intervalu $< a; b >$ existuje reálný kořen uvedené rovnice.

Metoda regula falsi stanoví nejprve průsečík sečny křivky $f(x)$ sestrojené v bodech $[a, f(a)]$ a $[b, f(b)]$ podle vztahu:

$$s = a - \frac{f(a)}{f(b) - f(a)} \cdot (b - a) \,. \tag{4.16}$$

V případě, že $\text{sign}(f(s)) = \text{sign}(f(a))$, změní se zkoumaný interval na $< s, b >$, v opačném případě na $< a, s >$ a výpočet průsečíku sečny křivky $f(x)$ pokračuje s využitím rekurentního vzorce (4.16) dokud není splněna zastavovací podmínka.

Postup výpočtu kořene nelineární rovnice $f(x) = 0$ metodou regula falsi je vyjádřen algoritmem 7.

Vstup : $\varepsilon > 0$, a, b
Výstup: x_c

$s \leftarrow a - \dfrac{f(a)}{f(b) - f(a)} \cdot (b - a)$

while $|f(s)| \geqq \varepsilon$ **do**

  if $\text{sign}(f(s)) = \text{sign}(f(a))$ **then**

    $a \leftarrow s$ `/* nové hranice intervalu jsou` $< s, b >$ `*/`

  else

    $b \leftarrow s$ `/* nové hranice intervalu jsou` $< a, s >$ `*/`

  end

  $s \leftarrow a - \dfrac{f(a)}{f(b) - f(a)} \cdot (b - a)$

end

$x_c \leftarrow s$

Algoritmus 7: Algoritmus metody regula falsi

Příklad 4.17. Metodou regula falsi stanovte aproximaci kořene rovnice (4.14) z příkladu 4.12. Vstupní parametry zvolte obdobné $\varepsilon = 0,001$, $a_0 = 0$ a $b_0 = 1$.

Řešení. Zápis funkce pro výpočet kořene nelineární rovnice metodou regula falsi ve formě skriptu programu MATLAB může být následující:

```
function xc=regula(f,a,b,tol)
if sign(f(a))*sign(f(b))>=0
  error('Není splněna podmínka f(a)*f(b)<0!')
end
fa=f(a);
fb=f(b);
s=a-fa/(fb-fa)*(b-a);
fs=f(s);
```

```
while abs(fs)>=tol
  if sign(fs)==sign(fa)  %nové hranice intervalu jsou <s,b>
    a=s;
    fa=fs;
  else                   %nové hranice intervalu jsou <a,s>
    b=s;
    fb=fs;
  end
  s=a-fa/(fb-fa)*(b-a);
  fs=f(s);
end
xc=s;
```

V seznamu parametrů je opět obsažena i řešená funkce f, kterou lze definovat již známou funkcí `inline`. Výpočet bude spuštěn příkazem:

```
xc=regula(f,0,1,0.001)
```

Výsledek je pak:

```
xc =
   0.682175815962540
```

Činnost funkce regula v průběhu iterace se dá zobrazit v následující tabulce. Lze z ní vyčíst měnící se obsah proměnných a, b a s během iterace i zda-li je příslušná hodnota $f(a)$, $f(s)$ a $f(s)$ záporná či kladná.

```
  i     a    f(a)    s    f(s)    b    f(b)
-----------------------------------------
  0   0.0000  -1   0.5000  -1   1.0000   1
  1   0.5000  -1   0.6364  -1   1.0000   1
  2   0.6364  -1   0.6712  -1   1.0000   1
  3   0.6712  -1   0.6797  -1   1.0000   1
  4   0.6797  -1   0.6817  -1   1.0000   1
  5   0.6817  -1   0.6822  -1   1.0000   1
```

▲

Poznámka 4.18. Metoda regula falsi konverguje k výslednému kořenu vždy, pokud je jeho existence zaručena vstupními parametry. Konvergence metodou regula falsi probíhá rychleji než tomu bylo u metody bisekce, čemuž odpovídá i průběh výpočtů v příkladu 4.12 a 4.17. Zatímco při požadované toleranci nepřesnosti výsledku $\varepsilon = 0.001$ bylo metodou bisekce dosaženo řešení po 10 krocích, metoda regula falsi nalezla hledaný kořen již po 5 iteracích.

Příklad 4.19. Pro nosník z příkladu 3.1 stanovte podobně jako v příkladu 4.14 největší průhyb na konstrukci, tentokráte ovšem s využitím metody regula falsi.

Řešení. Pro vstupní údaje $a = 0$, $b = l - 0,001$ a $\varepsilon = 0,0001$ vychází s využitím metody regula falsi největší průhyb v průřezu se souřadnicí:

```
xmax =
   2.529471870690230
```

Pootočení je v tomto řezu:

```
poot =
   -2.201632719885452e-006
```

a samotný maximální průhyb pak v milimetrech vychází:

```
wmax =
   23.769036245827937
```

▲

Poznámka 4.20. Pro znázornění principu fungování algoritmu metody regula falsi byl na obr. 4.2 vytvořen graf pootočení nosníku z předchozího příkladu. V grafu je vyznačena cesta iteračního výpočtu od počáteční aproximace směrem k řešenému kořenu rovnice pootočení.

4.2.4 Metoda sečen

Při uplatnění tohoto výpočetního postupu se řešená funkce $f(x)$ nahradí lineární funkcí:

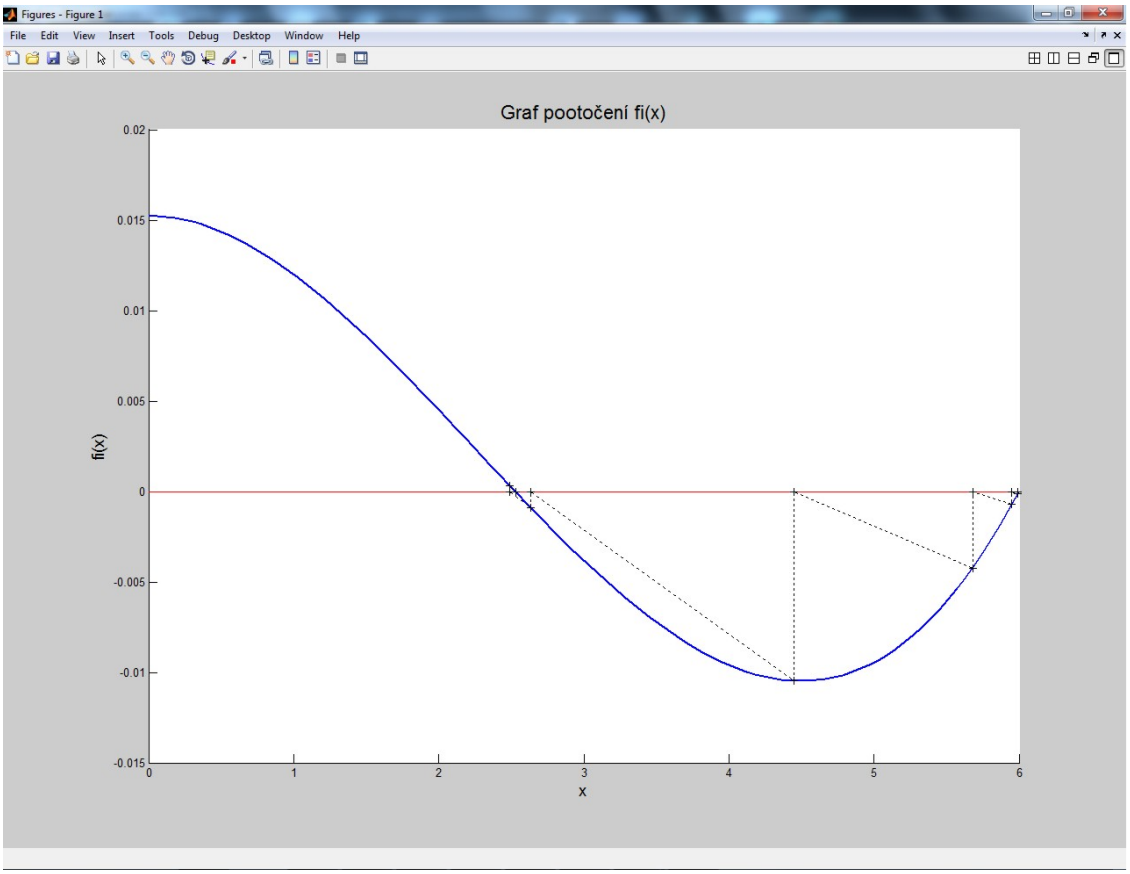
$$g(x) = \frac{f(x_k - f(x_{k-1}))}{x_k - x_{k-1}} \cdot (x - x_k) + f(x_k) \ . \tag{4.17}$$

Určení kořene rovnice $g(x)$ pak spočívá v uplatnění rekurentního vzorce:

$$x_{k+1} = x_k - f(x_k) \cdot \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \ . \tag{4.18}$$

Kořen x_{k+1} lze považovat za aproximaci kořene rovnice $f(x) = 0$. Vztah (4.18) představuje dvoukrokovou iterační formuli, neboť k zahájení iteračního výpočtu je potřeba určit dvě počáteční aproximace x_0 a x_1.

Poznámka 4.21. Pokud dvě počáteční aproximace x_0 a x_1 nebudou vhodně zvoleny, nemusí metoda sečen konvergovat. Z tohoto důvodu je nutné algoritmus výpočtu opatřit kontrolou konvergence, příp. počáteční aproximace stanovit s využitím jiné metody.



Obr. 4.2 Graf pootočení staticky neurčitého nosníku z příkladu 3.1 a iterace k průřezu s největším průhybem metodou regula falsi

Příklad 4.22. Metou sečen vyřešte kořen rovnice z příkladu 4.17. Vstupní parametry zvolte: $x_0 = 0$, $x_1 = 1$ a $\varepsilon = 0,001$.

Řešení. Při tvorbě algoritmu i instrukcí m-souboru je možno vyjít z výpočetního postupu metody regula falsi, který je nutné upravit v souvislosti s odlišným rekurentním vzorcem (4.18) metody sečen.

M-funkci s názvem např. `m_secen` pak lze vyvolat příkazem:

```
xc=m_secen(f,0,1,0.001)
```

Pro dané vstupní parametry vychází výsledek:

```
xc =
   0.682020419648186
```

Výpočetní postup metody sečen lze nejlépe sledovat postupným výpisem hodnot x_{k-1}, x_k a x_{k+1} i jejích funkčních hodnot:

```
  i    x(k-1)   f(x(k-1))      x(k)     f(x(k))    x(k+1)   f(x(k+1))
-------------------------------------------------------------------
  0    0.0000   -1.000000    1.0000    1.000000    0.5000   -0.375000
  1    1.0000    1.000000    0.5000   -0.375000    0.6364   -0.105935
  2    0.5000   -0.375000    0.6364   -0.105935    0.6901    0.018636
  3    0.6364   -0.105935    0.6901    0.018636    0.6820   -0.000737
```

Pro dosažení výsledku s tolerancí $varepsilon = 0,001$ postačily pouze 3 iterační cykly. ▲

Příklad 4.23. Pro nosník z příkladu 3.1 stanovte podobně jako v příkladech 4.14 nebo 4.19 největší průhyb na konstrukci, tentokráte ovšem s využitím metody sečen.

Řešení. Pro vstupní údaje $a = 0$, $b = l/2$ a $\varepsilon = 0,0001$ vychází s využitím metody sečen největší průhyb v průřezu se souřadnicí:

```
xmax =
   2.534161490683230
```

Pootočení je v tomto řezu:

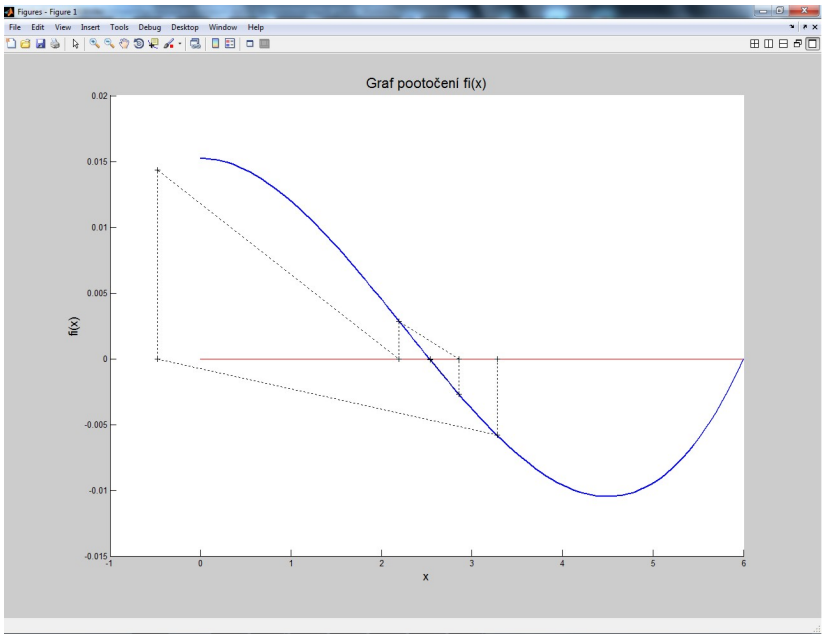
```
poot =
   -4.176775334049400e-005
```

a samotný maximální průhyb pak v milimetrech vychází:

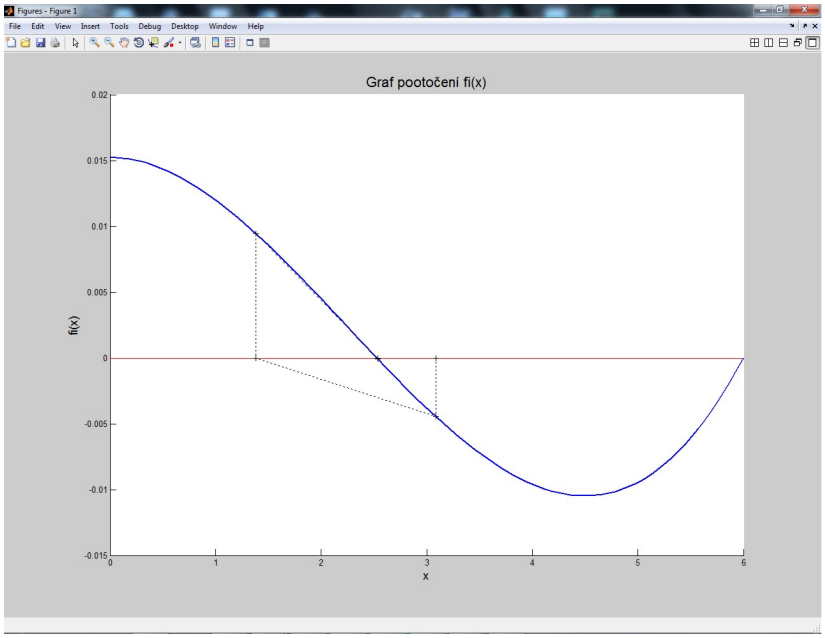
```
wmax =
   23.768933137770031
```

▲

Poznámka 4.24. Pro znázornění principu fungování algoritmu metody sečen byly na obr. 4.3 a 4.4 vytvořeny grafy pootočení nosníku z předchozího příkladu. V grafech je vyznačena cesta iteračního výpočtu od počáteční aproximace x_k směrem k řešenému kořenu rovnice pootočení. Oba grafy se liší nepatrně pozměněnou hodnotou vstupního parametru b, který byl zadán $b = l - 0,85$, resp. $b = l - 1,0$. Jak již bylo řečeno v pozn. 4.21, metoda je velice citlivá na hodnoty prvních dvou aproximací x_0 a x_1.



Obr. 4.3 Graf pootočení nosníku z příkladu 3.1 a iterace k průřezu s největším průhybem metodou sečen se vstupními parametry $a = 0$, $b = l - 0,85$ a $\varepsilon = 0,0001$.



Obr. 4.4 Graf pootočení nosníku z příkladu 3.1 a iterace k průřezu s největším průhybem metodou sečen se vstupními parametry $a = 0$, $b = l - 1,0$ a $\varepsilon = 0,0001$.

4.2.5 Newtonova metoda (metoda tečen)

Pokud jednoduchý reálný kořen rovnice $f(x) = 0$ leží v intervalu $< a, b >$, ve kterém existují i derivace $f'(x), f''(x)$, lze vyjádřit funkci f ve tvaru Taylorova rozvoje v bodě x_0:

$$f(x) = f(x_0) + f'(x_0) \cdot (x - x_0) + \frac{1}{2} \cdot f''(\xi_0) \cdot (x - x_0)^2 \, . \tag{4.19}$$

Rovnici $f(x) = 0$ lze aproximovat lineární rovnicí, kterou tvoří první dva členy tohoto rozvoje (4.19):

$$f(x_0) + f'(x_0) \cdot (x - x_0) = 0 \, . \tag{4.20}$$

Kořen lineární rovnice (4.20) se dá stanovit:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \, . \tag{4.21}$$

Pokud se rovnice (4.20) vyjádří obecně pro x_k:

$$f(x_k) + f'(x_k) \cdot (x - x_k) = 0 \, , \tag{4.22}$$

při řešení lze získat posloupnost, definovanou rekurentní formulí

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \, , \tag{4.23}$$

která vyjadřuje základní myšlenku Newtonovy metody.

Poznámka 4.25. Podle [9] je vhodné volit nultou aproximaci x_0 mezi krajními body a, b tak, aby $\text{sign}(f(x_0)) = \text{sign}(f''(x_0))$. Pokud mezi skutečným kořenem počáteční aproximací leží inflexní bod, může posloupnost konvergovat k některému z ostatních kořenů řešené funkce, příp. divergovat nebo oscilovat.

Poznámka 4.26. Algoritmus Newtonovy metody odpovídá výpočetnímu postupu metody prosté iterace pro funkci:

$$\varphi(x) = x - \frac{f(x)}{f'(x)} \, . \tag{4.24}$$

Poznámka 4.27. Oproti předchozím postupům obsahuje rekurentní vzorec posloupnosti derivaci funkce $f(x)$. Numerické derivování bude vysvětleno v některé z dalších kapitol, proto je následující příklad zaměřen pouze na úlohu, kde je derivace řešené funkce $f'(x)$ známa.

Příklad 4.28. Pro nosník z příkladu 3.1 stanovte podobně jako v příkladech 4.14, 4.19 nebo 4.23 největší průhyb na konstrukci. Pro určení průřezu s největším průhybem použijte Newtonovu metodu.

Řešení. Jak již bylo odvozeno v příkladu 3.1, derivace pootočení podle vztahu 3.15 je dána:

$$w_z(x)'' = \varphi_y'(x) = -\frac{1}{EI_y} \cdot M_y(x) \,, \tag{4.25}$$

konkrétně tedy podle (3.25):

$$\varphi_y'(x) = \frac{1}{EI_y} \cdot \left(q_z \frac{x^2}{2} - \frac{3}{8} q_z l x \right) \,, \tag{4.26}$$

Vztáhne-li se odvozená rovnice pro pootočení 3.26 ke vztahu 1.1 s obecným vyjádřením polynomu n-tého stupně, vektor `c` s prvky $[c_n, c_{n-1}, \ldots, c_1, c_0]$ pak nabývá tvar:

$$\mathbf{c} = \left[\; \frac{q_z}{2EI_y} \,, \quad -\frac{3q_z l}{8EI_y} \,, \quad 0 \; \right] \,. \tag{4.27}$$

Sled příkazů pro daný výpočet může vypadat následovně:

```
format long
qz=4000;
l=6;
b=0.02;
h=0.15;
Iy=1/12*b*h^3;
E=2.1*10^11;
c=[0 1/(48*E*Iy)*qz*l^3 0 -3/(48*E*Iy)*qz*l qz/(24*E*Iy)];
fi=[1/(48*E*Iy)*qz*l^3 0 -3/(16*E*Iy)*qz*l qz/(6*E*Iy)];
m=[0 -3*qz*l/(8*E*Iy) qz/(2*E*Iy)];
xmax=newton_it(3,fi,m,0.0001)
horner(3,fi,xmax)
horner(4,c,xmax)*1000
```

kde `newton_it` představuje m-funkci, vyvolanou se vstupním parametrem nulté aproximace $x_0 = 3$ metry a tolerancí nepřesnosti výsledku $\varepsilon = 0,0001$, obsahující např. příkazy:

```
function xc=newton_it(a,fi,m,tol)
f1=horner(2,m,a);
f2=horner(3,fi,a);
s=a-f2/f1;
fs=horner(3,fi,s);
```

```
while abs(fs)>=tol
  a=s;
  f1=horner(2,m,a);
  f2=horner(3,fi,a);
  s=a-f2/f1;
  fs=horner(3,fi,s);
end
xc=s;
```

Řešením je pak pro dané vstupní údaje:

```
xmax =
   2.529166666666667
```

Pootočení je v tomto řezu:

```
poot =
   3.740855052600245e-007
```

Maximální průhyb pak byl v milimetrech určen hodnotou:

```
wmax =
   23.769036524717556
```

Pokud provedeme výpis hodnot x_i, $f(x_i)$, $f'(x_i)$, x_{i+1} a $f(x_{i+1})$:

```
  i      x(i)     f(x(i))    df(x(i))    x(i+1)   f(x(i+1))
-------------------------------------------------------------
  0    3.0000   -0.007619   -0.003810    2.5000    0.000247
  1    2.5000   -0.008466    0.000247    2.5292    0.000000
```

zjistíme, že touto metodou jsme k dostatečně přesnému řešení dospěli již po prvním iteračním kroku. ▲

Příklady k procvičení

1. Výše vysvětlenými výpočetními postupy stanovte největší průhyb na staticky neurčitém nosníku, který je schématicky zobrazen na obr. 3.2,

2. Vypočtěte rovněž největší průhyb na staticky určitém nosníku, jehož schéma je zobrazeno na obr. 3.3.

Kapitola 5

Metody pro třídění souboru prvků

Cíle

Kapitola studentům přiblíží:

- základy algoritmu pro třídění, které najdou uplatnění v mnoha inženýrských aplikacích,
- dvojný cyklus `for`,
- procedury pro čtení dat z textového souboru i jejich zápis do diskového souboru.

5.1 Třídící algoritmy

Třídící (řadící) algoritmy najdou uplatnění v celé řadě inženýrských úloh, např. při zpracování experimentálně získaných dat či měření. Souhrnně jsou uvedeny např. v [1].

5.1.1 Bublinkové třídění (Bubble Sort)

Bublinkové řazení (anglicky *Bubblesort*, česky též *řazení záměnou*) je implementačně jednoduchý řadicí algoritmus. Název algoritmu vyjadřuje průběh zpracování. Při představě, že řazená čísla jsou přirovnány k bublinkám, které stoupají k vodní hladině s různou rychlostí podle své velikosti, pak řazené prvky s větší hodnotou „probublávají“ směrem ke konci vzestupně řazeného seznamu. Algoritmus opakovaně prochází posloupnost n prvků, přičemž porovnává každé dva sousedící prvky, a pokud nejsou ve správném pořadí, prohodí je. Porovnávání prvků běží do té doby, dokud není seznam seřazený.

Algoritmus je univerzální (pracuje na základě porovnávání dvojic prvků), pracuje lokálně (nevyžaduje pomocnou paměť), je stabilní (prvkům se stejným klíčem nemění vzájemnou polohu) a patří mezi přirozené řadicí algoritmy (částečně seřazený seznam zpracuje rychleji než neseřazený).

Vstup : x $= \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: x

for $j \leftarrow 1, 2, 3, \ldots, n-1$ **do**
  for $i \leftarrow n-1, n-2, \ldots, j+1, j$ **do**
    if $x_i > x_{i+1}$ **then**
      $c \leftarrow x_i$
      $x_i \leftarrow x_{i+1}$
      $x_{i+1} \leftarrow c$
    end
  end
end

Algoritmus 8: Algoritmus třídění typu Bubble sort

Výpočetní postup je schématicky znázorněn algoritmem 8 a jeho přepis do programovacího jazyka systému MATLAB může vypadat následovně:

```
function y=bubblesort(x);
n=length(x);
for j=1:n-1
  for i=n-1:-1:j
    if x(i)>x(i+1);
      c=x(i);
      x(i)=x(i+1);
      x(i+1)=c;
    end
  end
end
y=x;
```

Pro praktické účely je však postup neefektivní, využívá se hlavně pro výukové účely či v nenáročných aplikacích. Tento algoritmus řazení je jedním z nejpomalejších, oproti jiným algoritmům se stejnou složitostí vyžaduje velké množství zápisů do paměti.

Poznámka 5.1. Z hlediska použití cyklů typu `for` lze podotknout, že algoritmus obsahuje dvojný cyklus s řídícími proměnnými i a j. K seřazení pole o n prvcích bude zapotřebí $n-1$ provedení vnějšího cyklu, zatímco vnitřní cyklus se postupně provede $(n-1), (n-2), (n-3), \ldots, 2, 1$ krát. Koncová hodnota řídící proměnné j tedy není konstantní, ale závisí na hodnotě řídící proměnné i. Celkový počet operací při tomto typu třídění bude $\frac{n^2-n}{2}$, což řádově odpovídá složitosti úlohy s přibližným počtem výpočetních operací n^2.

5.1.2 Třídění přímým výběrem minima (Select Sort)

Třídění přímým výběrem minima neboli *Selection sort* (zkráceně *Selectsort*) je jednoduchý algoritmus uspořádávání. Pro svou jednoduchou implementaci bývá často používán pro uspořádávání malých množství dat. Pro větší objem dat se používají algoritmy s nižší časovou složitostí.

Algoritmus postupně prochází tříděný vektor a pro daný prvek vektoru hledá minimum ze zbývajících prvků. Postup lze popsat následovně:

1. nalezne se prvek s nejmenší hodnotou v posloupnosti n hodnot,
2. zamění se s prvkem na první pozici, takže se zde nachází prvek s nejmenší hodnotou,
3. zbytek posloupnosti se uspořádá opakováním těchto kroků pro zbylých $n-1$ prvků.

Algoritmus *Selectsortu* tedy vychází z myšlenky, že pokud se řadí pole od nejmenšího prvku k největšímu, tak na první pozici posloupnosti bude prvek s nejmenší hodnotou, za ním prvek s nejmenší hodnotou ze zbytku pole atd., čili stačí pouze postupně vybírat nejmenší prvky z neseřazené části pole a umísťovat je na konec seřazené části.

Vstup : $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: $\mathbf{x}$

```
for i ← 1, 2, 3, ..., n − 1 do
    for j ← i + 1, i + 2, i + 3, ..., n do
        if x_i > x_j then
            c ← x_i
            x_i ← x_j
            x_j ← c
        end
    end
end
```

Algoritmus 9: Algoritmus metody třídění přímým výběrem minima

Schématické znázornění řadícího výpočtu je obsaženo v algoritmu 9. Následuje jeho interpretace v programovacím jazyce systému MATLAB:

```
function y=selectsort(x);
n=length(x);
for i=1:n-1
  for j=i+1:n
    if x(i)>x(j);
      c=x(i);
      x(i)=x(j);
      x(j)=c;
    end
  end
end
y=x;
```

Poznámka 5.2. Z hlediska použití cyklu `for` lze podobně jako v předchozím případě konstatovat, že algoritmus obsahuje dvojný cyklus s řídícími proměnnými i a j a složitost úlohy zůstává s přibližným počtem výpočetních operací n^2.

5.1.3 Třídění přímým vkládáním (Insert Sort)

Třídění přímým vkládáním neboli *Insertion sort* (zkráceně *Insertsort*) je jednoduchý řadicí algoritmus založený na porovnávání. Algoritmus výpočtu pracuje tak, že prochází prvky postupně a každý další nesetříděný prvek zařadí na správné místo do již setříděné posloupnosti. Je to jeden z nejrychlejších algoritmů. Je sice pomalejší než pokročilé algoritmy typu *Quicksort* nebo *Shellsort* (viz dále), ale zato má jiné výhody:

- jednoduchá implementace,
- efektivní na malých množinách,
- efektivní na částečně seřazených množinách,
- efektivnější než předchozí algoritmy (Selectsort, Bubblesort),
- řadí stabilně (nemění vzájemné pořadí prvků se stejnými klíči),
- jedná se o tzv. online algoritmus, neboť umožňuje řadit data tak, jak vstupují do výpočtu.

Algoritmus zatřiďuje prvky z pravé části posloupnosti přímo do části levé, kde se utváří vzestupně seřazený seznam hodnot. Jediný rozdíl oproti předchozímu algoritmu třídění `Selectsort` tkví v tom, že tento způsob manipuluje při řazení přímo

Vstup : x $= \{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$
Výstup: x

for $i \leftarrow 2, 3, \ldots, n-1, n$ **do**
  $c \leftarrow x_i$
  for $j \leftarrow i-1, i-2, i-3, \ldots, 1$ **do**
    if $x_j > x_{j+1}$ **then**
      $x_{j+1} \leftarrow x_j$
      $x_j \leftarrow c$
    end
  end
end

Algoritmus 10: Algoritmus metody třídění přímým vkládáním

s jednotlivými prvky a ne jenom s jejich indexy, takže lze výpočetní postup využít např. při postupném načítání dat ze souboru.

Schématické znázornění výpočtu je uvedeno v algoritmu 10 a jeho přepis do programovacího jazyka systému MATLAB je následující:

```
function y=insertsort(x);
n=length(x);
for i=2:n
  c=x(i);
  for j=i-1:-1:1
    if x(j)>x(j+1);
      x(j+1)=x(j);
      x(j)=c;
    end
  end
end
y=x;
```

Řadící algoritmy Quicksort a Shellsort, které jsou uvedeny v dalším výkladu, jsou již z kategorie pokročilých. Jejich uvedení v tomto učebním textu má za cíl nahlédnout do pozadí kvalitních výpočetních postupů.

5.1.4 Rychlé (rekurzivní) řazení (Quick Sort)

Rychlé (rekurzivní) řazení do tříd neboli zkráceně *Quicksort* je jeden z nejrychlejších známých algoritmů řazení založených na porovnávání prvků. Algoritmus vymyslel v roce 1962 Sir Charles Antony Richard Hoare. Jeho průměrná časová složitost je pro algoritmy této skupiny nejlepší možná $(n \cdot \log n)$, v nejhorším případě (kterému se ale v praxi jde obvykle vyhnout) může být jeho časová náročnost opět i n^2.

Základní myšlenkou *Quicksortu* je rozdělení řazené posloupnosti čísel na dvě přibližně stejné části. V jedné části jsou čísla větší a ve druhé menší, než je hodnota jednoho ze zvolených prvků posloupnosti, nazývaného **pivot**. Pokud je pivot zvolen optimálně, jsou obě části posloupnosti přibližně stejně velké. Obě části se pak řadí samostatně.

Důležitým aspektem pro výkonnost tohoto výpočetního postupu je tedy volba pivota. Používanou volbou pivota bývá tzv. fixní prvek (první nebo poslední) a náhodný prvek. Fixní prvek má problém na částečně uspořádaných polích, případně na polích s nějakou strukturou (kde nedochází k optimálnímu dělení problému a složitost narůstá až k n^2).

Schématicky lze výpočetní postup *Quicksortu* popsat takto:

```
procedure quicksort(seznam_hodnot)
if length(seznam_hodnot) <= 1
  return
pivot = náhodný prvek ze seznamu_hodnot

Rozdělení seznam_hodnot do 3 částí
  seznam1 = {prvky menší než pivot}
  seznam2 = {pivot}
  seznam3 = {prvky větší než pivot}
seznam_hodnot = quicksort(seznam1) + seznam2 + quicksort(seznam3)
```

Algoritmus třídění *QuickSort* vychází z tzv. rekurzivního přístupu.

Rekurze je často používaná technika v matematice a informatice. Termín je pravděpodobně odvozen z latinského slovesa *recurso* (vrátit se) nebo *recursus* (návrat, zpětný běh). V programování představuje rekurze opakované volání stejné funkce (funkce vyvolává „sama sebe“), v takovém případě se hovoří o rekurzivní funkci. Nedílnou součástí rekurzivní funkce musí být ukončující podmínka určující, kdy se má vnořování zastavit. Jelikož bývá nejčastějším zdrojem chyb, je třeba ji navrhnout dostatečně robustním způsobem a prověřit veškeré možné situace jejího chodu. (Blíže viz např. [15])

Při menších velikostech problému je *Quicksort* poměrně pomalý, protože velmi pravděpodobně nedojde rozdělení pole na ideální poloviny. Z tohoto důvodu se

Quicksort používá pouze u velkých polí, pro řešení malých polí bývá výrazně rychlejší třídění pomocí *Insertsortu* nebo *Shellsortu* (viz dále).

Jedna z možností, jak rekurzivní způsob řazení pomocí algoritmu Quicksort v prostředí systému MATLAB naprogramovat, může vypadat například takto:

```
function y=quicksort(x)
n=length(x);
if(n<=1);y=x;return;end;
if(n==2)
  if(x(1)>x(2))
     x=[x(2); x(1)];
  end
  y=x;
  return;
end
m=fix(n/2);
pivot=x(m);
Mensi=find(x<pivot);
if(isempty(Mensi))
  ind=find(x>pivot);
  if(isempty(ind));y=x;return;end;
  pivot=x(ind(1));
  Mensi=find(x<pivot);
end
Vetsi=find(x>=pivot);
y=[quicksort(x(Mensi));quicksort(x(Vetsi))];
```

Poznámka 5.3. Uvedený algoritmus předpokládá, že řazený vektor je ve sloupcového tvaru. Z tohoto důvodu je nutné vyvolávat m-funkci s vektorem v příslušném tvaru, např:

```
A=[76 5 44 90 59 63 4 1 28 57];
B=quicksort(A')
```

Poznámka 5.4. V této m-funkci byly použity funkce programu MATLAB `fix` pro operaci celočíselné dělení a funkce `find`, která vrací vektor prvků posloupnosti hodnot x, které splňují podmínku obsaženou ve vstupním parametru. Logická funkce `isempty` testuje obsah proměnné (vektoru) v parametru. V případě, že je proměnná prázdná, výsledkem funkce je logická hodnota PRAVDA (tedy hodnota 1). Příkaz `return` nepřeruší běh výpočtu, jako např. příkaz `break`, pouze ukončí činnost vyvolané funkce (u rekurzivních výpočtů to neznamená konec výpočtu, ale návrat do té části algoritmu, ze které byla funkce vyvolaná).

5.1.5 Shellovo řazení (Shell Sort)

Shellovo řazení (zkráceně *Shellsort*) nebo také řazení se snižujícím se přírůstkem je řadicí algoritmus podobný *Insertsortu*, který objevil a v roce 1959 publikoval Donald Shell. Časová složitost *Shellsortu* je přibližně rovna $n^{\frac{3}{2}}$ a je z algoritmů složitosti typu n^2 nejvýkonnější.

Obecným problémem řadících algoritmů je zatřídění prvků do opačné části pole, než je jejich původní umístění. Tyto prvky musí v běžných kvadratických algoritmech „procestovat“ postupně celé pole. Přístup *Shellsortu* je v tomto ohledu jiný, protože neporovnává sousední prvky, ale v každém svém kroku prvky v určité vzdálenosti (tato vzdálenost se v každém kroku snižuje, až se zmenší na 1). Tímto způsobem se zajistí nejrychlejší způsob, jak prvky zařazené na špatné straně pole přesunout na jejich správnou pozici.

Základním problémem algoritmu je volba ideální vzdálenosti pro porovnávání jednotlivých prvků. Donald Shell původně navrhoval, aby se při porovnávání začínalo s hodnotou mezery $\frac{n}{2}$, kde n je velikost pole, a vzdálenost mezi porovnávanými prvky se tedy vždy půlila. Tento přístup má nevýhodu, protože prvky na sudých a lichých místech se porovnávají až v posledním kroku algoritmu. Dalšími pokusy s volbou mezery byla například volba posloupnosti $2 \cdot k - 1$ (Hibbard) se složitostí $n^{\frac{3}{2}}$ nebo posloupnost $9 \cdot 4^i - 9 \cdot 2^i$ (Sedgewick) se složitostí $n^{\frac{4}{2}}$, případně Fibonacciho posloupnost umocněná na dvojnásobek zlatého řezu. Nejlepší výsledky lze získat s posloupností $1, 4, 10, 23, 57, 132, 301, 701, 1750$ nebo $mezera \cdot 2,2$, jejímž autorem je Marcin Ciura.

Výpočet s využitím algoritmu *Shellsortu* lze naprogramovat v m-funkci např. následujícím způsobem:

```
function y=shellsort(x)
n=length(x);
mezera=floor(n/2);
while mezera>0
  for j=mezera:n
    for i=j-mezera:-mezera:1
      if x(i+mezera)>=x(i)
        break;
      else
        c=x(i);
        x(i)=x(i+mezera);
        x(i+mezera)=c;
      end
    end
  end
  mezera=floor(mezera/2);
end
y=x;
```

Poznámka 5.5. V m-funkci byla použita funkce `floor`, která zaokrouhluje na celé číslo směrem k $-\infty$, tj. např. pro `floor(-0.4)` vychází výsledek -1.

