Numerické metody I

Jan Schee
FPF SÚ Opava

2. října 2018

1 Strojová reprezentace čísel

Ve smyslu numerických výpočtů rozlišujeme dva číselné typy: celočíselný a s plovoucí desetinnou čárkou. V jazyce C jsou celočíselné typy implementovány v datových typech unsigned int, int, unsigned char, char. Co se týče čísel s plovoucí desetinnou čárkou tak jsou to datové typy float a double. Aritmetika číslicové techniky je postavena dvojkové soustavě. Čísla jsou ukládaná do bitových polí, která jsou organizována po osmi do jednotky byte (B).

1.1 Formát celého čísla

Celé číslo zapsané do dvojkové (binární) baze bude mít tvar

$$a = \sum_{k=0}^{n-1} a_k 2^k \tag{1}$$

kde koeficienty rozvoje a k jsou jednotlivé hodnoty bitového pole, viz. tabulka ukazující zápis čísla 136 do 8-bitového pole.

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	0	0	1	0	0	0

Uvažujme například datový typ unsigned char, ketrý je tvořen polem 8 bitů, tj. $n = 8$. Mějme binární číslo $(10001000)_2$ které převedeme na číslo v desítkové soustavě a obdržíme

$$\begin{aligned}(10001000)_2 &= (1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 \\ &\quad +0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0)_{10} \\ &= (128 + 8)_{10} = (136)_{10} \end{aligned} \tag{2}$$

V případě znaménkových typů je nejvyšší bit vyhrazen hodnotě znaménka a to tak, že

$$\text{sign}(a) = (-1)^{a_7} \tag{3}$$

(v případě datového typu *char*).

1.2 Formát čísla s plovoucí desetinou čárkou

Norma IEEE 754 definuje single-precision binary floating-point format: *binary32*. Jeho 32 bitů je rozděleno do tří polí:

- Znaménkový typ s: 1 bit
- Exponent e: 8 bitů

- Mantisa m: 23 bitů

Následující tabulka ilustruje upspořádání jednotlivých polí v datovém typu float:

s	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	m_{22}	m_{21}	$\cdots$	m_0

Hodnotu takto zakódovaného desetinného čísla získáme ze vztahu

$$r = (-1^s) \times \left(1 + \sum_{k=1}^{23} m_{23-k} 2^{-k}\right) \times 2^{e-127}. \qquad (4)$$

Jako příklad uveďme číslo 5.125 zakódované podle normy IEEE 754. Bitové pole je zaplněno tak jak ukazuje tabulka

0	10000001	01001000000000000000000

Odkud je zřejmě exponent $e = (10000001)_2 = (129)_{10}$ a mantisa je $m = (010\,0100\,0000\,0000\,0000\,0000)_2 = (2359296)_{10}$. Po dosazení do předchozího vztahu obdržíme hodnotu

$$\begin{aligned} r &= 1 \times (1 + 1 \times 2^{-2} + 1 \times 2^{-5}) \times 2^{129-127} = (1 + 0.25 + 0.03125) \times 2^2 \\ &= 1.28125 \times 4 = 5.125. \end{aligned} \qquad (5)$$

Cvičení

1. Vytvořte program, který má na vstupu celé číslo v desístkové soustavě a převeďte jej do dvojkové soustavy a zobrazte na obrazovce (hint: použijte datový typ *unsigned int*, výsledek uložte do pole *char a[32]*).

2. Napište program, který zobrazí bitové pole čísla uloženého v proměnnné o datovém typu float a určete exponent a mantisu. Výsledky zobrazte na obrazovce.

2 Typy chyb a jejich šíření

Díky konečné velikosti datových typů kódujících čísla s plovoucí detetinou čárkou není většina čísel strojově reprezento- vatelná. Například číslo $(0.1)_{10}$ není strojově reprezovatelné, protože má nekonečný rozklad do dvojkové baze, jak ukazuje následující rozklad

$$\begin{aligned} 0.1_{10} \quad = \quad & 0.0001100110011001100110011001100110011001100110011001 \\ & 1001100110011001100110011001100110011001100110011001100 \\ & 110011001 \cdots_2 . \end{aligned} \tag{6}$$

Nechť je množina strojově reprezovatelných čísel A. Pokud pro číslo x platí, že $x \notin A$ pak hledáme takové číslo $g \in A$ pro které platí

$$|x - g| \leq |x - b| \,, \forall b \in A, \tag{7}$$

tj., hledáme nejbližší strojově reprezentovatelné číslo g k číslu x. Základním mechanizmem, kterým lze toho dosáhnou je zaokrouhlování.

Uvažujme číslo $a < 1$ a definujme následující reprezentaci

$$a = 0.a_1a_2a_3 \cdots a_ia_{i+1} \cdots , \; 0 \leq a_i \leq 9 \; \wedge \; a_1 \neq 0. \tag{8}$$

Z čísla a vytoříme zaokrouhlené číslo na t -platných číslic takto

$$a' = \begin{cases} 0.a_1a_2a_3 \cdots a_t & \text{pro } a_{t+1} \leq 4 \\ 0.a_1a_2a_3 \cdots (a_t + 1) & \text{pro } a_{t+1} \geq 5 \end{cases} \tag{9}$$

Zaokrouhlené číslo potom píšeme ve tvaru

$$\mathrm{rd}(x) = \mathrm{sign}(x) \times a' \times 10^b. \tag{10}$$

Pomocí této definice snadno určíme relativní chybu zaokrouhlení, tj.

$$\epsilon_r = \left| \frac{\mathrm{rd}(x) - x}{x} \right|. \tag{11}$$

Nejprve určeme, čemu se rovná $\Delta \equiv \mathrm{rd}(x) - x$:

- $a_{t+1} \leq 4$

$$\begin{aligned} \Delta &= \left| 0.a_1a_2 \cdots a_t \times 10^b - 0.a_1a_2 \cdots a_ta_{t+1} \cdots \times 10^b \right| \\ &= \left| a_{t+1} \cdot a_{t+2} \cdots \times 10^{b-(t+1)} \right| \leq 5 \times 10^{b-(t+1)}, \end{aligned} \tag{12}$$

protože je $a_{t+1} \leq 4$ a $a_{t+2} \in [0, 1, 2, \ldots, 9]$.

- $a_{t+1} \geq 5$

$$\begin{aligned} \Delta &= \left|0.a_1a_2\cdots(a_t+1)\times 10^b - 0.a_1a_2\cdots a_ta_{t+1}\cdots\times 10^b\right| \\ &= \left|b_{t+1}\dot{b}_{t+2}\cdots\times 10^{b-(t+1)}\right| \leq 5\times 10^{b-(t+1)} \end{aligned} \tag{13}$$

protože zřejme platí $b_{t+1} = 10 - a_{t+1} \leq 5$ a současně $a_{t+1} \geq 5$.

Pro relativní chybu zaokrouhlování tak dostaneme odhad

$$\epsilon_r \leq \frac{5\times 10^{b-(t+1)}}{a\times 10^b} = \frac{5}{a}\times 10^{-t+1} \leq 5\times 10^{-t} \tag{14}$$

což je důsledkem faktu, že $a \geq 10^{-1}$. Zavádíme označení $EPS = 5\times 10^{-t}$ a nazýváme tuto veličinu *strojová přesnost*. Chyba, které se při zaokrouhlování dopouštíme je tzv. zaokrouhlovací chyba (trucation error). Tato chyba se netýká pouze vstupních dat ale mohou a jsou touto chybou zatíženy i aritmetické operace , které nad čísly s plovoucí desetinou čárkou provádíme. Může se totiž stát, že pro $x, y \in A$ bude $x + y \notin A$.

Je tedy důležité vědět, jak se při danám algoritmu (který je tvořen aritmetickými operacemi) zaokrouhlovací chyba šíří.

Nechť ϕ je funkce reprezentující algoritmus, tj.

$$\phi : D \to R^n,\ \phi(x) = \begin{pmatrix} \phi_1(x_1,\ldots,x_n) \\ \vdots \\ \phi_m(x_1,\ldots,x_n) \end{pmatrix} \tag{15}$$

dále nechť je $\tilde{x}$ aproximace čísla x a označme $\Delta x_i \equiv \tilde{x}_i - x_i$ a $\Delta x = \tilde{x} - x$ absolutní chybu a $\epsilon_{\tilde{x}_i} \equiv \Delta x_i / x_i$ relativní chybu

aproximace. Dosadíme-li na vstupu $\tilde{x}$ za x tak dostaneme výsledek $\tilde{y} = \phi(\tilde{x})$, který se liší od $y = \phi(x)$. Absolutní chyba výsledku bude (Taylorův rozvoj do prvího řádu v Δx_i)

$$\Delta y_i \equiv \tilde{y}_i - y_i = \phi_i(\tilde{x}) - \phi_i(x) = \sum_{j=1}^{n} \Delta x_j \frac{\partial \phi_i}{\partial x_j}. \qquad (16)$$

Veličina $\partial\phi_i/\partial x_j$ reprezentuje "citlivost algoritmu"na změnu Δx. Analogicky, pro relativní chybu výsledku dostaneme vztah

$$\epsilon_{y_i} = \sum_{j=1}^{n} \frac{x_j}{\phi_i(x)} \frac{\partial \phi_i(x)}{\partial x_j} \epsilon_{x_j}. \qquad (17)$$

Dokažme poslední vztah.