5.2 Práce s textovým souborem

Činnost třídících algoritmů lze doplnit načítáním číselných hodnot z textového souboru do pole vektoru v systému MATLAB, které se pak může setřídit některým z popsaných algoritmů. Naopak zápis dat do textového souboru může sloužit pro ukládání výsledků výpočtu, příp. pro přípravu dat.

Příklad 5.6. Do textového souboru s názvem `exp.txt` uložte tabulku s hodnotami funkce e^x pro $x = 0; 0,1; 0,2; \ldots; 1,0$.

Řešení. Zápis dat do textového souboru lze provádět např. s využitím příkazů systému MATLAB s názvy `fopen`, `fprintf` a `fclose`. Úlohu je pak možno vyřešit např. s pomocí následujícího sledu příkazů:

```
x=0:.1:1;
y=[x;exp(x)];
fid=fopen('exp.txt','w');
fprintf(fid,'%6.2f %12.8f\n',y);
fclose(fid);
type exp.txt
```

Ve funkci `fopen` je přitom nutno specifikovat druhým parametrem mód přístupu k souboru, jehož název obsahuje první parametr. K základním možnostem patří parametr **'r'** pro otevření souboru pro čtení a **'w'** pro zápis (v takovém případě se případně vymaže obsah již existujícího souboru).

Výsledný obsah textového souboru s názvem `exp.txt` se provede příkazem `type`. Výpis by pak měl vypadat následovně:

```
  0.00   1.00000000
  0.10   1.10517092
  0.20   1.22140276
  0.30   1.34985881
  0.40   1.49182470
  0.50   1.64872127
  0.60   1.82211880
  0.70   2.01375271
  0.80   2.22554093
  0.90   2.45960311
  1.00   2.71828183
```

▲

Příklad 5.7. Vytvořte textový soubor s názvem `hodnoty.txt`, do kterého zapíšete 10000 náhodně vygenerovaných čísel s rozsahem v rozmezí 0 až 1000000 pomocí funkce `randi`.

Řešení. Správné vyvolání příkazu pro vygenerování určeného počtu náhodných čísel v předem stanoveném rozsahu hodnot má tvar:

```
randi(1000000,1,10000)
```

Pokud se zkombinuje i se zápisem do textového souboru, pak může být posloupnost povelů např.:

```
x=randi(1000000,1,10000);
fid=fopen('hodnoty.txt','w');
fprintf(fid,'%7g\n',x);
fclose(fid);
type hodnoty.txt
```

▲

Poznámka 5.8. Funkce `randi` není obsažena v programu Octave, kde je nutno použít funkci `unidrnd`.

Příklad 5.9. Načtěte obsah vytvořeného souboru `hodnoty.txt` do proměnné A pomocí příkazu `dlmread`.

Řešení. Příkaz je velice jednoduchý s tvarem:

```
A=dlmread('hodnoty.txt')
```

Soubor hodnoty.txt ovšem musí obsahovat pouze číselné hodnoty. ▲

Poznámka 5.10. Obecněji lze operaci z předchozího příkladu provést následujícím způsobem:

```
clear;
fid=fopen('hodnoty.txt');
k=0;
while feof(fid)==0
    cti=fscanf(fid,'%f',1);
    if ~isempty(cti)
       k=k+1;
       A(k)=cti;
    end
end
fclose(fid);
A'
```

Uvedená m-funkce obsahuje funkci `feof`, která vrací hodnotu PRAVDA, tedy 1, v případě, že se při čtení textového souboru dostal program na jeho konec (v opačném případě vrací hodnotu NEPRAVDA čili 0).

Příklad 5.11. Setřiďte obsah proměnné A z předchozího příkladu pomocí třídících algoritmů, jenž byly vysvětleny v této kapitole.

Poznámka 5.12. Pokud je zapotřebí sledovat strojový čas výpočtu, je možné s výhodou použít dvojici funkcí `tic` a `toc`, např. tímto způsobem:

```
clc
tic;
B=bubblesort(A);
toc
```

Výpis o trvání výpočtu pak může vypadat např.:

```
Elapsed time is 1.790291 seconds.
```

Příklad 5.13. Upravte výpočetních postupů řadících algoritmů tak, aby výsledkem byl seznam sestupně setříděných hodnot.

Poznámka 5.14. Pro kontrolu správnosti práce vytvořených m-funkcí lze využít příkazy systému MATLAB s názvy `sort` (tříd vektory a matice vzestupně i sestupně) a `issorted` (vrací hodnotu PRAVDA, tedy 1, pro setříděný vektor či matici, v opačném případě je výslednou hodnotou NEPRAVDA, čili 0), ve kterých vstupní parametr obsahuje předmětný vektor či matici.

Kapitola 6

Soustavy lineárních rovnic

Cíle

Kapitola je zaměřena na problematiku:

- algoritmů řešení soustav lineárních rovnic přímými metodami,
- iteračních metod řešení soustav lineárních rovnic,
- trojných cyklů typu `for`,
- maticového počtu.

Velké množství úloh stavební mechaniky vede k řešení soustavy lineárních rovnic. Numerické metody pro jejich řešení jsou tedy velmi častým programátorským úkolem.

Obecně může být soustava n lineárních rovnic s n proměnnými zapsána jako:

$$\begin{array}{ccccccccc} a_{1,1}\cdot x_1 & + & a_{1,2}\cdot x_2 & + & \cdots & + & a_{1,n}\cdot x_n & = & b_1 \\ a_{2,1}\cdot x_1 & + & a_{2,2}\cdot x_2 & + & \cdots & + & a_{2,n}\cdot x_n & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n,1}\cdot x_1 & + & a_{n,2}\cdot x_2 & + & \cdots & + & a_{n,n}\cdot x_n & = & b_n \end{array}, \tag{6.1}$$

kde proměnné $x_1, \ldots, x_n$, obecně x_i pro $i = 1, \ldots, n$, jsou neznámé, $a_{i,j}$ pro $i, j = 1, \ldots, n$ jsou koeficienty soustavy rovnic a čísla b_i pro $i = 1, \ldots, n$, jsou absolutní členy soustavy (nebo také tzv. pravá strana soustavy).

K řešení kořenů soustav lineárních rovnic se využívá maticového počtu. Koeficienty soustavy lze zapsat ve tvaru matice:

$$[A] = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}, \tag{6.2}$$

která bývá označována jako matice soustavy.

Neznámé a pravou stranu soustavy je možno vyjádřit jako vektory:

$$\{x\} = \begin{Bmatrix} x_1 & x_2 & \cdots & x_n \end{Bmatrix}^T , \tag{6.3}$$

$$\{b\} = \begin{Bmatrix} b_1 & b_2 & \cdots & b_n \end{Bmatrix}^T . \tag{6.4}$$

Celou soustavu lineárních rovnic pak lze vyjádřit maticově:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{Bmatrix} , \tag{6.5}$$

nebo zkráceně v maticovém tvaru:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} , \tag{6.6}$$

kde **A** značí matici soustavy (levých stran rovnic), **x** sloupcový vektor neznámých kořenů soustavy a **b** sloupcový vektor pravých stran rovnic.

Jednou z podmínek jednoznačného řešení soustavy lineárních rovnic je skutečnost, že matice **A** musí být **regulární**.

Poznámka 6.1. Regulární matice je čtvercová matice, jejíž determinant je různý od nuly. Opakem regulární matice je tzv. **singulární matice** s nulovým determinantem. Důležitou vlastností regulární matice je možnost vypočítat jednoznačně inverzní matici. Toho lze využít např při řešení soustavy lineárních rovnic.

6.1 Přímé metody řešení soustav lineárních rovnic

Metody pro řešení soustav lineárních rovnic, které vedou k přesnému řešení (pokud se neberou v úvahu chyby numerického řešení) při konečném počtu výpočetních kroků, se označují jako *metody přímé*. Jejich základním rysem je eliminace neznámých. Pro plné matice bývají tyto metody nejefektivnější, při velkém počtu rovnic však může být výpočet omezen pamětí počítače.

6.1.1 Řešení trojúhelníkové soustavy lineárních rovnic

Obecnou horní trojúhelníkovou soustavu lineárních rovnic lze zapsat ve tvaru:

$$\begin{array}{ccccccccc} a_{1,1} \cdot x_1 & + & a_{1,2} \cdot x_2 & + & \cdots & + & a_{1,n} \cdot x_n & = & b_1 \\ & & a_{2,2} \cdot x_2 & + & \cdots & + & a_{2,n} \cdot x_n & = & b_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & a_{n,n} \cdot x_n & = & b_n \end{array} , \tag{6.7}$$

nebo maticově

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ & a_{2,2} & \cdots & a_{2,n} \\ & & \ddots & \vdots \\ & & & a_{n,n} \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{Bmatrix}, \tag{6.8}$$

Řešení, označované jako zpětná substituce (zpětný chod), je možno popsat algoritmem 11.

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

for $i \leftarrow n, n-1, \ldots, 2, 1$ **do**

$$x_i \leftarrow \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j}{a_{i,i}}$$

end

Algoritmus 11: Algoritmus zpětné substituce

Výpočet horní trojúhelníkové soustavy lineárních rovnic lze v programu MATLAB naprogramovat například takto:

```
n=input('Zadejte pocet neznamych v soustave rovnic:\n n=');
A=zeros(n,n);
fprintf('\n Zadejte prvky matice soustavy A:');
for i=1:n
  for j=i:n
    fprintf('\n A[%d,%d]=',i,j)
    A(i,j)=input('');
  end
end
if det(A)==0
  error('Soustava rovnic je singulární! Det(A) je roven 0!')
end
fprintf('\n Zadejte prvky vektoru pravych stran b:');
for i=1:n
    fprintf('\n b[%d]=',i)
    b(i)=input('');
end
if n==1
  x(1)=b(1)/A(1,1);
else
  for i=n:-1:1
    s=0;
```

```
    if i<n
      for j=i+1:n
        s=s+A(i,j)*x(j);
      end
    end
    x(i)=(b(i)-s)/A(i,i);
  end
end
fprintf('\n')
disp('Kořeny soustavy')
disp('--------------------------------')
for i=1:n
  fprintf('x[%d]=%16.8f\n',i,x(i))
end
```

Poznámka 6.2. V ukázce je pro zadání vstupních údajů použito příkazu `input`, který umožňuje výpis popisu zadávané veličiny na obrazovku a přiřazení hodnoty do proměnné zadáním přímo z klávesnice.

Příklad 6.3. Určete kořeny trojúhelníkové soustavy lineárních rovnic řádu 4:

$$\begin{array}{rcrcrcrcl} x_1 & + & 2\cdot x_2 & + & 3\cdot x_3 & + & 4\cdot x_4 & = & 2 \\ & & 2\cdot x_2 & + & 6\cdot x_3 & + & 12\cdot x_4 & = & 8 \\ & & & & 6\cdot x_3 & + & 24\cdot x_4 & = & 18 \\ & & & & & & 24\cdot x_4 & = & 24 \end{array} \tag{6.9}$$

Řešení. Výsledný vektor neznámých kořenů má tvar $x = \{-1 \quad 1 \quad -1 \quad 1\}^T$. ▲

Poznámka 6.4. Kontrolu správnosti řešení lze provést např. opětovným dosazením kořenů do jednotlivých rovnic soustavy nebo lépe odečtením levé strany soustavy od pravé, čímž je možno získat tzv. *reziduální vektor* řešení **r**, např. následujícím způsobem:

```
fprintf('\n')
disp('Reziduální vektor')
disp('--------------------------------')
for i=1:n
  r(i)=0;
  for j=i:n
    r(i)=r(i)+A(i,j)*x(j);
  end
  r(i)=r(i)-b(i);
  fprintf('r[%d]=%16.8f\n',i,r(i))
end
```

Jednotlivé prvky by se měly blížit k nule. Pro trojúhelníkovou soustavu vychází reziduální vektor:

```
r[1] = 0.00000000
r[2] = 0.00000000
r[3] = 0.00000000
r[4] = 0.00000000
```

Poznámka 6.5. Kontrolu přesnosti řešení lze provést rovněž pomocí *euklidovské normy reziduálního vektoru* **r**, danou výrazem $\sqrt{\sum_i |r_i|^2}$, kterou lze v programu MATLAB vyvolat např. příkazem `norm(A*x-b)`.

Poznámka 6.6. Při realizaci algoritmu vzniknou potíže, pokud čísla na diagonále, tj. $a_{i,i}$, budou malá. Matice soustavy **A** pak může být téměř singulární ($\det \mathbf{A} \approx 0$). Řešením je přeuspořádání soustavy lineárních rovnic tak, aby na diagonále byly největší prvky matice soustavy **A** .

Příklady k procvičení

1. Navrhněte algoritmus pro řešení obecné trojúhelníkové soustavy lineárních rovnic, která má dolní matici soustavy **A** (nuly nad diagonálou).

6.1.2 Gaussova eliminační metoda

Gaussova eliminace je jednou z nejstarších numerických metod, při které se matice soustavy **A** převádí na horní trojúhelníkovou matici. Řádkovými úpravami s využitím tzv. *multiplikátorů* se tato matice upraví do tvaru, kdy se pod hlavní diagonálou nachází pouze nuly. Upravená matice pak odpovídá soustavě rovnic, která je ekvivalentní s původní soustavou, a lze ji řešit podobně jako trojúhelníkovou soustavu lineárních rovnic s pomocí tzv. zpětné substituce (zpětného chodu). Celý výpočetní postup lze schématicky popsat algoritmem 12.

Samotný výpis m-funkce v programu MATLAB může nabývat tvaru:

```
function x=gauss(A,b)
if det(A)==0
  error('Soustava rovnic je singulární! Det(A) je roven 0!')
  return
end
n=length(A);
if n==1
  x(1)=b(1)/A(1,1);
  return
end
```

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

for $k \leftarrow 1, 2, \ldots, n-2, n-1$ **do**
  for $i \leftarrow k+1, k+2, \ldots, n-1, n$ **do**
    $m \leftarrow -\frac{a_{i,k}}{a_{k,k}}$
    for $j \leftarrow k, k+1, \ldots, n-1, n$ **do**
      $a_{i,j} \leftarrow a_{i,j} + m \cdot a_{k,j}$
    end
    $b_i \leftarrow b_i + m \cdot b_k$
  end
end
for $i \leftarrow n, n-1, \ldots, 2, 1$ **do**
  $x_i \leftarrow \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j}{a_{i,i}}$
end

Algoritmus 12: Algoritmus Gaussovy eliminace

```
for k=1:n-1
  if A(k,k)==0
    error('Na diagonále je nula!')
    return
  end
  for i=k+1:n
    m=-A(i,k)/A(k,k);
    for j=k:n
      A(i,j)=A(i,j)+m*A(k,j);
    end;
    b(i)=b(i)+m*b(k);
  end
end
for i=n:-1:1
  s=0;
  if i<n
    for j=i+1:n
      s=s+A(i,j)*x(j);
    end
  end
  x(i)=(b(i)-s)/A(i,i);
end
```

Příklad 6.7. S využitím vytvořené m-funkce vyřešte kořeny soustavy 4 lineárních rovnic:

$$\begin{array}{rcrcrcrcl} 2\cdot x_1 & - & x_2 & + & 3\cdot x_3 & - & x_4 & = & 7 \\ x_1 & - & x_2 & + & 4\cdot x_3 & - & 2\cdot x_4 & = & 5 \\ 3\cdot x_1 & + & 2\cdot x_2 & + & x_3 & + & 4\cdot x_4 & = & 31 \\ 4\cdot x_1 & - & 3\cdot x_2 & + & 3\cdot x_3 & - & 3\cdot x_4 & = & -5 \end{array} \tag{6.10}$$

Řešení. Řešením je vektor neznámých kořenů $\{x\} = \begin{Bmatrix} 1 & 2 & 4 & 5 \end{Bmatrix}^T$. V tomto příkladě lze demonstrovat chod algoritmu Gaussovy eliminace:

```
Původní matice A              Původní vektor b
---------------------------   ---------------------------
  2.000 -1.000  3.000 -1.000    7.000
  1.000 -1.000  4.000 -2.000    5.000
  3.000  2.000  1.000  4.000   31.000
  4.000 -3.000  3.000 -3.000   -5.000

Upravená matice A             Upravený vektor b
---------------------------   ---------------------------
  2.000 -1.000  3.000 -1.000    7.000
  0.000 -0.500  2.500 -1.500    1.500
  0.000  0.000 14.000 -5.000   31.000
  0.000  0.000  0.000 -0.857   -4.286

Kořeny soustavy
-------------------
x[ 1] =    1.000
x[ 2] =    2.000
x[ 3] =    4.000
x[ 4] =    5.000

Reziduální vektor
-------------------
r[ 1] = 0.000e+000
r[ 2] = 0.000e+000
r[ 3] = -7.105e-015
r[ 4] = -1.776e-015
```

▲

Příklad 6.8. Určete kořeny soustavy 3 lineárních rovnic s maticí soustavy

$$[A] = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \tag{6.11}$$

a vektorem pravých stran:

$$\{b\} = \begin{Bmatrix} 1 & 4 & 1 \end{Bmatrix}^T . \tag{6.12}$$

Řešení. Vektor neznámých kořenů je roven $\{x\} = \begin{Bmatrix} 3 & -5 & 3 \end{Bmatrix}^T$. ▲

Příklad 6.9. Vyřešte soustavu 4 lineárních rovnic, danou maticí soustavy

$$[A] = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \tag{6.13}$$

a vektorem pravých stran:

$$\{b\} = \begin{Bmatrix} 1 & 2 & 0 & 1 \end{Bmatrix}^T . \tag{6.14}$$

Řešení. Výsledný vektor neznámých kořenů je $\{x\} = \begin{Bmatrix} 0.5 & 0.75 & 0.25 & 0.5 \end{Bmatrix}^T$. ▲

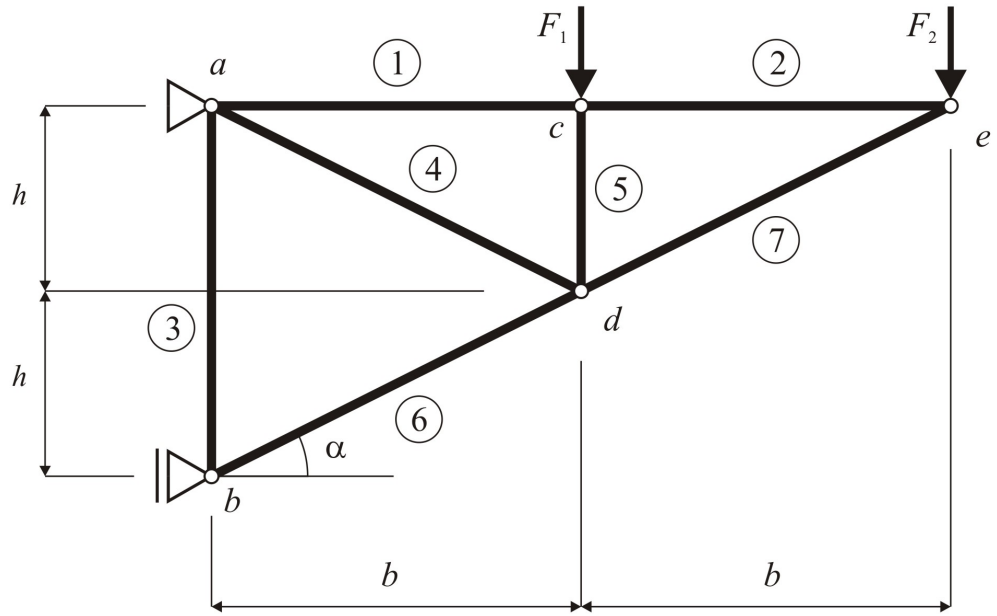
Příklad 6.10. Stanovte strojový čas a přesnost řešení (normu reziduálního vektoru) náhodně vygenerované soustavy 600 lineárních rovnic.

Řešení. Příklad lze vyřešit např. s využitím následujícího sledu příkazů:

```
clc;
clear;
n=600;
m=1200;
A=randn(n,m);
A=A*A';
b=randn(n,1);
tic, x=gauss(A,b); toc
norm(A*x'-b)
```

▲

Příklad 6.11. Vyřešte reakce a vnitřní síly u příhradové konstrukce z obr. 6.1 pomocí obecné styčníkové metody. Vstupní parametry jsou $b = 3$ m, $h = 1,5$ m, $F_1 = 5$ kN a $F_2 = 12$ kN. Soustavu lineárních rovnic pak vyřešte Gaussovou eliminací.



Obr. 6.1 Statické schéma řešené příhradové konstrukce

Řešení. Pokud se příhradová konstrukce z obr. 6.1 interpretuje jako soustava hmotných bodů, z kinematického hlediska obsahuje $2 \cdot s$ stupňů volnosti, kde s je počet hmotných bodů, tedy styčníků. Jelikož je konstrukce tvořena 5 styčníky ($s = 5$), vychází pak $n_v = 2 \cdot s = 10$ stupňů volnosti, které jsou odebrány 3 vnějšími vazbami ($v_\mathrm{e} = 3$) a 7 vnitřními ($v_i = 7$) (počet prutů). Vzhledem ke skutečnosti, že $n_v = v_\mathrm{e} + v_i$, jedná se o konstrukci staticky i kinematicky určitou.

Pokud se v každém ze styčníků stanoví 2 podmínky rovnováhy, lze získat celkem 10 podmínek rovnováhy, tvořících soustavu lineárních rovnic, ze kterých lze stanovit 10 neznámých - 3 reakce ($R_{ax}, R_{az}aR_{bx}$) a 7 vnitřních sil ($N_1, N_1, \ldots, N_7$).

Jednotlivé podmínky rovnováhy nabývají tvaru:

- Styčník a:
  1. $R_x = 0 : -R_{ax} + N_1 + N_4 \cdot \cos(\alpha) = 0$
  2. $R_z = 0 : -R_{az} + N_3 + N_4 \cdot \sin(\alpha) = 0$
- Styčník b:
  3. $R_x = 0 : +R_{bx} + N_6 \cdot \cos(\alpha) = 0$
  4. $R_z = 0 : -N_3 - N_6 \cdot \sin(\alpha) = 0$
- Styčník c:
  5. $R_x = 0 : -N_1 + N_2 = 0$
  6. $R_z = 0 : +F_1 + N_5 = 0$

- Styčník d:
  7. $R_x = 0 : -N_4 \cdot \cos(\alpha) - N_6 \cdot \cos(\alpha) + N_7 \cdot \cos(\alpha) = 0$
  8. $R_z = 0 : -N_4 \cdot \sin(\alpha) - N_5 + N_6 \cdot \sin(\alpha) - N_7 \cdot \sin(\alpha) = 0$
- Styčník e:
  9. $R_x = 0 : -N_2 - N_7 \cdot \cos(\alpha) = 0$
  10. $R_z = 0 : +F_2 + N_7 \cdot \sin(\alpha) = 0$

Celou soustavu lineárních rovnic řádu 10 lze přehledně zapsat maticově:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \,, \tag{6.15}$$

kde $\mathbf{A}$ značí matici levých stran rovnic, obsahující údaje geometrie konstrukce:

$$\begin{bmatrix}
-1 & 0 & 0 & +1 & 0 & 0 & +\cos(\alpha) & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & +1 & +\sin(\alpha) & 0 & 0 & 0 \\
0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & +\cos(\alpha) & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -\sin(\alpha) & 0 \\
0 & 0 & 0 & -1 & +1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\cos(\alpha) & 0 & -\cos(\alpha) & +\cos(\alpha) \\
0 & 0 & 0 & 0 & 0 & 0 & -\sin(\alpha) & -1 & +\sin(\alpha) & -\sin(\alpha) \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -\cos(\alpha) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +\sin(\alpha)
\end{bmatrix} , \tag{6.16}$$

$\mathbf{x}$ představuje sloupcový vektor neznámých kořenů, obsahující 10 neznámých reakcí a vnitřních sil:

$$\begin{Bmatrix} R_{ax} & R_{az} & R_{bz} & N_1 & N_2 & N_3 & N_4 & N_5 & N_6 & N_7 \end{Bmatrix}^T \tag{6.17}$$

a $\mathbf{b}$ vyjadřuje sloupcový vektor pravých stran rovnic, obsahující uzlové zatížení příhradové konstrukce:

$$\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & -F_1 & 0 & 0 & 0 & -F_2 \end{Bmatrix}^T \,. \tag{6.18}$$

Goniometrické funkce $\cos(\alpha)$ a $\sin(\alpha)$, obsažené v matici $\mathbf{A}$, lze vyjádřit přímo z rozměru konstrukce:

$$\cos(\alpha) = \frac{b}{l} \,, \quad \sin(\alpha) = \frac{h}{l} \,, \tag{6.19}$$

kde l je délka prutů č.4, 6 a 7:

$$l = l_4 = l_6 = l_7 = \sqrt{b^2 + h^2} \,. \tag{6.20}$$

Vzhledem k podmínce řešení Gaussovy metody, že prvky na diagonále matice $\mathbf{A}$ se nesmí rovnat 0, matici $\mathbf{A}$ i vektor pravých stran $\mathbf{b}$ je nutno upravit vhodným přeuspořádáním pořadí styčníkových rovnic, a to:

- na 4.řádku bude původně 5. styčníková rovnice,
- na 5.řádku bude původně 9. styčníková rovnice,
- na 6.řádku bude původně 4. styčníková rovnice,
- na 8.řádku bude původně 6. styčníková rovnice,
- na 9.řádku bude původně 8. styčníková rovnice,

takže pozměněná matice levých stran **A** bude mít výsledný tvar:

$$\begin{bmatrix} -1 & 0 & 0 & +1 & 0 & 0 & +\cos(\alpha) & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & +1 & +\sin(\alpha) & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & +\cos(\alpha) & 0 \\ 0 & 0 & 0 & -1 & +1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -\cos(\alpha) \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -\sin(\alpha) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\cos(\alpha) & 0 & -\cos(\alpha) & +\cos(\alpha) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\sin(\alpha) & -1 & +\sin(\alpha) & -\sin(\alpha) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +\sin(\alpha) \end{bmatrix}, \tag{6.21}$$

a sloupcový vektor pravých stran rovnic **b**:

$$\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -F_1 & 0 & -F_2 \end{Bmatrix}^T . \tag{6.22}$$

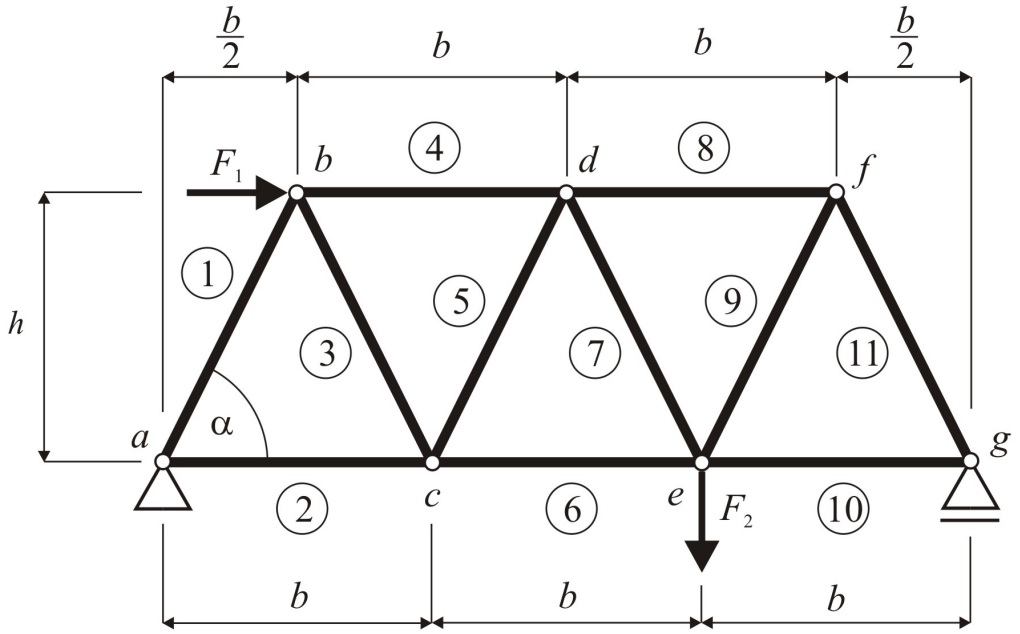
Sloupcový vektor neznámých kořenů **x** zůstane nezměněn.

Takto sestavenou soustavu lineárních rovnic lze vyřešit pro konkrétně zadané vstupní veličiny s pomocí Gaussovy eliminační metody:

```
Kořeny soustavy
-------------------
x[ 1] =   29.000 kN
x[ 2] =   17.000 kN
x[ 3] =   29.000 kN
x[ 4] =   24.000 kN
x[ 5] =   24.000 kN
x[ 6] =   14.500 kN
x[ 7] =    5.590 kN
x[ 8] =   -5.000 kN
x[ 9] =  -32.423 kN
x[10] =  -26.833 kN

Norma reziduálního vektoru:
---------------------------
n =
     0
```

▲

Příklad 6.12. Vyřešte reakce a vnitřní síly u příhradové konstrukce z obr. 6.2 pomocí obecné styčníkové metody. Vstupní parametry jsou $b = 4$ m, $h = 4$ m, $F_1 = 8$ kN a $F_2 = 14$ kN. Soustavu lineárních rovnic pak vyřešte Gaussovou eliminační metodou.



Obr. 6.2 Statické schéma řešené příhradové konstrukce

Řešení. Pokud se příhradová konstrukce z obr. 6.2 interpretuje podobně jako v příkladě 6.11, tedy jako soustava hmotných bodů, z kinematického hlediska obsahuje $2 \cdot s$ stupňů volnosti, kde s je počet hmotných bodů, tedy styčníků. Jelikož je konstrukce tvořena 7 styčníky ($s = 7$), vychází pak $n_v = 2 \cdot s = 14$ stupňů volnosti, které jsou odebrány 3 vnějšími vazbami ($v_\mathrm{e} = 3$) a 11 vnitřními ($v_i = 11$) (počet prutů). Vzhledem ke skutečnosti, že $n_v = v_\mathrm{e} + v_i$, jedná se o konstrukci staticky i kinematicky určitou a lze ji řešit pouze s využitím podmínek rovnováhy.

Pokud se v každém ze styčníků stanoví 2 podmínky rovnováhy, lze získat celkem 14 podmínek rovnováhy, tvořících soustavu lineárních rovnic, ze kterých lze stanovit 14 neznámých - 3 reakce ($R_{ax}, R_{az}aR_{gz}$) a 11 vnitřních sil ($N_1, N_1, \ldots, N_{11}$).

Jednotlivé podmínky rovnováhy nabývají tvaru:

- Styčník a:
  1. $R_x = 0 : -R_{ax} + N_1 \cdot \cos(\alpha) + N_2 = 0$
  2. $R_z = 0 : -R_{az} - N_1 \cdot \sin(\alpha) = 0$

- Styčník b:

  3. $R_x = 0 : +F_1 - N_1 \cdot \cos(\alpha) + N_3 \cdot \cos(\alpha) + N_4 = 0$
  4. $R_z = 0 : +N_1 \cdot \sin(\alpha) + N_3 \cdot \sin(\alpha) = 0$

- Styčník c:

  5. $R_x = 0 : -N_2 - N_3 \cdot \cos(\alpha) + N_5 \cdot \cos(\alpha) + N_6 = 0$
  6. $R_z = 0 : +N_3 \cdot \sin(\alpha) + N_5 \cdot \sin(\alpha) = 0$

- Styčník d:

  7. $R_x = 0 : -N_4 \cdot \cos(\alpha) - N_5 \cdot \cos(\alpha) + N_7 \cdot \cos(\alpha) + N_8 = 0$
  8. $R_z = 0 : +N_5 \cdot \sin(\alpha) + N_7 \cdot \sin(\alpha) = 0$

- Styčník e:

  9. $R_x = 0 : -N_8 - N_9 \cdot \cos(\alpha) + N_{11} \cdot \cos(\alpha) + N_{12} = 0$
  10. $R_z = 0 : +N_9 \cdot \sin(\alpha) + N_{11} \cdot \sin(\alpha) = 0$

- Styčník f:

  11. $R_x = 0 : -N_{10} - N_{11} \cdot \cos(\alpha) + N_{13} \cdot \cos(\alpha) = 0$
  12. $R_z = 0 : +N_{11} \cdot \sin(\alpha) + N_{13} \cdot \sin(\alpha) = 0$

- Styčník g:

  13. $R_x = 0 : -N_{12} - N_{13} \cdot \cos(\alpha) = 0$
  14. $R_z = 0 : -N_{13} \cdot \sin(\alpha) - R_{gz} = 0$

Celou soustavu lineárních rovnic řádu 14 lze podobně jako v příkladu 6.11 přehledně zapsat maticově:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \,, \tag{6.23}$$

kde $\mathbf{A}$ značí matici levých stran rovnic, obsahující údaje geometrie konstrukce, $\mathbf{x}$ představuje sloupcový vektor neznámých kořenů, obsahující 14 neznámých reakcí a vnitřních sil (zvoleno pořadí $\mathbf{x} = \left\{R_{ax} \quad R_{az} \quad N_1 \quad N_2 \quad \dots \quad N_{11} \quad R_{gz}\right\}^T$), a $\mathbf{b}$ vyjadřuje sloupcový vektor pravých stran rovnic, obsahující uzlové zatížení příhradové konstrukce.

Na některých řádcích matice $\mathbf{A}$ se vyskytuje nula, a proto je třeba provést vzájemné přeuspořádání pořadí styčníkových rovnic:

- na 4.řádku bude původně 5. styčníková rovnice,
- na 6.řádku bude původně 7. styčníková rovnice,
- na 8.řádku bude původně 9. styčníková rovnice,
- na 10.řádku bude původně 11. styčníková rovnice,
- na 12.řádku bude původně 13. styčníková rovnice,

Výpis výsledných hodnot kořenů soustavy pak může vypadat následovně:

```
Kořeny soustavy
-------------------
x[ 1] =    8.000 kN
x[ 2] =    2.000 kN
x[ 3] =   -2.236 kN
x[ 4] =    9.000 kN
x[ 5] =    2.236 kN
x[ 6] =  -10.000 kN
x[ 7] =   -2.236 kN
x[ 8] =   11.000 kN
x[ 9] =    2.236 kN
x[10] =  -12.000 kN
x[11] =   13.416 kN
x[12] =    6.000 kN
x[13] =  -13.416 kN
x[14] =   12.000 kN

Reziduální vektor:
-------------------
r[ 1] = 0.000e+000
r[ 2] = 0.000e+000
r[ 3] = 0.000e+000
r[ 4] = 0.000e+000
r[ 5] = 0.000e+000
r[ 6] = 0.000e+000
r[ 7] = 0.000e+000
r[ 8] = -8.882e-016
r[ 9] = -6.661e-016
r[10] = 8.882e-016
r[11] = 1.776e-015
r[12] = 0.000e+000
r[13] = -1.776e-015
r[14] = -1.776e-015

Norma reziduálního vektoru:
---------------------------
n =
  3.3894e-015
```

▲

Poznámka 6.13. Matice soustavy **A** z příkladu 6.12 je vzhledem k vhodnému označení uzlů, a tedy i vhodném sestavení rovnic soustavy, pásová (blíže viz kap. 6.2.3). V takto vzniklé matici soustavy jsou nenulové prvky od diagonály vzdáleny maxi-

málně o dva sloupce vlevo a o čtyři sloupce vpravo (šířka pásu je tedy 7). Pokuste se upravit algoritmus Gaussovy eliminační metody tak, aby jste zohlednili šířku pásu a snížili tak počet výpočetních operací.

6.1.3 Gauss-Jordanova metoda

Gaussovu eliminační metodu lze jednoduše modifikovat způsobem, který je označován jako Gauss-Jordanova metoda. Základní myšlenkou tohoto výpočetního postupu je úprava matice soustavy **A** na diagonální nebo dokonce jednotkovou matici. Výpočetní postup je schématicky popsán algoritmem 13 a lze jej implementovat do m-funkce programu MATLAB např. následujícím způsobem:

```
function x=gauss_jordan(A,b)
if det(A)==0
  error('Soustava rovnic je singulární! Det(A) je roven 0!')
  return
end
n=length(A);
if n==1
  x(1)=b(1)/A(1,1);
  return
end
for k=1:n
  if A(k,k)==0
    error('Na diagonále je nula!')
    return
  end
  for i=1:n
    if ~(i==k)
      m=-A(i,k)/A(k,k);
      for j=k:n
        A(i,j)=A(i,j)+m*A(k,j);
      end;
      b(i)=b(i)+m*b(k);
    end
  end
end
for i=1:n
  x(i)=b(i)/A(i,i);
end
```

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

for $k \leftarrow 1, 2, \ldots, n-2, n$ **do**
  for $i \leftarrow 1, 2, \ldots, k-1, k+1, \ldots, n$ **do**
    $m \leftarrow -\frac{a_{i,k}}{a_{k,k}}$
    for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
      $a_{i,j} \leftarrow a_{i,j} + m \cdot a_{k,j}$
    end
    $b_i \leftarrow b_i + m \cdot b_k$
  end
end
for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
  $x_i \leftarrow \frac{b_i}{a_{i,i}}$
end

Algoritmus 13: Algoritmus Gauss-Jordanovy metody

Gauss-Jordanovu metodu je možno použít i k řešení tzv. maticových rovnic, jenž lze zkráceně vyjádřit:

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B}\,, \tag{6.24}$$

kde $\mathbf{X}$ představuje matici kořenů soustavy a $\mathbf{B}$ matici levých stran, obě s obecným rozměrem $[n, r]$. Výpočetní postup je tedy nutno upravit tak, aby pracoval s více pravými stranami, jak je např. schématicky popsánu v upraveném algoritmu 14.

Příklad 6.14. Pomocí Gauss-Jordanovy metody vypočtěte maticovou rovnici:

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}^T = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 2 & 1 \\ 3 & -1 & 2 \end{bmatrix}^T . \tag{6.25}$$

Řešení. Řešením je matice:

$$[X] = \begin{bmatrix} 3 & -5 & 3 \\ 2 & -2 & 1 \\ -2 & 7 & -3 \end{bmatrix}^T . \tag{6.26}$$

▲

Řešení maticových rovnic lze využít např. k řešení **inverzní matice**, kdy je matice pravých stran tvořena jednotkovou diagonální maticí„ nebo pro výpočet příhradové konstrukce z příkladu 6.11 s více zatěžovacími stavy.

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{B} = [b_{i,j}] = [b_{1,1}, \ldots, b_{n,r}]$
Výstup: $\mathbf{X} = [x_{i,j}] = [x_{1,1}, \ldots, x_{n,r}]$

for $k \leftarrow 1, 2, \ldots, n-2, n$ **do**
  for $i \leftarrow 1, 2, \ldots, k-1, k+1, \ldots, n$ **do**
    $m \leftarrow -\frac{a_{i,k}}{a_{k,k}}$
    for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
      $a_{i,j} \leftarrow a_{i,j} + m \cdot a_{k,j}$
    end
    for $j \leftarrow 1, 2, \ldots, r-1, r$ **do**
      $b_{i,j} \leftarrow b_{i,j} + m \cdot b_{k,j}$
    end
  end
end
for $j \leftarrow 1, 2, \ldots, r-1, r$ **do**
  for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
    $x_{i,j} \leftarrow \frac{b_{i,j}}{a_{i,i}}$
  end
end

Algoritmus 14: Algoritmus Gauss-Jordanovy metody pro řešení tzv. maticových rovnic

Příklad 6.15. Pomocí Gauss-Jordanovy metody vypočtěte inverzní matici k matici:

$$\begin{bmatrix} 1 & 2 & 6 \\ 2 & 5 & 15 \\ 6 & 15 & 46 \end{bmatrix} \tag{6.27}$$

Řešení. Řešením je inverzní matice:

$$[A]^{-1} = \begin{bmatrix} 5 & -2 & 0 \\ -2 & 10 & -3 \\ 0 & -3 & 1 \end{bmatrix}, \tag{6.28}$$

o čemž se lze přesvědčit např. výpisem:

```
A =                    B =                    A*B =
     1     2     6          5    -2     0          1     0     0
     2     5    15         -2    10    -3          0     1     0
     6    15    46          0    -3     1          0     0     1
```

▲

Příklady k procvičení

1. Určete reakce a vnitřní síly příhradové konstrukce z příkladu z příkladu 6.11 rovněž pro zatěžovací stav, tvořený dvojicí svislých uzlových zatížení $F = 10$ kN ve styčnících d a e.

6.1.4 LU rozklad

Princip metody LU rozkladu je založen na principu, kdy regulární matici $\mathbf{A}$ lze rozložit na součin dvou trojúhelníkových matic $\mathbf{L}$ a $\mathbf{U}$ tak, že platí:

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} \,. \tag{6.29}$$

Soustava lineárních rovnic se pak řeší ve dvou výpočetních krocích, kdy v prvním se vyřeší trojúhelníková soustava lineárních rovnic:

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \,, \tag{6.30}$$

obdobně pak

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \,. \tag{6.31}$$

V dolní trojúhelníkové matici $\mathbf{L}$ se přitom volí na diagonále hodnota 1. Matice $\mathbf{U}$ je horní trojúhelníková. Výhodou metody LU rozkladu je její použití u úloh s více soustavami lineárních rovnic se stejnou maticí soustavy $\mathbf{A}$, která se rozloží pouze jednou a dále se již opakovaně řeší pouze trojúhelníkové soustavy.

Výpočetní postup řešení soustavy lineárních rovnic LU rozkladem je schématicky vyjádřen v algoritmu 15.

Výpočetní algoritmus metody LU rozkladu je možno implementovat do m-funkce programu MATLAB např. následujícím způsobem:

```
function x=lu_rozklad(A,b)
if det(A)==0
  error('Soustava rovnic je singulární! Det(A) je roven 0!')
  return
end
n=length(A);
if n==1
  x(1)=b(1)/A(1,1);
  return
end
L=eye(n,n);
U=zeros(n,n);
```

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
  for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
    $u_{i,j} \leftarrow 0$
  end
end

for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
  for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
    if $i = j$ **then**
      $l_{i,j} \leftarrow 1$
    else
      $l_{i,j} \leftarrow 0$
    end
  end
end

for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
  for $i \leftarrow 1, 2, \ldots, j-1, j$ **do**
    $u_{i,j} \leftarrow a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} \cdot u_{k,j}$
  end
  for $i \leftarrow j+1, j+2, \ldots, n-1, n$ **do**
    $l_{i,j} \leftarrow \dfrac{a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} \cdot u_{k,j}}{u_{j,j}}$
  end
end

for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
  $y_i \leftarrow b_i - \sum_{j=1}^{i-1} l_{i,j} \cdot y_j$
end

for $i \leftarrow n, n-1, \ldots, 2, 1$ **do**
  $x_i \leftarrow \dfrac{y_i - \sum_{j=i+1}^{n} u_{i,j} \cdot x_j}{u_{i,i}}$
end

Algoritmus 15: Algoritmus metody LU rozkladu

```
for j=1:n
  for i=1:j
    s=0;
      if i>1
      for k=1:i-1
        s=s+L(i,k)*U(k,j);
      end
    end
    U(i,j)=A(i,j)-s;
  end
  if U(j,j)==0
    error('Na diagonále je nula!')
    return
  end
  for i=j+1:n
    s=0;
    if j>1
      for k=1:j-1
        s=s+L(i,k)*U(k,j);
      end
    end
    L(i,j)=(A(i,j)-s)/U(j,j);
  end
end
for i=1:n
  s=0;
  if i>1
    for j=1:i-1
      s=s+L(i,j)*y(j);
    end
  end
  y(i)=(b(i)-s);
end
for i=n:-1:1
  s=0;
  if i<n
    for j=i+1:n
      s=s+U(i,j)*x(j);
    end
  end
  if U(i,i)==0
    error('Na diagonále je nula!')
    return
```

```
  end
  x(i)=(y(i)-s)/U(i,i);
end
```

Příklad 6.16. Pomocí metody LU rozkladu vypočtěte kořeny soustavy lineárních rovnic z příkladu 6.7.

Řešení. Řešení lze demonstrovat podrobným výpisem průběžných i konečných výsledků:

```
Matice soustavy A             Vektor pravých stran b
---------------------------   ---------------------------
  2.000 -1.000  3.000 -1.000    7.000
  1.000 -1.000  4.000 -2.000    5.000
  3.000  2.000  1.000  4.000   31.000
  4.000 -3.000  3.000 -3.000   -5.000


Matice L                      Matice U
---------------------------   ---------------------------
  1.000  0.000  0.000  0.000    2.000 -1.000  3.000 -1.000
  0.500  1.000  0.000  0.000    0.000 -0.500  2.500 -1.500
  1.500 -7.000  1.000  0.000    0.000  0.000 14.000 -5.000
  2.000  2.000 -0.571  1.000    0.000  0.000  0.000 -0.857

Matice L*U
---------------------------
  2.000 -1.000  3.000 -1.000
  1.000 -1.000  4.000 -2.000
  3.000  2.000  1.000  4.000
  4.000 -3.000  3.000 -3.000

Kořeny soustavy
---------------------------
x[ 1] = 1.000e+000
x[ 2] = 2.000e+000
x[ 3] = 4.000e+000
x[ 4] = 5.000e+000

Norma reziduálního vektoru:
---------------------------
n = 7.324106877635580e-015
```

▲

6.1.5 Choleského metoda (dekompozice)

Choleského metoda (také Choleského dekompozice nebo rozklad) je modifikace metody LU rozkladu pro řešení soustav lineárních rovnic se symetrickou regulární pozitivně definitní čtvercovou maticí soustavy $\mathbf{A}$, která se upraví na součin dolní a horní trojúhelníkové matice, přičemž jedna trojúhelníková matice je transpozicí matice druhé:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{U}^T \,. \tag{6.32}$$

Dolní trojúhelníková matice $\mathbf{U}$ z tohoto rozkladu se nazývá Choleského trojúhelník matice $\mathbf{A}$.

Poznámka 6.17. Pozitivně definitní matice je symetrická čtvercová matice, jejíž vlastní čísla jsou větší než nula. Musí platit:

$$\mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x}^T > 0 \,. \tag{6.33}$$

Výpočetní postup řešení soustavy lineárních rovnic Choleského metodou obsahuje v porovnání s metodou LU rozkladu zhruba poloviční počet výpočetních operací a je schématicky popsán algoritmem 16.

Příklad 6.18. Proveďte Choleského dekompozici matice:

$$[A] = \begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 0 & 0 & 0 \\ -1 & 0 & 3 & 1 & 1 \\ -1 & 0 & 1 & 4 & 2 \\ -1 & 0 & 1 & 2 & 5 \end{bmatrix} \,. \tag{6.34}$$

Řešení. Řešením je matice:

$$[U] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \,, \tag{6.35}$$

o čemž se lze přesvědčit:

```
A =                                   U =
     1    -1    -1    -1    -1            1     0     0     0     0
    -1     2     0     0     0           -1     1     0     0     0
    -1     0     3     1     1           -1    -1     1     0     0
    -1     0     1     4     2           -1    -1    -1     1     0
    -1     0     1     2     5           -1    -1    -1    -1     1
```

```
U*U' =
     1    -1    -1    -1    -1
    -1     2     0     0     0
    -1     0     3     1     1
    -1     0     1     4     2
    -1     0     1     2     5
```

▲

Vstup : n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**
  for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
    if $i = j$ **then**
      $u_{i,j} \leftarrow 1$
    else
      $u_{i,j} \leftarrow 0$
    end
  end
end

for $j \leftarrow 1, 2, \ldots, n-1, n$ **do**
  for $i \leftarrow j, j+1, \ldots, n-1, n$ **do**

$$u_{j,j} \leftarrow \sqrt{a_{j,j} - \sum_{k=1}^{j-1} u_{j,k}^2}$$

$$u_{i,j} \leftarrow \frac{a_{i,j} - \sum_{k=1}^{j-1} u_{i,k} \cdot u_{j,k}}{u_{j,j}}$$

  end
end

for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**

$$y_i \leftarrow \frac{b_i - \sum_{j=1}^{i-1} u_{i,j} \cdot y_j}{u_{i,i}}$$

end

for $i \leftarrow n, n-1, \ldots, 2, 1$ **do**

$$x_i \leftarrow \frac{y_i - \sum_{j=1}^{i-1} u_{j,i} \cdot x_j}{u_{i,i}}$$

end

Algoritmus 16: Algoritmus Choleského metody

M-funkce, implementující postup Choleského metody, může vypadat následovně:

```
function x=choleskeho_met(A,b)
n=length(A);
U=eye(n,n);
for j=1:n
  for i=j:n
    s=0;
    if i>1
      for k=1:j-1
        s=s+U(j,k)^2;
      end
    end
    U(j,j)=sqrt(A(j,j)-s);
    s=0;
    if i>1
      for k=1:j-1
        s=s+U(i,k)*U(j,k);
      end
    end
    U(i,j)=(A(i,j)-s)/U(j,j);
  end
end
for i=1:n
  s=0;
  if i>1
    for j=1:i-1
      s=s+U(i,j)*y(j);
    end
  end
  y(i)=(b(i)-s)/U(i,i);
end
for i=n:-1:1
  s=0;
  if i<n
    for j=i+1:n
      s=s+U(j,i)*x(j);
    end
  end
  x(i)=(y(i)-s)/U(i,i);
end
```

Příklad 6.19. Vypočtěte kořeny soustavy rovnic z příkladu 6.9 Choleského metodou.

6.2 Iterační metody řešení soustav lineárních rovnic

Řešení soustav lineárních rovnic iteračními metodami spočívá narozdíl od přímých způsobů výpočtů v postupném přibližování se k přesnému výsledku. Existuje množství způsobů tvorby takové posloupnosti aproximací, jejíž limitou je vektor $\mathbf{x}$ přesně určených kořenů soustavy lineárních rovnic.

Soustava lineárních rovnic s reálnou maticí soustavy, vyjádřena maticově, může být vyjádřena např. vztahem (6.6). Lze předpokládat, že existuje jediné přesné řešení:

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b} \,. \tag{6.36}$$

Rovnici (6.6) pak lze upravit do tvaru, který je vhodný k iterování:

$$\mathbf{x} = \mathbf{H} \cdot \mathbf{x} + \mathbf{g} \,, \tag{6.37}$$

kde $\mathbf{H}$ představuje *iterační matici*. Na vztahu (6.37) je založeno sestrojení posloupnosti iterací $\mathbf{x}^{(k)} = \{x_1^k, x_2^k, \ldots, x_n^k\}^T$ podle rekurentní formule:

$$\mathbf{x}^{(k+1)} = \mathbf{H} \cdot \mathbf{x}^{(k)} + \mathbf{g} \,, \tag{6.38}$$

pro iterační kroky $k = 0, 1, 2, \ldots$.

Kromě iterační formule je nutno definovat také volbu nulté aproximace $\mathbf{x}^{(0)} = \{x_1^0, x_2^0, \ldots, x_n^0\}^T$ a způsob ukončení iteračního cyklu, který lze provést:

(a) stanovením přesného počtu iteračních cyklů, které se mají provést (např. s využitím cyklu typu `for`),

(b) zastavovací podmínkou se zadanou tolerancí nepřesnosti $\varepsilon > 0$, např. s využitím vektorové normy:

$$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| < \varepsilon \,. \tag{6.39}$$

Iterační matice $\mathbf{H}$ i vektor $\mathbf{g}$ se může při každém iteračním kroku k měnit. Hovoří se pak o tzv. *nestacionárním iteračním procesu*, narozdíl od procesu *stacionárního*, kdy jsou matice $\mathbf{H}$ i vektor $\mathbf{g}$ nezávislými na iteračním cyklu k.

Iterační výpočet soustavy lineárních rovnic konverguje k řešení, pokud je matice soustavy $\mathbf{A}$ *diagonálně dominantní* (převažují hodnoty prvků na diagonále), což lze vyjádřit:

$$|a_{i,i}| > \sum_{j=1}^{i-1} |a_{i,j}| + \sum_{j=i+1}^{n} |a_{i,j}| \,, \quad i = 1, 2, \ldots, n \,. \tag{6.40}$$

Příklad 6.20. Sestrojte funkci pro posouzení, zda-li je matice $\mathbf{A}$ diagonálně dominantní. Vyzkoušejte podle (6.40) např. na matici:

$$[A] = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \,. \tag{6.41}$$

6.2.1 Jacobiho iterace

V případě obecné soustavy lineárních rovnic $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ jsou všechny rovnice obecně vyjádřeny:

$$a_{i,1} \cdot x_1 \quad + \quad a_{i,2} \cdot x_2 \quad + \quad \cdots \quad + \quad a_{i,n} \cdot x_n \quad = \quad b_i \,, \quad i = 1, 2, \ldots, n \,. \tag{6.42}$$

Pokud je $a_{i,i} \neq 0$, lze každou z rovnic upravit:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j}{a_{i,i}} \,, \quad i = 1, 2, \ldots, n \,, \tag{6.43}$$

což znamená, že z i-té rovnice lze určit i-tý neznámý kořen soustavy.

Jacobiho iterační rekurentní formule pak má tvar:

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k-1)}}{a_{i,i}} \,, \quad i = 1, 2, \ldots, n \,, \tag{6.44}$$

kde k je číslo iteračního cyklu ($k = 1, 2, \ldots, m$).

Výpočetní postup Jacobiho iterační metody pro m iteračních cyklů je schématicky vyjádřen algoritmem 17. Pokud se pro ukončení iteračního výpočtu použije zakončovací podmínka 6.39, algoritmus se nepatrně změní (viz algoritmus 18).

Vstup : m, n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$,
$\mathbf{x}^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\}^T$
Výstup: $\mathbf{x}^{(m)} = \{x_1^{(m)}, x_2^{(m)}, \ldots, x_n^{(m)}\}^T$
for $k \leftarrow 1, 2, \ldots, m-1, m$ **do**
  for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k-1)}}{a_{i,i}}$$

  end
end

Algoritmus 17: Algoritmus Jacobiho iterační metody

Algoritmus Jacobiho iterace lze do programovacího jazyka programu MATLAB přepsat různými způsoby. První m-funkce obsahuje implementaci výpočetního postupu řešení systému lineárních rovnic, který pracuje Jacobiho metodou s konečným počtem cyklů, tedy s využitím cyklu typu `for`. K zápisu tohoto algoritmu se využívá možnosti maticových operací programového systému MATLAB:

Vstup : n, ε, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$,
$\mathbf{x}^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\}^T$
Výstup: $\mathbf{x}^{(m)} = \{x_1^{(m)}, x_2^{(m)}, \ldots, x_n^{(m)}\}^T$

$k \leftarrow 1$
for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**

$$x_i^{(1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(0)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(0)}}{a_{i,i}}$$

end
while $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \geqq \varepsilon$ **do**
  $k \leftarrow k + 1$
  for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k-1)}}{a_{i,i}}$$

  end
end

Algoritmus 18: Algoritmus Jacobiho iterace, doplněné o zakončovací podmínku

```
function x=jacobi(A,b,x,m)
n=length(A);
d=diag(A);
r=A-diag(d);
for k=1:m
  x=(b-r*x)./d;
end
```

Druhá m-funkce je zapsána obecnějším způsobem. Výpočet je rovněž doplněn kontrolou, zda-li je matice soustavy **A** regulární a diagonálně dominantní:

```
function x=jacobi(A,b,x,m)
if det(A)==0
  error('Soustava rovnic je singulární! Det(A) je roven 0!')
  return
end
n=length(A);
for i=1:n
  if A(i,i)==0
    error('Na diagonále je nula!')
    return
```

```
  end
  s=0;
  for j=1:n
    if ~(i==j)
      s=s+abs(A(i,j));
    end
  end
  if abs(A(i,i))<s
    disp('Matice A není diagonálně dominantní!')
    return
  end
end
for k=1:m
  y=x;
  for i=1:n
    s1=0;
    if i>1
      for j=1:i-1
        s1=s1+A(i,j)*y(j);
      end
    end
    s2=0;
    if i<n
      for j=i+1:n
        s2=s2+A(i,j)*y(j);
      end
    end
    x(i)=(b(i)-s1-s2)/A(i,i);
  end
end
```

Třetí m-funkce řeší soustavu lineárních rovnic Jacobiho iterací cyklem typu `while`, jenž je ukončen zakončovací podmínkou 6.39:

```
function x=jacobi(A,b,x,eps)
y=x;
k=1;
for i=1:n
  s1=0;
  if i>1
    for j=1:i-1
      s1=s1+A(i,j)*y(j);
    end
```

```
  end
  s2=0;
  if i<n
    for j=i+1:n
      s2=s2+A(i,j)*y(j);
    end
  end
  if A(i,i)==0
    error('Na diagonále je nula!')
    return
  end
  x(i)=(b(i)-s1-s2)/A(i,i);
end
while ~(norm(y-x)<eps) & k<1000
  y=x;
  k=k+1;
  for i=1:n
    s1=0;
    if i>1
      for j=1:i-1
        s1=s1+A(i,j)*y(j);
      end
    end
    s2=0;
    if i<n
      for j=i+1:n
        s2=s2+A(i,j)*y(j);
      end
    end
    if A(i,i)==0
      error('Na diagonále je nula!')
      return
    end
    x(i)=(b(i)-s1-s2)/A(i,i);
  end
end
```

Všechny tři způsoby zápisu předpokládají, že musí být zadána i tzv. nultá aproximace řešených kořenů soustavy $\mathbf{x}^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\}^T$.

Příklad 6.21. Vyřešte kořeny soustavu lineárních rovnic Jacobiho iterací:

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{Bmatrix} . \tag{6.45}$$

Výpočet proveďte nejprve pro konečný počet cyklů $m = 5$, 10 a 20, přičemž sledujte vzrůstající přesnost dosaženého řešení. Nakonec výpočet proveďte m-funkcí se zakončovací podmínkou 6.39 se zadanou tolerancí nepřesnosti $\varepsilon = 1 \cdot 10^{-6}$.

Řešení. Správné řešení kořenů soustavy se rovná vektoru

$$\{x\} = \begin{Bmatrix} 0.5 & 0.75 & 0.25 & 0.5 \end{Bmatrix}^T . \tag{6.46}$$

Samotný průběh iteračního výpočtu může vypadat při zadání $\mathbf{x}^{(0)} = \{1, 1, 1, 1\}^T$ následujícím způsobem:

```
Průběh Jacobiho iterace
-------------------------------------------------------
 c.it.       x[1]         x[2]         x[3]         x[4]
-------------------------------------------------------
    0     1.000000     1.000000     1.000000     1.000000
    1     0.750000     1.000000     0.500000     0.750000
    2     0.625000     0.875000     0.375000     0.625000
    3     0.562500     0.812500     0.312500     0.562500
    4     0.531250     0.781250     0.281250     0.531250
    5     0.515625     0.765625     0.265625     0.515625
    6     0.507813     0.757813     0.257813     0.507813
    7     0.503906     0.753906     0.253906     0.503906
    8     0.501953     0.751953     0.251953     0.501953
    9     0.500977     0.750977     0.250977     0.500977
   10     0.500488     0.750488     0.250488     0.500488
   11     0.500244     0.750244     0.250244     0.500244
   12     0.500122     0.750122     0.250122     0.500122
   13     0.500061     0.750061     0.250061     0.500061
   14     0.500031     0.750031     0.250031     0.500031
   15     0.500015     0.750015     0.250015     0.500015
   16     0.500008     0.750008     0.250008     0.500008
   17     0.500004     0.750004     0.250004     0.500004
   18     0.500002     0.750002     0.250002     0.500002
   19     0.500001     0.750001     0.250001     0.500001
   20     0.500000     0.750000     0.250000     0.500000
```

K dosažení výsledku s tolerancí nepřesností $\varepsilon = 1 \cdot 10^{-6}$ je tedy při výpočtu Jacobiho iterací celkem 20-ti iteračních cyklů.

▲

6.2.2 Gauss-Seidelova iterační metoda

Obecnou soustavu lineárních rovnic $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ lze opět vyjádřit vztahem (6.42), přičemž lze každou z rovnic definovat při splnění podmínky $a_{i,i} \neq 0$ jako:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j}{a_{i,i}}, \quad i = 1, 2, \ldots, n\,. \tag{6.47}$$

Z této rovnice lze odvodit Gauss-Seidelovu iterační rekurentní formuli:

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k-1)}}{a_{i,i}}, \quad i = 1, 2, \ldots, n\,, \tag{6.48}$$

která se oproti Jacobiho iterační formule liší ve skutečnosti, že vypočtené kořeny soustavy $x_j^{(k)}$ jsou k dalšímu výpočtu použity ihned a nikoliv až v dalším iteračním cyklu. Celý výpočet pak konverguje k přesnému řešení rychleji.

Výpočet Gauss-Seidelovou iterační metodou pro konečný počet m iteračních cyklů je schématicky vyjádřen algoritmem 19. Postup lze samozřejmě opět doplnit i zakončovací podmínkou 6.39.

Vstup : m, n, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$,
$\mathbf{x}^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\}^T$
Výstup: $\mathbf{x}^{(m)} = \{x_1^{(m)}, x_2^{(m)}, \ldots, x_n^{(m)}\}^T$

for $k \leftarrow 1, 2, \ldots, m-1, m$ **do**
  for $i \leftarrow 1, 2, \ldots, n-1, n$ **do**

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j^{(k-1)}}{a_{i,i}}$$

  end
end

Algoritmus 19: Algoritmus Gauss-Seidelovy iterační metody

Příklad 6.22. Vyřešte soustavu lineárních rovnic z příkladu 6.21 Gauss-Seidelovou iterační metodou. Při výpočtu uvažujte tolerancí nepřesnosti $\varepsilon = 1 \cdot 10^{-6}$ a nultou aproximaci kořenů řešené soustavy $\mathbf{x}^{(0)} = \{1, 1, 1, 1\}^T$.

Řešení. Samotný průběh iteračního výpočtu lze zobrazit např. takto:

```
Průběh Gauss-Seidelovy iterace
-------------------------------------------------------
 c.it.       x[1]         x[2]         x[3]         x[4]
-------------------------------------------------------
     0    1.000000     1.000000     1.000000     1.000000
     1    0.750000     0.937500     0.437500     0.593750
     2    0.593750     0.796875     0.296875     0.523438
     3    0.523438     0.761719     0.261719     0.505859
     4    0.505859     0.752930     0.252930     0.501465
     5    0.501465     0.750732     0.250732     0.500366
     6    0.500366     0.750183     0.250183     0.500092
     7    0.500092     0.750046     0.250046     0.500023
     8    0.500023     0.750011     0.250011     0.500006
     9    0.500006     0.750003     0.250003     0.500001
    10    0.500001     0.750001     0.250001     0.500000
    11    0.500000     0.750000     0.250000     0.500000
    12    0.500000     0.750000     0.250000     0.500000
```

K dosažení výsledku s tolerancí nepřesností $\varepsilon = 1 \cdot 10^{-6}$ je tedy při výpočtu Gauss-Seidelovou iterací celkem 12-ti iteračních cyklů.

▲

6.2.3 Řídké a pásové matice

V numerické matematice se při řešení soustav rovnic často vyskytují matice soustavy, které jsou řídké a pásové.

Řídkou maticí se rozumí matice, která obsahuje značný počet nulových prvků.

Pásová matice pak má nenulové prvky pouze v těsné blízkosti diagonály. Šířka pásu matice pak představuje maximální počet sloupců v blízkosti diagonály matice s nenulovými prvky. Rovněž lze rozlišit i tzv. šířky polopásu p, q, kterými se rozumí maximální počet sloupců s nenulovými prvky vlevo a vpravo od diagonály.

Příklad 6.23. Vyřešte kořeny soustavu lineárních rovnic s pásovou maticí soustavy:

$$\begin{bmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & & \ddots & & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{Bmatrix} \tag{6.49}$$

pro $n = 100$ Gauss-Seidelovou metodou, upravenou pro řešení pásových matic. Uvažujte tolerancí nepřesnosti hodnotou $\varepsilon = 1 \cdot 10^{-6}$ a nultou aproximaci kořenů řešené soustavy $\mathbf{x}^{(0)} = \{0, 0, \ldots, 0, 0\}^T$.

Řešení. Matici soustavy **A** i vektor pravých stran lze vygenerovat příkazy programu MATLAB:

```
clc;
clear;
n=1000;
E=ones(n,1);
A=spdiags([-E 3*E -E],-1:1,n,n);
b=ones(n,1);
b(1)=2;
b(n)=2;
x=zeros(n,1);
```

Výpočet je pak možno provést m-funkcí, která implementuje výpočetní postup Gauss-Seidelovy iterační metody, upravené pro řešení pásových matic. Šířky polopásu p, q jsou v tomto případě rovny 1.

```
function x=gauss_seidel_pas(A,b,x,p,q,eps)
n=length(A);
maxkrok=1000;
y=x;
k=1;
for i=1:n
  s1=0;
  if i>1
    od=i-p;
    if od<1
      od=1;
    end
    for j=od:i-1
      s1=s1+A(i,j)*x(j);
    end
```

```
  end
  s2=0;
  if i<n
    do=i+q;
    if do>n
      do=n;
    end
    for j=i+1:do
      s2=s2+A(i,j)*y(j);
    end
  end
  if A(i,i)==0
    error('Na diagonále je nula!')
    return
  end
  x(i)=(b(i)-s1-s2)/A(i,i);
end
while ~(norm(y-x)<eps) & k<maxkrok
  y=x;
  k=k+1;
  for i=1:n
    s1=0;
    if i>1
      od=i-p;
      if od<1
        od=1;
      end
      for j=od:i-1
        s1=s1+A(i,j)*x(j);
      end
    end
    s2=0;
    if i<n
      do=i+q;
      if do>n
        do=n;
      end
      for j=i+1:do
        s2=s2+A(i,j)*y(j);
      end
    end
    if A(i,i)==0
      error('Na diagonále je nula!')
```

```
      return
    end
    x(i)=(b(i)-s1-s2)/A(i,i);
  end
end
if k==maxkrok
  disp('Výpočet nebyl ukončen ukončovací podmínkou!')
end
```

Výpočetní utilita je rovněž doplněna zakončovací podmínkou, že maximální počet iteračních cyklů může být nejvýše 1000. Pokud bude výpočet ukončen splněním této podmínky, znamená to, že iterační výpočet nekonverguje dostatečně rychle nebo diverguje. ▲

Příklad 6.24. Proveďte výpočet soustavy lineárních rovnic s pásovou maticí soustavy z příkladu 6.23 Gauss-Seidelovou iterační metodou pro $n = 1000$. Uvažujte tolerancí nepřesnosti hodnotou $\varepsilon = 1 \cdot 10^{-6}$ a nultou aproximaci kořenů řešené soustavy $\mathbf{x}^{(0)} = \{0, 0, \ldots, 0, 0\}^T$.

Příklad 6.25. Vyřešte kořeny soustavu lineárních rovnic s pásovou maticí soustavy:

$$\begin{bmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & & \ddots & & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{Bmatrix} \tag{6.50}$$

pro $n = 100$ Gauss-Seidelovou metodou, upravenou pro řešení pásových matic. Uvažujte tolerancí nepřesnosti hodnotou $\varepsilon = 1 \cdot 10^{-6}$ a nultou aproximaci kořenů řešené soustavy $\mathbf{x}^{(0)} = \{0, 0, \ldots, 0, 0\}^T$.

Řešení. Správné řešení kořenů soustavy se rovná vektoru

$$\{x\} = \begin{Bmatrix} 1 & -1 & 1 & -1 & \cdots & 1 & -1 \end{Bmatrix}^T . \tag{6.51}$$

▲

Příklad 6.26. Proveďte výpočet soustavy lineárních rovnic s pásovou maticí soustavy z příkladu 6.25 Gauss-Seidelovou iterační metodou pro $n = 1000$. Uvažujte tolerancí nepřesnosti hodnotou $\varepsilon = 1 \cdot 10^{-6}$ a nultou aproximaci kořenů řešené soustavy $\mathbf{x}^{(0)} = \{0, 0, \ldots, 0, 0\}^T$.

6.2.4 Metoda sdružených gradientů

Metoda sdružených gradientů (Conjugate Gradient Method) je vhodným výpočetním postupem pro řešení soustav lineárních rovnic, u nichž je matice soustavy $\mathbf{A}$ čtvercová, symetrická a pozitivně definitní.

Princip metody sdružených gradientů spočívá ve vhodně zvolené posloupnosti vektorů $\mathbf{d}^{(k)}$, které určují směr postupu od $\mathbf{x}^{(k)}$ k další aproximaci $\mathbf{x}^{(k+1)}$. Vektory $\mathbf{d}^{(k)}$ se postupně konstruují z posloupnosti reziduálních vektorů $\mathbf{r}^{(k)}$ tak, aby pro skalární součin $(\mathbf{d}^{(k)}, \mathbf{A}\cdot\mathbf{d}^{(k-1)})$ platilo $(\mathbf{d}^{(k)}, \mathbf{A}\cdot\mathbf{d}^{(k-1)}) = 0$. Posloupnost vektoru $\mathbf{d}^{(k)}$ je pak $\mathbf{A}$-ortogonální.

Postup výpočtu je schématicky popsán algoritmem 20, ve kterém se nejdříve vytvoří počáteční aproximace $\mathbf{x}^{(0)} = 0$ a reziduální vektor $\mathbf{r}^{(0)} = \mathbf{b}$. Za první směr $\mathbf{d}^{(0)}$ se zvolí $\mathbf{r}^{(0)}$. Určí se hodnota $\alpha^{(0)}$, pro kterou funkce (kvadratická forma) $F(\mathbf{x}^{(0)} + \alpha\cdot\mathbf{r}^{(0)}) = F(\mathbf{x}) = \frac{1}{2}\cdot(\mathbf{A}\cdot\mathbf{x},\mathbf{x}) - (\mathbf{b},\mathbf{x})$ nabývá svého minima, takže platí:

$$\alpha^{(0)} = \frac{(\mathbf{d}^{(0)}, \mathbf{r}^{(0)})}{(\mathbf{d}^{(0)}, \mathbf{A}\cdot\mathbf{d}^{(0)})} = \frac{(\mathbf{r}^{(0)})^T\cdot\mathbf{r}^{(0)}}{(\mathbf{d}^{(0)})^T\cdot\mathbf{A}\cdot\mathbf{d}^{(0)}}\,. \tag{6.52}$$

Nyní se opraví aproximace řešení soustavy $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha^{(0)}\cdot\mathbf{d}^{(0)}$, vypočte se nový reziduální vektor $\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha^{(0)}\cdot\mathbf{A}\cdot\mathbf{d}^{(0)}$ a určí se nový směr postupu $\mathbf{d}^{(1)}$ ve tvaru $\mathbf{d}^{(1)} = \mathbf{r}^{(1)} + \beta^{(0)}\cdot\mathbf{d}^{(0)}$ tak, aby platilo $(\mathbf{d}^{(1)}, \mathbf{A}\cdot\mathbf{d}^{(0)}) = 0$. Hodnota $\beta^{(0)}$ se přitom určí ze vztahu:

$$\beta^{(0)} = -\frac{(\mathbf{d}^{(0)}, \mathbf{A}\cdot\mathbf{r}^{(1)})}{(\mathbf{d}^{(0)}, \mathbf{A}\cdot\mathbf{d}^{(0)})} = \frac{(\mathbf{r}^{(1)})^T\cdot\mathbf{r}^{(1)}}{(\mathbf{r}^{(0)})^T\cdot\mathbf{r}^{(0)}}\,. \tag{6.53}$$

Dále se určí hodnota $\alpha^{(1)}$ tak, aby funkce $F(\mathbf{x}^{(1)} + \alpha\cdot\mathbf{r}^{(1)})$ nabývala minima, a celý výpočet se opakuje tak dlouho, dokud není splněna zakončovací podmínka.

Výpočetní postup řešení soustavy rovnic metodou sdružených gradientů lze naprogramovat v programu MATLAB např. následujícím způsobem:

```
function x=sdruz_grad(A,b,eps)
n=length(A);
maxkrok=1000;
x=zeros(n,1);
d=b;
r=b;
for k=1:maxkrok
  if norm(r)<eps
    return
  end
  alfa=r'*r/(d'*A*r);
  x=x+alfa*d;
  rs=r;
```

Vstup : m, n, ε, $\mathbf{A} = [a_{i,j}] = [a_{1,1}, \ldots, a_{n,n}]$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}^T$
Výstup: $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$

$\mathbf{x}^{(0)} \leftarrow 0$
$\mathbf{d}^{(0)} \leftarrow \mathbf{r}^{(0)} \leftarrow \mathbf{b}$
for $k \leftarrow 1, 2, \ldots, m-1, m$ **do**
  if $\|\mathbf{r}^{(k-1)}\| < \varepsilon$ **then**
    stop
  else
    $\alpha^{(k-1)} \leftarrow \dfrac{(\mathbf{r}^{(k-1)})^T \cdot \mathbf{r}^{(k-1)}}{(\mathbf{d}^{(k-1)})^T \cdot \mathbf{A} \cdot \mathbf{d}^{(k-1)}}$
    $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k-1)} + \alpha^{(k-1)} \cdot \mathbf{d}^{(k-1)}$
    $\mathbf{r}^{(k)} \leftarrow \mathbf{r}^{(k-1)} - \alpha^{(k-1)} \cdot \mathbf{A} \cdot \mathbf{d}^{(k-1)}$
    $\beta^{(k-1)} \leftarrow \dfrac{(\mathbf{r}^{(k)})^T \cdot \mathbf{r}^{(k)}}{(\mathbf{r}^{(k-1)})^T \cdot \mathbf{r}^{(k-1)}}$
    $\mathbf{d}^{(k)} \leftarrow \mathbf{r}^{(k)} + \beta^{(k-1)} \cdot \mathbf{d}^{(k-1)}$
  end
end

Algoritmus 20: Algoritmus metody sdružených gradientů

```
  r=r-alfa*A*d;
  beta=r'*r/(rs'*rs);
  d=r+beta*d;
  if k==maxkrok
    disp('Výpočet nebyl ukončen ukončovací podmínkou!')
  end
end
```

Výpočet hodnot $\alpha^{(k-1)}$ a $\beta^{(k-1)}$ lze v m-funkci rovněž určit s využitím příkazu programu MATLAB pro skalární součin dvou vektorů s názvem `dot`, např.:

```
alfa=dot(d,r)/dot(d,A*d);
```

resp.

```
beta=-dot(d,A*r)/dot(d,A*d);
```

Příklad 6.27. Proveďte výpočet soustavy lineárních rovnic s pásovou maticí soustavy z příkladu 6.23 metodou sdružených gradientů pro $n = 10000$ a tolerancemi nepřesnosti $\varepsilon = 1 \cdot 10^{-6}$ a $\varepsilon = 1 \cdot 10^{-12}$.