$\diamond$ Relativní chybu definujeme vztahem

$$\epsilon_{y_i} \equiv \frac{\Delta y_i}{y_i}. \qquad (18)$$

Dosazením za Δy_i obdržíme vztah

$$\begin{aligned} \epsilon_{y_i} &= \frac{\phi_i(\tilde{x}) - \phi_i(x)}{\phi_i(x)} \simeq \sum_{j=1}^{n} \frac{\Delta x_j}{\phi_i(x)} \frac{\partial \phi_i(x)}{\partial x_j} = \sum_{j=1}^{n} \frac{\Delta x_j}{\phi_i(x)} \frac{\partial \phi_i(x)}{\partial x_j} \frac{x_j}{x_j} \\ &= \sum_{j=1}^{n} \frac{x_j}{\phi_i(x)} \frac{\partial \phi_i(x)}{\partial x_j} \epsilon_{x_j} \qquad (19) \end{aligned}$$

C.B.D.

Ilustrujme šíření zaokrouhlovací chyby na příkladu $y = \phi(a, b, c) = a + b + c$. Pro relativní chybu výsledku obdržíme výsledek

$$\epsilon_y = \sum_{j=1}^{3} \frac{x_j}{\phi(x)} \frac{\partial \phi(x)}{\partial x_j} \epsilon_{x_j} = \frac{a}{a+b+c}\epsilon_a + \frac{b}{a+b+c}\epsilon_b + \frac{c}{a+b+c}\epsilon_c. \tag{20}$$

Z výsledku vyčteme, tento problém je dobře definován pokud je součet $a + b + c$ mnohem větší než jednotlivé hodnoty a, b, c.

Podívejme se na další příklad. Nechť je tentokráte $y = -p + \sqrt{p^2 + q}$. Pro relativní chybu výpočtu tohoto vzorce dostaneme výraz

$$\epsilon_y = -\frac{p}{\sqrt{p^2+q}}\epsilon p + \frac{p + \sqrt{p^2+q}}{2\sqrt{p^2+q}}\epsilon_q. \tag{21}$$

Pro $q > 0$ jsou oba faktory < 1. Tento algoritmus je špatně definován pro $q = -p^2$. Nakonec určeme relativní chyby aritmetických operací $\pm$, $*$ a $/$. Obdržíme následující výsledky

$$\phi(x, y) \equiv x * y \rightarrow \epsilon_{xy} = \epsilon_x + \epsilon_y, \tag{22}$$

$$\phi(x, y) \equiv x/y \rightarrow \epsilon_{xy} = \epsilon_x - \epsilon_y, \tag{23}$$

$$\phi(x, y) \equiv x \pm y \rightarrow \epsilon_{xy} = \frac{x}{x \pm y}\epsilon_x \pm \frac{y}{x \pm y}\epsilon_y. \tag{24}$$

Vydíme, že násobení a dělení nejsou nebezpečné operace. Pozor si musíme dávat při odečítání, kdy dochází k amplifikaci chyby pro $x \simeq y$.

3 Hledání kořene nelineární 1D-rovnice $f(x) = 0$

Pokud je funkce $f(x)$ spojitá a hladká, tak podmínkou pro existenci kořene r na intervalu $[x_1, x_2]$ je

- $f(x_1)f(x_2) < 0$,
- $\forall x \in [x_1, x_2]$ bude buď $f'(x) < 0$ nebo $f'(x) > 0$.

Pokud hledáme všechny kořeny rovnice $f(x) = 0$ tak musíme provést na intervalu $[a, b]$ tzv. "bracketing", tj. musíme najít subintervaly na kterých funkce $f(x)$ splnuje výše uvedené podmínky.

Bracketing

Nejednoduší způsob jak zajistit aby na pod-intervalech byla funkce $f(x)$ monotónní, je rozdělit definiční obor na co nejmenší části. Následuje pseudokód ilustrující myšlenku bracketingu.

```
1  (*bracketing*)
2  h=(b-a)/N
3  indx=0
4  for i=0 to N step 1
5          x1=a+i*h;
6          x2=x1+h;
7          if f(x1)*f(x2)<0 then
8                  xl[indx]=x1
9                  xu[indx]=x2
10                 indx=indx+1
11         end
12 end
```

Po ukončení tohoto algoritmu budou sub-intervaly na kterých existuje kořen rovnice $f(x) = 0$, $J_i = [xl[i], xu[i]]$. Jejich počet bude uložen v proměnné $indx$.

Metoda bisekce

Půlením intervalu $[x_1, x_2]$ postupně udoláme kořen. Jestliže po n - tém kroku je šířka subintervalu ϵ_n pak v následujícím bude zřejmě poloviční, tj.

$$\epsilon_{n+1} = \frac{\epsilon_n}{2}. \tag{25}$$

Počáteční interval má šířku ϵ_0. Počet kroků n potřebných k dosažení šířky subintervalu ϵ_n potom je

$$n = \log_2 \frac{\epsilon_0}{\epsilon_n}. \tag{26}$$

Dokažme toto tvrzení. Pro tři po sobě následující kroky bisekce postupně dostáváme

- $\epsilon_1 = \epsilon_0/2$
- $\epsilon_2 = \epsilon_1/2 = \epsilon_0/4$
- $\epsilon_3 = \epsilon_2/2 = \epsilon_0/8$
- ...

Z této posloupnosti je vidět, že pro obecné n platí

$$\frac{\epsilon_0}{\epsilon_n} = 2^n \Rightarrow n = \log_2 \frac{\epsilon_0}{\epsilon_n}. \tag{27}$$

Když budeme chtít najít kořen s přesností odpovídající $\epsilon_n = 10^{-12}$ v okolí $x = 1$ bude potřeba $n \simeq 40$ kroků. Následující pseudokód ilustruje implementaci metody bisekce při hledání kořene 1D rovnice $f(x) = 0$.

```
(*Bisekce*)
x1=a
x2=b
m=(x1+x2)/2
while abs(x1-x2)>eps and abs(f(m))>eps do
        if f(m)*f(x2)<0 then
                x1=m
        else
                x2=m
        end
        m=(x1+x2)/2
end
```

Na konci algoritmu bisekce bude přibližný kořen rovnice (určený s přesností *eps*) uložen v proměnné m.

Metoda sečen

Tato metoda je založena na lineární aproximací fuknce $f(x)$ na daném intervalu. Označme interval monotónosti $I \equiv [a, b]$. Přímka aproximující $f(x)$ na daném intervalu bude předepsána formulí

$$y(x) = \frac{f_b - f_a}{b - a}(x - a) + f_a \tag{28}$$

kde je $f_a \equiv f(a)$ a $f_b \equiv f(b)$. Odhad kořene rovnice $f(x) = 0$ potom plyne jako řešení rce $y(x) = 0$, což nám dá výsledek

$$x_c = b - f_a \frac{b - a}{f_b - f_a}. \tag{29}$$

Pokud je $f(x_c) < \epsilon$ tak jsme našli kořen rovnice $f(x) = 0$. Pokud zmíněná podmínka neplatí, pak je interval $[a, b]$ rozdělen na dva podintervaly $[a, x_c]$ a $[x_c, b]$ a skutečný kořen leží v jednom z nich. Pokud leží v $[a, x_c]$ tak položíme $b = x_c$ v opačném případě položíme $a = x_c$. Celý postup opakujeme dokud není dosaženo požadované přesnosti . Metodu sečen shrnuje následující pseudokód:

```
1       ai=a
2       bi=b
3       fc=f((a+b)/2)
4       while abs(ai-bi)>eps  do
5               fa=f(ai)
6               fb=f(bi)
7               xc=b-fa*(b-a)/(fb-fa)
8               fc=f(xc)
9               if abs(fc)<eps then break
10              if fc*fb<0 then ai=xc
11              else bi=xc
12      end
```

Kořen hledaý na intervalu $[a, b]$ bude po skončení tohoto algoritmu uložen v proměnné x_c s přesostí *eps*.