Příklad 6.28. Proveďte výpočet soustavy lineárních rovnic s pásovou maticí soustavy z příkladu 6.25 metodou sdružených gradientů pro $n = 1000$ a tolerancemi nepřesnosti $\varepsilon = 1 \cdot 10^{-6}$ a $\varepsilon = 1 \cdot 10^{-12}$.

Příklad 6.29. Vyřešte kořeny soustavu lineárních rovnic s řídkou maticí soustavy:

$$\begin{bmatrix} 3 & -1 & & & & & & & & 0.5 \\ -1 & 3 & -1 & & & & & & 0.5 & \\ & -1 & 3 & -1 & & & & 0.5 & & \\ & & & \ddots & & & \iddots & & & \\ & & & -1 & 3 & -1 & & & & \\ & & & & -1 & 3 & -1 & & & \\ & & & \iddots & & & \ddots & & & \\ & & 0.5 & & & & -1 & 3 & -1 & \\ & 0.5 & & & & & & -1 & 3 & -1 \\ 0.5 & & & & & & & & -1 & 3 \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ \\ \vdots \\ \\ \\ \vdots \\ \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} 2.5 \\ 1.5 \\ \vdots \\ 1.5 \\ 1 \\ 1 \\ 1.5 \\ \vdots \\ 1.5 \\ 2.5 \end{Bmatrix} \tag{6.54}$$

pro $n = 10000$ metodou sdružených gradientů s požadovanou tolerancí nepřesnosti $1 \cdot 10^{-6}$.

Řešení. Matici soustavy **A** i vektor pravých stran lze vygenerovat příkazy programu MATLAB:

```
clc;
clear;
n=100;
E=ones(n,1);
n2=n/2;
A=spdiags([-E 3*E -E],-1:1,n,n);
C=spdiags([E/2],0,n,n);
C=fliplr(C);
A=A+C;
A(n2+1,n2)=-1;
A(n2,n2+1)=-1;
b=zeros(n,1);
b(1)=2.5;
b(n)=2.5;
b(2:n-1)=1.5;
b(n2:n2+1)=1;
```

Správné řešení kořenů soustavy se rovná vektoru

$$\{x\} = \begin{Bmatrix} 1 & 1 & \cdots & 1 & 1 \end{Bmatrix}^T . \tag{6.55}$$

▲

Kapitola 7

Numerická integrace určitého integrálu

Cíle

Kapitola slouží k bližšímu seznámení:

- se základními algoritmy pro přibližný výpočet určitých integrálů,
- s pokročilými výpočetními postupy, využívanými v souvislosti s integrálním počtem.

Při numerické integraci se provádí přibližné řešení určitého integrálu:

$$\int_a^b f(x)\,\mathrm{d}x\ , \tag{7.1}$$

kde $f(x)$ je spojitou funkcí v intervalu $\langle a, b\rangle$, a, b představují meze určitého integrálu.

Poznámka 7.1. Pro numerickou integraci se vžilo synonymum *kvadratura*, spojované zejména s jednorozměrnými integrály. Dvojrozměrná integrace bývá někdy označovaná jako *kubatura*.

Interval $\langle a, b\rangle$ se rozdělí na n stejně velkých intervalů:

$$\langle a, b\rangle = \langle a \equiv x_0, x_1\rangle \cup \langle x_1, x_2\rangle \cup \ldots \cup \langle x_{n-1}, x_n \equiv b\rangle\ , \tag{7.2}$$

přičemž šířka všech subintervalů je stejná:

$$h = \frac{b-a}{n}\ . \tag{7.3}$$

V každém subintervalu se aproximuje integrovaná funkce $f(x)$ jednodušší interpolační nebo aproximační funkcí (polynomem stupně m) $\varphi_m(x)$:

$$\int_a^b f(x)\,\mathrm{d}x = \int_a^b \varphi_m(x)\,\mathrm{d}x + R_m(f)\ , \tag{7.4}$$

kde $R_m(f)$ je chyba použité výpočetní formule.

7.1 Obdélníková metoda

V případě použití obdélníkové metody numerické integrace se integrovaná funkce $f(x)$ aproximuje v každém ze subintervalů polynomem nultého stupně, tedy konstantní funkcí $\varphi_0(x) = \text{konst}$. Potom platí:

$$\int_a^b f(x)\,\mathrm{d}x = h \cdot \sum_{i=0}^{n-1} f(x_i) + R_0(f)\ , \tag{7.5}$$

kde $R_0(f)$ je chyba výpočtu, kterou je možno minimalizovat zvýšením počtu subintervalů n.

Příklad 7.2. Stanovte aproximaci:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.6}$$

s využitím obdélníkové metody. Při výpočtu postupně zvyšujte počet subintervalů $n = 5, 10, 20, 100$ a sledujte dosaženou přesnost řešení porovnáním s analytickým přesným řešením integrálu

Řešení. Řešení úlohy lze založit na vztahu (7.5), který lze v programu MATLAB naprogramovat například následujícím způsobem:

```
function y=obd(f,a,b,n)
if n<1
  error('Počet intervalů n musí být > 0 !')
end;
if ~(a<b)
  error('Meze integrálu musí být a > b !')
end;
h=(b-a)/n;
y=0;
for x=a:h:b-h
    y=y+f(x);
end
y=y*h;
```

Funkci `obd` lze vyvolat s čtveřicí vstupních parametrů: f je předpis integrované funkce, kterou lze v programu MATLAB definovat s pomocí příkazu `inline`, a, b jsou integrační meze ($a < b$) a n počet subintervalů, kterými byl interval $\langle a; b\rangle$ rozdělen.

Výpočet pak lze vyvolat například následující sekvencí příkazů:

```
clc;
format long;
g=inline('log(x)');
integral=obd(g,1,2,5)
```

Výsledek integrování obdélníkovou metodou s pěti subintervaly pak vychází:

```
integral =
   0.315316817512604
```

Porovná-li se výsledek s přesnou hodnotou analytického řešení:

$$\int_1^2 \ln(x)\,\mathrm{d}x = \ln(4) - 1 \approx 0,386294361119891\ . \tag{7.7}$$

například s pomocí příkazů

```
res=log(4)-1;
fprintf('Odchylka od přesného řešení = %8.6e\n\n',res-int)
```

odchylka dosažené aproximace od přesného analytického řešení činí:

```
Odchylka od přesného řešení = 7.097754e-002
```

Pro zvýšený počet subintervalů $n = 10, 20, 100$ pak výsledná hodnota aproximace řešeného integrálu i s odchylkou od přesného analytického řešení vychází:

```
integral =
   0.351220577717757

Odchylka od přesného řešení = 3.507378e-002

integral =
   0.368861530118207

Odchylka od přesného řešení = 1.743283e-002

integral =
   0.382824458574729

Odchylka od přesného řešení = 3.469903e-003
```

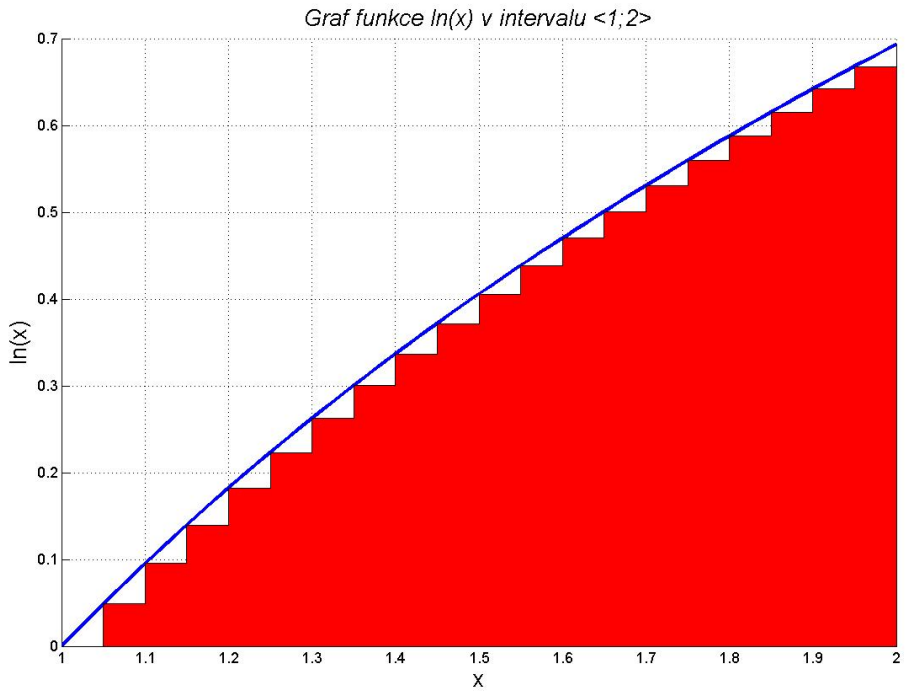
což svědčí o velké nepřesnosti a neefektivitě obdélníkové metody, jejíž princip lze schématicky pro $n = 20$ zobrazit na obrázku 7.1 (aproximace řešeného integrálu se rovná červeně zbarvené ploše). ▲

Poznámka 7.3. Analytické řešení integrálu (7.6) lze pro kontrolu získat např. v programu MATLAB příkazem

```
int(log(sym('x')),1,2)
```

který se používá pro výpočet tzv. symbolické integrace. Výsledkem je pak

```
log(4) - 1
```



Obr. 7.1 Princip výpočtu integrálu obdélníkovou metodou s počtem subintervalů $n = 20$

7.2 Lichoběžníková metoda

Pokud se k numerickému integrování použije lichoběžníková metoda numerické integrace, na jednotlivých subintervalech se integrovaná funkce $f(x)$ aproximuje polynomem prvního stupně, tedy lineární funkcí $\varphi_1(x) = k \cdot x + q$. Potom platí:

$$\int_a^b f(x)\,\mathrm{d}x = h \cdot \left(\frac{f(a \equiv x_0) + f(x_1)}{2} + \frac{f(x_1) + f(x_2)}{2} + \cdots + \right.$$
$$\left. + \frac{f(x_{n-1}) + f(x_n \equiv b)}{2} \right) + R_1 =$$
$$= h \cdot \left(\frac{1}{2} \cdot f(a \equiv x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2} \cdot f(x_n \equiv b) \right) + R_1 \,, \quad (7.8)$$

kde $R_1(f)$ je chyba výpočtu, pro kterou platí:

$$R_1(f) = -\frac{b-a}{12} \cdot h^2 \cdot f''(\xi) \,, \quad (7.9)$$

pro $\xi \in \langle a, b \rangle$. Má-li integrovaná funkce spojitou druhou derivaci, pak lze vhodnou volbou počtu subintervalů n docílit libovolně malé chyby výpočtu.

V případě, že meze integrálu tvoří x_0, x_1 a $y_0 = f(x_0), y_1 = f(x_1)$ jsou jejich příslušné funkční hodnoty ($n = 1$), lze definovat tzv. lichoběžníkové pravidlo:

Definice 7.4. Lichoběžníkové pravidlo (Trapezoid Rule):

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = \frac{h}{2}\cdot(y_0 + y_1) - \frac{h^3}{12}\cdot f''(c)\ , \tag{7.10}$$

kde $h = x_1 - x_0$ a c leží mezi x_0 a x_1.

Příklad 7.5. Využijte lichoběžníkového pravidla ke stanovení aproximace integrálu z příkladu 7.2:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.11}$$

pro $n = 1$ subinterval a určete maximální odchylku této aproximace od přesného řešení.

Řešení. Uplatněním lichoběžníkového pravidla lze získat:

$$\int_1^2 \ln(x)\,\mathrm{d}x \approx \frac{h}{2}\cdot(y_0 + y_1) = \frac{1}{2}\cdot(\ln 1 + \ln 2) \approx 0,346573590279973\ . \tag{7.12}$$

Chyba výpočtu s využitím Lichoběžníkového pravidla je dána pro $1 < c < 2$:

$$R_1(f) = -\frac{h^3}{12}\cdot f''(c)\ . \tag{7.13}$$

Platí:

$$f'(x) = \frac{1}{x} \tag{7.14}$$

a

$$f''(x) = -\frac{1}{x^2}\ , \tag{7.15}$$

takže chyba výpočtu činí:

$$R_1(f) = \frac{1^3}{12\cdot c^2}\ . \tag{7.16}$$

Největší nepřesnost výpočtu tedy bude:

$$R_1(f) \leqq \frac{1^3}{12\cdot 1^2} = \frac{1}{12} = 0,08\overline{3}\ . \tag{7.17}$$

Jinými slovy lichoběžníkové pravidlo říká:

$$\int_1^2 \ln(x)\,\mathrm{d}x = 0,346573590279973 \pm 0,08\overline{3}\ , \tag{7.18}$$

což lze porovnat s přesným řešením úlohy:

$$\int_1^2 \ln(x)\,\mathrm{d}x = \ln(4) - 1 \approx 0,386294361119891\ . \tag{7.19}$$

▲

Poznámka 7.6. Analytické řešení derivací (7.15) a (7.16) lze v programu MATLAB provést příkazy

```
diff(log(sym('x')),1)
diff(log(sym('x')),2)
```

jenž se používají pro výpočet tzv. symbolické derivace (řád požadované derivace je obsažen ve 2. vstupním parametru). Výsledný výpis obou analytických vztahů pak vypadá následovně:

```
1/x
-1/x^2
```

Příklad 7.7. Stanovte aproximace integrálu z příkladu 7.2:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.20}$$

lichoběžníkovou metodou pro $n = 5, 10, 20, 100$ subintervalů a porovnejte dosažené výsledky s přesným analytickým řešením.

Řešení. Výpočet integrálu lichoběžníkovou metodou lze založit na vztahu (7.8), který lze aplikovat v programu MATLAB následovně:

```
function y=lichobeznik(f,a,b,n)
if n<1
  error('Počet intervalů n musí být > 0 !')
end;
if ~(a<b)
  error('Meze integrálu musí být a > b !')
end;
h=(b-a)/n;
y=f(a)/2+f(b)/2;
for x=a+h:h:b-h
    y=y+f(x);
end
y=y*h;
```

Funkci `lichobeznik` lze vyvolat se stejnou čtveřicí vstupních parametrů, jak tomu bylo i v případě obdélníkové metody: f je předpis integrované funkce, kterou lze v programu MATLAB definovat s pomocí příkazu `inline`, a, b jsou integrační meze ($a < b$) a n počet subintervalů, kterými byl interval $\langle a; b\rangle$ rozdělen.

Výpočet i porovnání přesnosti s přesným analytickým řešením pak lze podobně jako v případě příkladu 7.2 vyvolat například následující sekvencí příkazů:

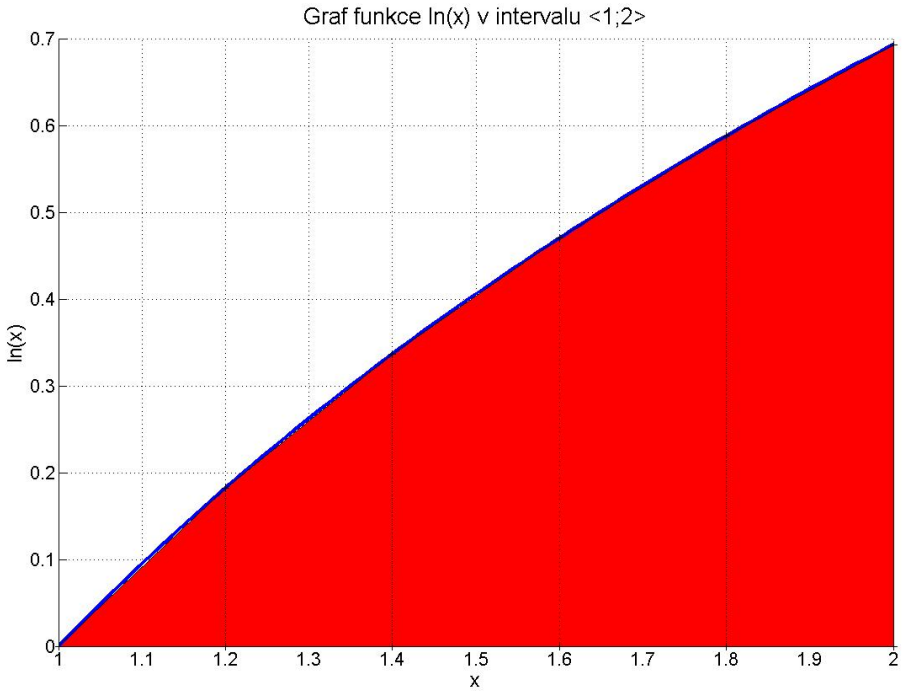
```
clc;
format long;
g=inline('log(x)');
integral=lichobeznik(g,1,2,5)
res=log(4)-1;
fprintf('Odchylka od přesného řešení = %8.6e\n\n',res-int)
```

Výsledek integrování obdélníkovou metodou s pěti, deseti, dvaceti a sto subintervaly pak vychází:

```
integral =
   0.384631535568599

Odchylka od přesného řešení = 1.662826e-003

integral =
   0.385877936745754

Odchylka od přesného řešení = 4.164244e-004

integral =
   0.386190209632206

Odchylka od přesného řešení = 1.041515e-004

integral =
   0.386290194477529

Odchylka od přesného řešení = 4.166642e-006
```

což svědčí o větší přesnosti i efektivitě řešení nežli u obdélníkové metody.

Princip výpočtu řešeného integrálu lichoběžníkovým pravidlem lze schématicky znázornit pro $n = 5$ subintervalů na obrázku 7.2 (aproximace řešeného integrálu se rovná červeně zbarvené ploše). ▲



Obr. 7.2 Princip výpočtu integrálu lichoběžníkovou metodou s počtem subintervalů $n = 5$

7.3 Simpsonova metoda

Zvolí-li se pro aproximaci funkce $f(x)$ na jednotlivých subintervalech polynomy druhého stupně, tedy kvadratické funkce $\varphi_2(x) = a \cdot x^2 + b \cdot x + c$, provádí se numerické integrování Simpsonovou metodou numerické integrace. Počet subintervalů n přitom ale musí být sudý. Potom platí:

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{3} \cdot \Big(f(a \equiv x_0) + 4 \cdot f(x_1) + 2 \cdot f(x_2) + 4 \cdot f(x_3) + \cdots + \\ + 4 \cdot f(x_{n-3}) + 2 \cdot f(x_{n-2}) + 4 \cdot f(x_{n-1}) + f(x_n \equiv b) \Big) + R_2(f)\ , \quad (7.21)$$

kde $R_2(f)$ je chyba výpočtu, kterou lze stanovit:

$$R_2(f) = -\frac{b-a}{180} \cdot h^4 \cdot f''''(\xi)\ , \quad (7.22)$$

pro $\xi \in \langle a, b \rangle$. Má-li integrovaná funkce spojitou čtvrtou derivaci, pak lze vhodnou volbou počtu subintervalů n docílit libovolně malé chyby výpočtu.

V případě, že meze integrálu tvoří x_0, x_2 a $y_0 = f(x_0), y_2 = f(x_2)$ jsou jejich příslušné funkční hodnoty ($n = 2$), a že existuje $f''''(x)$, která je spojitá, lze definovat tzv. Simpsonovo pravidlo:

Definice 7.8. Simpsonovo pravidlo (Simpson's Rule):

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x = \frac{h}{3}\cdot(y_0 + 4\cdot y_1 + y_2) - \frac{h^5}{90}\cdot f''''(c)\ , \tag{7.23}$$

kde $h = x_2 - x_1 = x_1 - x_0$, $y_1 = f(x_1)$, a c leží mezi x_0 a x_2.

Příklad 7.9. Využijte Simpsonova pravidla ke stanovení aproximace integrálu z příkladů 7.2 a 7.5:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.24}$$

pro $n = 2$ subintervalů a určete maximální odchylku této aproximace od přesného řešení.

Řešení. Podle Simpsonova pravidla vychází pro $h = 2 - 1,5 = 1,5 - 1 = 0,5$:

$$\int_1^2 \ln(x)\,\mathrm{d}x \approx \frac{h}{3}\cdot(y_0+4\cdot y_1+y_2) = \frac{0,5}{3}\cdot(\ln 1+4\cdot\ln 1,5+\ln 2) \approx 0,385834602165434\ . \tag{7.25}$$

Chyba výpočtu se podle Simpsonova pravidla pro $1 < c < 2$ stanoví:

$$R_2(f) = -\frac{h^5}{90}\cdot f''''(c)\ . \tag{7.26}$$

Platí:

$$f'''(x) = \frac{2}{x^3} \tag{7.27}$$

a

$$f''''(x) = -\frac{6}{x^4}\ , \tag{7.28}$$

takže největší nepřesnost výpočtu bude:

$$R_2(f) \leqq \frac{0,5^5\cdot 6}{90\cdot 1^4} = \frac{0,5^5\cdot 6}{60} = \frac{1}{480} = 0,002083\overline{3}\ . \tag{7.29}$$

Výsledek výpočtu Simpsonovým pravidlem je tedy:

$$\int_1^2 \ln(x)\,\mathrm{d}x = 0,385834602165434 \pm 0,002083\overline{3}\ , \tag{7.30}$$

což je podstatně přesnější hodnota výsledné aproximace integrálu (7.24) než při výpočtu lichoběžníkovým pravidlem. ▲

Poznámka 7.10. Analytické řešení derivací (7.27) a (7.28) lze opět provést v programu MATLAB příkazy pro symbolické derivování

```
diff(log(sym('x')),3)
diff(log(sym('x')),4)
```

Výsledný výpis obou analytických vztahů pak vypadá následovně:

```
2/x^3
-6/x^4
```

Příklad 7.11. Využijte Simpsonovu metodu pro výpočet aproximace integrálu z příkladu 7.9:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.31}$$

pro $n = 4, 8, 16, 32$ subintervalů a porovnejte dosažené výsledky s přesným analytickým řešením.

Řešení. Řešení integrálu Simpsonovou metodou je založeno na vztahu (7.21), který lze v programu MATLAB naprogramovat např. pomocí následující sekvence příkazů:

```
function y=simpson(f,a,b,n)
if n<1
  error('Počet intervalů n musí být > 0 !')
end;
if ~(mod(n,2)==0)
  error('Počet intervalů n musí být sudý !')
end;
if ~(a<b)
  error('Meze integrálu musí být a > b !')
end;
h=(b-a)/n;
y=f(a)+f(b);
k=1;
for x=a+h:h:b-h
    if mod(k,2)==0
      y=y+2*f(x);
    else
      y=y+4*f(x);
    end
    k=k+1;
end
y=y*h/3;
```

Funkci `simpson` lze vyvolat se stejnou čtveřicí vstupních parametrů, jak tomu bylo i v případě obdélníkové či lichoběžníkové metody: f je předpis integrované funkce, kterou lze v programu MATLAB definovat s pomocí příkazu `inline`, a, b jsou integrační meze ($a < b$) a n počet subintervalů, kterými byl interval $\langle a; b\rangle$ rozdělen (musí být sudé číslo, což je také v programu kontrolováno pomocí příkazu pro výpočet zbytku celočíselného dělení `mod`).

Výpočet i porovnání přesnosti s přesným analytickým řešením pak lze vyvolat příkazy:

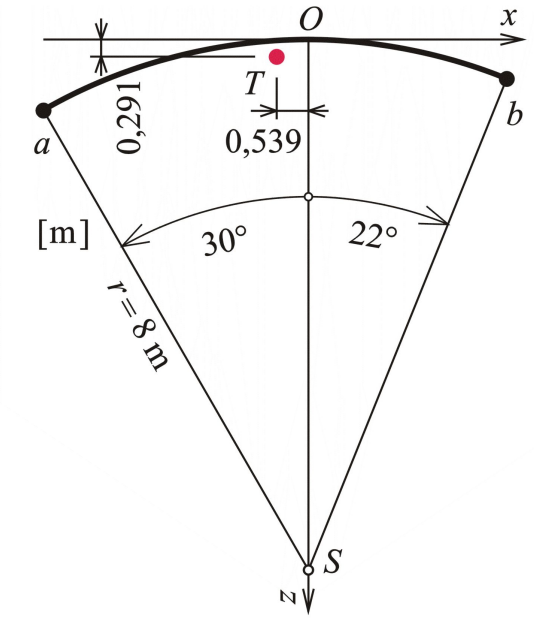
```
clc;
format long;
g=inline('log(x)');
integral=simpson(g,1,2,4)
res=log(4)-1;
fprintf('Odchylka od přesného řešení = %8.6e\n\n',res-int)
```

Výsledky integrování Simpsonovou metodou se čtyřmi, osmi, šestnácti a dvaatřiceti subintervaly jsou následující:

```
integral =
   0.386259562814567

Odchylka od přesného řešení = 3.479831e-005

integral =
   0.386292043466313

Odchylka od přesného řešení = 2.317654e-006

integral =
   0.386294213675793

Odchylka od přesného řešení = 1.474441e-007

integral =
   0.386294351862333

Odchylka od přesného řešení = 9.257558e-009
```

což svědčí o největší přesnosti i efektivitě řešení nežli u předchozích dvou numerických metod integrace. ▲

Příklad 7.12. Využijte Simpsonovu metodu s dělením na $n = 32$ subintervalů ke stanovení souřadnic těžiště kružnicového oblouku, jehož schéma je zobrazeno na obr. 7.3. Poloměr kružnicového oblouku je $r = 8$ m, středové úhly $\varphi_a = -30°$ a $\varphi_b = 22°$.



Obr. 7.3 Schéma kružnicového oblouku

Řešení. Délka kružnicového oblouku je dána vztahem:

$$s = \int_s \mathrm{d}s \,, \tag{7.32}$$

přičemž $\mathrm{d}s = r \cdot \mathrm{d}\varphi$, takže vztah (7.32) lze upravit:

$$s = \int_{\varphi_a}^{\varphi_b} r \,\mathrm{d}\varphi = r \cdot (\varphi_b - \varphi_a) \,. \tag{7.33}$$

Potřebné statické momenty se určí:

$$S_x = \int_s z \,\mathrm{d}s = \int_{\varphi_a}^{\varphi_b} r \cdot z \,\mathrm{d}\varphi \,. \tag{7.34}$$

a

$$S_z = \int_s x \,\mathrm{d}s = \int_{\varphi_a}^{\varphi_b} r \cdot x \,\mathrm{d}\varphi \,. \tag{7.35}$$

Při převodu z kartézských souřadnic na polární lze využít vztahů $x = r \cdot \sin\varphi$ a $z = r \cdot (1 - \cos\varphi)$, takže vztahy (7.34) a (7.35) se upraví do tvarů:

$$S_x = \int_{\varphi_a}^{\varphi_b} r^2 \cdot (1 - \cos\varphi)\,\mathrm{d}\varphi\ . \tag{7.36}$$

a

$$S_z = \int_{\varphi_a}^{\varphi_b} r^2 \cdot \sin\varphi\,\mathrm{d}\varphi\ . \tag{7.37}$$

Výsledné souřadnice těžiště parabolického oblouku jsou pak v daném souřadnicovém systému rovny:

$$x_T = \frac{S_z}{s} \tag{7.38}$$

a

$$z_T = \frac{S_x}{s}\ . \tag{7.39}$$

Konkrétní řešení v programu MATLAB lze provést pomocí sekvence příkazů:

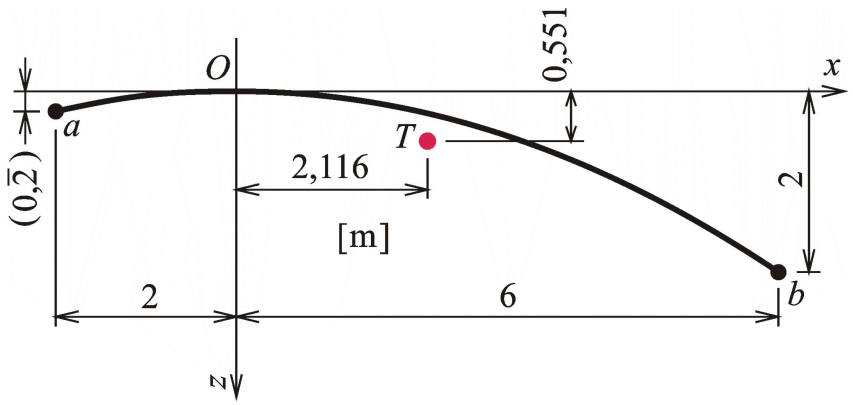
```
clc;
format long;
fia=-30/180*pi;
fib=22/180*pi;
r=8;
g1=inline('1-cos(fi)');
g2=inline('sin(fi)');
int1=simpson(g1,fia,fib,32);
int2=simpson(g2,fia,fib,32);
s=r*(fib-fia)
xT=int2*r^2/s
zT=int1*r^2/s
```

jejichž vyvolání vede k následujícím výsledkům:

```
s =
   7.260569688296410


xT =
  -0.539095557536041


zT =
   0.290574351201034
```

▲

Příklad 7.13. Pomocí Simpsonovy metody určete souřadnice těžiště parabolického oblouku, jehož schéma je zobrazeno na obr. 7.4. Tvar střednice je dán kvadratickou parabolou s rovnicí $z(x) = k \cdot x^2$. Vodorovné souřadnice obou krajních bodů jsou $x_a = -2$ m a $x_b = 6$ m, svislá pořadnice bodu b pak je $z_b = 2$ m.



Obr. 7.4 Schéma parabolického oblouku

Řešení. Parametr parabolického oblouku k se určí ze vztahu:

$$k = \frac{z_b}{x_b^2} \,. \tag{7.40}$$

Délka oblouku je dána vztahem:

$$s = \int_s \mathrm{d}s = \int_{x_a}^{x_b} \sqrt{1 + (z')^2}\,\mathrm{d}x = \int_{x_a}^{x_b} \sqrt{1 + 4 \cdot k^2 \cdot x^2}\,\mathrm{d}x \,. \tag{7.41}$$

Potřebné statické momenty se určí:

$$S_x = \int_s z\,\mathrm{d}s = \int_{x_a}^{x_b} k \cdot x^2 \cdot \sqrt{1 + 4 \cdot k^2 \cdot x^2}\,\mathrm{d}x \,. \tag{7.42}$$

a

$$S_z = \int_s x\,\mathrm{d}s = \int_{x_a}^{x_b} x \cdot \sqrt{1 + 4 \cdot k^2 \cdot x^2}\,\mathrm{d}x \,. \tag{7.43}$$

Výsledné souřadnice těžiště parabolického oblouku jsou pak v daném souřadnicovém systému rovny:

$$x_T = \frac{S_z}{s} \tag{7.44}$$

a

$$z_T = \frac{S_x}{s} \,. \tag{7.45}$$

Konkrétní řešení v programu MATLAB může být aplikováno následující posloupností příkazů:

```
clc;
format long;
xa=-2;
xb=6;
g1=inline('sqrt(1+(2*(2/(6^2))*x)^2)');
g2=inline('(2/(6^2))*x^2*sqrt(1+(2*(2/(6^2))*x)^2)');
g3=inline('x*sqrt(1+(2*(2/(6^2))*x)^2)');
int1=simpson(g1,xa,xb,32);
int2=simpson(g2,xa,xb,32);
int3=simpson(g3,xa,xb,32);
xT=int3/int1
zT=int2/int1
```

jejichž vyvolání vede k následujícím výsledkům:

```
xT =
   2.115895489649506


zT =
   0.550954275587375
```

▲

Poznámka 7.14. Určitou vadou na kráse předchozího výpočtu je definice trojice inline funkcí pomocí konkrétních vstupních hodnot pro $x_b = 2$ a $z_b = 6$, které do těchto funkcí vstupují jako definice parametru paraboly $k = \frac{2}{6^2}$. Tímto způsobem se dalo obejít použití inline funkcí se dvěma proměnnými $g(k, x)$, pro které by se musela upravit i m-funkce `simpson`. Pokuste se uvedený výpočet zobecnit.

Příklady k procvičení

1. Simpsonovu metodu použijte k určení aproximace integrálů:

   a) $\int_0^1 x^2 \,\mathrm{d}x\,,$

   b) $\int_0^\pi \sin^2(x) \,\mathrm{d}x\,,$

   c) $\int_0^{\frac{\pi}{2}} \cos(x) \,\mathrm{d}x\,,$

   d) $\int_0^1 \mathrm{e}^x \,\mathrm{d}x\,,$

   Počet intervalů postupně volte $n = 4, 8, 16, 32$. Výsledky porovnejte s přesným řešením.

7.4 Rombergova metoda

Rombergova metoda numerické integrace vychází z myšlenky, související s podrobnějším vyjádřením řešení lichoběžníkovou metodou:

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{2}\cdot\left(f(x_0) + 2\cdot\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right) + c_2\cdot h^2 + c_4\cdot h^4 + c_6\cdot h^6 + \dots \quad (7.46)$$

kde c_i závisí pouze na derivacích $f(x)$ v intervalu $\langle a; b\rangle$ a nikoliv na h.

Lze rozepsat následující posloupnost diferencí h_i pro $i = 1, 2, \dots, j$:

$$\begin{aligned} h_1 &= b - a \\ h_2 &= \frac{1}{2}\cdot(b-a) \\ h_3 &= \frac{1}{4}\cdot(b-a) \\ &\vdots \end{aligned} \quad (7.47)$$

$$h_j = \frac{1}{2^{j-1}}\cdot(b-a)\,, \quad (7.48)$$

ale také příslušné aproximace řešeného integrálu:

$$\begin{aligned} R_{1,1} &= \frac{h_1}{2}\cdot(f(a)+f(b)) \\ R_{2,1} &= \frac{h_2}{2}\cdot\left(f(a)+f(b)+2\cdot f\left(\frac{a+b}{2}\right)\right) = \frac{1}{2}\cdot R_{1,1} + h_2\cdot f\left(\frac{a+b}{2}\right) \\ &\vdots \end{aligned} \quad (7.49)$$

$$R_{j,1} = \frac{1}{2}\cdot R_{j-1,1} + h_j\cdot\sum_{i=1}^{2^{j-2}} f(a + (2\cdot i - 1)\cdot h_j) \quad (7.50)$$

pro $j = 2, 3, \dots, n$.

Dalším krokem Rombergovy integrace je stanovení zpřesněných aproximací integrálů $R_{j,k}$ pro $k = 2, \dots, j$, které lze určit s využitím předchozích hodnot aproximací $R_{j,k-1}$ a $R_{j-1,k-1}$:

$$R_{j,k} = \frac{4^{k-1}\cdot R_{j,k-1} - R_{j-1,k-1}}{4^{k-1} - 1}\,. \quad (7.51)$$

Nejpřesnější aproximací řešeného integrálu je pak $R_{j,j}$, kterou lze určit s využitím vztahů (7.48) a (7.51) v cyklu, jak je schématicky naznačeno v algoritmu 21.

Vstup : n, a, b

Výstup: $\mathbf{R}$

$R_{1,1} = (b-a) \cdot \dfrac{f(a)+f(b)}{2}$

for $j \leftarrow 2, 3, \ldots, n$ **do**

$h_j = \dfrac{b-a}{2^{j-1}}$

$R_{j,1} = \dfrac{1}{2} \cdot R_{j-1,1} + h_j \cdot \sum_{i=1}^{2^{j-2}} f(a + (2 \cdot i - 1) \cdot h_j)$

for $k \leftarrow 2, 3, \ldots, j$ **do**

$R_{j,k} = \dfrac{4^{k-1} \cdot R_{j,k-1} - R_{j-1,k-1}}{4^{k-1} - 1}$

end

end

Algoritmus 21: Algoritmus Rombergovy metody numerické integrace

Příklad 7.15. Pomocí Rombergovy metody numerické integrace vypočtěte aproximaci integrálu:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.52}$$

pro dělení $n = 3$. Výslednou aproximaci porovnejte s výsledkem přesného analytického řešení.