Brentova metoda

Brentova metoda kombinuje metodu bisekce s inverzni kvadratickou aproximací funkce $f(x)$, resp. s metodou sečen. Opět řešíme problém $f(x) = 0$. Řešením zřejmě bude $x = f^{-1}(y = 0)$. Iverzní funkci $f^{-1}(y)$ aproximujeme Legendrovým polynomem a dostaneme

$$f^{-1}(y) \simeq \frac{(y-y_1)(y-y_2)}{(y_0-y_1)(y_0-y_2)}x_0 + \frac{(y-y_0)(y-y_2)}{(x_1-x_0)(x_1-x_2)}x_1$$

$$+\frac{(y-y_0)(y-y_1)}{(y_2-y_0)(y_1-y_0)}x_2. \tag{30}$$

Odhad kořene x_c potom bude

$$\begin{aligned} x_c &= f^{-1}(0) \simeq \frac{(y_1)(y_2)}{(y_0-y_1)(y_0-y_2)}x_0 \\ &+\frac{(y_0)(y_2)}{(x_1-x_0)(x_1-x_2)}x_1 + \frac{(y_0)(y_1)}{(y_2-y_0)(y_1-y_0)}x_2 \end{aligned} \tag{31}$$

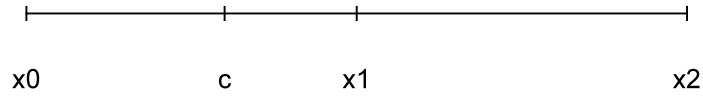
V případě, že je $y_1 == y_0$ nebo $y_2 == y_1$ pak nelze použít odhad kořene Lagrangovým polynomem druhého řádu a použijeme výraz odpovídající odhadu kořene metodou sečen.

Pro určení kvadratického polyomu jsou potřeba tři body $x_0 < x_1 < x_2$. Nechť bod x_1 je přesně v polovině intervalu $[x_0, x_2]$. Body x_1 a x_c dělí interval na tři části. Nejprve otestujeme jestli kořen rovnice neleží mezi body (x_1, x_c). Pokud ano tak nastavíme nové krajní body intervalu na $[x_c, x_1]$. Pokud ne tak otestujeme na přítomnost kořene interval $[x_c, x_2]$ a v případě kladné odpovědi nastavíme nový interval tak, že $x_0 = x_c$. Pokud ani tento interval neobsahuje kořen rovnice tak zbývá posladní možnost a to interval $[x_0, x_c]$. Nakonec, kořenem se stává bod $min(f(x_c), f(x_2))^{-1}$. Následující pseudokód ilustruje implementaci této metody hledání kořene problému $f(x) = 0$:

```
1       x0=a
2       x1=(a+b)/2
3       x2=b
4       c=a
5       while abs(x0-x2)>eps and (f(c)!=0 and f(x2)!=0) do
6           if f0!=f1 and f2!=f1 then
7               c=f1*f2*x0/((f0-f1)*(f0-f2))
```

```
8                               c=c+f0*f2*x1/((f1-f0)*(f1-f2))
9                               c=c+f0*f1*x2/((f2-f0)*(f2-f1))
10                      else
11                              c=x2-f1*(x2-x0)/(f2-f0)
12                      end
13                      fc=f(c)
14                      if x1<c then swap(x1,c)
15                      if fc*f1<0 then
16                              x0=c
17                              x2=x1
18                      else if f1*f2<0 then
19                                      x0=x1
20                      else
21                                      x2=c
22                      end
23                      end
24
25                      x1=(x0+x2)/2
26                      f0=f(x0)
27                      f1=f(x1)
28                      f2=f(x2)
29              end while
30              if f(x2)<f(c) return x2
31              return c
```

Následující obrázek ukazuje typické dělení intervalu $[x_0, x_2]$ během Brentovy metody.



4 Polynomiální rovnice

Polynomem n -tého řádu je funkce daná předpisem

$$P_n(x) \equiv a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n. \qquad (32)$$

Pokud je $n > 4$ tak neznáme explicitní vyjádření pro kořeny polynomiálních rovnic $P_n(x) = 0$. Použijeme účinnou numerickou metodu založenou na Laugerre algoritmu. Nechť jsou $x_1, x_2, \ldots, x_n$ kořeny rovnice $P_n(x) = 0$. Pak lze $P_n(x)$ pomocí těchto kořenů zapsat vztahem

$$P_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n). \qquad (33)$$

Nyní na poslední rovnici aplikujeme přirozený logaritmus a obdržíme

$$\ln(P_n(x)) = \ln|x - x_1| + ln|x - x_2| + \cdots + \ln|x - x_n|. \qquad (34)$$

Nakonec, pro poslední rovici určíme první a druhou derivaci

$$\frac{\mathrm{d} \ln P_n(x)}{\mathrm{d}x} = \frac{1}{x - x_1} + \frac{1}{x - x_2} + \cdots + \frac{1}{x - x_n} = \frac{P_n'(x)}{P_n(x)} \equiv G \qquad (35)$$

a

$$\begin{aligned} -\frac{\mathrm{d}^2 \ln P_n(x)}{\mathrm{d}x^2} &= \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2^2)} + \cdots + \frac{1}{(x - x_n)^2} \\ &= \left(\frac{P_n'(x)}{P_n(x)}\right)^2 - \frac{P_n''(x)}{P_n(x)}. \end{aligned} \qquad (36)$$

Hledaný kořen x_1 nechť je vzdálen od aktuálního odhadu o délku a a ostatní kořeny o délku b, tj.

$$x_1 - x = a \text{ a } x_i - x = b, \forall\, i = 2, 3, \ldots, n. \tag{37}$$

Po dosazení do (35) a (36) obdržíme rovnice

$$\frac{1}{a} + \frac{n-1}{b} = G, \tag{38}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H. \tag{39}$$

Po několika algebraických úpravách posledních dvou rovnic najdeme pro délku a výraz

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}. \tag{40}$$

Kořen x_1 teď snadno najdeme iterativně. Pro odhad kořene x určíme, ze vztahu (40), parametr a a získáme nový odhad $x := x + a$ (operátor := zde reprezetuje operaci přiřazení!). Takto pokračujeme dokud není $|a| < \epsilon$ nebo $|f(x)| < \epsilon$. Pro určení zbývajících kořenů nejprve zredukujeme původní polynom

$$P_n(x) = (x - x_1)Q_{n-1}(x) \Rightarrow Q_{n-1}(x) = P_n(x)/(x - x_1) \tag{41}$$

a aplikujeme celou předchozí iteraci na polynom stupně $n-1$, $Q_{n-1}(x)$, atd. dokud neurčíme všechny kořeny.

Redukce polynomu n -tého stupně na polynom stupně $n-1$ si zaslouží detailní diskuzi. Nechť je z komplexní, pak jsou-li koeficienty a_i polynomu $P_n(z)$ reálná čísla pak komplexním sdružením tohoto polynomu dostaneme $[Pn(z)]* = Pn(z*)$. Dokažme toto tvrzení.

◇ Počítejme

$$\begin{aligned}[P_n(z)]^* &= (a_n z^n)^* + (a_{n-1}z^{n-1})^* + \cdots + (a_1 z)^* + a_0 \\ &= a_n(z^n)^* + a_{n-1}(z^{n-1})^* + \cdots + a_1 z^* + a_0 \\ &= a_n(z^*)^n + a_{n-1}(z^*)^{n-1} + \cdots + a_1 z^* + a_0 = P_n(z^*) \end{aligned} \quad (42)$$

Zde jsme využili následující identity. Nechť $v = a + ib$ a $w = c + id$ jsou komplexní, pak plati:

- $$\begin{aligned}(v+w)^* &= (a+ib+c+id)^* = (a+c+i(b+d))^* = a+c-i(b+d) \\ &= v^* + w^* \end{aligned} \quad (43)$$

- $$(w^n)^* = (|w|e^{in\phi})^* = |w|^n (e^{in\phi})^* = |w|^n e^{-in\phi} = (w^*)^n \quad (44)$$

C.B.D.

Nyní, pokud je z kořenem rovnice $P_n(z) = 0$ pak je kořenem i příslušné komplexně sdružené číslo $z*$,tj. platí $P_n(z*) = 0$. To znamená, že v algoritmu pro redukci polynomu vystačíme s reálnými poli $a[]$ a $v[]$ neboť zřejmě platí

$$P_n(z) = (z - z_1)(z - z_1^*)Q_{n-2}(z) = (z^2 + 2az + a^2 + b^2)Q_{n-2}(z), \quad (45)$$

kde a a b určují kořen $z_1 = a + ib$ a zjevně jsou a a b reálná čísla. Polynom $n-2$ stupně, $Q_{n-2}(z)$, má reálné koeficienty. Dále si, na základě této diskuze, musíme uvědomit, že kořeny rovnice

$P_n(z) = 0$ jsou bud reálná čísla, nebo komplexní. A pokud je nalezený kořen komplexní tak řešením této rovnice je určitě i kořen komplexně sdružený.

Následující algoritmy ilustrují realizace Laguerrovy iterace, dělení polynomu a vyčíslení polynomu a jeho prvních dvou derivací.

```
1  (*Laguerrova iterace*)
2  laguerre(c,n,x,eps)
3          imax =100
4          z=0+I*0
5          while(i<=imax )
6                  poly(z,c,n,p)
7                  if abs(p[0])<eps break
8                  F=(n-1)
9                  F=F*((n-1)*p[1]*p[1]-n*p[0]*p[2])
10                 G1=p[1]+sqrt(F)
11                 G2=p[1]-sqrt(F)
12                 if(abs(G1)>abs(G2)) then G=G1
13                 else G=G2
14                 z=z-n*p[0]/G
15         end while
16 end
```

Redukci polynomu n- tého stupně na $n-1$ stupeň provede algoritmus:

```
1  (*Deleni polynomu*)
2  poldiv(a,n,z1,v)
3          p=a[n]
4          for i=n-1 to 0 step -1
5                  v[i]=p
6                  p=p*z1+a[i]
7          end for
8  end
```

Dále je potřeba vyčíslit polynom P_n v z a jeho první a druhou derivaci. To zajistí algoritmus:

```
1  (* Vycisleni P, P' a P''*)
2  poly(z,a,n,p)
3          //polynom
4          p[0]=a[n]
5          for i=n-1 to 0 step -1
6                  p[0]=p[0]*z
7                  p[0]=p[0]+a[i]
8          end for
9          //prvni derivace polynomu
10         p[1]=n*a[n]
11         for i=n-1 to 1 step -1
12                 p[1]=p[1]*z
13                 p[1]=p[1]+i*a[i]
14         end for
15         //druha derivace polynomu
16         p[2]=n*(n-1)*a[n]
17         for i=n-1 to 2 step -1
18                 p[2]=p[2]*z
19                 p[2]=p[2]+i*(i-1)*a[i]
20         end for
21 end
```

5 Lagrangova interpolace

Mějme N bodů (x_1, y_1), (x_2, y_2), ..., (x_N, y_N), kde je $y_i = f(x_i)$ pro $i = 1, 2, ..., N$. Těmito body vede jedinečný polynom $N-1$ stupně určený Lagrangeovou formulí

$$P_{N-1}(x) \sum_{i=1}^{N} y_i \prod_{j=1, j\neq i}^{N-1} \frac{x - x_j}{x_i - x_j}. \tag{46}$$

Například pro polynom 2. stupně dostaneme výraz

$$P_2(x) \quad = \quad \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2$$

$$+\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3. \tag{47}$$

Pro určení chyby, které se dopouštíme při Lagrangově interpolaci definujeme funkci $F(z)$

$$F(z) \equiv f(z) - y(z) - [f(x) - y(x)]\frac{p_n(z)}{p_n(x)} \tag{48}$$

kde je

$$p_n(x) = \prod_{i=1}^{n}(x - x_i) \tag{49}$$

Funkce $F(z)$ má $n+1$ uzlových bodů. Když n-krát zderivujeme funkci $F(z)$ podle z dostaneme výraz

$$F^{(n)}(z) = f^{(n)}(z) - y^{(n)}(z) - \frac{f(x) - y(x)}{p_n(x)}n!. \tag{50}$$

Tato fukce má na intervalu, které určují nulové body x_1, x_2,... a bod x aspoň jeden nulový bod (n-krát jsme derivovali funkci, která jich má $n+1$). Označme tento nulový bod $z = \xi$ a dosadíme jej do vztahu pro $F^{(n)}(z)$ a obdržíme rovnici

$$0 = F^{(n)}(\xi) = f^{(n)}(\xi) - y^{(n)}(\xi) - \frac{f(x) - y(x)}{p_n(x)}n! \tag{51}$$

Všimněme si, že $f^{(n)}(z) = 0$, protože f je polynom $n-1$ stupně. Označíme-li chybu interpolace $E(x) = f(x) - y(x)$ tak z předchozí rovnice dostaneme výsledek

$$E(x) = \frac{p_n(x)}{n!}f^{(n)}(\xi). \tag{52}$$

K vyčíslení Lagrangova polynomu se využívá Nevillův algoritmus. Pošme jej pomocí příkladu pro $N = 4$. Strom Nevillova pak vypadá následovně:

$$
\begin{array}{lllll}
x_1: & y_1 = P_1 & & & \\
 & & P_{12} & & \\
x_2: & y_2 = P_2 & & P_{123} & \\
 & & P_{23} & & P_{1234} \\
x_3: & y_3 = P_3 & & P_{234} & \\
 & & P_{34} & & \\
x_4: & y_4 = P_4 & & &
\end{array}
$$

kde jsou $P_1, \ldots, P_4$ polynomy 0-tého stupně odpovídající funkčním hodnotám $f(x_i)$. P_{12} je polynom 1. řádu procházející body (x_1, y_1), (x_2, y_2) a analogicky ostatní P_{ij}. P_{123} je polynom druhého stupně procházející body (x_1, y_1), (x_2, y_2) a (x_3, y_3). Nakonec P_{1234} je požadovaný výstup, polynom třetího stupně procházející všemi čtyřmi body. Vztah mezi rodičovskými a dceřinými polynomy vyjadřuje výraz

$$
\begin{aligned}
P_{i(i+1)\cdots(i+m)} \quad = \quad & \frac{1}{x_i - x_{i+m}} \left((x - x_{i+m}) P_{i(i+1)\cdots(i+m-1)} + \right. \\
& \left. (x_i - x) P_{(i+1)(i+2)\cdots(i+m)} \right) . \qquad (53)
\end{aligned}
$$

Následující pseudokód implementuje Nevillův algoritmus:

```
1  lagrange(xx, x[], y[], n)
2          for m =1 to n
3                  A[m ]=y[m ]
4          end for
5          for m =1 to n-1
```

```
6                   for j=1 to n-m
7                       B[j]=((xx-x[j+m])*A[j]
8                         -(xx-x[j])*A[j+1])/(x[j]-x[j+m])
9                   end for
10                  for j=1 to n-m
11                      A[j]=B[j]
12                  end for
13          end for
14          return B[1]
15  end
```

6 Spline interpolace

Máme $N+1$ bodů $(x_0, y_0), \ldots, (x_N, y_N)$, kde je $y_i = f(x_i)$. Na intervalech $[x_i, x_{i+1}]$ budeme aproximovat funkci $f(x)$ lineární funkcí

$$g_i(x) = a_i x + b_i. \tag{54}$$

Mluvíme pak o *lineárním spline*. Podmínkou pro konstrukci lineárního spline je aby byl spojitý, tj. aby platilo

$$g_i(x_{i+1}) = g_{i+1}(x_{i+1}) \tag{55}$$

kde $g_i(x)$ je lineární funkce na intervalu $[x_i, x_{i+1}]$. Lineární spline má potom tvar

$$g_i(x) = \frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)}(x - x_i) + y_i, \, \forall \, i = 0, \ldots, N-1. \tag{56}$$

Snadno ověříme, že platí výše zavedená podmínka spojitosti. Platí totiž

$$g_i(x_{i+1}) = y_{i+1} - y_i + y_i = y_{i+1} = g_{i+1}(x_{i+1}). \tag{57}$$

Lineární spline $g(x)$ zapisujeme ve tvaru

$$g(x) = \begin{cases} g_0(x) & \text{pro } x \in [x_0, x_1] \\ g_1(x) & \text{pro } x \in [x_1, x_2] \\ \vdots & \\ g_{N-1}(x) & \text{pro } x \in [x_{N-1}, x_N] \end{cases} \tag{58}$$

Chceme-li aby se lineární spline $g(x)$ co nejvíce přiblížil funkci $f(x)$, pak musíme mít k dispozici co nejvíce bodů (x_i, y_i). Lepší aproximace dosáhneme volbou kubické funkce na intervalech $[x_i, x_{i+1}]$.

V případě *kubického spline* nahradíme lineární funkce (54) kubickou funkcí

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i. \tag{59}$$

Každá funkce $S_i$je určena čtyřmi parametry (a_i, b_i, c_i, d_i). To je celkem $4N$ neznámých parametrů. Následující podmínky, jejichž splnění u kubického spline uplatňujeme nám dá potřebných $4N$ rovnic.