Řešení. Výpočet integrálu Rombergovou metodou lze v programu MATLAB naprogramovat např. následujícím způsobem:

```
function r=romberg(f,a,b,n)
h=(b-a)./(2.^(0:n-1));
r(1,1)=(b-a)*(f(a)+f(b))/2;
for j=2:n
  s=0;
  for i=1:2^(j-2)
    s=s+f(a+(2*i-1)*h(j));
  end
  r(j,1)=r(j-1,1)/2+h(j)*s;
  for k=2:j
    r(j,k)=(4^(k-1)*r(j,k-1)-r(j-1,k-1))/(4^(k-1)-1);
  end
end
```

Výsledek pro $n = 3$ pak nabývá hodnot:

```
integral =
   0.346573590279973                   0                   0
   0.376019349194069   0.385834602165434                   0
   0.383699509409442   0.386259562814567   0.386287893524509

Odchylka od přesného řešení = 6.467595e-006
```

▲

7.5 Adaptivní integrace

Adaptivní metoda numerické integrace se oproti předchozím způsobům numerického integrování liší nerovnoměrným dělením intervalu integrace $\langle a; b\rangle$. V místech, kde je integrovaná funkce dostatečně hladká a mění se pomalu, lze použít dělení intervalů hrubší. Naopak, v místech, kde se integrovaná funkce mění výrazně je vhodné použít jemnější dělení intervalů.

Metoda vychází z úpravy lichoběžníkové nebo Simpsonovy metody. V případě metody lichoběžníkové lze napsat:

$$\int_a^b f(x)\,\mathrm{d}x = S_{a,b} - h^3 \cdot \frac{f''(c_0)}{12}\,, \tag{7.53}$$

kde $h = b - a$ a $a < c_0 < b$. Pokud se bod c_0 zvolí v polovině intervalu $\langle a; b\rangle$, lze vztah (7.53) upravit:

$$\int_a^b f(x)\,\mathrm{d}x = S_{a,c} - \frac{h^3}{8} \cdot \frac{f''(c_1)}{12} + S_{c,b} - \frac{h^3}{8} \cdot \frac{f''(c_2)}{12} = \\ = S_{a,c} + S_{c,b} - \frac{h^3}{4} \cdot \frac{f''(c_3)}{12}\,, \tag{7.54}$$

Rozdílem vztahů (7.54) od (7.53) je možno získat odhad chyby dané výpočetní operace:

$$S_{a,b} - (S_{a,c} + S_{c,b}) = h^3 \cdot \frac{f''(c_0)}{12} - \frac{h^3}{4} \cdot \frac{f''(c_3)}{12} \approx \frac{3}{4} \cdot h^3 \cdot \frac{f''(c_3)}{12}\,, \tag{7.55}$$

která je zhruba trojnásobná, než nepřesnost výpočtu výrazu (7.53). Při zadání požadované tolerance nepřesnosti numerické integrace ε je pak možno zakončovací podmínku definovat:

$$S_{a,b} - (S_{a,c} + S_{c,b}) < 3 \cdot \varepsilon\,. \tag{7.56}$$

Při nesplnění kritéria zakončovací podmínky se provede dělení obou intervalů půlením. Na každé rozdělené části se pak zakončovací podmínka vyhodnocuje samostatně, což vede k nerovnoměrnému rozdělení integrovaného intervalu $\langle a; b\rangle$ na úseky se stejnou nepřesností.

Příklad 7.16. Stanovte aproximace integrálu:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.57}$$

s využitím adaptivní metody numerické integrace, vycházející z lichoběžníkové metody, pro požadovanou toleranci nepřesnosti $1 \cdot 10^{-6}$. Obě vypočtené aproximace porovnejte s výsledkem přesného analytického řešení.

Řešení. Výpočet integrálu adaptivní metodou, vycházející z lichoběžníkového pravidla, lze v programovém systému MATLAB provést např. následující m-funkcí:

```
function s=adap_int(f,a0,b0,tol0)
s=0;
n=1;
a(1)=a0;
b(1)=b0;
tol(1)=tol0;
S(1)=lich(f,a,b);
while n>0
  c=(a(n)+b(n))/2;
  oldS=S(n);
  S(n)=lich(f,a(n),c);
  S(n+1)=lich(f,c,b(n));
  if abs(oldS-(S(n)+S(n+1)))<3*tol(n)
     s=s+S(n)+S(n+1);
     n=n-1;
  else
     b(n+1)=b(n);
     b(n)=c;
     a(n+1)=c;
     tol(n)=tol(n)/2;
     tol(n+1)=tol(n);
     n=n+1;
  end
end
```

kde funkce `lich` aplikuje lichoběžníkové pravidlo:

```
function s=lich(f,a,b)
s=(f(a)+f(b))*(b-a)/2;
```

Po zadání požadované tolerance nepřesnosti řešení $\varepsilon = 1 \cdot 10^{-6}$ bude výsledek integrálu:

```
integral =
   0.386293831301211

Odchylka od přesného řešení = 5.298187e-007
```

Práci algoritmu lze demonstrovat výpisem jednotlivých subintervalů řešeného integrálu. Při zadání požadované tolerance nepřesnosti $\varepsilon = 1 \cdot 10^{-3}$ se původní interval $\langle a; b\rangle$ rozdělí na deset subintervalů s různou šířkou, na nichž se aplikuje lichoběžníková metoda (subintervaly jsou zobrazeny ve smyslu činnosti výpočetního algoritmu, tedy v opačném pořadí):

```
   i     ai       bi       hi
-----------------------------
   1   1.8750   2.0000   0.125000
   2   1.7500   1.8750   0.125000
   3   1.6250   1.7500   0.125000
   4   1.5000   1.6250   0.125000
   5   1.3750   1.5000   0.125000
   6   1.2500   1.3750   0.125000
   7   1.1875   1.2500   0.062500
   8   1.1250   1.1875   0.062500
   9   1.0625   1.1250   0.062500
  10   1.0000   1.0625   0.062500
```

▲

Poznámka 7.17. Výpočetní algoritmus adaptivní metody numerické integrace lze upravit, aby řešení vycházelo ze Simpsonovy metody integrace (např. [9]).

7.6 Gaussova metoda

Gaussova metoda numerické integrace (Gaussova kvadratura) vychází ze vztahu:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i \cdot f(x_i)\,, \tag{7.58}$$

kde koeficienty c_i i kořeny x_i pro $n = 1, 2, \ldots, 5$ integračních bodů jsou uvedeny v tabulce 7.1.

n	x_i	c_i
1	0	2
2	$-\sqrt{\frac{1}{3}} \approx -0,5774$	1
	$\sqrt{\frac{1}{3}} \approx 0,5774$	1
3	$-\sqrt{\frac{3}{5}} \approx -0,7746$	$\frac{5}{9} \approx 0,5556$
	0	$\frac{8}{9} \approx 0,8889$
	$\sqrt{\frac{3}{5}} \approx 0,7746$	$\frac{5}{9} \approx 0,5556$
4	$-\sqrt{\frac{15+2\cdot\sqrt{30}}{35}} \approx -0,8611$	$\frac{90-5\cdot\sqrt{30}}{180} \approx 0,3479$
	$-\sqrt{\frac{15-2\cdot\sqrt{30}}{35}} \approx -0,3400$	$\frac{90+5\cdot\sqrt{30}}{180} \approx 0,6521$
	$\sqrt{\frac{15-2\cdot\sqrt{30}}{35}} \approx 0,3400$	$\frac{90+5\cdot\sqrt{30}}{180} \approx 0,6521$
	$\sqrt{\frac{15+2\cdot\sqrt{30}}{35}} \approx 0,8611$	$\frac{90-5\cdot\sqrt{30}}{180} \approx 0,3479$
5	$-\sqrt{\frac{35+2\cdot\sqrt{70}}{63}} \approx -0,9062$	$\frac{322-13\cdot\sqrt{70}}{900} \approx 0,2369$
	$-\sqrt{\frac{35-2\cdot\sqrt{70}}{63}} \approx -0,5385$	$\frac{322+13\cdot\sqrt{70}}{900} \approx 0,4786$
	0	$\frac{128}{225} \approx 0,5689$
	$\sqrt{\frac{35-2\cdot\sqrt{70}}{63}} \approx 0,5385$	$\frac{322+13\cdot\sqrt{70}}{900} \approx 0,4786$
	$\sqrt{\frac{35+2\cdot\sqrt{70}}{63}} \approx 0,9062$	$\frac{322-13\cdot\sqrt{70}}{900} \approx 0,2369$

Tab. 7.1 Kořeny x_i a koeficienty c_i Gaussovy kvadratury pro $n = 1, 2, \ldots, 5$ bodů

Při řešení integrálu s obecně zadanými mezemi $\langle a; b\rangle$ je nutno integrál (7.58) transformovat:

$$\int_a^b f(x)\,\mathrm{d}x = \int_{-1}^{1} f\left(\frac{(b-a)\cdot t + b + a}{2}\right)\cdot\frac{b-a}{2}\,\mathrm{d}t \approx$$
$$\approx \sum_{i=1}^{n} c_i \cdot f\left(\frac{(b-a)\cdot t_i + b + a}{2}\right)\cdot\frac{b-a}{2}\,, \quad (7.59)$$

kde t je výsledek substituce

$$t = \frac{2\cdot x - a - b}{b-a}\,, \quad (7.60)$$

a t_i příslušný kořen x_i Gaussovy kvadratury.

Příklad 7.18. Stanovte aproximace integrálu:

$$\int_1^2 \ln(x)\,\mathrm{d}x \tag{7.61}$$

s využitím Gaussovy metody numerické integrace postupně pro $n = 1, 2, \ldots, 5$ integračních bodů a určete odchylky těchto aproximací od přesného řešení.

Řešení. M-funkce pro výpočet Gaussovy kvadratury pro počet $n = 1, 2, \ldots, 5$ integračních bodů může nabývat tvaru:

```
function s=gauss_int(f,a,b,n)
if ~((n==1)|(n==2)|(n==3)|(n==4)|(n==5))
  error('Počet intervalů n musí být 1,2,3,4 nebo 5 !')
end;
if ~(a<b)
  error('Meze integrálu musí být a > b !')
end;
if n==1
  x(1)=0; c(1)=2;
end
if n==2
  x(1)=-sqrt(1/3); x(2)=sqrt(1/3);
  c(1)=1; c(2)=1;
end
if n==3
  x(1)=-sqrt(3/5); x(2)=0; x(3)=sqrt(3/5);
  c(1)=5/9; c(2)=8/9; c(3)=5/9;
end
if n==4
  x(1)=-sqrt((15+2*sqrt(30))/35);
  x(2)=-sqrt((15-2*sqrt(30))/35);
  x(3)=sqrt((15-2*sqrt(30))/35);
  x(4)=sqrt((15+2*sqrt(30))/35);
  c(1)=(90-5*sqrt(30))/180;
  c(2)=(90+5*sqrt(30))/180;
  c(3)=(90+5*sqrt(30))/180;
  c(4)=(90-5*sqrt(30))/180;
end
if n==5
  x(1)=-sqrt((35+2*sqrt(70))/63);
  x(2)=-sqrt((35-2*sqrt(70))/63);
  x(3)=0;
  x(4)=sqrt((35-2*sqrt(70))/63);
```

```
  x(5)=sqrt((35+2*sqrt(70))/63);
  c(1)=(322-13*sqrt(70))/900;
  c(2)=(322+13*sqrt(70))/900;
  c(3)=128/225;
  c(4)=(322+13*sqrt(70))/900;
  c(5)=(322-13*sqrt(70))/900;
end
s=0;
for i=1:n
  s=s+(f(((b-a)*x(i)+b+a)/2)*(b-a)/2)*c(i);
end
```

Pro porovnání přesnosti řešení při zvyšujícím se počtu integračních bodů $n = 1, 2, \ldots, 5$ pak lze získat následující výsledky:

```
integral =
   0.405465108108164

Odchylka od přesného řešení = -1.917075e-002

integral =
   0.386594944116741

Odchylka od přesného řešení = -3.005830e-004

integral =
   0.386300421584011

Odchylka od přesného řešení = -6.060464e-006

integral =
   0.386294496938714

Odchylka od přesného řešení = -1.358188e-007

integral =
   0.386294364348948

Odchylka od přesného řešení = -3.229058e-009
```

▲

Kapitola 8

Numerické derivování

Cíle

Kapitola studentům přiblíží:

- základní výpočetní postupy pro numerické stanovení přibližné hodnoty derivací funkce,
- pokročilý způsob numerického derivování,
- základy numerického řešení parciálních derivací funkce dvou proměnných.

Numerické derivování spočívá v určení hodnoty aproximace derivace funkce $f(x)$ v konkrétním bodě x s využitím funkčních hodnot v okolních bodech a interpolačních polynomů stupně m, pro něž platí:

$$f'(x) \approx p'_m(x) \ . \tag{8.1}$$

Derivace funkce $f(x)$ v bodě x udává směrnici tečny k funkci v daném bodě.

8.1 Metoda konečných diferencí

Nejjednodušší používané vzorce pro výpočet derivace funkce $f(x)$ v bodech x_0 a x_1 $(x_0 < x_1)$ nabývají tvaru:

$$f'(x_0) = \frac{f(x_1) - f(x_0)}{h} - \frac{h}{2} \cdot f''(\xi_0) \tag{8.2}$$

a

$$f'(x_1) = \frac{f(x_1) - f(x_0)}{h} + \frac{h}{2} \cdot f''(\xi_1) \tag{8.3}$$

kde $h = x_1 - x_0$ a $\xi_0, \xi_1 \in \langle x_0, x_1 \rangle$. Předpokladem výpočtu je existence druhé derivace $f''(x)$ v řešeném intervalu $\langle x_0, x_1 \rangle$. Posledním členem ve vztazích (8.2) a (8.3) je chyba vypočtené aproximace, která se při výpočtu $f'(x)$ zanedbává.

Oba body x_0 a x_1 jsou tedy vzdáleny o diferenci h. Vztah (8.2) lze zobecnit:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} \cdot f''(\xi) \approx \frac{f(x+h) - f(x)}{h} \,, \tag{8.4}$$

kde ξ leží v intervalu $\langle x, x+h \rangle$. K určení derivace v bodě x podle (8.4) je tedy potřeba znát hodnotu funkce $f(x)$ i v druhém bodě $x+h$. Vztah (8.4) bývá označován jako *dvoubodová dopředná diferenční formule*.

Definice 8.1. Derivace funkce f v bodě x je definována předpisem:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \,. \tag{8.5}$$

Podobně je možno zobecnit i vztah (8.3):

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{h}{2} \cdot f''(\xi) \approx \frac{f(x) - f(x-h)}{h} \,, \tag{8.6}$$

V tomto případě se jedná o tzv. *dvoubodovou zpětnou diferenční formuli*, protože k určení derivace v bodě x je potřeba znát také hodnotu funkce $f(x)$ v bodě $x-h$.

Příklad 8.2. Stanovte aproximaci derivace funkce:

$$f(x) = \frac{1}{x} \tag{8.7}$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.4).

Řešení. Použitím dvoubodové dopředné diferenční formule lze získat:

$$f'(x=2) \approx \frac{f(2+0,1) - f(2)}{0,1} = \frac{\frac{1}{2,1} - \frac{1}{2}}{0,1} \approx -0,238095238095238 \,. \tag{8.8}$$

Analyticky určená první derivace funkce (8.7) je definovaná:

$$f'(x) = -\frac{1}{x^2} \,, \tag{8.9}$$

a pro $x = 2$ se přesná hodnota řešení tedy rovná:

$$f'(2) = -\frac{1}{2^2} = -\frac{1}{4} = -0,25 \,, \tag{8.10}$$

takže odchylka dosažené aproximace od přesného analytického řešení a tedy chyba numerického výpočtu derivace je rovna:

$$-0,238095238095238 - (-0,25) \approx 0,011904761904762 \,. \tag{8.11}$$

Ve vztahu (8.4) je chyba výpočtu vyjádřena vzorcem:

$$\frac{h}{2} \cdot f''(\xi) \,. \tag{8.12}$$

Druhá derivace funkce (8.7) je rovna:

$$f''(x) = \frac{2}{x^3} \,, \tag{8.13}$$

takže lze vztah (8.12) vyčíslit pro $\xi = 2$:

$$\frac{0,1}{2} \cdot \frac{2}{2^3} = 0,0125 \,. \tag{8.14}$$

i $\xi = 2,1$:

$$\frac{0,1}{2} \cdot \frac{2}{(2,1)^3} \approx 0,010797969981643 \,. \tag{8.15}$$

Chyba dosažená při výpočtu (8.11) skutečně leží v rozmezí hodnot (8.14) a (8.15).

▲

Příklad 8.3. Stanovte aproximaci derivace funkce:

$$f(x) = \frac{1}{x} \tag{8.16}$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.6).

Řešení. Použitím dvoubodové zpětné diferenční formule lze získat:

$$f'(x = 2) \approx \frac{f(2) - f(2 - 0,1)}{0,1} = \frac{\frac{1}{2} - \frac{1}{1,9}}{0,1} \approx -0,263157894736842 \,. \tag{8.17}$$

Chyba dosažená při výpočtu je rovna:

$$-0,25 - (-0,263157894736842) \approx 0,013157894736842 \tag{8.18}$$

a leží v rozmezí hodnot:

$$\frac{0,1}{2} \cdot \frac{2}{2^3} = 0,0125 \,. \tag{8.19}$$

a

$$\frac{0,1}{2} \cdot \frac{2}{(1,9)^3} \approx 0,014579384749964 \,. \tag{8.20}$$

▲

Výpočet derivace podle (8.2) a (8.3) je založen na derivaci interpolačního polynomu prvního stupně p_1 v uzlech x_0 a $x_1 = x_0 + h$. Použitím polynomu druhého stupně p_2 lze získat:

$$f'(x_0) = \frac{-3 \cdot f(x_0) + 4 \cdot f(x_1) - f(x_2)}{2 \cdot h} + \frac{h^2}{3} \cdot f'''(\xi_0) \,, \tag{8.21}$$

$$f'(x_2) = \frac{f(x_0) - 4 \cdot f(x_1) + 3 \cdot f(x_2)}{2 \cdot h} + \frac{h^2}{3} \cdot f'''(\xi_1) \tag{8.22}$$

a

$$f'(x_1) = \frac{f(x_2) - f(x_0)}{2 \cdot h} - \frac{h^2}{12} \cdot f'''(\xi_0) - \frac{h^2}{12} \cdot f'''(\xi_1) \,, \tag{8.23}$$

kde $h = x_1 - x_0 = x_2 - x_1$, $\xi_0 \in \langle x_0, x_1 \rangle$ a $\xi_1 \in \langle x_1, x_2 \rangle$. Předpokladem výpočtu je existence třetí derivace $f'''(x)$ v řešeném intervalu $\langle x_0, x_2 \rangle$.

Vztah (8.21) lze zobecnit:

$$f'(x) = \frac{-3 \cdot f(x) + 4 \cdot f(x+h) - f(x+2 \cdot h)}{2 \cdot h} + \frac{h^2}{3} \cdot f'''(\xi_0) \approx$$
$$\approx \frac{-3 \cdot f(x) + 4 \cdot f(x+h) - f(x+2 \cdot h)}{2 \cdot h} \,. \tag{8.24}$$

K určení derivace v bodě x podle (8.24) je tedy potřeba znát hodnotu funkce $f(x)$ i v dalších dvou bodech $x + h$ a $x + 2 \cdot h$. Vztah (8.24) lze označit jako *tříbodovou dopřednou diferenční formuli*.

Definice 8.4. Derivace funkce f v bodě x je definována předpisem:

$$f'(x) = \lim_{h \to 0} \frac{-3 \cdot f(x) + 4 \cdot f(x+h) - f(x+2 \cdot h)}{2 \cdot h} \,. \tag{8.25}$$

Podobně je možno zobecnit i vztah (8.22):

$$f'(x) = \frac{f(x-2 \cdot h) - 4 \cdot f(x-h) + 3 \cdot f(x)}{2 \cdot h} + \frac{h^2}{3} \cdot f'''(\xi_1) \approx$$
$$\approx \frac{f(x-2 \cdot h) - 4 \cdot f(x-h) + 3 \cdot f(x)}{2 \cdot h} \,. \tag{8.26}$$

V tomto případě se jedná o tzv. *tříbodovou zpětnou diferenční formuli*, protože k určení derivace v bodě x je potřeba znát také hodnotu funkce $f(x)$ ve dvou dalších bodech $x - h$ a $x - 2 \cdot h$.

Výpočet derivace funkce $f(x)$ lze provést i s pomocí zobecněného vztahu (8.23):

$$f'(x) = \frac{f(x+h) - f(x-h)}{2 \cdot h} - \frac{h^2}{12} \cdot f'''(\xi_0) - \frac{h^2}{12} \cdot f'''(\xi_1) \approx$$
$$\approx \frac{f(x+h) - f(x-h)}{2 \cdot h} - \frac{h^2}{6} \cdot f'''(\xi) \approx \frac{f(x+h) - f(x-h)}{2 \cdot h} \,, \tag{8.27}$$

kde $\xi \in \langle x+h, x-h\rangle$. V tomto případě se jedná o *tříbodovou středovou diferenční formuli*. Bod x přitom leží uprostřed výpočetního intervalu $\langle x+h, x-h\rangle$.

Příklad 8.5. Stanovte aproximaci derivace funkce:

$$f(x) = \frac{1}{x} \tag{8.28}$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.24).

Řešení. Použitím tříbodové dopředné diferenční formule lze získat:

$$f'(x=2) \approx \frac{-3 \cdot f(2) + 4 \cdot f(2+0,1) - f(2+2 \cdot 0,1)}{2 \cdot 0,1} = \\ = \frac{-\frac{3}{2} + \frac{4}{2,1} - \frac{1}{2,2}}{0,2} \approx -0,248917748917749 \; . \tag{8.29}$$

Chyba numerického výpočtu derivace je rovna:

$$-0,248917748917749 - (-0,25) \approx 0,001082251082251 \; . \tag{8.30}$$

Ve vztahu (8.24) je chyba výpočtu vyjádřena vzorcem:

$$-\frac{h^2}{3} \cdot f'''(\xi_0) \; . \tag{8.31}$$

Třetí derivace funkce (8.28) je rovna:

$$f'''(x) = -\frac{6}{x^4} \; , \tag{8.32}$$

takže lze vztah (8.31) vyčíslit pro $\xi = 2$:

$$-\frac{(0,1)^2}{3} \cdot -\frac{6}{2^4} = 0,00125 \; . \tag{8.33}$$

i $\xi = 2,1$:

$$-\frac{(0,1)^2}{3} \cdot -\frac{6}{(2,1)^4} \approx 0,001028378093490 \; . \tag{8.34}$$

Chyba dosažená při výpočtu (8.30) skutečně leží v rozmezí hodnot (8.33) a (8.34).

▲

Příklad 8.6. Stanovte aproximaci derivace funkce:

$$f(x) = \frac{1}{x} \tag{8.35}$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.26).

Řešení. Použitím tříbodové zpětné diferenční formule lze získat:

$$f'(x=2) \approx \frac{f(2-2\cdot 0,1) - 4\cdot f(2-0,1) + 3\cdot f(2)}{2\cdot 0,1} = \\ = \frac{\frac{1}{1,8} - \frac{4}{1,9} + \frac{3}{2}}{0,2} \approx -0,248538011695906 \,. \quad (8.36)$$

Chyba numerického výpočtu derivace je rovna:

$$-0,248538011695906 - (-0,25) \approx 0,001461988304094 \,. \quad (8.37)$$

Chybu ve vztahu (8.26) lze vyčíslit pro $\xi = 2$:

$$-\frac{(0,1)^2}{3} \cdot -\frac{6}{2^4} = 0,00125 \,. \quad (8.38)$$

i $\xi = 1,9$:

$$-\frac{(0,1)^2}{3} \cdot -\frac{6}{(1,9)^4} \approx 0,001534672078944 \,. \quad (8.39)$$

Chyba dosažená při výpočtu (8.37) leží v rozmezí hodnot (8.38) a (8.39). ▲

Příklad 8.7. Stanovte aproximaci derivace funkce:

$$f(x) = \frac{1}{x} \quad (8.40)$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.27).

Řešení. Derivaci funkce (8.40) lze určit s pomocí tříbodové středové diferenční formule:

$$f'(x=2) \approx \frac{f(2+0,1) - f(2-0,1)}{2\cdot 0,1} = \frac{\frac{1}{2,1} - \frac{1}{1,9}}{0,2} \approx -0,250626566416040 \,. \quad (8.41)$$

Chyba dosažená při výpočtu je rovna:

$$-0,25 - (-0,250626566416040) \approx 6,265664160400863 \cdot 10^{-4} \,. \quad (8.42)$$

Ve vztahu (8.27) je chyba výpočtu vyjádřena vzorcem:

$$-\frac{h^2}{6} \cdot f'''(\xi) \,. \quad (8.43)$$

Třetí derivace funkce (8.40) se podle (8.32) rovná:

$$f'''(x) = -\frac{6}{x^4} \,, \quad (8.44)$$

takže lze vztah (8.43) vyčíslit nejprve pro $\xi = 1,9$:

$$-\frac{(0,1)^2}{6} \cdot -\frac{6}{(1,9)^4} \approx 7,673360394717661 \cdot 10^{-4} \,. \tag{8.45}$$

a pak pro $\xi = 2,1$:

$$-\frac{(0,1)^2}{6} \cdot -\frac{6}{(2,1)^4} \approx 5,141890467449263 \cdot 10^{-4} \,. \tag{8.46}$$

Chyba dosažená při výpočtu (8.42) skutečně leží v rozmezí hodnot (8.45) a (8.46). ▲

Má-li řešená funkce čtvrtou derivaci $f''''(x)$, lze pak s využitím polynomu druhého stupně p_2 stanovit i druhou derivaci $f''(x)$ v bodě x_1:

$$f''(x_1) = \frac{f(x_0) - 2 \cdot f(x_1) + f(x_2)}{h^2} - \frac{h^2}{12} \cdot f''''(\xi) \,, \tag{8.47}$$

pro $\xi \in \langle x_0, x_2 \rangle$.

Příklad 8.8. Stanovte aproximaci druhé derivace funkce:

$$f(x) = \frac{1}{x} \tag{8.48}$$

v bodě $x = 2$ s diferencí $h = 0,1$ pomocí vztahu (8.47).

Řešení. Aproximaci druhé derivace funkce (8.48) lze určit dosazením příslušných hodnot do výrazu (8.47):

$$f''(2) = \frac{f(2-0,1) - 2 \cdot f(2) + f(2+0,1)}{(0,1)^2} \approx 0,250626566416034 \tag{8.49}$$

Analyticky určená druhá derivace funkce (8.48) je definovaná podle (8.13):

$$f''(x) = \frac{2}{x^3} \,, \tag{8.50}$$

a pro $x = 2$ se přesná hodnota řešení tedy rovná:

$$f''(2) = \frac{2}{2^3} = \frac{2}{8} = 0,25 \,, \tag{8.51}$$

takže chyba dosažená při výpočtu podle (8.49) je rovna:

$$0,250626566416034 - 0,25 \approx 6,265664160344797 \cdot 10^{-4} \,. \tag{8.52}$$

Ve vztahu (8.47) je chyba výpočtu vyjádřena vzorcem:

$$\frac{h^2}{12} \cdot f''''(\xi) \ . \tag{8.53}$$

Čtvrtá derivace funkce (8.48) je rovna:

$$f''''(x) = \frac{24}{x^5} \ , \tag{8.54}$$

takže lze vztah (8.53) vyčíslit nejprve pro $\xi = 1,9$:

$$\frac{(0,1)^2}{12} \cdot \frac{24}{(1,9)^5} \approx 8,077221468123856 \cdot 10^{-4} \ . \tag{8.55}$$

a pak pro $\xi = 2,1$:

$$\frac{(0,1)^2}{12} \cdot \frac{24}{(2,1)^5} \approx 4,897038540427868 \cdot 10^{-4} \ . \tag{8.56}$$

Chyba dosažená při výpočtu (8.52) se nachází v rozmezí hodnot (8.55) a (8.56). ▲

Vztah (8.47) lze získat rovněž jako derivaci z prvních derivací:

$$f''(x) \approx \frac{f'(x) - f'(x-h)}{h} =$$

$$= \frac{\dfrac{f(x+h) - f(x)}{h} - \dfrac{f(x) - f(x-h)}{h}}{h} =$$

$$= \frac{f(x+h) - 2 \cdot f(x) + f(x-h)}{h^2} \ , \tag{8.57}$$

kde derivace $f'(x)$ je určena pomocí dvoubodové dopředné diferenční formule (8.4).

Podobně lze stanovit i třetí derivaci funkce $f(x)$, např. pomocí tříbodové středové diferenční formule (8.27):

$$f'''(x) \approx \frac{f''(x+h) - f''(x-h)}{2 \cdot h} =$$

$$= \frac{\dfrac{f(x+2 \cdot h) - 2 \cdot f(x+h) + f(x)}{h^2} - \dfrac{f(x) - 2 \cdot f(x-h) + f(x-2 \cdot h)}{h^2}}{2 \cdot h} =$$

$$= \frac{f(x+2 \cdot h) - 2 \cdot f(x+h) + 2 \cdot f(x-h) - f(x-2 \cdot h)}{2 \cdot h^3} \ , \tag{8.58}$$

nebo čtvrtou derivaci funkce $f(x)$, např. pomocí diferenční formule (8.47):

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

$$= \frac{\dfrac{f(x+2\cdot h) - 2 \cdot f(x+h) + f(x)}{h^2} - 2 \cdot \dfrac{f(x+h) - 2 \cdot f(x) + f(x-h)}{h^2} + \dfrac{f(x) - 2 \cdot f(x-h) + f(x-2\cdot h)}{h^2}}{h^2} =$$

$$= \frac{f(x+2\cdot h) - 4 \cdot f(x+h) + 6 \cdot f(x) - 4 \cdot f(x-h) + f(x-2\cdot h)}{h^4} \,. \quad (8.59)$$

Příklady k procvičení

1. Vztahy pro numerické určení první, druhé, třetí a čtvrté derivace využijte pro výpočet pootočení, ohybového momentu, posouvající síly a zatížení derivováním ohybové čáry konstrukce z příkladů
   a) 3.1
   b) 3.2
   c) 3.3.

   Uvedené veličiny stanovte nejprve tabelizací v průřezech s roztečí 10 cm a nakonec graficky zobrazte. Porovnejte s přesným řešením.

8.2 Numerické derivování s proměnnou diferencí

U numerického výpočtu derivace vyvstane otázka, jak velkou volit diferenci h. Řešením je tzv. Nevillův algoritmus (podobný Rombergově integraci - viz kap. 7.4), který byl definován anglickým matematikem Ericem Haroldem Nevillem. Výpočetní postup vychází z tříbodové středové diferenční formule 8.27.

Výpočet probíhá v cyklu s řídící proměnnou i celkem n-krát, přičemž se vždy změní krok h na hodnotu:

$$h_i = \frac{h_0}{10^{i-1}} \,, \tag{8.60}$$

kde h_0 je počáteční hodnota diference.

Velikost derivace je pak rovna:

$$a_{i,1} = \frac{f(x+h_i) - f(x-h_i)}{2 \cdot h_i} \,, \tag{8.61}$$

což lze dále zpřesňovat:

$$a_{i,j} = \frac{a_{i,j-1} \cdot 10^{2 \cdot j-2} - a_{i-1,j-1}}{10^{2 \cdot j-2} - 1} \,, \tag{8.62}$$

pro $j = 2, 3, \ldots, n$. Nejpřesnější odhad požadované derivace je na konci výpočtu obsažen v proměnné $a_{n,n}$. Tento výpočetní postup lze schématicky znázornit algoritmem 22.

Vstup : f, x, h_0, n

Výstup: a

$h_1 = h_0$

$a_{1,1} = \dfrac{f(x+h_1) - f(x-h_1)}{2 \cdot h_1}$

for $i \leftarrow 2, 3, \ldots, n$ **do**

  $h_i = \dfrac{h_0}{10^{i-1}}$

  $a_{i,1} = \dfrac{f(x+h_i) - f(x-h_i)}{2 \cdot h_i}$

  for $j \leftarrow 2, 3, \ldots, i$ **do**

    $a_{i,j} = \dfrac{a_{i,j-1} \cdot 10^{2 \cdot j-2} - a_{i-1,j-1}}{10^{2 \cdot j-2} - 1}$

  end

end

Algoritmus 22: Nevillův algoritmus numerického derivování

Příklad 8.9. Vypočtěte aproximaci derivace:

$$f(x) = \frac{1}{x} \tag{8.63}$$

v bodě $x = 2$ s využitím Nevillova algoritmu numerického derivování. Výslednou aproximaci porovnejte s výsledkem přesného analytického řešení.

Řešení. Výpočet derivace pomocí Nevillova výpočetního postupu lze v programu MATLAB naprogramovat např. následujícím způsobem:

```
function a=neville(f,x,h0,n)
h(1)=h0;
a(1,1)=(f(x+h(1))-f(x-h(1)))/(2*h(1));
for i=2:n
  h(i)=h0/(10^(i-1));
  a(i,1)=(f(x+h(i))-f(x-h(i)))/(2*h(i));
  for j=2:i
    a(i,j)=(a(i,j-1)*10^(2*j-2)-a(i-1,j-1))/(10^(2*j-2)-1);
  end
end
```

Výsledek pro $n = 3$ pak nabývá hodnot:

```
deriv =
  -0.250626566416040                   0                   0
  -0.250006250156248  -0.249999984335442                   0
  -0.250000062499978  -0.249999999998399  -0.249999999999966

Odchylka od přesného řešení = 3.438916e-014
```

▲

8.3 Parciální derivace

Při řešení funkce dvou proměnných:

$$z = f(x, y) \tag{8.64}$$

lze stanovit parciální derivace. Pokud bude y považováno za konstantu, parciální derivaci podle x lze definovat:

$$\frac{\partial z}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x} \,. \tag{8.65}$$

Naopak parciální derivace funkce $f(x, y)$ podle y nabývá tvaru:

$$\frac{\partial z}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y} \,. \tag{8.66}$$

Druhé parciální derivace funkce $f(x, y)$ pak mohou být určeny pomocí vztahů:

$$\frac{\partial^2 z}{\partial x^2} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - 2 \cdot f(x, y) + f(x - \Delta x, y)}{\Delta x^2} \tag{8.67}$$

a

$$\frac{\partial^2 z}{\partial y^2} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - 2 \cdot f(x, y) + f(x, y - \Delta y)}{\Delta y^2} \,. \tag{8.68}$$

Lze rovněž definovat smíšenou parciální derivaci funkce $f(x, y)$:

$$\frac{\partial^2 z}{\partial x \partial y} = \lim_{\Delta x, \Delta y \to 0} \frac{f(x + \Delta x, y + \Delta y) - f(x + \Delta x, y - \Delta y) -}{4 \cdot \Delta x \cdot \Delta y} \\ \frac{-f(x - \Delta x, y + \Delta y) + f(x - \Delta x, y - \Delta y)}{4 \cdot \Delta x \cdot \Delta y} \,. \tag{8.69}$$

Třetí parciální derivace funkce $f(x, y)$ je možno určit podobně jako v (8.58):

$$\frac{\partial^3 z}{\partial x^3} = \lim_{\Delta x \to 0} \frac{f(x + 2 \cdot \Delta x, y) - 2 \cdot f(x + \Delta x, y) +}{2 \cdot \Delta x^3} \\ \frac{+2 \cdot f(x - \Delta x, y) - f(x - 2 \cdot \Delta x, y)}{2 \cdot \Delta x^3} \tag{8.70}$$

a

$$\frac{\partial^3 z}{\partial y^3} = \lim_{\Delta y \to 0} \frac{f(x, y + 2 \cdot \Delta y) - 2 \cdot f(x, y + \Delta y) +}{2 \cdot \Delta y^3} \\ \frac{+2 \cdot f(x, y - \Delta y) - f(x, y - 2 \cdot \Delta y)}{2 \cdot \Delta y^3} \,. \tag{8.71}$$

Stejným způsobem lze postupovat i v případě čtvrtých parciálních derivací funkce $f(x, y)$, např. podobně jako v (8.59):

$$\frac{\partial^4 z}{\partial x^4} = \lim_{\Delta x \to 0} \frac{f(x + 2 \cdot \Delta x, y) - 4 \cdot f(x + \Delta x, y) + 6 \cdot f(x, y) -}{\Delta x^4} \\ \frac{-4 \cdot f(x - \Delta x, y) + f(x - 2 \cdot \Delta x, y)}{\Delta x^4} \tag{8.72}$$

a

$$\frac{\partial^4 z}{\partial y^4} = \lim_{\Delta y \to 0} \frac{f(x, y + 2 \cdot \Delta y) - 4 \cdot f(x, y + \Delta y) + 6 \cdot f(x, y) -}{\Delta y^4} \\ \frac{-4 \cdot f(x, y - \Delta y) + f(x, y - 2 \cdot \Delta y)}{\Delta y^4} \,. \tag{8.73}$$

Rovněž lze definovat smíšenou čtvrtou parciální derivaci funkce $f(x, y)$:

$$\frac{\partial^4 z}{\partial x^2 \partial y^2} = \lim_{\Delta x, \Delta y \to 0} \frac{f(x + \Delta x, y + \Delta y) - 2 \cdot f(x + \Delta x, y) + f(x + \Delta x, y - \Delta y) -}{\Delta x^2 \cdot \Delta y^2}$$

$$\frac{-2 \cdot (f(x, y + \Delta y) - 2 \cdot f(x, y) + f(x, y - \Delta y)) +}{\Delta x^2 \cdot \Delta y^2}$$

$$\frac{+f(x - \Delta x, y + \Delta y) - 2 \cdot f(x - \Delta x, y) + f(x - \Delta x, y - \Delta y)}{\Delta x^2 \cdot \Delta y^2} . \quad (8.74)$$

Kapitola 9

Řešení diferenciálních rovnic

Cíle

Kapitola má za cíl:

- seznámit studenty s numerickým řešením jednoduchých diferenciálních rovnic,
- ukázat jim jejich uplatnění v elementárních úlohách stavební mechaniky.