1. Požadujeme, aby náš spline interpoloval všech $N+1$ bodů, dostáváme tak podmínky

   $$S_i(x_i) = y_i \text{ pro } 0 \leq i \leq N-1 \text{ a } S_{N-1}(x_N) = y_N. \tag{60}$$

   Tím jsme získali $n+1$ podmínek.

2. Požadujeme spojitost slpline, jeho prví a druhé derivace. To nám dává následující podmínky

   $$S_{i-1}(x_i) \quad = \quad S_i(x_i) \text{ pro } 1 \leq i \leq N-1 \text{ dává } N-1 \text{ podmínek}$$

$$
\begin{aligned}
S'_{i-1}(x_i) &= S'_i(x_i) \text{ pro } 1 \le i \le N-1 \text{ dává } N-1 \text{ podmínek} \\
S''_{i-1}(x_i) &= S''_i(x_i) \text{ pro } 1 \le i \le N-1 \text{ dává } N-1 \text{ podmínek}
\end{aligned}
$$

Máme celkem $(N+1)+3(N-1) = 4N-2$ podmínek. Chybí nám ještě dvě. V případě tzv. *přirozeného spline* se požaduje aby byly navíc splněny podmínky

$$S''(x_0) = S''(x_N) = 0 \tag{61}$$

tj. v krajních bodech interpolované funkce požadujeme nulovost druhých derivací kubického spline.

Uplatněním výše uvedených podmínek určíme kubický spline následovně. Nechť pro všechna $0 \le i \le N$ platí

$$z_i \equiv S''_i(x_i) \tag{62}$$

a dále nechť je

$$h_i \equiv x_{i+1} - x_i. \tag{63}$$

Protože je $S_i(x)$ kubickým polynomem an intervalu $[x_i, x_{i+1}]$, tak $S''_i(x)$ je na tomto intervalu lineární, tj.

$$S''_i(x) = z_i + m_i\,(x - x_i) \tag{64}$$

kde je

$$m_i = \frac{z_{i+1} - z_i}{h_i} \tag{65}$$

Rovnici (64) snadno převedeme na tvar

$$S''_i(x) = \frac{z_{i+1}}{h_i}\,(x - x_i) + \frac{z_i}{h_i}\,(x_{i+1} - x)\,. \tag{66}$$

Ze znalosti $z_i$jsme schopni určit druhou derivaci kubického spline. Nás ale zajímá funkce $S_i(x)$ takže budeme funkci (66) dvakrát integrovat a dostaneme

$$S_i(x) = \frac{z_{i+1}}{6h_i}\left(x - x_i\right)^3 + \frac{z_i}{6h_i}\left(x_{i+1} - x\right)^3 + C_i\left(x - x_i\right) + D_i\left(x_{i+1} - x\right) \tag{67}$$

kde jsou C_ia D_i integrační konstanty, které určíme z podmínek

$$S_i(x_i) = y_i \text{ a } S_i(x_{i+1}) = y_{i+1}. \tag{68}$$

Takto obdržíme pro C_ia D_i vztahy

$$\begin{aligned} C_i &= \frac{y_{i+1}}{h_i} - \frac{h_i}{6} z_{i+1}, \\ D_i &= \frac{y_i}{h_i} - \frac{h_i}{6} z_i. \end{aligned}$$

Ke konečnému výpočtu funkce (67) musíme určit samotné z_i. K jejich určení použijeme poslední podmínku spojisti prvních derivací kubického spline, tj.

$$S'_{i-1}(x_i) = S'_i(x_i). \tag{69}$$

Derivací rovnice (67) dostaneme následující dvě rovnice

$$\begin{aligned} S'_{i-1}(x_i) &= \frac{h_{i-1}}{6} z_{i-1} + \frac{h_{i-1}}{3} z_i + b_{i-1}, \\ S'_i(x_i) &= -\frac{h_i}{6} z_{i+1} - \frac{h_i}{3} z_i + b_i, \end{aligned}$$

kde je $b_i \equiv (y_{i+1} - y_i)/h_i$. Dosazením posledních dvou rovnic do (69) dostáváme rovnici pro $1 \leq i \leq N-1$

$$h_{i-1} z_{i-1} + 2\left(h_{i-1} + h_i\right) z_i + h_i z_{i+1} = 6\left(b_i - b_{i-1}\right). \tag{70}$$

Nezapomeňme, že pro přirozený spline platí $z_0 = z_n = 0$. Dostali jsme tak tridiagonální lineární systém

$$\begin{bmatrix} u_1 & h_1 & & & & & \\ h_1 & u_2 & h_2 & & & & \\ & h_2 & u_3 & h_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & h_{N-3} & u_{N-2} & h_{N-2} \\ & & & & h_{N-2} & u_{N-1} \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{N-2} \\ z_{N-1} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_{N-2} \\ p_{N-1} \end{bmatrix}. \tag{71}$$

$$u_i = 2(h_i + h_{i-1}) \tag{72}$$

a

$$p_i = 6(b_i - b_{i-1}). \tag{73}$$

Tridiagonální systém

K vyřešení tridiagonálního lineárního systému použijeme následující algoritmus. Hledáme řešení systému, který má tvar

$$\begin{bmatrix} b_0 & c_0 & 0 & \dots & & & \\ a_1 & b_1 & c_1 & \dots & & & \\ & & & \dots & & & \\ & & & \dots & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & \dots & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \dots \\ r_{N-2} \\ r_{N-1} \end{bmatrix} \tag{74}$$

Řešení spočíva v postupné eliminaci prvního nenulového prvku v řádcích 1 až $N-1$. Tímto určíme hodnotu N-té složky vektoru

$\vec{u}$, tj. u_{N-1}. Postupnou zpětnou substitucí od řádku č.$N-2$ po řádek 0 získáme zbylé složky vektoru $\vec{u}$. V maticové notaci bude soustava vypadat následovně

$$\mathbf{M} \cdot \mathbf{u} = \mathbf{r}. \tag{75}$$

Budeme ekvavalentními operaceni upravovat matici

$$(\mathbf{M}|\mathbf{r}) \tag{76}$$

nebo ve složkovém zápise

$$\left(\begin{array}{ccccc|c} M_{00} & M_{01} & M_{02} & \dots & M_{0N-1} & M_{0N} = r_0 \\ M_{10} & M_{11} & M_{12} & \dots & M_{1N-1} & M_{1N} = r_1 \\ \vdots & & & & & \vdots \\ M_{N-10} & M_{N-11} & M_{N-12} & \dots & M_{N-1N-1} & M_{N-1N} = r_{N-1} \end{array}\right). \tag{77}$$

Protože máme tridiagonální systém, tak stačí eliminovat na i-tém řádku prvek M_{ii-1}. Provedeme to tak, že i-tý řádek vynásobíme zlomkem $-M_{i-1i-1}/M_{ii-1}$ a přičteme řádek $i-1$, tzn. pro j-tý sloupec i-tého řádku provedeme následující výpočet

$$\tilde{M}_{ij} = -\frac{M_{i-1i-1}}{M_{ii-1}} M_{ij} + M_{i-1j}. \tag{78}$$

Tento vztah je aplikován na řádky matice M_{ij} pro $i = 1, \dots, N-1$. Novou matici soustavy $\tilde{\mathbf{M}}$ budeme opět značit $\mathbf{M}$.

Řešení soustavy (77) najdeme zpětnou substitucí, tj. pro $i = N-1$ bude

$$u_{N-1} = M_{N-1N}/M_{N-1N-1}. \tag{79}$$

Zbývající složky vektoru **u** vycházejí ze vztahu

$$u_i = \frac{1}{M_{ii}} \left(M_{iN} - \sum_{j=i-1}^{N-1} u_j M_{ij} \right) \tag{80}$$

Tento postup je detailně ukázách na následujícím pseudokódu:

```
1  (*transformace tridiag systemu*)
2  for i=1 to N-1 step 1
3          MM=M[i-1][i-1]/M[i][i-1];
4          for j=0 to N step 1
5                  M[i][j]=-MM*M[i][j]+M[i-1][j]
6          end
7  end
8  (* vypocet korenu u_i *)
9  u[N-1]=M[N-1][N]/M[N-1][N-1]
10 for i=N-2 to 0 step -1
11         u[i]=M[i][N]
12         for j=i-1 to N-1 step 1
13                 u[i]=u[i]-u[j]*M[i][j]
14         end
15 end
```

Spline je definován na intervalech $[x_i, x_{i+1}]$ a je tedy potřeba, k určení jeho hodnoty v bodě x, znát do kterého intervalu x náleží. Pokud jsou jsou body x_i ekvidistantní se šířkou intervalu h, pak index i, který identifikuje levý, krajní bod intervalu do kterého x náleží, určíme takto

$$x = x_0 + i * h \Rightarrow i = \mathrm{Chop}\left(\frac{x - x_0}{h} \right). \tag{81}$$

Funkce Chop odtrhává od reálného čísla jeho desetinnou část, takže nám zůstane celé číslo.