V diferenciálních rovnicích se jako proměnné objevují derivace funkcí. Podle počtu proměnných a typu derivací funkcí lze diferenciální rovnice členit na:

- *obyčejné diferenciální rovnice*, jenž obsahují derivace hledané funkce jen podle jedné proměnné.
- *parciální diferenciální rovnice*, které obsahují derivace hledané funkce podle více proměnných, tedy parciální derivace.

Řád diferenciální rovnice je definován podle nejvyšší derivace, která je v dané diferenciální rovnice obsažena.

Řešením diferenciální rovnice je funkce, která má příslušné derivace a vyhovuje dané diferenciální rovnici - integrál diferenciální rovnice, kterých může být nekonečně mnoho. V praktických úlohách definují jednoznačné řešení počáteční podmínky.

9.1 Obyčejné diferenciální rovnice prvního řádu

Obyčejné diferenciální rovnice prvního řádu obsahují jednu derivaci funkce jedné závisle proměnné $y(x)$. Numericky lze např. řešit funkci $y = y(x)$, která v intervalu $\langle a, b\rangle$ vyhovuje rovnici:

$$y'(x) = f(x, y(x)) \,, \tag{9.1}$$

kde $f(x, y(x))$ je pravá strana obyčejné diferenciální rovnice prvního řádu, pro jejíž jednoznačné určení musí být splněna počáteční podmínka ve tvaru:

$$y(a) = c\ . \tag{9.2}$$

9.1.1 Eulerova metoda

Nejjednodušší výpočetní postup numerického řešení obyčejných diferenciálních rovnic s počáteční podmínkou publikoval v roce 1768 švýcarský matematik a fyzik Leonhard Euler. Řešení je založeno na přibližném výpočtu derivace funkce $y'(x)$ v rovnici (9.1) pomocí aproximace metodou konečných diferencí s využitím dvoubodové dopředné diferenční formule (8.4):

$$y'(x_i) = f(x_i, y(x_i)) \approx \frac{y(x_{i+1}) - y(x_i)}{h} = f(x_i, y_i)\ , \tag{9.3}$$

kde h je krok, odpovídající hodnotě $(x_{i+1} - x_i)$. Ze vztahu (9.3) je možno jednoduchou úpravou získat rekurentní vzorec Eulerovy metody:

$$y_{i+1} = y_i + h \cdot f(x_i, y_i)\ , \tag{9.4}$$

pro $i = 0, 1, \ldots, n-1$, kde n je počet diferencí v řešeném intervalu $\langle a, b\rangle$. Hodnotu y_0 je nutno jednoznačně určit pomocí počáteční podmínky podle (9.2).

Příklad 9.1. Eulerovou metodou určete v intervalu $\langle -2; 3\rangle$ přibližné řešení obyčejné diferenciální rovnice:

$$y'(x) = x^2 - 0,2 \cdot y(x)\ , \tag{9.5}$$

s počáteční podmínkou $y(-2) = -1$. Výpočetní krok h postupně volte $h = 1$, $h = 0,5$, $h = 0,1$ příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením:

$$y(x) = 5 \cdot x^2 - 50 \cdot x + 250 - \frac{371}{e^{0,4}} \cdot e^{-0,2 \cdot x}\ , \tag{9.6}$$

Řešení. Řešení Eulerovou metodou vychází z rekurentního vzorce (9.4) a lze je naprogramovat např. následujícím způsobem:

```
f=inline('x^2-0.2*y');
a=-2; b=3;
c=-1;
h=0.5;
n=(b-a)/h;
x(1)=a; y(1)=c;
yp(1)=c;
```

```
for i=1:n
  x(i+1)=x(i)+h; y(i+1)=y(i)+h*f(x(i),y(i));
  yp(i+1)=5*x(i+1)^2-50*x(i+1)+250-(371/exp(0.4))*exp(-0.2*x(i+1));
end
[x' y' yp' (y-yp)']
plot(x,y,'r',x,yp,'b');
legend('přesné řešení','aproximace');
title('Eulerova aproximace');
xlabel('x'); ylabel('y(x)');
```

Výsledné řešení např. pro výpočetní krok $h = 0,5$ je následující:

```
     x          y          yp         y-yp
-----------------------------------------
   -2.0000   -1.0000   -1.0000    0.0000
   -1.5000    1.1000    0.5553    0.5447
   -1.0000    2.1150    1.2509    0.8641
   -0.5000    2.4035    1.4064    0.9971
    0.0000    2.2882    1.3113    0.9769
    0.5000    2.0593    1.2271    0.8322
    1.0000    1.9784    1.3909    0.5875
    1.5000    2.2806    2.0169    0.2637
    2.0000    3.1775    3.2990   -0.1214
    2.5000    4.8598    5.4127   -0.5529
    3.0000    7.4988    8.5167   -1.0179
```

Srovnání dosažené přesnosti řešení pro jednotlivé hodnoty výpočetních kroků $h = 1$, $h = 0,5$, $h = 0,1$ a $h = 0,01$ je zobrazeno na obrázku 9.1.

▲

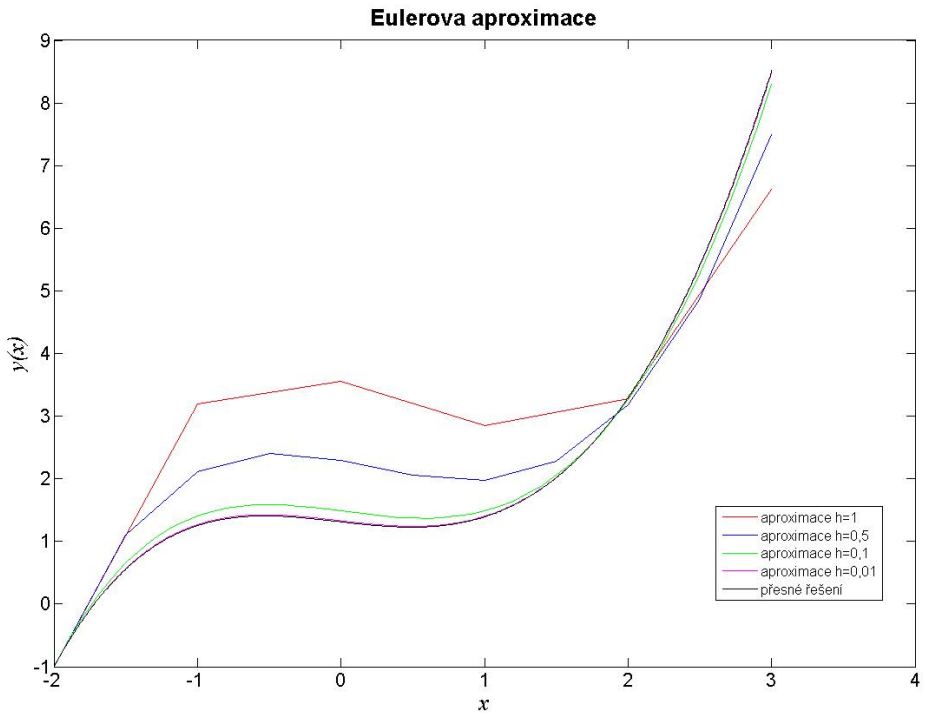
Poznámka 9.2. Řešení úlohy z příkladu 9.1 je možno zkontrolovat rovněž matematickými prostředky programového systému MATLAB. Jednou z možností je využití funkce `ode45`, např. s požadovanou tolerancí nepřesnosti řešení $1 \cdot 10^{-9}$. Nejprve je potřeba zadat řešenou diferenciální rovnici s využitím samostatné `m-funkce`, např.:

```
function y=fce(x,y);
y=x^2-0.2*y;
```

na kterou se pak lze odkázat:

```
options=odeset('AbsTol',1e-9);
[x,y]=ode45(@fce,[-2 3],-1,options)
```

Poslední příkaz vypíše hodnoty výsledné funkce $y(x)$ v bodech x_i. Pokud se příkaz upraví na tvar:



Obr. 9.1 Výsledná aproximace funkce $y(x)$ z příkladu 9.1 pro kroky $h = 1$, $h = 0,5$, $h = 0,1$ a $h = 0,01$

```
ode45(@fce,[-2 3],-1)
```

zobrazí se graf vyřešené funkce $y(x)$ (viz obrázek 9.2).

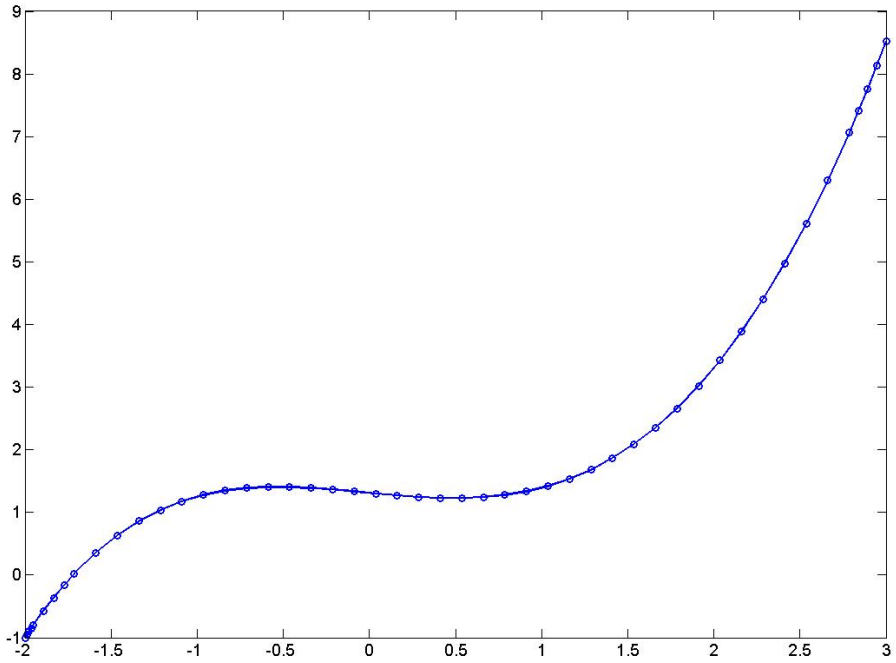
Druhou možností, jak úlohu z příkladu 9.1 vyřešit příkazy programu MATLAB, je použití funkce `dsolve` pro symbolické řešení obyčejných diferenciálních rovnic s následnou vektorizací stanovené funkce $y(x)$ pomocí sekvence povelů:

```
y=dsolve('Dy=x^2-0.2*y','y(-2)=-1','x')
x=linspace(-2,3,1000);
z=eval(vectorize(y));
plot(x,z)
```

Výsledný výraz, který byl funkcí `dsolve` stanoven:

```
y =
5*x^2 - 371/(exp(2/5)*exp(x/5)) - 50*x + 250
```

je identický s výrazem přesného řešení (9.6).



Obr. 9.2 Výsledná aproximace funkce $y(x)$ z příkladu 9.1 určená funkcí `ode45`

Příklad 9.3. Stanovte průběh posouvající síly na konzolovém nosníku, schématicky znázorněném na obr. 9.3, Eulerovou metodou. Konkrétní vstupní údaje jsou uvedeny v tabulce 9.1. Výpočetní krok h zvolte $h = 1$, příp. $h = 0,5$. Výslednou aproximaci porovnejte s přesným řešením.

Spojité silové zatížení q_z :	4 kN/m
Rozpětí konzolového nosníku l :	6 m

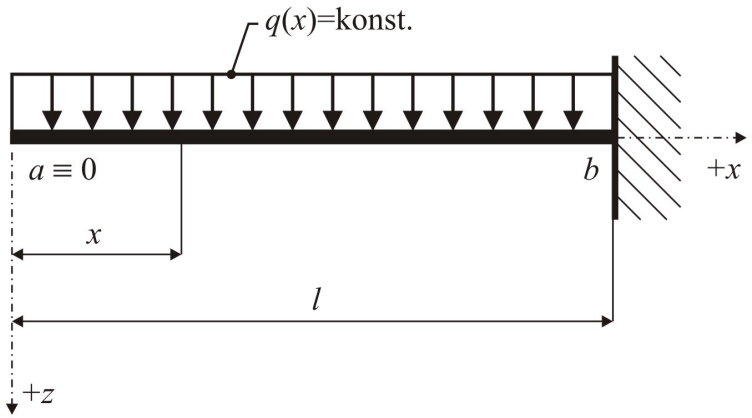
Tab. 9.1 Vstupní údaje příkladu 9.3

Řešení. Obyčejná diferenciální rovnice prvního řádu vyplývá ze Schwedlerových vztahů:

$$\frac{V_z(x)}{\mathrm{d}x} = -q_z(x) = \text{konst} \rightarrow y'(x) = -q_z \cdot x^0 \ . \tag{9.7}$$

Počáteční podmínka vychází ze statické okrajové podmínky, udávající nulovou hodnotu posouvající síly na volném okraji konzolového nosníku, tedy:

$$V_z(x = 0) = y(x = 0) = 0 \ . \tag{9.8}$$



Obr. 9.3 Statické schéma řešeného staticky určitého konzolového nosníku

Výpočet aproximace průběhu posouvající síly Eulerovou metodou je založen na rekurentním vztahu (9.4). Přesné řešení odpovídá analyticky odvozené rovnici pro posouvající sílu $V_z(x)$:

$$V_z(x) = -q_z \cdot x \ . \tag{9.9}$$

Vzhledem ke skutečnosti, že funkce posouvající síly je lineární, lze v tomto případě získat Eulerovou metodou přesné řešení:

x	y	yp	y-yp
0.0000	0.0000	0.0000	0.0000
0.5000	-2.0000	-2.0000	0.0000
1.0000	-4.0000	-4.0000	0.0000
1.5000	-6.0000	-6.0000	0.0000
2.0000	-8.0000	-8.0000	0.0000
2.5000	-10.0000	-10.0000	0.0000
3.0000	-12.0000	-12.0000	0.0000
3.5000	-14.0000	-14.0000	0.0000
4.0000	-16.0000	-16.0000	0.0000
4.5000	-18.0000	-18.0000	0.0000
5.0000	-20.0000	-20.0000	0.0000
5.5000	-22.0000	-22.0000	0.0000
6.0000	-24.0000	-24.0000	0.0000

▲

Příklad 9.4. Určete na konzolovém nosníku z příkladu 9.3 průběh ohybových momentů Eulerovou metodou. Výpočetní krok h zvolte $h = 1$, příp. $h = 0,5$. Výslednou aproximaci porovnejte s přesným řešením.

Řešení. Obyčejná diferenciální rovnice prvního řádu vyplývá opět ze Schwedlerových vztahů:

$$\frac{M_y(x)}{\mathrm{d}x} = V_z(x) \to y'(x) = -q_z \cdot x \,. \tag{9.10}$$

Počáteční podmínka vychází ze statické okrajové podmínky, udávající nulovou hodnotu ohybového momentu na volném okraji konzolového nosníku, tedy:

$$M_y(x = 0) = y(x = 0) = 0 \,. \tag{9.11}$$

Výpočet aproximace průběhu ohybového momentu Eulerovou metodou je založen na rekurentním vztahu (9.4). Přesné řešení odpovídá analyticky odvozené rovnici pro ohybový moment $M_y(x)$:

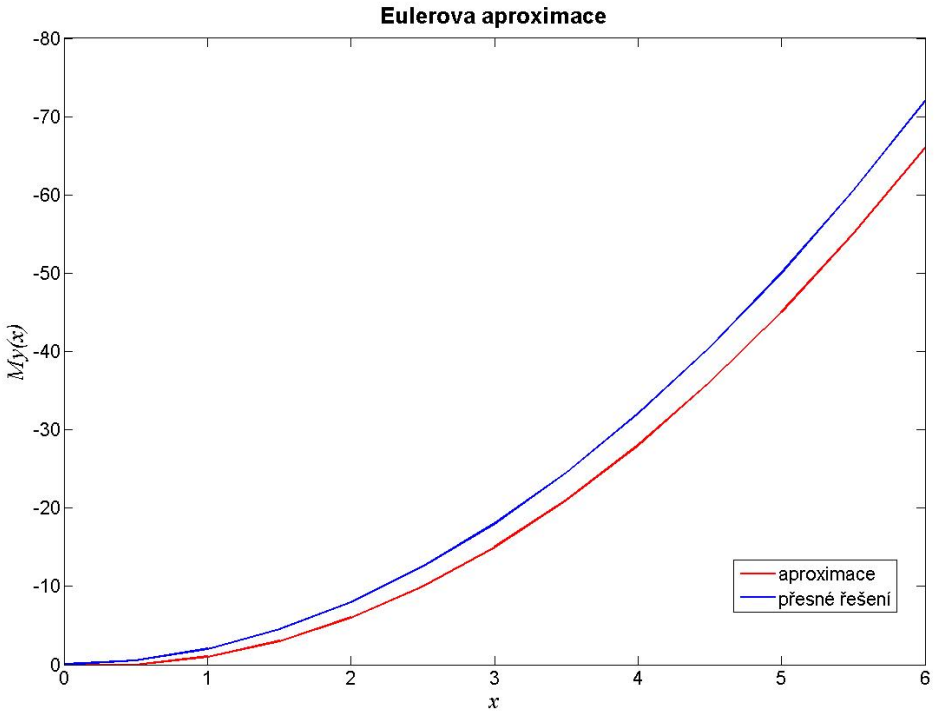
$$M_y(x) = -\frac{q_z \cdot x^2}{2} \,. \tag{9.12}$$

Pro výpočetní krok $h = 0,5$ lze Eulerovou metodou získat tyto výsledky:

x	y	yp	y-yp
0.0000	0.0000	0.0000	0.0000
0.5000	0.0000	-0.5000	0.5000
1.0000	-1.0000	-2.0000	1.0000
1.5000	-3.0000	-4.5000	1.5000
2.0000	-6.0000	-8.0000	2.0000
2.5000	-10.0000	-12.5000	2.5000
3.0000	-15.0000	-18.0000	3.0000
3.5000	-21.0000	-24.5000	3.5000
4.0000	-28.0000	-32.0000	4.0000
4.5000	-36.0000	-40.5000	4.5000
5.0000	-45.0000	-50.0000	5.0000
5.5000	-55.0000	-60.5000	5.5000
6.0000	-66.0000	-72.0000	6.0000

Průběh vypočtených ohybových momentů je pak zobrazen na obrázku 9.4.

▲



Obr. 9.4 Výsledná aproximace ohybových momentů na konzolovém nosníku z příkladu 9.3

9.1.2 Metoda Runge-Kutta

Metody založené na výpočetním postupu Runge-Kutta jsou vhodné pro řešení obyčejných diferenciálních rovnic tvaru (9.1). Tyto výpočetní techniky byly vyvinuty na přelomu 19. a 20. století německými matematiky Carlem Rungem a Martinem Wilhelmem Kuttou. Lze je vyjádřit obecným rekurentním vztahem:

$$y_{i+1} = y_i + h \cdot \sum_{i=1}^{s} (b_i \cdot k_i) \ , \tag{9.13}$$

kde koeficienty k_i jsou dány obecným vztahem:

$$k_i = f(x_i + c_i \cdot h, y_i + h \cdot \sum_{j=1}^{i-1} (a_{i,j} \cdot k_j)) \ . \tag{9.14}$$

Koeficienty $a_{i,j}$, b_i a c_i pro $i, j = 1, \ldots, s$ jsou uvedeny pro Kuttovu metodu třetího řádu ($s = 3$) v tabulce 9.2 a pro klasickou metodu Runge-Kutta čtvrtého řádu ($s = 4$) v tabulce 9.3.

$c_1 = 0$	$a_{1,1} = 0$	$a_{1,2} = 0$	$a_{1,3} = 0$
$c_2 = \frac{1}{2}$	$a_{2,1} = \frac{1}{2}$	$a_{2,2} = 0$	$a_{2,3} = 0$
$c_3 = 1$	$a_{3,1} = -1$	$a_{3,2} = 2$	$a_{3,3} = 0$
	$b_1 = \frac{1}{6}$	$b_2 = \frac{2}{3}$	$b_3 = \frac{1}{6}$

Tab. 9.2 Koeficienty $a_{i,j}$, b_i a c_i Kuttovy metody

$c_1 = 0$	$a_{1,1} = 0$	$a_{1,2} = 0$	$a_{1,3} = 0$	$a_{1,4} = 0$
$c_2 = \frac{1}{2}$	$a_{2,1} = \frac{1}{2}$	$a_{2,2} = 0$	$a_{2,3} = 0$	$a_{2,4} = 0$
$c_3 = \frac{1}{2}$	$a_{3,1} = 0$	$a_{3,2} = \frac{1}{2}$	$a_{3,3} = 0$	$a_{3,4} = 0$
$c_4 = 1$	$a_{4,1} = 0$	$a_{4,2} = 0$	$a_{4,3} = 1$	$a_{4,4} = 0$
	$b_1 = \frac{1}{6}$	$b_2 = \frac{1}{3}$	$b_3 = \frac{1}{3}$	$b_4 = \frac{1}{6}$

Tab. 9.3 Koeficienty $a_{i,j}$, b_i a c_i klasické metody Runge-Kutta

Poznámka 9.5. Pomocí vztahů (9.13) a (9.14) lze vyjádřit i rekurentní výraz (9.4) pro výpočet Eulerovou metodou, která je řádu $s = 1$. Příslušné koeficienty $a_{i,j}$, b_i a c_i jsou obsaženy v tabulce 9.4.

$c_1 = 0$	$a_{1,1} = 0$
	$b_1 = 1$

Tab. 9.4 Koeficienty $a_{i,j}$, b_i a c_i Eulerovy metody

Z principu metody Runge-Kutta vychází řada adaptivních metod. Jedná se např. o metodu Heun-Euler ($s = 2$, tabulka 9.5), metodu Ralstonovu ($s = 3$, tabulka 9.6) nebo metodu Bogacki-Shampine ($s = 4$, tabulka 9.7).

$c_1 = 0$	$a_{1,1} = 0$	$a_{1,2} = 0$
$c_2 = 1$	$a_{2,1} = 1$	$a_{2,2} = 0$
	$b_1 = 1$	$b_2 = 0$

Tab. 9.5 Koeficienty $a_{i,j}$, b_i a c_i metody Heun-Euler

$c_1 = 0$	$a_{1,1} = 0$	$a_{1,2} = 0$	$a_{1,3} = 0$
$c_2 = \frac{1}{2}$	$a_{2,1} = \frac{1}{2}$	$a_{2,2} = 0$	$a_{2,3} = 0$
$c_3 = \frac{3}{4}$	$a_{3,1} = 0$	$a_{3,2} = -\frac{3}{4}$	$a_{3,3} = 0$
	$b_1 = \frac{2}{9}$	$b_2 = \frac{3}{9}$	$b_3 = \frac{4}{9}$

Tab. 9.6 Koeficienty $a_{i,j}$, b_i a c_i Ralstonovy metody

$c_1 = 0$	$a_{1,1} = 0$	$a_{1,2} = 0$	$a_{1,3} = 0$	$a_{1,4} = 0$
$c_2 = \frac{1}{2}$	$a_{2,1} = \frac{1}{2}$	$a_{2,2} = 0$	$a_{2,3} = 0$	$a_{2,4} = 0$
$c_3 = \frac{3}{4}$	$a_{3,1} = 0$	$a_{3,2} = \frac{3}{4}$	$a_{3,3} = 0$	$a_{3,4} = 0$
$c_4 = 1$	$a_{4,1} = \frac{2}{9}$	$a_{4,2} = \frac{1}{3}$	$a_{4,3} = \frac{4}{9}$	$a_{4,4} = 0$
	$b_1 = \frac{7}{24}$	$b_2 = \frac{1}{4}$	$b_3 = \frac{1}{3}$	$b_4 = \frac{1}{8}$

Tab. 9.7 Koeficienty $a_{i,j}$, b_i a c_i metody Bogacki-Shampine

Příklad 9.6. Metodou Runge-Kutta stanovte přibližné řešení obyčejné diferenciální rovnice z příkladu 9.1:

$$y'(x) = x^2 - 0,2 \cdot y(x) \,, \tag{9.15}$$

v intervalu $\langle -2; 3\rangle$ s počáteční podmínkou $y(-2) = -1$. Výpočetní krok h postupně volte $h = 1$, $h = 0,5$, $h = 0,1$ příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením.

Řešení. Vztahy (9.13) a (9.14), popisující podstatu metody Runge-Kutta, mohou být s využitím hodnot koeficientů $a_{i,j}$, b_i a c_i z tabulky 9.3 aplikovány pro velikost výpočetního kroku $h = 1$ např. následujícím způsobem:

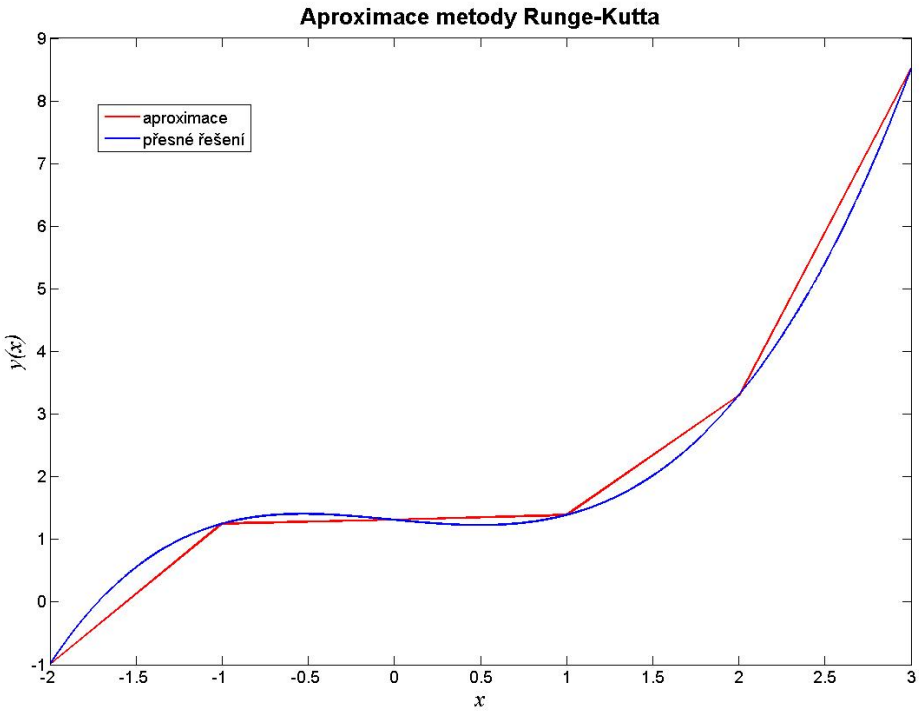
```
f=inline('x^2-0.2*y');
a=-2; b=3;
c=-1;
h=1;
n=(b-a)/h;
x(1)=a; y(1)=c;
for i=1:n
  x(i+1)=x(i)+h;
  K1=h*f(x(i),y(i));
  K2=h*f(x(i)+h/2,y(i)+K1/2);
  K3=h*f(x(i)+h/2,y(i)+K2/2);
  K4=h*f(x(i)+h,y(i)+K3);
  y(i+1)=y(i)+(K1+2*K2+2*K3+K4)/6;
end
[x' y'], plot(x,y,'r');
```

Porovnají-li se dosažené výsledky s přesným řešením (viz obrázek 9.5):

x	y	yp	y-yp
-2.0000	-1.0000	-1.0000	0.0000
-1.0000	1.2508	1.2509	-0.0001
0.0000	1.3112	1.3113	-0.0001
1.0000	1.3910	1.3909	0.0001
2.0000	3.2994	3.2990	0.0004
3.0000	8.5175	8.5167	0.0008

ukáže se podstatně větší přesnost vypočtených aproximací, nežli tomu bylo v případě Eulerovy metody.

▲



Obr. 9.5 Výsledná aproximace metodou Runge-Kutta

Příklad 9.7. Stanovte přibližné řešení obyčejné diferenciální rovnice z příkladu 9.1:

$$y'(x) = x^2 - 0,2 \cdot y(x) \,, \tag{9.16}$$

v intervalu $\langle -2; 3\rangle$ s počáteční podmínkou $y(-2) = -1$ s využitím ostatních metod, vycházejících z klasické metody Runge-Kutta, tedy Kuttovy metody třetího řádu, metody Heun-Euler, Ralstonovy metody a metody Bogacki-Shampine. Výpočetní krok h postupně volte $h = 1$, $h = 0,5$, $h = 0,1$ příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením.

Příklad 9.8. Určete na konzolovém nosníku z příkladu 9.3 průběh ohybových momentů metodou Runge-Kutta. Výpočetní krok h zvolte $h = 1$, příp. $h = 0,5$. Výslednou aproximaci porovnejte s přesným řešením.

Řešení. Výpočet aproximace průběhu ohybových momentů metodou Runge-Kutta vychází z rekurentních vztahů (9.13) a (9.14) a příslušných hodnot koeficientů $a_{i,j}$, b_i a c_i z tabulky 9.3.

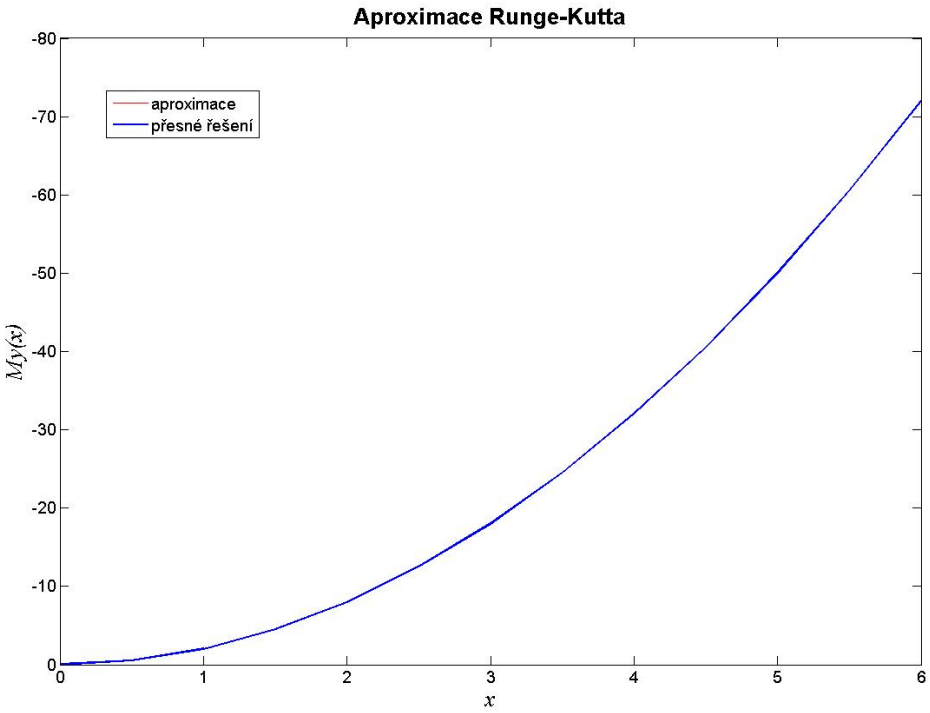
Pro výpočetní krok $h = 0,5$ lze metodou Runge-Kutta získat oproti Eulerovy metody podstatně přesnější výsledky:

x	y	yp	y-yp
0.0000	0.0000	0.0000	0.0000
0.5000	-0.5000	-0.5000	0.0000
1.0000	-2.0000	-2.0000	0.0000
1.5000	-4.5000	-4.5000	0.0000
2.0000	-8.0000	-8.0000	0.0000
2.5000	-12.5000	-12.5000	0.0000
3.0000	-18.0000	-18.0000	0.0000
3.5000	-24.5000	-24.5000	0.0000
4.0000	-32.0000	-32.0000	0.0000
4.5000	-40.5000	-40.5000	0.0000
5.0000	-50.0000	-50.0000	0.0000
5.5000	-60.5000	-60.5000	0.0000
6.0000	-72.0000	-72.0000	0.0000

Průběh vypočtených ohybových momentů je pak zobrazen na obrázku 9.6.

▲

Příklad 9.9. Určete na konzolovém nosníku z příkladu 9.3 průběh ohybových momentů s využitím ostatních metod, vycházejících z klasické metody Runge-Kutta, tedy Kuttovy metody třetího řádu, metody Heun-Euler, Ralstonovy metody a metody Bogacki-Shampine. Výpočetní krok h postupně volte $h = 1$, $h = 0,5$, $h = 0,1$ příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením.



Obr. 9.6 Výsledná aproximace ohybových momentů na konzolovém nosníku z příkladu 9.8

9.1.3 Metoda skákající žáby

Metoda skákající žáby je příkladem dvoukrokové metody, jenž vychází z rekurentního vzorce:

$$y_{i+1} = y_{i-1} + 2 \cdot h \cdot f(x_i, y_i) \ . \tag{9.17}$$

K výpočtu hodnoty y_{i+1} je tedy nutno znát hodnoty funkce y_i a y_{i-1} ve dvou předchozích bodech. Při zahájení výpočtu se obě hodnoty v počátečním úseku y_0 a y_1 stanoví z počáteční podmínky a s využitím některé z jednokrokových metod.

Název metody skákající žáby vystihuje skutečnost, že hodnota vypočtené aproximace osciluje okolo řešení přesného.

Příklad 9.10. Metodou skákající žáby stanovte přibližné řešení obyčejné diferenciální rovnice z příkladu 9.1:

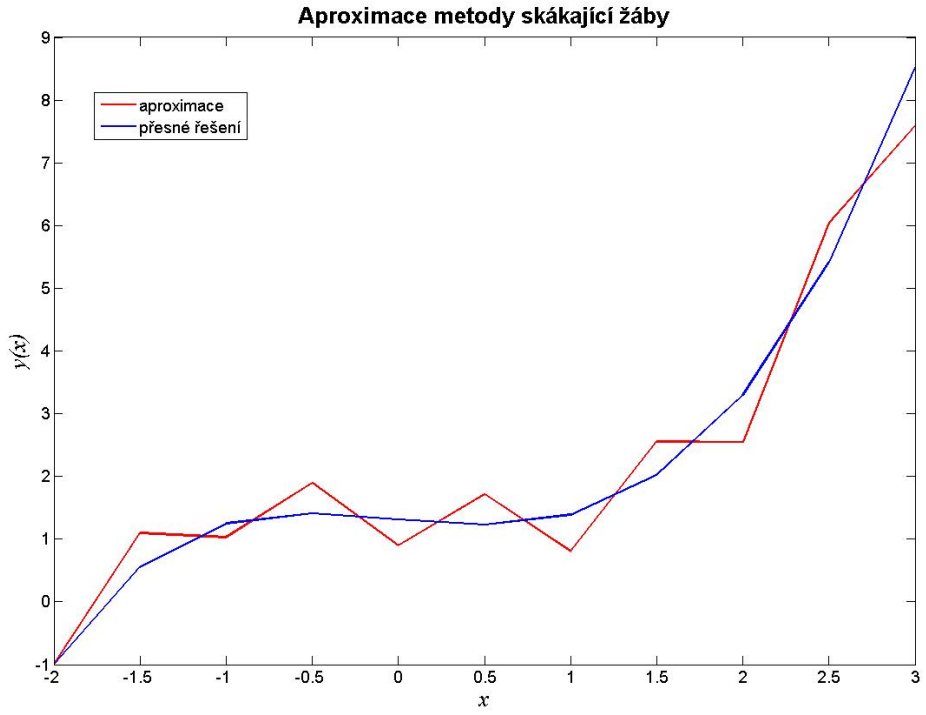
$$y'(x) = x^2 - 0,2 \cdot y(x) \ , \tag{9.18}$$

v intervalu $\langle -2; 3 \rangle$ s počáteční podmínkou $y(-2) = -1$. Pro určení hodnoty funkce ve druhém bodu výpočtu $y(-2 + h)$ použijte Eulerovy metody. Výpočetní krok h postupně volte $h = 1$, $h = 0,5$, $h = 0,1$ příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením.

Řešení. Výpočet metodou skákající žáby je založen na rekurentním vztahu (9.17). Výsledný zdrojový text programu pro velikost výpočetního kroku $h = 0,5$ může vypadat následujícím způsobem:

```
f=inline('x^2-0.2*y');
a=-2; b=3;
c=-1;
h=0.5;
n=(b-a)/h;
x(1)=a; y(1)=c;
x(2)=x(1)+h; y(2)=y(1)+h*f(x(1),y(1));;
for i=2:n
  x(i+1)=x(i)+h;
  y(i+1)=y(i-1)+2*h*f(x(i),y(i));
end
[x' y'], plot(x,y,'r');
```

Na obrázku 9.7 je zobrazena vypočtená aproximace metodou skákající žáby pro výpočetní krok $h = 0,5$. Na obrázku je patrná charakteristická vlastnost výpočetního postupu metody skákající žáby, a to oscilace okolo přesného řešení.



Obr. 9.7 Výsledná aproximace metody skákající žáby s výpočetním krokem $h = 0,5$

Dosažené numerické výsledky i odchylky od přesného řešení jsou tyto:

```
     x          y          yp         y-yp
-------------------------------------------
  -2.0000   -1.0000   -1.0000    0.0000
  -1.5000    1.1000    0.5553    0.5447
  -1.0000    1.0300    1.2509   -0.2209
  -0.5000    1.8940    1.4064    0.4876
   0.0000    0.9012    1.3113   -0.4101
   0.5000    1.7138    1.2271    0.4866
   1.0000    0.8084    1.3909   -0.5824
   1.5000    2.5521    2.0169    0.5352
   2.0000    2.5480    3.2990   -0.7509
   2.5000    6.0425    5.4127    0.6298
   3.0000    7.5895    8.5167   -0.9272
```

▲

9.2 Obyčejné diferenciální rovnice druhého řádu

Podobně jako v případě obyčejné diferenciální rovnice prvního řádu (9.1) lze řešit i obyčejné diferenciální rovnice druhého řádu:

$$y''(x) = f(x, y(x), y'(x)) \; . \tag{9.19}$$

Typů diferenciálních rovnic 2. řádu je mnoho typů. Např. lze řešit obyčejné diferenciální rovnice s konstantními koeficienty a,b a c, které lze vyjádřit ve tvaru:

$$a \cdot y''(x) + b \cdot y'(x) + c \cdot y(x) = f(x) \; . \tag{9.20}$$

Numerické řešení rovnic typu (9.20) spočívá v jejich převedení na soustavu dvou diferenciálních rovnic:

$$z(x) = y'(x) \tag{9.21}$$

a

$$z'(x) = \frac{f(x) - b \cdot z(x) - c \cdot y(x)}{a} \; . \tag{9.22}$$

Nutností řešení jsou v daném případě dvě počáteční podmínky, např.:

$$y(a) = c \tag{9.23}$$

a

$$z(b) = d \; . \tag{9.24}$$

Příklad 9.11. Určete na konzolovém nosníku z příkladu 9.3 průběh ohybových momentů řešením obyčejné diferenciální rovnice 2.řádu, vycházející ze Schwedlerových vztahů:

$$\frac{M_y(x)}{\mathrm{d}x^2} = -q_z(x) = \text{konst} \rightarrow y''(x) = -q_z \cdot x^0 \,. \tag{9.25}$$

Počáteční podmínky vychází ze statických okrajových podmínek, které udávají nulovou hodnotu posouvající síly i ohybového momentu na volném okraji konzolového nosníku, tedy:

$$V_z(x=0) = y'(x=0) = 0 \tag{9.26}$$

a

$$M_y(x=0) = y(x=0) = 0 \,. \tag{9.27}$$

Pro numerické řešení použijte Eulerovu metodu. Výpočetní krok h zvolte $h = 1$, příp. $h = 0,5$. Výslednou aproximaci porovnejte s přesným řešením.

Řešení. Celý výpočetní postup je zřejmý ze zápisu programu s výpočetním krokem $h = 0,5$ do `m-souboru` programu MATLAB:

```
f=inline('-4*x^0');
q=4; a=0; b=6;
c=0; d=0; h=0.5;
n=(b-a)/h;
x(1)=a; y(1)=c;
y2(1)=d;
for i=1:n
  x(i+1)=x(i)+h;
  y(i+1)=y(i)+h*f(x(i));
  y2(i+1)=y2(i)+h*(y(i));
end
[x' y2'], plot(x,y2,'r');
```

Zápis zdrojového textu programu lze vylepšit následujícím způsobem (řešená funkce $y(x)$ i její derivace $y'(x)$ je uložena v jedné proměnné):

```
f=inline('-4*x^0');
q=4; a=0; b=6;
c=0; d=0;
h=0.5; x=a:h:b;
f=inline('-4*x^0');
y(1,:)=[d c];
for i=2:length(x)
   k1=[y(i-1,2) -q];
   y(i,:)=y(i-1,:)+k1*h;
end
[x' y(:,1)], plot(x,y(:,1),'r');
```

Pro výpočetní krok $h = 0,5$ lze Eulerovou metodou získat tyto výsledky:

```
     x          y          yp         y-yp
-----------------------------------------
    0.0000     0.0000     0.0000     0.0000
    0.5000     0.0000    -0.5000     0.5000
    1.0000    -1.0000    -2.0000     1.0000
    1.5000    -3.0000    -4.5000     1.5000
    2.0000    -6.0000    -8.0000     2.0000
    2.5000   -10.0000   -12.5000     2.5000
    3.0000   -15.0000   -18.0000     3.0000
    3.5000   -21.0000   -24.5000     3.5000
    4.0000   -28.0000   -32.0000     4.0000
    4.5000   -36.0000   -40.5000     4.5000
    5.0000   -45.0000   -50.0000     5.0000
    5.5000   -55.0000   -60.5000     5.5000
    6.0000   -66.0000   -72.0000     6.0000
```

▲

Příklad 9.12. Diferenciální rovnici (9.25) z příkladu 9.11 pro určení průběhu ohybových momentů vyřešte s využitím klasické metody Runge-Kutta.

Řešení. Celý výpočetní postup je opět zřejmý ze zápisu programu s výpočetním krokem $h = 0,5$ do `m-souboru` programu MATLAB:

```
f=inline('-4*x^0');
q=4; a=0; b=6; c=0; d=0; h=0.5;
n=(b-a)/h;
x(1)=a; y(1)=c; y2(1)=d;
for i=1:n
K1=f(x(i));
  K2=f(x(i)+h/2);
  K3=f(x(i)+h/2);
  K4=f(x(i)+h);
  y(i+1)=y(i)+h*((K1+K4)/6+(K2+K3)/3);
  K1=y(i);
  K2=(y(i)+y(i+1))/2;
  K3=(y(i)+y(i+1))/2;
  K4=y(i+1);
  y2(i+1)=y2(i)+h*((K1+K4)/6+(K2+K3)/3);
  x(i+1)=x(i)+h;
end
[x' y2'], plot(x,y2,'r');
```

I v tomto případě je možno zápis zdrojového textu programu vylepšit uložením řešená funkce $y(x)$ i její derivace $y'(x)$ do jediné proměnné:

```
f=inline('-4*x^0');
q=4; a=0; b=6;
c=0; d=0;
h=0.5; x=a:h:b;
f=inline('-4*x^0');
y(1,:)=[d c];
for i=2:length(x)
   K1=f(x(i-1));
   K2=f(x(i-1)+h/2);
   K3=f(x(i-1)+h/2);
   K4=f(x(i-1)+h);
   y(i,2)=y(i-1,2)+h*((K1+K4)/6+(K2+K3)/3);
   K1=y(i-1,2);
   K2=(y(i-1,2)+y(i,2))/2;
   K3=(y(i-1,2)+y(i,2))/2;
   K4=y(i,2);
   y(i,1)=y(i-1,1)+h*((K1+K4)/6+(K2+K3)/3);
end
[x' y(:,1)], plot(x,y(:,1),'r');
```

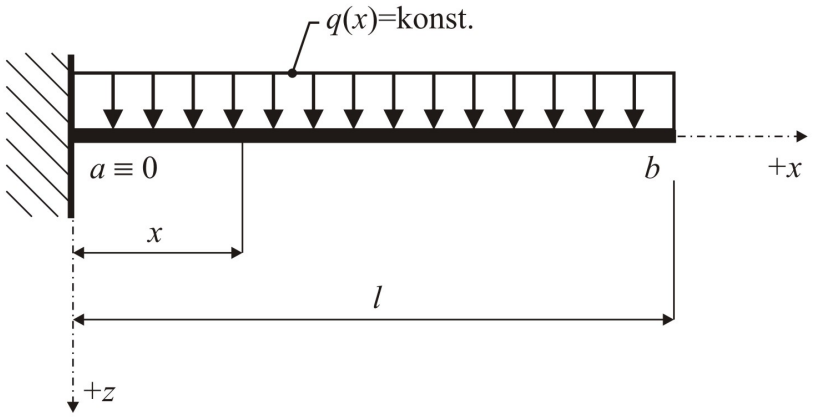
Pro výpočetní krok $h = 0,5$ lze metodou Runge-Kutta získat oproti Eulerově metodě podstatně přesnější výsledky:

```
     x          y          yp         y-yp
-----------------------------------------
    0.0000     0.0000     0.0000     0.0000
    0.5000    -0.5000    -0.5000     0.0000
    1.0000    -2.0000    -2.0000     0.0000
    1.5000    -4.5000    -4.5000     0.0000
    2.0000    -8.0000    -8.0000     0.0000
    2.5000   -12.5000   -12.5000     0.0000
    3.0000   -18.0000   -18.0000     0.0000
    3.5000   -24.5000   -24.5000     0.0000
    4.0000   -32.0000   -32.0000     0.0000
    4.5000   -40.5000   -40.5000     0.0000
    5.0000   -50.0000   -50.0000     0.0000
    5.5000   -60.5000   -60.5000     0.0000
    6.0000   -72.0000   -72.0000     0.0000
```

▲

Příklad 9.13. Diferenciální rovnici (9.25) z příkladu 9.11 pro určení průběhu ohybových momentů vyřešte s využitím ostatních metod, které vycházejí z klasické metody Runge-Kutta, tedy Kuttovy metody třetího řádu, metody Heun-Euler, Ralstonovy metody a metody Bogacki-Shampine.

Příklad 9.14. Stanovte tvar ohybové čáry konzolového nosníku, schématicky znázorněném na obr. 9.8. Konkrétní vstupní údaje jsou uvedeny v tabulce 9.8. K numerickému řešení použijte Eulerovu metodu. Výpočetní krok h zvolte $h = 0,5$, $h = 0,25$, $h = 0,1$, příp. $h = 0,01$. Výslednou aproximaci porovnejte s přesným řešením.



Obr. 9.8 Statické schéma řešeného staticky určitého konzolového nosníku

Spojité silové zatížení q_z :	4 kN/m
Rozpětí konzolového nosníku l :	3 m
Šířka obdélníkového průřezu b :	0, 02 m
Výška obdélníkového průřezu h :	0, 15 m
Moment setrvačnosti I_y :	$\frac{1}{12} \cdot 0,02 \cdot 0,15^3 = 5,625 \cdot 10^{-6}$ m^4
Modul pružnosti v tahu a tlaku E :	$2,1 \cdot 10^{11}$ Pa

Tab. 9.8 Vstupní údaje příkladu 9.14

Řešení. Obyčejná diferenciální rovnice druhého řádu nabývá tvaru:

$$EI_y w_z(x)'' = -M_y(x) \,, \tag{9.28}$$

kde EI_y je ohybová tuhost nosníku (konstantní a nenulová).

Počáteční podmínky vychází z deformačních okrajových podmínek, které udávají nulovou hodnotu průhybu i pootočení ve vetknutí konzolového nosníku, tedy:

$$\varphi_y(x=0) = w_z'(x=0) = 0 \tag{9.29}$$

a

$$w_z(x=0) = y(x=0) = 0\,. \tag{9.30}$$

Z podmínek rovnováhy lze určit nejprve velikost silové reakce ve vetknutí konzoly:

$$R_{a,z} = q_z \cdot l\,(\uparrow)\,, \tag{9.31}$$

momentovou reakci ve vetknutí konzolového nosníku:

$$M_{a,y} = \frac{q_z \cdot l^2}{2}\,(\circlearrowleft)\,, \tag{9.32}$$

a nakonec i samotnou rovnici ohybového momentu:

$$M_y(x) = -\frac{q_z \cdot l^2}{2} + q_z \cdot l \cdot x - \frac{q_z \cdot x^2}{2} = q_z \cdot \left(-\frac{l^2}{2} + l \cdot x - \frac{x^2}{2}\right)\,, \tag{9.33}$$

která figuruje v řešené diferenciální rovnici (9.28).

Výpočet Eulerovou metodou pak lze provést pro výpočetní krok $h = 0,25$ s využitím funkce `horner` (viz kapitola 3.1) a následujícího sledu příkazů:

```
qz=4000; l=3; E=2.1*10^11;
sirka=0.02; vyska=0.15; Iy=1/12*sirka*vyska^3;
M=[-qz/2*l^2 qz*l -qz/2];
a=0; b=l;
c=0; d=0;
h=0.25;
n=(b-a)/h;
x(1)=a; y(1)=c; y2(1)=d;
for i=1:n
  x(i+1)=x(i)+h;
  y(i+1)=y(i)-h*horner(2,M,x(i+1))/(E*Iy);
  y2(i+1)=y2(i)+h*y(i)*1000;
end
[x' y2'], plot(x,y2,'r');
```

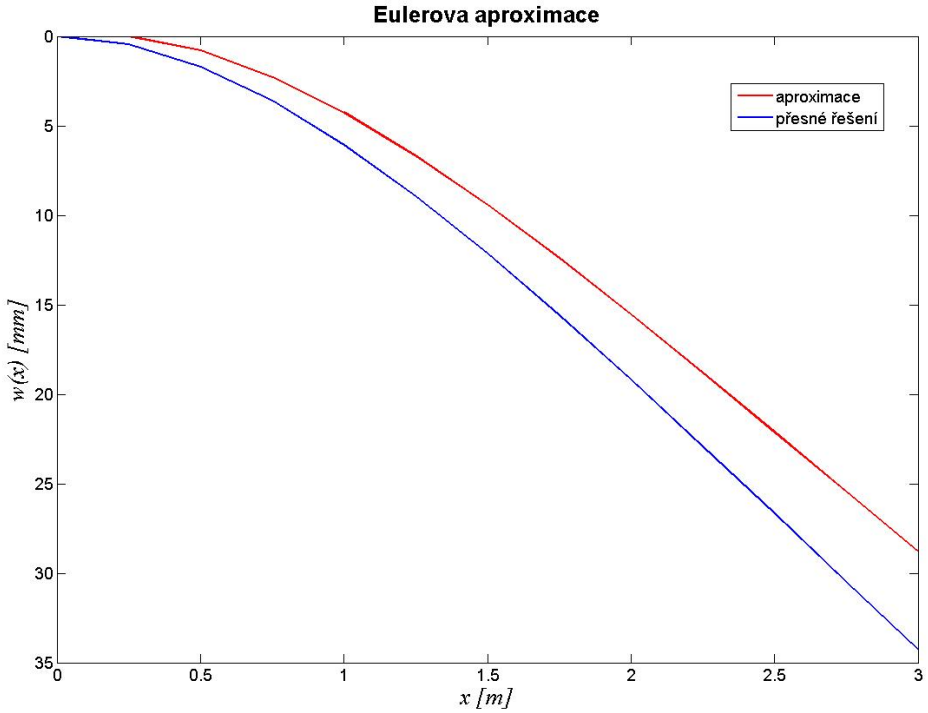
Dosažené výsledky lze porovnat s přesnou hodnotou, vycházející z analyticky určené rovnice ohybové čáry:

$$w_z(x) = \frac{q_z}{EI_y} \cdot \left(\frac{l^2 \cdot x^2}{4} - \frac{l \cdot x^3}{6} + \frac{x^4}{24}\right)\,. \tag{9.34}$$

Pro výpočetní krok $h = 0,25$ pak vycházejí následující numerické hodnoty výsledného řešení:

x	y1	y1p	y[mm]	yp[mm]	y-yp
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.2500	0.0032	0.0035	0.0000	0.4503	-0.4503
0.5000	0.0058	0.0064	0.8003	1.7019	-0.9017
0.7500	0.0080	0.0088	2.2619	3.6161	-1.3542
1.0000	0.0097	0.0107	4.2593	6.0670	-1.8078
1.2500	0.0110	0.0122	6.6799	8.9424	-2.2625
1.5000	0.0119	0.0133	9.4246	12.1429	-2.7183
1.7500	0.0126	0.0141	12.4074	15.5826	-3.1752
2.0000	0.0130	0.0147	15.5556	19.1887	-3.6332
2.2500	0.0133	0.0150	18.8095	22.9018	-4.0923
2.5000	0.0134	0.0152	22.1230	26.6755	-4.5525
2.7500	0.0134	0.0152	25.4630	30.4767	-5.0138
3.0000	0.0134	0.0152	28.8095	34.2857	-5.4762

Na obrázku 9.9 je zobrazena vypočtená aproximace ohybové čáry konzolového nosníku určená Eulerovou metodou pro výpočetní krok $h = 0,25$.



Obr. 9.9 Výsledná aproximace ohybové čáry konzolového nosníku s výpočetním krokem $h = 0,25$

▲

Příklad 9.15. Určete aproximaci ohybové čáry konzolového nosníku z příkladu 9.14 metodou Runge-Kutta.

Řešení. Výpočet aproximace ohybové čáry může být proveden `m-skriptem`:

```
qz=4000; l=3; E=2.1*10^11; sirka=0.02; vyska=0.15;
Iy=1/12*sirka*vyska^3; M=[-qz/2*l^2 qz*l -qz/2];
a=0; b=l; h=0.25; n=(b-a)/h;
c=0; d=0; x(1)=a; y(1)=c; y2(1)=d;
for i=1:n
  x(i+1)=x(i)+h;
  K1=horner(2,M,x(i))/(E*Iy);
  K2=horner(2,M,x(i)+h/2)/(E*Iy);
  K3=horner(2,M,x(i)+h/2)/(E*Iy);
  K4=horner(2,M,x(i)+h)/(E*Iy);
  y(i+1)=y(i)-h*((K1+K4)/6+(K2+K3)/3);
  K1=y(i)*1000;
  K2=(y(i)+y(i+1))*1000/2;
  K3=(y(i)+y(i+1))*1000/2;
  K4=y(i+1)*1000;
  y2(i+1)=y2(i)+h*((K1+K4)/6+(K2+K3)/3);
end
[x' y2'], plot(x,y2,'r');
```

Pro výpočetní krok $h = 0,5$ lze metodou Runge-Kutta získat oproti Eulerově metodě podstatně přesnější výsledky:

x	y1	y1p	y[mm]	yp[mm]	y-yp
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.2500	0.0035	0.0035	0.4376	0.4503	-0.0127
0.5000	0.0064	0.0064	1.6777	1.7019	-0.0243
0.7500	0.0088	0.0088	3.5813	3.6161	-0.0347
1.0000	0.0107	0.0107	6.0229	6.0670	-0.0441
1.2500	0.0122	0.0122	8.8900	8.9424	-0.0524
1.5000	0.0133	0.0133	12.0833	12.1429	-0.0595
1.7500	0.0141	0.0141	15.5170	15.5826	-0.0656
2.0000	0.0147	0.0147	19.1182	19.1887	-0.0705
2.2500	0.0150	0.0150	22.8274	22.9018	-0.0744
2.5000	0.0152	0.0152	26.5983	26.6755	-0.0772
2.7500	0.0152	0.0152	30.3979	30.4767	-0.0788
3.0000	0.0152	0.0152	34.2063	34.2857	-0.0794

▲

Příklad 9.16. Určete aproximaci ohybové čáry konzolového nosníku z příkladu 9.14 s využitím ostatních metod, vycházejících z klasické metody Runge-Kutta, tedy Kuttovy metody třetího řádu, metody Heun-Euler, Ralstonovy metody a metody Bogacki-Shampine.

Příklad 9.17. Stanovte přibližné řešení obyčejné diferenciální rovnice druhého řádu:

$$y''(t) + 100 \cdot y'(t) + 10000 \cdot y(t) = 10000 \cdot |\sin(377 \cdot t)| \tag{9.35}$$

v intervalu $\langle 0; 0,08\rangle$ s počátečními podmínkami $y(0) = 0$ a $y'(0) = 0$. Pro numerické řešení použijte Eulerovu metodu a klasickou metodu Runge-Kutta 4. řádu. Výpočetní krok h postupně volte $h = 0,01$, $h = 0,0025$ a $h = 0,0001$. Výslednou aproximaci porovnejte s řešením pomocí funkce `ode45` programu MATLAB .

Řešení. Programový systém MATLAB nedisponuje funkcemi pro řešení obyčejných diferenciálních rovnic druhého řádu. Řešení je založeno na převedení úlohy na soustavu dvou diferenciálních rovnic popsaných vztahy (9.21) a (9.22). V případě řešené úlohy se daný rozklad provede s využitím samostatné `m-funkce`, např.:

```
function y=fce(t,y);
y=[y(2);-100*y(2)-10000*y(1)+10000*abs(sin(377*t))];
```

na kterou se v `m-skriptu` lze odkázat například takto:

```
t0=[0 0]; interval=[0,0.08]; options = odeset('AbsTol',1e-9);
[t,y]=ode45(@fce,interval,t0,options); plot(t,y(:,1));
```

Program, který provede výpočet řešené diferenciální rovnice 2. řádu Eulerovou metodou, klasickou metodou Runge-Kutta čtvrtého řádu a funkcí `ode45` programu MATLAB pak pro výpočetní krok $h = 0,0025$ vypadá následovně:

```
a=0; b=0.08; h=0.0025; n=(b-a)/h;
c=0; d=0; x(1)=a; ye(1)=c; yrk(1)=c; y2e(1)=d; y2rk(1)=d;
for i=1:n
  x(i+1)=x(i)+h;
  ye(i+1)=ye(i)+h*(1E4*abs(sin(377*x(i)))-100*ye(i)-1E4*y2e(i));
  y2e(i+1)=y2e(i)+h*(ye(i));
  K1=h*(1E4*abs(sin(377*x(i)))-100*yrk(i)-1E4*y2rk(i));
  K2=h*(1E4*abs(sin(377*(x(i)+h/2)))-100*(yrk(i)+K1/2)-1E4*y2rk(i));
  K3=h*(1E4*abs(sin(377*(x(i)+h/2)))-100*(yrk(i)+K2/2)-1E4*y2rk(i));
  K4=h*(1E4*abs(sin(377*(x(i)+h)))-100*(yrk(i)+K3)-1E4*y2rk(i));
  yrk(i+1)=yrk(i)+(K1+2*K2+2*K3+K4)/6;
  K1=yrk(i); K2=(yrk(i)+yrk(i+1))/2;
  K3=(yrk(i)+yrk(i+1))/2; K4=yrk(i+1);
  y2rk(i+1)=y2rk(i)+h*((K1+K4)/6+(K2+K3)/3);
end
```

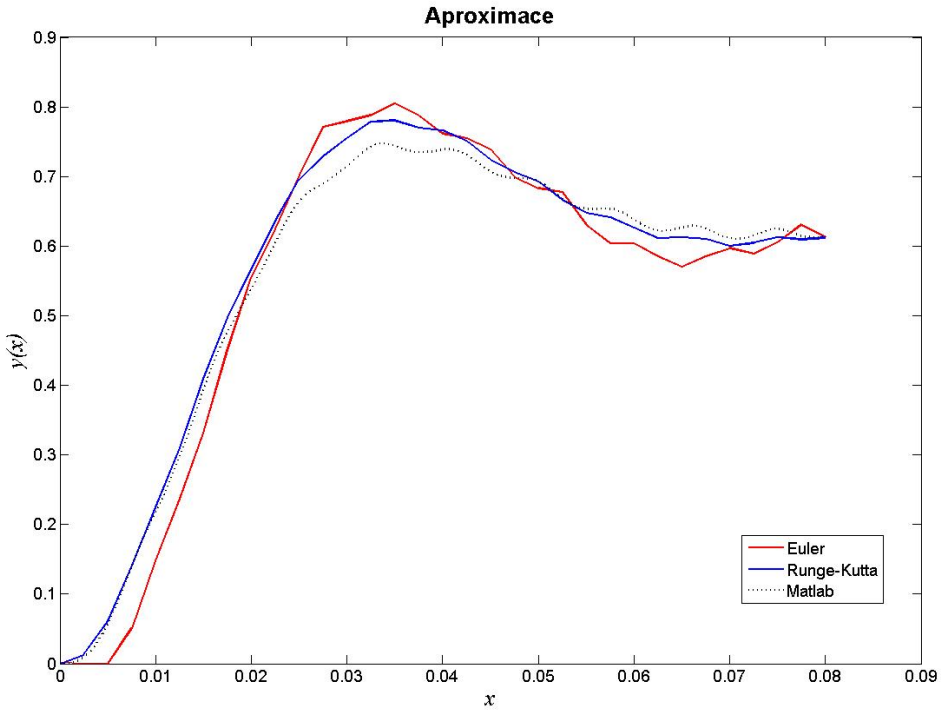
```
t0=[0 0]; interval=[0,0.08]; options=odeset('AbsTol',1e-9);
[xm,y2m]=ode45(@fce,interval,t0,options);
[x' y2e' y2rk'], plot(x,y2e,'r',x,y2rk,'b',xm,y2m(:,1),'k');
```

Numerické výsledky, získané Eulerovou metodou a metodou Runge-Kutta pro výpočetní krok $h = 0,0025$, jsou následující:

x	y_eul	y_rk
0.0000	0.0000	0.0000
0.0025	0.0000	0.0126
0.0050	0.0000	0.0610
0.0075	0.0506	0.1415
0.0100	0.1479	0.2242
0.0125	0.2371	0.3098
0.0150	0.3315	0.4091
0.0175	0.4499	0.4969
0.0200	0.5548	0.5663
0.0225	0.6246	0.6364
0.0250	0.7018	0.6956
0.0275	0.7712	0.7291
0.0300	0.7794	0.7547
0.0325	0.7879	0.7785
0.0350	0.8050	0.7803
0.0375	0.7879	0.7698
0.0400	0.7615	0.7662
0.0425	0.7549	0.7513
0.0450	0.7392	0.7238
0.0475	0.6995	0.7071
0.0500	0.6830	0.6923
0.0525	0.6774	0.6663
0.0550	0.6306	0.6475
0.0575	0.6037	0.6413
0.0600	0.6036	0.6266
0.0625	0.5850	0.6112
0.0650	0.5702	0.6125
0.0675	0.5849	0.6100
0.0700	0.5971	0.6003
0.0725	0.5890	0.6048
0.0750	0.6051	0.6128
0.0775	0.6308	0.6095
0.0800	0.6124	0.6122

Na obrázku 9.10 jsou zobrazeny vypočtené aproximace Eulerovou metodou, me-

todou Runge-Kutta i funkcí `ode45` programového systému MATLAB pro výpočetní krok $h = 0,0025$.



Obr. 9.10 Srovnání dosažených aproximací Eulerovou metodou, metodou Runge-Kutta a funkcí `ode45` pro výpočetní krok $h = 0,0025$

▲

Kapitola 10

Interpolace a aproximace

Cíle

Kapitola by měla sloužit:

- k vysvětlení pojmů interpolace a aproximace,
- k uplatnění algoritmů pro interpolaci a aproximaci v inženýrských úlohách.

Úlohou **interpolace** je například:

- najít k funkci f_x mnohočlen $\Phi_n(x)$ n-tého stupně, který nabývá pro $n+1$ argumentů x_k, kde $k = 0, 1, \ldots, n$ týchž hodnot jako funkce f_x.

- počítat z tabulky funkce f_x sestavené pro $x = x_k$, přibližné hodnoty f_x pomocí mnohočlenu $\Phi_n(x)$ pro body x, které jsou různé od uzlových bodů x_i.

Nejsou-li dané hodnoty funkce y_i $(i = 0, 1, \ldots, n)$ v uzlových bodech x_0, x_1 až x_n dány přesně (jsou získány například měřením, které je vždy zatíženo chybou), nemá význam, aby se hledaná funkce ztotožnila s funkcí přesně v uzlových bodech, jako v případě interpolace. Úlohou **aproximace** je tedy nalezení jednodušší a matematicky přesně definované spojité aproximační funkce F_x v intervalu $\langle a, b\rangle$, která by co nejlépe přiléhala k empirickým bodům $x_0, x_1, \ldots, x_n$.

10.1 Lineární interpolace

Lineární interpolace umožňuje nahradit průběh funkce mezi dvěma body o souřadnicích x_k, y_k a x_{k+1}, y_{k+1} úsečkou, která je definovaná rovnicí přímky:

$$\frac{y(x) - y_k}{x - x_k} = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \,. \tag{10.1}$$

Pro úpravě (10.1) lze získat rovnici pro výpočet $y(x)$ s parametrem x:

$$y(x) = \frac{y_k \cdot (x - x_{k+1}) - y_{k+1} \cdot (x - x_k)}{x_k - x_{k+1}} \,. \tag{10.2}$$

Příklad 10.1. S využitím lineární interpolace pro dva body o souřadnicích $[x_0, y_0] = [1, 1.8]$ a $[x_1, y_1] = [2, 2.27]$ stanovte hodnotu interpolační funkce $y(x = 1.5)$.

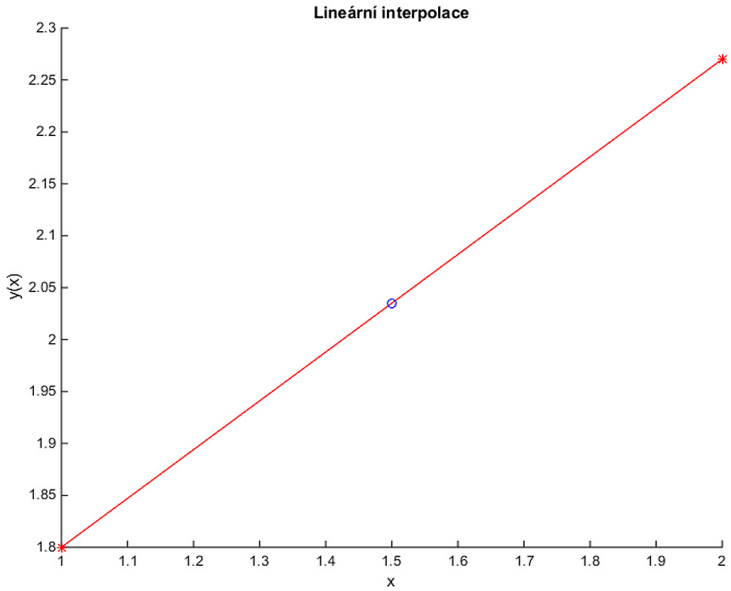
Řešení. Funkce pro lineární interpolaci v souboru `lin_interpol.m` může vypadat např. takto:

```
function y=lin_interpol(x,xy2)
y=(xy2(1,2)*(x-xy2(2,1))-xy2(2,2)*(x-xy2(1,1)))/(xy2(1,1)-xy2(2,1));
```

Při vyvolání této funkce `y=lin_interpol(x,sour_xy_2b)` s parametry `x=1.5` a `sour_xy_2b=[1  1.8; 2 2.27]` lze získat výsledek:

```
y =
   2.0350
```

Lze rovněž znázornit graficky - viz obr. 10.1.



Obr. 10.1 Výsledná lineární interpolace pro bod se souřadnicí $x = 1.5$, který je označen kroužkem

▲

10.2 Lagrangeova interpolace

Pokud $f(x)$ je reálná funkce definovaná v intervalu $\langle a, b\rangle$, lze uvažovat také o funkci:

$$\Phi(x) = a_0 \cdot \varphi_0(x) + a_1 \cdot \varphi_1(x) + a_2 \cdot \varphi_2(x) + \ldots + a_i \cdot \varphi_i(x) + \ldots + a_n \cdot \varphi_n(x) \ , \quad (10.3)$$

kde a_i jsou reálné koeficienty a $\varphi_i(x)$ je rovna x^i pro $i = 0, 1, \ldots, n$. Řešením je pak nalezení interpolačního polynomu $\Phi(x)$, pro který platí:

$$\Phi(x_i) = f(x_i) \ , \quad (10.4)$$

kde x_i se nachází v intervalu $\langle a, b\rangle$ pro $i = 0, 1, 2, \ldots, n$. Znamená to, že hledaná funkce interpolačního polynomu $\Phi(x)$ by měla mít totožné hodnoty s danou funkcí $f(x)$ pro $n + 1$ vstupních parametrů $x_0, x_1, x_2, \ldots, x_n$.

Uvedený problém lze řešit např. postupným dosazováním $x = x_i$, $i = 0, 1, 2$ až n do rovnice (10.4), čímž lze získat soustavu $n + 1$ lineárních rovnic s neznámými koeficienty a_i:

$$\begin{array}{ccccccccc} a_0 \cdot \varphi(x_0) & + & a_1 \cdot \varphi(x_0) & + & \ldots & + & a_n \cdot \varphi(x_0) & = & f(x_0) \\ a_0 \cdot \varphi(x_1) & + & a_1 \cdot \varphi(x_1) & + & \ldots & + & a_n \cdot \varphi(x_1) & = & f(x_1) \\ & & & & \vdots & & & & \\ a_0 \cdot \varphi(x_n) & + & a_1 \cdot \varphi(x_n) & + & \ldots & + & a_n \cdot \varphi(x_n) & = & f(x_n) \end{array} \ . \quad (10.5)$$

Jeden ze způsobů, jak se při určování interpolačního polynomu $\Phi(x_i)$ řešení zmiňované soustavy lineárních rovnic (10.5) vyhnout, je Lagrangeova metoda.

Poznámka 10.2. I když je metoda pojmenovaná podle Josepha Louise Lagrange, který ji publikoval v roce 1795, byla poprvé objevena v roce 1779 Edwardem Waringem a její důsledky částečně zveřejněny v roce 1783 Leonhardem Eulerem.

Pokud je v intervalu $\langle a, b\rangle$ dáno $n + 1$ různých uzlových bodů $x_0, x_1, x_2, \ldots, x_n$ a hodnoty funkce $y_i = f(x_i)$ pro $i = 0, 1, \ldots, n$, pak lze sestrojit interpolační polynom stupně nejvýše n-tého, pro který bude platit:

$$\Phi_n(x) = P_0(x) + P_1(x) + \ldots + P_n(x) = y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + \ldots y_n \cdot l_n(x) \ . \quad (10.6)$$

Pro $l_i(x)$ platí:

$$l_i(x_j) = \begin{cases} 1 & \text{pro } i = j \\ 0 & \text{pro } i \neq j \end{cases} \ . \quad (10.7)$$

Tuto podmínku splňuje polynom:

$$\begin{aligned} l_i(x) &= \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \\ &= \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdot \ldots \cdot \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdot \ldots \cdot \frac{x - x_n}{x_i - x_n} \ . \end{aligned} \quad (10.8)$$

Výsledný tvar Lagrangeova interpolačního polynomu je pak:

$$\begin{aligned} L_n(x) = y_0 \cdot \left(\frac{x-x_1}{x_0-x_1} \cdot \frac{x-x_2}{x_0-x_2} \cdot \ldots \cdot \frac{x-x_n}{x_0-x_n} \right) + \\ + y_1 \cdot \left(\frac{x-x_0}{x_1-x_0} \cdot \frac{x-x_2}{x_1-x_2} \cdot \ldots \cdot \frac{x-x_n}{x_1-x_n} \right) + \ldots \\ + y_i \cdot \left(\frac{x-x_0}{x_i-x_0} \cdot \frac{x-x_1}{x_i-x_1} \cdot \ldots \cdot \frac{x-x_{i-1}}{x_i-x_{i-1}} \cdot \frac{x-x_{i+1}}{x_i-x_{i+1}} \cdot \ldots \cdot \frac{x-x_n}{x_i-x_n} \right) + \ldots \\ + y_n \cdot \left(\frac{x-x_0}{x_n-x_0} \cdot \frac{x-x_1}{x_n-x_1} \cdot \ldots \cdot \frac{x-x_{n-1}}{x_n-x_{n-1}} \right) . \end{aligned} \tag{10.9}$$

Funkci, která stanoví pro zadaný bod se souřadnicí x ve vstupním parametru `par` hodnotu Lagrangeova interpolačního polynomu, sestaveného pro zadanou množinu bodů se souřadnicemi x a y uloženou ve vstupních parametrech s vektory `x` a `y`, lze naprogramovat v MATLABU např. pomocí skriptu `lagrange.m`:

```
function s=lagrange(x,y,par)
n=length(x);
s=0;
for i=1:n
  m=y(i);
  for j=1:n
    if ~(i==j)
      m=m*(par-x(j))/(x(i)-x(j));
    end
  end
  s=s+m;
end
```

Příklad 10.3. S využitím Lagrangeova interpolačního polynomu pro tři body o souřadnicích $[x_0, y_0] = [0, 1]$, $[x_1, y_1] = [2, 2]$ a $[x_2, y_2] = [3, 4]$ stanovte rovnici interpolační funkce $y(x)$.