Shrnutí

Interval $[x_0, x_N]$ rozdělíme na $N+1$ částí. Na každé z nich určíme funkci

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \tag{82}$$

a požadujeme aby tyto funkce na jednotlivých intervalech splňovali podmínky

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \tag{83}$$

a

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}). \tag{84}$$

Postupnými algebraickými úpravami dostaneme

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x-x_i)^3 + \frac{z_i}{6h_i}(x_{i+1}-x)^3 + C_i(x-x_i) + D_i(x_{i+1}-x) \tag{85}$$

kde je

$$C_i = \frac{y_{i+1}}{h_i} - \frac{h_i}{6} z_{i+1}, \tag{86}$$

$$D_i = \frac{y_i}{h_i} - \frac{h_i}{6} z_i, \tag{87}$$

$$b_i = \frac{y_{i+1} - y_i}{h_i}, \tag{88}$$

$$y_i = S_i(x_i), \tag{89}$$

$$z_i = S_i''(x_i), \tag{90}$$

$$h_i = x_{i+1} - x_i. \tag{91}$$

K určení splinu je tedy nutné určit pouze z_i. Hodnoty z_i jsou řešením tridiagonálního systému (71).

7 Numerická integrace

Neprve se seznámíme s integrací s fixním krokem kdy budeme aproximovat určitý integrál

$$I = \int_a^b f(x)\mathrm{d}x \tag{92}$$

pomocí kvadratury, tj. sumou

$$I = h\sum_{i-0}^{N} \alpha_i f_i \tag{93}$$

kde je N dělení intervalu $[a, b]$, $h \equiv (b-a)/N$, $f_i \equiv f(x_i)$ a $x_i = a + i \cdot h$ pro $i = 0, \cdots, N$. Koeficienty α_i jsou váhy součtu.

Aproximace pomocí kvadratur je založena na myšlence, že na funkci $f(x)$ na intervalu $[a, b]$ nahradíme Legengrovým interpolačním polynomem, který jsme studovali výše. Napišme jej ve tvaru

$$P_N(x) \equiv \sum_{i=0}^{N} f_i L_i(x) \tag{94}$$

kde zřejmě je

$$L_i(x) = \prod_{k=0,k\neq i}^{N} \frac{x - x_k}{x_i - x_k}. \tag{95}$$

Integrál I pak aproximujeme takto

$$\int_a^b f(x)\mathrm{d}x \simeq \int_a^b P_N(x)\mathrm{d}x = \sum_{i=0}^{N} f_i \int_a^b L_i(x)\mathrm{d}x. \tag{96}$$

Když dále zavedeme substituci

$$x = a + t \cdot h \tag{97}$$

bude mít funkce $L_i(x)$ tvar

$$L_i(x) = \phi_i(t) = \prod_{k=0,k\neq i}^{N} \frac{a + th - a - kh}{a + ih - a - kh} = \prod_{k=0,k\neq i}^{N} \frac{t-k}{i-k} \tag{98}$$

a diferenciál $dx = h\mathrm{d}t$. Výsledný vzorec pro aproximaci integrálu I kvadraturou bude mít tvar

$$I \simeq h \sum_{i=0}^{N} f_i \int_0^N \phi_i(t)\mathrm{d}t = h \sum_{i=0}^{N} f_i \alpha_i. \tag{99}$$

Koeficienty α_i jsou potom určeny vztahem

$$\alpha_i \equiv \int_0^N \phi_i(t)\mathrm{d}t. \tag{100}$$

Pro názornost odvodíme lichoběžníkové a Simpsnovo pravidlo.

Lichoběžníkové pravidlo

K aproximaci použijeme pouze dva krajní body intervalu $[a, b]$, tzn. $N = 1$. Určíme koeficienty $alpha_0$ a α_1 :

$$\alpha_0 = \int_0^1 \frac{t-1}{-1}\mathrm{d}t = -\left(\frac{t^2}{2} - t\right)_0^1 = -(1/2 - 1) = 1/2, \tag{101}$$

a

$$\alpha_1 = \int_0^1 \frac{t-0}{1-0}\mathrm{d}t \left(\frac{t^2}{2}\right)_0^1 = 1/2. \tag{102}$$

Výsledkem bude integrační pravidlo

$$I \simeq \frac{h}{2}(f_a + f_b). \tag{103}$$

Simpsonovo pravidlo

V tomto případě aproximace se použijí tři body intervalu $[a, b]$, oba krajní a jeden prostřední, tzn. $N = 2$. Opět určíme koeficienty α_i , tentokráte pro $i = 0, 1$ a 2. Dostaneme

$$\alpha_0 = \int_0^2 \frac{(t-1)(t-2)}{(0-1)(0-2)}\mathrm{d}t = \frac{1}{2}\int_0^2 (t^2 - 3t + 2)\mathrm{d}t = \frac{1}{2}\left(\frac{t^3}{3} - 3\frac{t^2}{3} + 2t\right)_0^2 (104) \frac{1}{3}$$

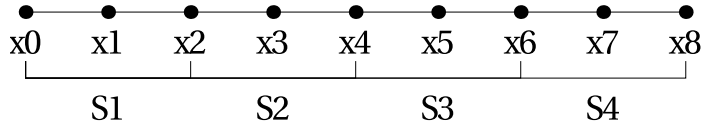
$$\alpha_1 = \int_0^2 \frac{(t-0)(t-2)}{(1-0)(1-2)}\mathrm{d}t = -\int_0^2 (t^2 - 2t)\mathrm{d}t = -\left(\frac{t^3}{3} - t^2\right)_0^2 = \frac{4}{3} \tag{105}$$

$$\alpha_2 = \int_0^2 \frac{(t-0)(t-1)}{(2-0)(2-1)}\mathrm{d}t = \frac{1}{2}\int_0^2 (t^2 - t)\mathrm{d}t = \frac{1}{2}\left(\frac{t^3}{3} - \frac{t^2}{2}\right)_0^2 = \frac{1}{3}. \tag{106}$$

Výsledné integrační pravidlo pak píšeme ve tvaru

$$I \simeq \frac{h}{3}(f_0 + 4f_1 + f_2). \tag{107}$$

Vytvořme pseudokód pro implementaci Simpsonova pravidla. Mějme reálnou funkci $f(x)$, na intervalu $[a, b]$, který rozdělíme na M podintervalů, přičemž M je sudé číslo. Pro ilustraci se podívejte na dělení intervalu $[a, b]$ pro $M = 8$ na obrázku.



Hodnoty S_i představují dílčí aproximace integrálu nad příslušným podintervalem. V následujícím pseudókdu je ukázána impementace Simpsonova pravidla.

```
1  simpson (f,a,b,M)
2          h=(b-a)/M
3          S=0
4          for i=0 to M/2
5                  x0=a+2*i*h
6                  x1=a+(2*i+1)*h
7                  x2=a+(2*i+2)*h
8                  f0=f(x0)
9                  f1=f(x1)
10                 f2=f(x2)
11                 S = S + h*(f0+4*f1+f2)/3
12         end for
13         return S
14 end
```

Gauss-Legendrova kvadratura

V předchozích podkapitolách byl určitý integrál funkce aproximován součtem funkčních hodnot na množině ekvidistantních bodů x_i, vynásobené vhodně zvoleným váhovým parametrem.

Gaussova kvadratura, nám umožňje vybrat jak váhové parametry tak souřadnice (abscissa). Vhodnou volbou vah a souřadnic lze zajistit, to aby pro integrandy typu $polynom \times W(x)$

($W(x)$ je známá fukce) byl určitý integrál exaktní. Funkci $W(x)$ můžeme zvolit tak, abychom odstraili integrovatelné singularity z požadovaného ntegrálu.

Pro danou funkci $W(x)$ a dané N najdeme množinu vah w_j a souřadnic x_j tak, že aproximace

$$\int_a^b W(x)f(x)\mathrm{d}x \simeq \sum_{j=1}^{N} w_j f(x_j) \tag{108}$$

je exaktní pokud je $f(x)$ polynom.

Určení vah a souřadnic je postaveno na existenci *ortogonálních polynomů*. Polynomy $f(x)$ a $g(x)$ jsou ortogonální, pokud je jejich skalární součin

$$< f|g > \equiv \int_a^b W(x)f(x)g(x)\mathrm{d}x \tag{109}$$

roven nule. Funkce $f(x)$ je normalizovaná pokud platí

$$< f|f >= 1. \tag{110}$$

Množina polynomů splňujících obě podmínky je ortonormální možina polynomů. Lze najít množiny polynomů které splňují:

- zahrnují přesně jeden polynom j-tého řádu $P_j(x)\,\forall\, j = 0, 1, 2, \ldots$.
- všechny jsou vzájemně ortogonální přes danou váhovou funkci $W(x)$.

Procedura pro nalezení takovéto množiny polynomů je určena rekurentním vztahem

$$P_{-1}(x) \equiv 0, \tag{111}$$
$$P_0(x) \equiv 1, \tag{112}$$
$$P_{j+1}(x) = (x - a_j)xP_j(x) - b_j P_{j-1}(x) \tag{113}$$

kde je

$$a_j \equiv \frac{< xP_j|P_j >}{< P_j|P_j >} \quad \text{pro } j = 0, 1, \ldots \tag{114}$$

$$b_j \equiv \frac{< P_j|P_j >}{< P_{j-1}|P_{j-1} >} \quad \text{pro } j = 1, 2, \ldots \tag{115}$$

Polynomy $P_j(x)$ mají přesně j kořenů na intervalu (a, b). Souřadnice(abscisy) N-bodové Gaussovy kvadratury s váhovou funkcí $W(x)$ na intervalu (a, b) jsou kořeny ortogoválního polynomu $P_N(x)$ pro stejný interval a stejnou váhovou funkci.

Soustřeďme pozrnost na Gauss-Legendrovu kvadraturu, tj. na případ kdy je váhová funkce $W(x) = 1$. Ortonormální polynomy na intervalu $-1 \leq x \leq 1$ jsou dány rekurencí

$$(j + 1)P_{j+1} = (2j + 1)xP_j - jP_{j-1}. \tag{116}$$

Po nalezení souřadnic x_i zbývá určit váhové koeficienty w_j, které vypočítáme podle vztahu

$$w_j = \frac{< P_{N-1}|P_{N-1} >}{P_{N-1}(x_j)P'_N(x_j)}. \tag{117}$$

8 Numerické řešení obyčejných diferenciálních rovnic

8.1 Eulerova metoda

Přímý útok na řešení problému

$$y'(t) = f(t, y) \tag{118}$$

je Eulerova metoda, která spočívá na hledání řešení ve tvaru Taylorova rozvoje do prvního řádu, tj.

$$y(t + h) = y(t) + y'(t)h + \mathcal{O}(h^2) \tag{119}$$

nebo ve tvaru

$$y(t + h) = y(t) + f(t, y)h. \tag{120}$$

Pro fixní krok h převedeme vztah pro numerické výpočty na tvar

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{121}$$

8.2 Runge-Kutta (klasická RK4)

Řešíme problém s počátečními podmínkami ve tvaru

$$\dot{y} = f(t, y) \text{ a } y(t_0) = y_0. \tag{122}$$

Pro fixní krok$h > 0$ definujeme

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{123}$$

kde je

$$k_1 = f(t_n, y_n), \tag{124}$$
$$k_2 = f(t_n + h/2, y_n + hk_1/2), \tag{125}$$
$$k_3 = f(t_n + h/2, y_n + hk_2/2), \tag{126}$$
$$k_4 = f(t_n + h, y_n + hk_3). \tag{127}$$

8.3 Runge-Kutta druhého řádu

Pro jednoduchost bude odvozeny vzorce Rungeho-Kuttovy metody druhého řádu. Případě RK metod vyšších řádů je postup analogický ale mnohem pracnější. Uvažujme 1-D problém

$$y'(t) = f(t, y). \tag{128}$$

Řešení $y(t)$ hledáme ve tvaru Taylorova rozvoje , tj.

$$y(t + h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \mathcal{O}(h^3) \tag{129}$$

kde je $y'(t) = f(t, y)$ a y'' obdržíme diferenciací y', tj.

$$y''(t) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}y' = f_t(t, y) + f_y(t, y)f(t, y). \tag{130}$$

Dostáváme tak řešení ve tvaru

$$y(t + h) = y(t) + hf(t, y) + \frac{h^2}{2}\left[f_t(t, y) + f_y(t, y)f(t, y)\right], \tag{131}$$

kterou po úpravě převedeme na výraz

$$y(t+h) = y(t)+\frac{h}{2}f(t,y)+\frac{h}{2}\left[f(t,y)+hf_t(t,y)+hf_y(t,y)f(t,y)\right]. \tag{132}$$

Taylorův rozvoj funkce dvou proměnných do prvního řáde je

$$f(t+h,y+k) = f(t,y)+hf_t(t,y)+kf_y(t,y)+\cdots. \tag{133}$$

Srovnáním s výrazem v závorce ve vztahu (132) dostáváme

$$f(t+h,y+hf(t,y)) = f(t,y)+\frac{h}{2}f_t(t,y)+\frac{h}{2}f(t+h,y+hf(t,y)) \tag{134}$$

a tak bude mít vztah pro RK2 krok tvar

$$y(t+h) = y(t)+\frac{h}{2}f(t,y)+\frac{h}{2}f(t+h,y+hf(t,y)). \tag{135}$$

Zapsáno pro účely numerických výpočtů máme výsledek

$$\begin{aligned} y_{n+1} &= y_n + h\left(\frac{1}{2}k_1+\frac{1}{2}k_2\right), \\ k_1 &= f(t_n,y_n), \\ k_2 &= f(t_n+h,y_n+hk_1). \end{aligned}$$

8.4 Chyby (Demonstrováno na Eulerově metotě integrace)

- *Local Truncation Error (LTE)* - rozdíl mezi numerickým řešením po ukončení jednoho kroku y_1a exaktním řešením

v čase $t_1 = t_0 + h$, tj.

$$\begin{aligned} y_1 &= y_0 + hf(y_0, t_0) \text{ (numerické řešení)}, \\ y(t_0 + h) &= y(t_0) + hy'(t_0) + \frac{1}{2}h^2 y''(t_0) + \mathcal{O}(h^3) \\ &\quad \text{(exaktní řešení rozvedené do Taylorova rozvoje)}, \\ LTE &= y(t_0 + h) - y_1 = \frac{1}{2}h^2 y''(t_0) + \mathcal{O}(h^3). \end{aligned}$$

- *Global Truncation Error (GTE)* - je chyba v určitém časovém okomžiku t dosženého po provedené n kroků potřebných k tomu aby metoda dospěla od počátečního okamžiku t_0k okamžiku t. Pro fixní krok h počet kroků bude

$$n = \frac{t - t_0}{h}. \tag{136}$$

  Tento druh chyby je kumulativní efekt LTE který je v případě Eulerovy integrace $\sim h^2$. To znamená, že po n krocích bude $GTE \sim nh^2 \sim h$.

8.5 Adaptivní krok a RK4

Dosud byl integrační krop pevně daný. Pro získání výsledku s předepsanou přesností *eps* bylo nutné vhodně nastavit integrační krok h tak aby maximální chyba výsledku byla právě *eps*. Nevýhoda spočívá v tom, že při fixním kroku jsou určité části řešení určovány se zbytečně malým krokem (zbytečně velkou přesností) za cenu velkého počtu integračních kroků.

Lepší je v každém integračním kroku určit chybu výsledku a pokud bude větší než požadovaná přesnost tak zmenšit integrační krok a v opačném případě integrační krok zmenšit.

Přímočarý algoritmus spočívá v "doubling"integračního kroku. Nechť je aproximace řešení ODE v bodě $t + 2h$ reprezentováno veličinou y_1. Určeme aproximaci řešení ODE v tomtéž bodě ale použitím dvou po sobě následujících RK4 kroků s krokem h. Tuto aproximaci označme y_2. Chybu dvojnásobného kroku určuje rozdíl

$$\Delta = |y_2 - y_1|. \tag{137}$$

Bude-li $\Delta > EPS$ pak integrační krok zmenšíme na polovinu, tj. $h \to h/2$, a zopakujeme výpočet y_1 a y_2. Pokud je $\Delta < EPS$ tak další krok zdojnásobíme, tj. $h \to 2h$, a určíme y_1 a y_2 v následujícím bodě. Následující pseudokód inlustruje myšlenku adaptivního kroku.

```
1  h=1e-6
2  t=0
3  y=y0
4  while t<tmax do
5
6            t=t+h
7
8            while true do
9
10
11                   rk4(t,2h,y1)
12
13                   rk4(t,h,y2)
14                   rk4(t+h,h,y2)
15
16                   err=abs(y1-y2)
17
18                   if err>eps then
19                                        h=h/2
```

```
20                                          y1=y
21                                          y2=y
22                  else
23                          h=2*h
24                          y=y1
25                          break
26                  end
27          end while
28
29          print t, y
30
31  end while
```