Řešení. Uvedený příklad lze řešit obecně dosazením zadaných souřadnic tří bodů do

obecné rovnice interpolačního polynomu (10.9):

$$\begin{aligned} L_2(x) &= y_0 \cdot \frac{(x-x_1)\cdot(x-x_2)}{(x_0-x_1)\cdot(x_0-x_2)} + y_1 \cdot \frac{(x-x_0)\cdot(x-x_2)}{(x_1-x_0)\cdot(x_1-x_2)} + \\ &+ y_2 \cdot \frac{(x-x_0)\cdot(x-x_1)}{(x_2-x_0)\cdot(x_2-x_1)} = \\ &= 1 \cdot \frac{(x-2)\cdot(x-3)}{(0-2)\cdot(0-3)} + 2 \cdot \frac{(x-0)\cdot(x-3)}{(2-0)\cdot(2-3)} + 4 \cdot \frac{(x-0)\cdot(x-2)}{(3-0)\cdot(3-2)} = \\ &= \frac{1}{6} \cdot (x^2 - 5\cdot x + 6) + 2 \cdot -\frac{1}{2} \cdot (x^2 - 3\cdot x) + 4 \cdot \frac{1}{3} \cdot (x^2 - 2\cdot x) = \\ &= \frac{1}{2} \cdot x^2 - \frac{1}{2} \cdot x + 1 \,. \end{aligned} \tag{10.10}$$

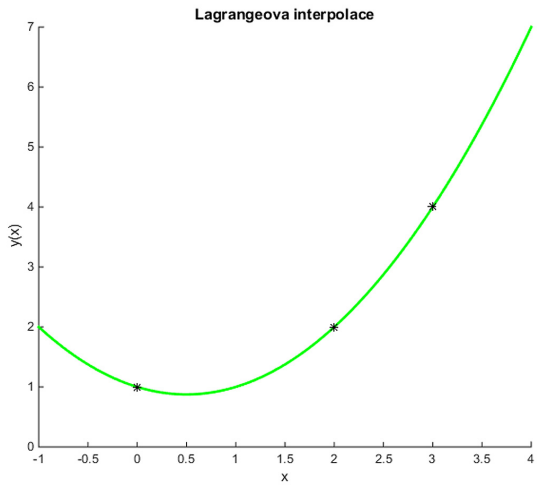
O správnosti odvozeného interpolačního polynomu se lze přesvědčit dosazením souřadnic zadaných bodů:

$$L_2(x_0) = \frac{1}{2} \cdot x_0^2 - \frac{1}{2} \cdot x_0 + 1 = \frac{1}{2} \cdot 0^2 - \frac{1}{2} \cdot 0 + 1 = 1 \,, \tag{10.11}$$

$$L_2(x_1) = \frac{1}{2} \cdot x_1^2 - \frac{1}{2} \cdot x_1 + 1 = \frac{1}{2} \cdot 2^2 - \frac{1}{2} \cdot 2 + 1 = 2 \,, \tag{10.12}$$

$$L_2(x_2) = \frac{1}{2} \cdot x_2^2 - \frac{1}{2} \cdot x_2 + 1 = \frac{1}{2} \cdot 3^2 - \frac{1}{2} \cdot 3 + 1 = 4 \,. \tag{10.13}$$

Sestrojený Lagrangeův interpolační polynom lze zobrazit rovněž graficky - viz obr. 10.2.



Obr. 10.2 Výsledná interpolace s využitím Lagrangeova interpolačního polynomu pro body o souřadnicích $[x_0, y_0] = [0, 1]$, $[x_1, y_1] = [2, 2]$ a $[x_2, y_2] = [3, 4]$

▲

Příklad 10.4. S využitím Lagrangeova interpolačního polynomu stanovte hodnotu ohybového momentu konstrukce popsané v příkladu 3.1 pro bod se souřadnicí $x = l/5 = 1.2$ m. Pro sestrojení Lagrangeova interpolačního polynomu využijte hodnoty skutečných ohybových momentů ve třech bodech o souřadnicích $[x_0, x_1, x_2] = [0, l/2, l] = [0, 3, 6]$ m.

Řešení. Nejprve je samozřejmě nutné stanovit v zadaných bodech hodnoty skutečných ohybových momentů, které pro dané zadání nabývají hodnot $M_y(x_0 = 0) = 0$ kNm, $M_y(x_1 = 3) = 9$ kNm a $M_y(x_2 = 6) = -18$ kNm. Vytvoření Lagrangeova interpolačního polynomu je možné již vytvořeným a dříve popsaným skriptem `lagrange.m`. Celý výpočet může být proveden např. následujícím sledem příkazů:

```
clc; clear; format short;
qz=4000;
l=6;
M=[0 3/8*qz*l -qz/2]/1000;
x=[0 l/2 l]; % 0 m, 3 m, 6 m
y=[horner(2,M,x(1)) horner(2,M,x(2)) horner(2,M,x(3))];
par=l/5;
res=lagrange(x,y,par)
```

Výsledkem celého řešení je pak hodnota v bodu o souřadnici $x = l/5 = 1.2$ m:

```
res =
    7.9200
```

Vzhledem ke skutečnosti, že výsledný Lagrangeův interpolační polynom tvoří stejně jako průběh ohybových momentů polynom 2. stupně, lze pozorovat naprostou shodu mezi touto dvojicí funkcí - viz obr. 10.3.

▲

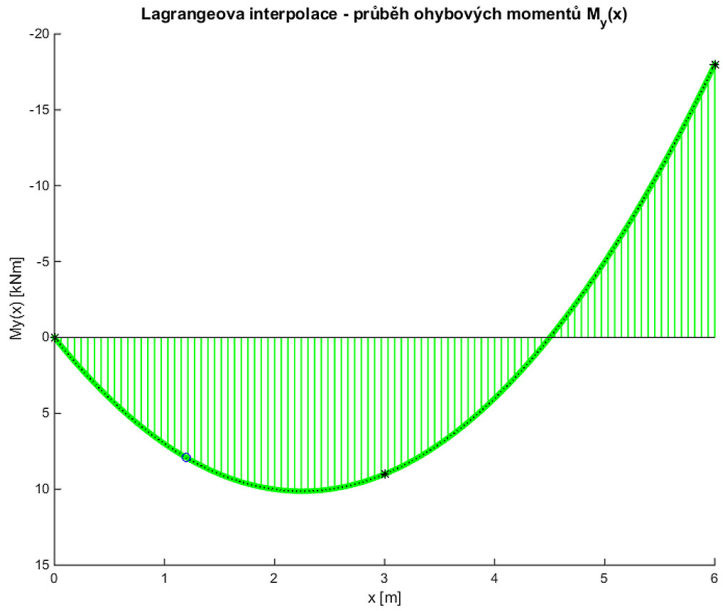
10.3 Newtonova interpolace

Interpolační polynom $\Phi_n(x)$ lze vyjádřit také funkcí obsahující diference:

$$\begin{aligned}\Phi_n(x) = a_0 + a_1 \cdot (x - x_0) + a_2 \cdot (x - x_0) \cdot (x - x_1) + \\ + \ldots + a_n \cdot (x - x_0) \cdot (x - x_1) \cdot \ldots \cdot (x - x_n)\ .\end{aligned} \tag{10.14}$$

Požadovaný interpolační polynom $\Phi_n(x)$ musí v bodech x_i nabývat hodnot:

$$\Phi_n(x_i) = f(x_i) \text{ pro } i = 0, 1, \ldots, n\ . \tag{10.15}$$



Obr. 10.3 Výsledná interpolace průběhu ohybových momentů na nosníku z příkladu 3.1 s využitím Lagrangeova interpolačního polynomu pro body $M_y(x_0 = 0) = 0$ kNm, $M_y(x_1 = 3) = 9$ kNm a $M_y(x_2 = 6) = -18$ kNm označené hvězdičkou. Hodnota ohybového momentu v zadaném bodu o souřadnici $x = l/5 = 1.2$ m je pak naznačena kroužkem.

Řešením úlohy je stanovení neznámých koeficientů a_k pro $k = 0, 1, \ldots, n$, jenž jsou obsaženy ve vztahu 10.14. Postupným dosazováním $x = x_i$ pro $i = 0, 1, \ldots, n$ do polynomu $\Phi_n(x)$ z 10.14 lze získat následující podmínky:

$$\begin{aligned}
\Phi_0(x_0) &= f(x_0) = a_0 \\
\Phi_1(x_1) &= f(x_1) = a_0 + a_1 \cdot (x_1 - x_0) \\
\Phi_2(x_2) &= f(x_2) = a_0 + a_1 \cdot (x_1 - x_0) + a_2 \cdot (x_1 - x_0) \cdot (x_2 - x_0) \\
&\vdots \\
\Phi_n(x_n) &= f(x_n) = a_0 + a_1 \cdot (x_1 - x_0) + a_2 \cdot (x_1 - x_0) \cdot (x_2 - x_0) + \\
&\quad + \ldots + a_n \cdot (x_1 - x_0) \cdot (x_2 - x_0) \cdot \ldots \cdot (x_n - x_0) \,.
\end{aligned} \tag{10.16}$$

Hledané koeficienty a_k lze z uvedených rovnic postupně určit, např.:

$$\begin{aligned}
a_0 &= f(x_0) \\
a_1 &= \frac{f(x_1) - a_0}{x_1 - x_0} \\
a_2 &= \frac{f(x_2) - a_0 - a_1 \cdot (x_1 - x_0)}{(x_1 - x_0) \cdot (x_2 - x_0)} \\
&\vdots \\
a_n &= \frac{f(x_n) - a_0 - a_1 \cdot (x_1 - x_0) - \ldots - a_{n-1} \cdot (x_1 - x_0) \cdot \ldots \cdot (x_{n-1} - x_0)}{(x_1 - x_0) \cdot (x_2 - x_0) \cdot \ldots \cdot (x_n - x_0)} \,.
\end{aligned} \tag{10.17}$$

Rovnice (10.16) lze pro $k = 1, \ldots, n$ vyjádřit rovněž pomocí následujícího vztahu:

$$\Phi_k(x_k) = f(x_k) = \Phi_{k-1}(x_k) + a_k \cdot (x_1 - x_0) \cdot (x_2 - x_0) \cdot \ldots \cdot (x_k - x_0) \,, \tag{10.18}$$

čehož lze využít i zobecnění výpočtu neznámých koeficientů a_k:

$$a_k = \frac{f(x_k) - \Phi_{k-1}(x_k)}{(x_1 - x_0) \cdot (x_2 - x_0) \cdot \ldots \cdot (x_k - x_0)} \,. \tag{10.19}$$

Uvedený problém lze popsat také s využitím tzv. dělených diferencí:

$$\begin{aligned}
f[\, x_k \,] &= f(x_k) \\
f[\, x_k \quad x_{k+1} \,] &= \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} \\
f[\, x_k \quad x_{k+1} \quad x_{k+2} \,] &= \frac{f[\, x_{k+1} \quad x_{k+2} \,] - f[\, x_k \quad x_{k+1} \,]}{x_{k+2} - x_k} \\
f[\, x_k \quad x_{k+1} \quad x_{k+2} \quad x_{k+3} \,] &= \frac{f[\, x_{k+1} \quad x_{k+2} \quad x_{k+3} \,] - f[\, x_k \quad x_{k+1} \quad x_{k+2} \,]}{x_{k+3} - x_k} \\
&\vdots \,.
\end{aligned} \tag{10.20}$$

Tato čísla odpovídají koeficientům a_k pro $k = 0, 1, \ldots, n$ Newtonova interpolačního polynomu, který lze definovat ve finální podobě:

$$\begin{aligned}
N_n(x) = f[\, x_1 \,] &+ f[\, x_1 \quad x_2 \,] \cdot (x - x_1) \\
&+ f[\, x_1 \quad x_2 \quad x_3 \,] \cdot (x - x_1) \cdot (x - x_2) \\
&+ f[\, x_1 \quad x_2 \quad x_3 \quad x_4 \,] \cdot (x - x_1) \cdot (x - x_2) \cdot (x - x_3) \\
&+ \ldots + \\
&+ f[\, x_1 \quad \cdots \quad x_n \,] \cdot (x - x_1) \cdot \ldots \cdot (x - x_{n-1}) \,,
\end{aligned} \tag{10.21}$$

resp. pro $k = 1, \ldots, n$:

$$N_k(x) = N_{k-1}(x) + f[\, x_1 \quad \cdots \quad x_k \,] \cdot (x - x_1) \cdot \ldots \cdot (x - x_{k-1}) \,. \tag{10.22}$$

Výpočet Newtonova interpolačního polynomu lze algoritmicky vyjádřit např. pomocí algoritmu 23.

Vstup : $x = [\ x_1 \quad \cdots \quad x_n\ ]$, $y = [\ y_1 \quad \cdots \quad y_n\ ]$, z
Výstup: $N_n(z)$

for $j \leftarrow 1, 2, \ldots, n$ **do**
| $f[x_j] \leftarrow y_j$
end

for $i \leftarrow 2, 3, \ldots, n$ **do**
| **for** $j \leftarrow 1, 2, \ldots, n+1-i$ **do**
| | $f[\ x_j \quad \cdots \quad x_{j+i-1}\ ] \leftarrow \dfrac{f[\ x_{j+1} \quad \cdots \quad x_{j+i-1}\ ] - f[\ x_j \quad \cdots \quad x_{j+i-2}\ ]}{x_{j+i-1} - x_j}$
| **end**
end

$$N_n(z) \leftarrow \sum_{i=1}^{n} f[\ x_1 \quad \cdots \quad x_i\ ] \cdot (x - x_1) \cdot \ldots \cdot (x - x_{i-1})$$

Algoritmus 23: Stanovení hodnoty Newtonova interpolačního polynomu $N_n(x)$

Pro rekurzivní vyjádření dělených diferencí Newtonova interpolačního polynomu se používá tabulkového vyjádření (pro tři body viz tab. 10.1). Koeficienty Newtonova interpolačního polynomu (10.22) pak lze odečíst z horní hrany zobrazeného trojúhelníka.

x_1	$f[\ x_1\ ]$		
		$f[\ x_1 \quad x_2\ ]$	
x_2	$f[\ x_2\ ]$		$f[\ x_1 \quad x_2 \quad x_3\ ]$
		$f[\ x_2 \quad x_3\ ]$	
x_3	$f[\ x_3\ ]$		

Tab. 10.1 Dělené diference Newtonova interpolačního polynomu pro tři body

Příklad 10.5. S využitím Newtonova interpolačního polynomu stanovte rovnici interpolační funkce $y(x)$ pro tři body z příkladu 10.3 o souřadnicích $[x_0, y_0] = [0, 1]$, $[x_1, y_1] = [2, 2]$ a $[x_2, y_2] = [3, 4]$.

Řešení. S využitím postupu (10.21) pro sestrojení Newtonova interpolačního polynomu lze sestavit tabulku 10.2, ve které se jednotlivé členy určí s pomocí (10.20):

$$f[\ x_1 \quad x_2\ ] = \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{2-1}{2-0} = \frac{1}{2}\,, \tag{10.23}$$

$$f[\ x_1 \quad x_2 \quad x_3\ ] = \frac{f[\ x_2 \quad x_3\ ] - f[\ x_1 \quad x_2\ ]}{x_3 - x_1} = \frac{2 - \frac{1}{2}}{3 - 0} = \frac{1}{2}\,, \tag{10.24}$$

$$f[\ x_2 \quad x_3\ ] = \frac{f[x_3] - f[x_2]}{x_3 - x_2} = \frac{4 - 2}{3 - 2} = 2\,. \tag{10.25}$$

$x_1 = 0$	$f[\ x_1\ ] = 1$		
		$f[\ x_1 \quad x_2\ ] = \frac{1}{2}$	
$x_2 = 2$	$f[\ x_2\ ] = 2$		$f[\ x_1 \quad x_2 \quad x_3\ ] = \frac{1}{2}$
		$f[\ x_2 \quad x_3\ ] = 2$	
$x_3 = 3$	$f[\ x_3\ ] = 4$		

Tab. 10.2 Dělené diference Newtonova interpolačního polynomu pro tři body z příkladu 10.3

Jak již bylo řečeno, koeficienty hledaného Newtonova interpolačního polynomu podle (10.22) pak lze odečíst z tabulky 10.2 z horní hrany zobrazeného trojúhelníka:

$$\begin{aligned} N_3(x) &= f[\ x_1\ ] + f[\ x_1 \quad x_2\ ] \cdot (x - x_1) + f[\ x_1 \quad x_2 \quad x_3\ ] \cdot (x - x_1) \cdot (x - x_2) = \\ &= 1 + \frac{1}{2} \cdot (x - 0) + \frac{1}{2} \cdot (x - 0) \cdot (x - 2) = \frac{1}{2} \cdot x^2 - \frac{1}{2} \cdot x + 1\,. \end{aligned} \tag{10.26}$$

Z výsledné rovnice vztahu Newtonova interpolačního polynomu (10.26) je zřejmé, že bylo dosaženo stejného polynomu druhého řádu jako v případě příkladu (10.3). ▲

Funkci, která stanoví pro zadaný bod se souřadnicí x ve vstupním parametru `par` hodnotu Newtonova interpolačního polynomu, sestaveného pro zadanou množinu bodů se souřadnicemi x a y uloženou ve vstupních parametrech s vektory `x` a `y`, lze naprogramovat v MATLABU např. pomocí skriptu `newton.m`:

```
function s=newton(x,y,par)
n=length(x);
for j=1:n
  tab(j,1)=y(j);
end
for i=2:n
  for j=1:n+1-i
    tab(j,i)=(tab(j+1,i-1)-tab(j,i-1))/(x(j+i-1)-x(j));
  end
end
s=tab(1,1);
for i=2:n
```

```
  m=tab(1,i);
  for j=1:i-1
      m=m*(par-x(j));
  end
  s=s+m;
end
```

Skript lze nepatrně upravit tak, aby bylo možno hodnoty Newtonova interpolačního polynomu efektivně určit i pro vektor, obsahující ve vstupním parametru `par` souřadnice x více bodů (skript je funkční i pro jednu souřadnici).

```
function s=newton(x,y,par)
n=length(x);
for j=1:n
  tab(j,1)=y(j);
end
for i=2:n
  for j=1:n+1-i
    tab(j,i)=(tab(j+1,i-1)-tab(j,i-1))/(x(j+i-1)-x(j));
  end
end
num=length(par);
for k=1:num
  tot=tab(1,1);
  for i=2:n
    m=tab(1,i);
    for j=1:i-1
      m=m*(par(k)-x(j));
    end
    tot=tot+m;
  end
  s(k)=tot;
end
```

Příklad 10.6. S využitím Newtonova interpolačního polynomu stanovte hodnotu ohybového momentu podle zadání v příkladu 10.6.

Poznámka 10.7. Pro konstrukci Newtonova interpolačního polynomu lze použít i velmi zajímavý skript - viz dále, který umožňuje zadávat jednotlivé body potřebné k sestrojení interpolačního polynomu přímo z grafu klikáním levým tlačítkem myši. Je zajímavé sledovat, jak se s přibývajícími body zvyšuje řád interpolačního polynomu. Výpočet se ukončí po kliknutí pravým tlačítkem myši.

```
xmin=-3; xmax=3;
x_p=xmin:.01:xmax;
ymin=-3; ymax=3;
plot([xmin xmax],[0 0],'k',[0 0],[ymin ymax],'k');
grid on;
x=[];    y=[];
tlac=1;  k=0;
while ~(tlac==3)
 [x_novy,y_novy,tlac]=ginput(1);
 if tlac==1
  k=k+1; x(k)=x_novy; y(k)=y_novy;
  y_p=newton(x,y,x_p);
  plot(x,y,'o',x_p,y_p,[xmin xmax],[0,0],'k',[0 0],[ymin ymax],'k');
  axis([xmin xmax ymin ymax]);
  grid on;
 end
end
```

10.4 Aproximace metodou nejmenších čtverců

Při interpolaci některou z předchozích metod se předpokládalo, že interpolovaná funkce je zadaná tabulkou s hodnotami x_i a $f(x_i) = y_i$, kde $i = 0, 1, \ldots, n$. V případě aproximace není úkolem najít funkci, která se ztotožní v zadaných bodech s hledanou funkcí, nýbrž určit aproximační funkci $F(x)$, která by co nejlépe přiléhala k $n + 1$ zadaným empirickým bodům $[x_0, y_0]$, $[x_1, y_1]$ až $[x_n, y_n]$.

V metodě nejmenších čtverců se jako kritérium přiléhavosti využívá součet druhých mocnin (čtverců) rozdílů mezi hodnotami aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i:

$$Q = \sum_{i=0}^{n} (F(x_i) - y_i)^2 \ . \tag{10.27}$$

Funkce $F(x)$ může být obecně dána jako:

$$F(x) = a_0 \cdot f_0(x) + a_1 \cdot f_1(x) + \ldots + a_m \cdot f_m(x) \ , \tag{10.28}$$

kde $f_0, f_1, \ldots, f_m$ jsou vhodně zvolené lineárně nezávislé funkce a $a_0, a_1, \ldots, a_m$ neznámé reálné koeficienty, které se určí tak, aby hodnota Q ve vztahu (10.27) byla minimální. Musí tedy platit:

$$\frac{\partial Q}{\partial a_k} = 2 \cdot \sum_{i=0}^{n} (F(x_i) - y_i) \cdot \frac{\partial F(x_i)}{\partial a_k} = 0 \ , \tag{10.29}$$

kde $k = 0, 1, \ldots, m$.

Při volbě

$$\frac{\partial F(x_i)}{\partial a_k} = f_i(x_i) \ , \tag{10.30}$$

musí platit:

$$\frac{\partial Q}{\partial a_k} = 2 \cdot \sum_{i=0}^{n} [a_0 \cdot f_0(x_i) + a_1 \cdot f_1(x_i) + \ldots + a_m \cdot f_m(x_i) - y_i] \cdot f_k(x_i) = 0 \ . \tag{10.31}$$

Vztah (10.31) lze dále upravit:

$$\begin{aligned} \sum_{i=0}^{n} [a_0 \cdot f_k(x_i) \cdot f_0(x_i) + a_1 \cdot f_k(x_i) \cdot f_1(x_i) + \ldots + a_m \cdot f_k(x_i) \cdot f_m(x_i)] = \\ = \sum_{i=0}^{n} f_k(x_i) \cdot y_i \ , \end{aligned} \tag{10.32}$$

resp.

$$\begin{aligned} a_0 \cdot \sum_{i=0}^{n} f_k(x_i) \cdot f_0(x_i) + a_1 \cdot \sum_{i=0}^{n} f_k(x_i) \cdot f_1(x_i) + \ldots + a_m \cdot \sum_{i=0}^{n} f_k(x_i) \cdot f_m(x_i) = \\ = \sum_{i=0}^{n} f_k(x_i) \cdot y_i \ , \end{aligned} \tag{10.33}$$

kde $k = 0, 1, \ldots, m$.

Vztah (10.33) lze vyjádřit i v maticovém tvaru:

$$\begin{aligned} &\begin{bmatrix} \sum_{i=0}^{n} f_0^2(x_i) & \sum_{i=0}^{n} f_0(x_i) \cdot f_1(x_i) & \ldots & \sum_{i=0}^{n} f_0(x_i) \cdot f_m(x_i) \\ \sum_{i=0}^{n} f_1(x_i) \cdot f_0(x_i) & \sum_{i=0}^{n} f_1^2(x_i) & \ldots & \sum_{i=0}^{n} f_1(x_i) \cdot f_m(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} f_m(x_i) \cdot f_0(x_i) & \sum_{i=0}^{n} f_m(x_i) \cdot f_1(x_i) & \ldots & \sum_{i=0}^{n} f_m^2(x_i) \end{bmatrix} \cdot \begin{Bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{Bmatrix} = \\ &= \begin{Bmatrix} \sum_{i=0}^{n} f_0(x_i) \cdot y_i \\ \sum_{i=0}^{n} f_1(x_i) \cdot y_i \\ \vdots \\ \sum_{i=0}^{n} f_m(x_i) \cdot y_i \end{Bmatrix} \ . \end{aligned} \tag{10.34}$$

10.4.1 Aproximace přímkou

Při lineární aproximaci platí mezi proměnnými x a y vztah:

$$F(x) = a \cdot x + b \,, \tag{10.35}$$

kde a, b jsou neznámé parametry, jenž lze určit z podmínky podle (10.27):

$$Q = \sum_{i=0}^{n} (a \cdot x_i + b - y_i)^2 = \min \ . \tag{10.36}$$

Řešení úlohy dané vztahem (10.28) vede k soustavě dvou rovnic:

$$\frac{\partial Q}{\partial a} = 0 \tag{10.37}$$

a

$$\frac{\partial Q}{\partial b} = 0 \,. \tag{10.38}$$

Po úpravě obou rovnic podle (10.31) až (10.33) lze získat jejich výsledný tvar:

$$\begin{gathered} n \cdot b + \left(\sum_{i=0}^{n} x_i \right) \cdot a = \sum_{i=0}^{n} y_i \\ \left(\sum_{i=0}^{n} x_i \right) \cdot b + \left(\sum_{i=0}^{n} x_i^2 \right) \cdot a = \sum_{i=0}^{n} x_i \cdot y_i \,, \end{gathered} \tag{10.39}$$

jenž lze vyjádřit maticově:

$$\begin{bmatrix} n & \sum\limits_{i=0}^{n} x_i \\ \sum\limits_{i=0}^{n} x_i & \sum\limits_{i=0}^{n} x_i^2 \end{bmatrix} \cdot \begin{Bmatrix} b \\ a \end{Bmatrix} = \begin{Bmatrix} \sum\limits_{i=0}^{n} y_i \\ \sum\limits_{i=0}^{n} x_i \cdot y_i \end{Bmatrix} \,. \tag{10.40}$$

10.4.2 Aproximace polynomem m-tého stupně

Pokud bude zvolen za aproximující funkci polynom m-tého stupně:

$$F_m(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + \ldots + a_m \cdot x^m \,, \tag{10.41}$$

po úpravách (10.29) až (10.33) lze po dosazení za $f_k(x) = x^k$, $k = 0, 1, \ldots, m$ získat soustavu $m + 1$ rovnic:

$$\begin{bmatrix} \sum\limits_{i=0}^{n} (x_i^0)^2 & \sum\limits_{i=0}^{n} x_i^0 \cdot x_i^1 & \ldots & \sum\limits_{i=0}^{n} x_i^0 \cdot x_i^m \\ \sum\limits_{i=0}^{n} x_i^1 \cdot x_i^0 & \sum\limits_{i=0}^{n} (x_i^1)^2 & \ldots & \sum\limits_{i=0}^{n} x_i^1 \cdot x_i^m \\ \vdots & \vdots & \ddots & \vdots \\ \sum\limits_{i=0}^{n} x_i^m \cdot x_i^0 & \sum\limits_{i=0}^{n} x_i^m \cdot x_i^1 & \ldots & \sum\limits_{i=0}^{n} (x_i^m)^2 \end{bmatrix} \cdot \begin{Bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{Bmatrix} = \begin{Bmatrix} \sum\limits_{i=0}^{n} x_i^0 \cdot y_i \\ \sum\limits_{i=0}^{n} x_i^1 \cdot y_i \\ \vdots \\ \sum\limits_{i=0}^{n} x_i^m \cdot y_i \end{Bmatrix} \,. \tag{10.42}$$

Soustavu rovnic (10.42) s neznámými koeficienty $a_0, a_1, \ldots, a_m$ pak lze dále upravit na tvar:

$$\begin{bmatrix} n+1 & \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \ldots & \sum_{i=0}^{n} x_i^m \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i^3 & \ldots & \sum_{i=0}^{n} x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots & \\ \sum_{i=0}^{n} x_i^m & \sum_{i=0}^{n} x_i^{m+1} & \sum_{i=0}^{n} x_i^{m+2} & \ldots & \sum_{i=0}^{n} x_i^{2\cdot m} \end{bmatrix} \cdot \begin{Bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{Bmatrix} = \begin{Bmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i \cdot y_i \\ \vdots \\ \sum_{i=0}^{n} x_i^m \cdot y_i \end{Bmatrix}. \tag{10.43}$$

Příklad 10.8. Proveďte lineární aproximaci i aproximaci polynomem 2.stupně pro data obsažená v tabulce 10.3. U obou případů stanovte součet druhých mocnin (čtverců) rozdílů mezi hodnotami aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i.

x	1	2	3	4	5
y	0	2	2	5	4

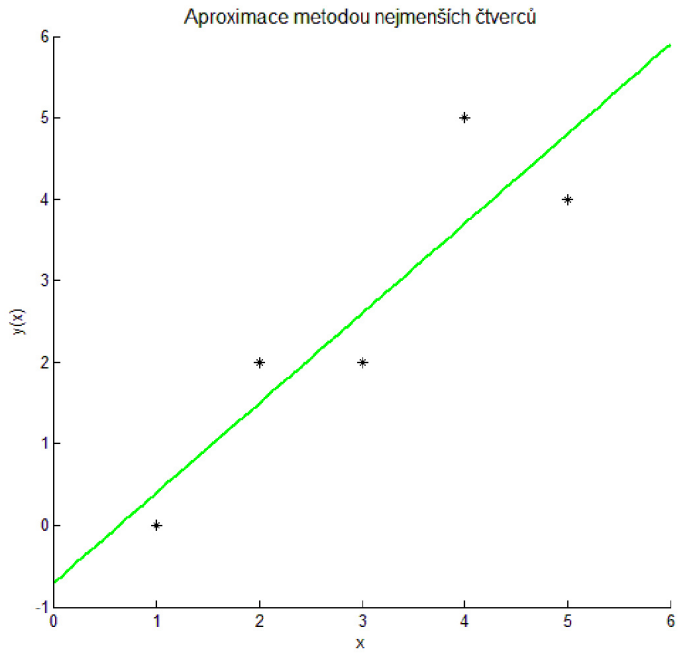
Tab. 10.3 Vstupní údaje pro výpočet aproximace v příkladu 10.8

Řešení. Výpočet lineární aproximace i polynomem m-tého stupně včetně součtu druhých mocnin (čtverců) rozdílů mezi hodnotami příslušné aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i lze provést následujícím sledem příkazů:

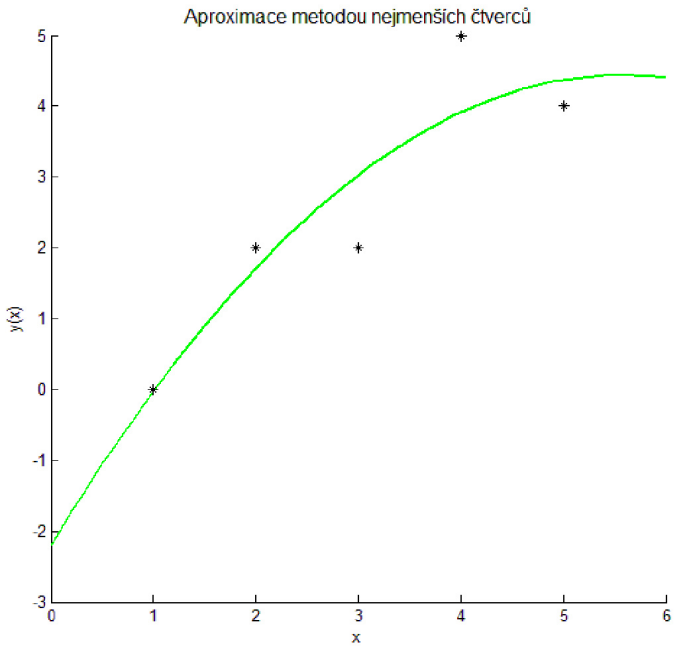
```
clear; clc;
x0=[1 2 3 4 5];
y0=[0 2 2 5 4];
m=1;
for i=1:m+1
  for j=i:m+1
    A(i,j)=sum(x0.^((i-1)+(j-1)));
    if ~(i==j)
      A(j,i)=A(i,j);
    end
  end
  b(i)=sum((x0.^(i-1)).*y0);
end
c=A\b';
x=0:.1:6;
for j=1:length(x)
```

```
  s=c(1);
  for i=1:m
    s=s+c(i+1)*x(j)^(i);
  end
  y(j)=s;
end
plot(x0,y0,'o',x,y);
soucet_ctvercu=0;
for j=1:length(x0)
  s=c(1);
  for i=1:m
    s=s+c(i+1)*x0(j)^(i);
  end
  soucet_ctvercu=soucet_ctvercu+(s-y0(j))^2;
end
soucet_ctvercu
```

Pro případ lineární aproximace lze získat přímku, zobrazenou na obr. 10.4, se součtem čtverců rozdílů mezi hodnotami příslušné aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i rovném 3.1. V případě aproximace polynomem 2. stupně - viz obr. 10.5 je součet čtverců rozdílů mezi hodnotami příslušné aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i roven hodnotě 2.4571.



Obr. 10.4 Lineární aproximace pro body, zadané v příkladu 10.8



Obr. 10.5 Aproximace polynomem 2. stupně pro body, zadané v příkladu 10.8

▲

Příklad 10.9. Vyberte nejvhodnější stupeň polynomu pro aproximaci naměřených hodnot krychelné pevnosti betonu v tlaku v závislosti na dnech zrání betonové směsi, které jsou zobrazeny v tabulce 10.4. Jako kritérium nejlepší přiléhavosti použijte součet druhých mocnin (čtverců) rozdílů mezi hodnotami aproximační funkce $F(x_i)$ a naměřenými hodnotami y_i.

x [dny]	0	0	0	7	7	7	14	14	14	28	28	28
y [MPa]	0	0	0	21.5	22.2	21.2	30.7	31.4	30.5	40.1	43.4	41.5

Tab. 10.4 Vstupní údaje pro výpočet aproximační funkce v příkladu 10.9

Literatura

[1] *Algoritmus*. Webové stránky zaměřené na tvorbu algoritmů. [on-line]. <http://www.algoritmy.net>. (Citováno na s 49.)

[2] Eaton, J.W. *Octave*. Programový systém pro provádění matematických výpočtů. Freeware, verze 4.2.1. [on-line]. <http://www.gnu.org/software/octave>. University of Wisconsin, Department of Chemical Engineering, 1998-2017. (Citováno na s 1.)

[3] Just, M. *Octave — český průvodce programem*. Elektronický manuál programového systému Octave. [on-line]. <http://www.octave.cz>. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky, 2006. (Citováno na s 12.)

[4] Kučera, R. *Numerické metody*. Učební texty. VŠB-TU Ostrava. (152 s). ISBN 80-248-1198-7.

[5] Materna, A. — Štěpánek, P. — Teplý, B. *Automatizace inženýrských úloh*. Skriptum. Vysoké učení technické v Brně, 1985. (132 s).

[6] MATLAB. Programový systém pro provádění matematických výpočtů. Komerční software, verze R2017b. [on-line]. <http://www.mathworks.com>. The MathWorks, prosinec 2017. (Citováno na s 1.)

[7] Mika, S. *Numerické metody algebry*. Matematika pro vysoké školy technické. 2. vydání. SNTL - Nakladatelství technické literatury, Praha, 1985. (176 s). (Citováno na s 34.)

[8] Okrouhlík, M. a kol. *Numerical methods in computational mechanics*. Elektronická publikace ve formátu PDF. [on-line]. <http://www.it.cas.cz/elektronicka-kniha-numerical-methods-computation-mechanics>. Institute of Thermomechanics, Prague 2008. Last revision January 23, 2012.

[9] Olehla, M. — Tišer, J. *Praktické použití Fortranu*. 2. upravené vydání. Nakladatelství dopravy a spojů, Praha, 1979. (432 s). (Citováno na s 46 a 117.)

[10] Ralston, A. *Základy numerické matematiky*. 1. vydání. Academia, Praha, 1973. (635 s).

[11] Rektorys, K. *Přehled užité matematiky*. 4. vydání. SNTL - Nakladatelství technické literatury, Praha, 1981. (1140 s).

[12] Sauer T. *Numerical Analysis*. George Mason University. Pearson Education, Inc., 2006. (669 s). ISBN 0-321-26898-9. (Citováno na s 16.)

[13] Sigmon K. *MATLAB Primer CZ*. Elektronický manuál programového systému MATLAB. Druhé vydání. [on-line]. <https://artax.karlin.mff.cuni.cz/~beda/cz/matlab/primercz/matlab-primer.html>. Department of Mathematics, University of Florida, 1989, 1992. Z anglického originálu přeložil Petr Klášterecký. (Citováno na s 12.)

[14] *Wikipedia*. Otevřená internetová encyklopedie. Webové stránky. [on-line]. <http://cs.wikipedia.org>. (Citováno na s 13.)

[15] Wirth N. *Algoritmy a štruktúry údajov*. 1. vydanie. Alfa, vydavateľstvo technickej a ekonomickej literatúry, Bratislava, 1988. (488 s). (Citováno na s 54.)