8.6 Adaptivní krok a embedované RK metody

Cílem embedovaných RK metod je co nejefektivnější vypočítat aproximaci řešení v p-tém a $p+1$ - řádu. Nechť je $\tilde{x}_{n+1}$ přesné řešení diferenciální rovnice

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(t,x) \tag{138}$$

v $n+1$ integračním kroku. Vyjádřeme jej pomocí aproximativních řešení x_1 a x_2 (první aproximace je určena krokem h a druhá krokem $h/2$)

$$\tilde{x}_{n+1} = x_1 + Ch^{p+1} + o(h^{p+2}), \tag{139}$$

$$\tilde{x}_{n+1} = x_2 + 2C\left(\frac{h}{2}h\right)^{p+1} + o(h^{p+2}). \tag{140}$$

Odtud vidíme, že pro rozdíl $x_1 - x_2$ platí

$$|x_1 - x_2| = Ch^{p+1}(1 - \frac{1}{2^p}) \Rightarrow C = \frac{|x_1 - x_2|}{h^{p+1}(1 - 1/2^p)}. \tag{141}$$

Označme

$$\epsilon = \frac{|x_1 - x_2|}{1 - 2^{-p}} \tag{142}$$

a dostáváme výraz pro C ve tvaru

$$C = \frac{\epsilon}{h^{p+1}}. \tag{143}$$

Odhadněme nyní největší optimální krok pro RK metodu při požadované toleranci ϵ_0. Jestliže bude krok h_{new} pak vztah mezi aproximací a skutečným řešením je

$$\tilde{x}_{n+1} = x_{n+1} + Ch_{new}^{p+1}. \tag{144}$$

Hledáme aproximaci s tolerancí ϵ_0, tj.

$$\tilde{x}_{n+1} = \tilde{x}_{n+1} + \epsilon_0. \tag{145}$$

To ale znamená, že musí platit

$$Ch_{new} \leq \epsilon_0 \Rightarrow h_{new} = \left(\frac{\epsilon_0}{\epsilon}\right)^{\frac{1}{1+p}} h. \tag{146}$$

V praktických aplikacích se ukazuje, že je užitečné použít tzn. safety faktor $\beta \simeq 1$ který modifikuje odhad optimálního kroku takto

$$h_{new} = \beta \left(\frac{\epsilon_0}{\epsilon}\right)^{\frac{1}{1+p}} h. \tag{147}$$

Volba ϵ_0 závisí na daném pro problému, který řešíme. Často se volí ϵ_0 úměrné h. Tady se musíme mít na pozoru, protože ve fázi kdy snižujeme krok h z velkých hodnot pak nový krok h_{new}

určený exponentem $1/(1+p)$ nevede k požadované přesnosti. Pro $\epsilon > \epsilon_0$ je lepší použít exponent $1/p$. Určení nového optimálního kroku h_{new} tak shrnuje vztah

$$h_{new} = \begin{cases} \beta h \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(1+p)} & \text{pro } \epsilon < \epsilon_0 \\ \beta h \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(p)} & \text{pro } \epsilon \geq \epsilon_0 \end{cases} \tag{148}$$

Pro embedovanou RK metodu čtvrtého řádu tak dostaneme ($p = 4$) výraz pro nový krok ve tvaru

$$h_{new} = \begin{cases} \beta h \left(\frac{\epsilon_0}{\epsilon}\right)^{0.2} & \text{pro } \epsilon < \epsilon_0 \\ \beta h \left(\frac{\epsilon_0}{\epsilon}\right)^{0.25} & \text{pro } \epsilon \geq \epsilon_0 \end{cases} \tag{149}$$

Algoritmus pro RK s adaptivním krokem pak vypadá takto

```
1   VSTUP: t0,x0,eps0,h,j=0, beta=0.8
2
3   KROK1: URCI x(t+h,h), x(h+h,h/2) a eps
4
5   KROK2: POKUD JE eps<eps0 TAK
6
7                                   h=h*beta*pow(eps0/eps,0.2)
8
9                                   t=t+h
10                  JINAK
11
12                                  h=h*beta*pow(eps0/eps,0.25)
13
14                  KONEC
15
16              POKRACUJ KROK1 DOKUD t<tmax
17  KROK3: TISK TAKULKU (tn,xn)
```

Šesti úrovňová RK metoda 4. a 5. řádu je

$$
\begin{aligned}
k_1 &= f(t, x), && (150)\\
k_2 &= f(t + a_2 h, x + h b_{21} k_1), && (151)\\
&\vdots && (152)\\
k_6 &= f(t + a_6 h, x + h b_{61} k_1 + h b_{62} k_2 + \cdots + h b_{65} k_5), && (153)\\
x_{n+1} &= x_n + h \sum_{i=1}^{6} (c_i k_i) + o(h^5), && (154)\\
x^*_{n+1} &= x_n + h \sum_{i=1}^{6} (c *_i k_i) + o(h^5), && (155)\\
& && (156)
\end{aligned}
$$

Výpočet chyby ϵ pak jednoduše bude

$$\epsilon = |x_{n+1} - x^*_{n+1}| = \sum_{i=1}^{6} (C_i - C^*_i) k_i. \tag{157}$$

8.7 Poznámky k odvození embedded RK schémat

K vysvětlení konstrukce embedded RK schémat se soustředíme na nejjednodušší variantu RK1(2). Tzn. hlavní integrátor je Eulerova metoda 1. řádu a odhad chyby plyne z integračního kroku 2. řádu. Taylorův rozvoj řešení diferenciální rovnice

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y) \tag{158}$$

do 1. a do 2. řádu v h je

$$\begin{aligned} y_{n+1}^{(1)} &= y_n + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)_n h + o(h^2) \\ &= y_n + f_n h + o(h^2) \end{aligned} \tag{159}$$

a

$$\begin{aligned} y_{n+1}^{(2)} &= y_n + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)_n h + \frac{1}{2}\left(\frac{\mathrm{d}^2y}{\mathrm{d}t^2}\right)_n h^2 + o(h^3) \\ &= y_n + f_n h + \frac{1}{2}\left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f\right)_n h^2 + o(h^3). \end{aligned} \tag{160}$$

Embeded RK schéma je následující:

$$k_1 = f(t_n, y_n), \tag{161}$$

$$k_2 = f(t_n + a_2 h, y_n + b_{21} h k_1), \tag{162}$$

$$y_{n+1}^{(1)} = y_n + (c_1 k_1 + c_2 k_2) h + o(h^2), \tag{163}$$

$$y_{n+1}^{(2)} = y_n + (c_1^* k_1 + c_2^* k_2) h + o(h^3). \tag{164}$$

Nyní rozvineme do Taylorova rozvoje k_2

$$k_2 = f(t_n, y_n) + \left(\frac{\partial f}{\partial t}\right)_n a_2 h + \left(\frac{\partial f}{\partial y} f\right)_n b_{21} h \tag{165}$$

a dosadíme do vztahů pro $y^{(1)}$ a $y^{(2)}$ a obdržíme

$$y_{n+1}^{(1)} = y_n + (c_1 + c_2) f_n h + c_2 a_2 \frac{\partial f}{\partial t}_n h^2 + c_2 b_{21} \left(\frac{\partial f}{\partial y} f\right)_n h^2, \tag{166}$$

$$y_{n+1}^{(2)} = y_n + (c_1^* + c_2^*) f_n h + c_2^* a_2 \frac{\partial f}{\partial t}_n h^2 + c_2^* b_{21} \left(\frac{\partial f}{\partial y} f\right)_n h^2 \tag{167}$$

Srovnáním rovnic (159) a (160) s rovnicemi (166) a (167) obdržíme rovnice pro koeficienty c_1, c_2, c_1^*, c_2^*, a_2 a b_21 ve taru

$$\begin{aligned} c_1 + c_2 &= 1, && (168)\\ c_1^* + c_2^* &= 1, && (169)\\ c_2^* a_2 &= 1/2, && (170)\\ c_2^* b_{21} &= 1/2. && (171) \end{aligned}$$

Odkud plyne,že $b_{21} = a_2$. Když zvolíme $a_2 = 1$ a $c_2 = 0$ tak zbylé koeficienty budou

$$b_{21} = 1,\ c_1 = 1,\ c_2^* = 1/2,\ c_1^* = 1/2. \qquad (172)$$

Výsledné embedded schéma RK1(2) bude

$$\begin{aligned} k_1 &= f(t_n, y_n), && (173)\\ k_2 &= f(t_n + h, y_n + hk_1), && (174)\\ y_{n+1}^{(1)} &= y_n + hk_1, && (175)\\ y_{n+1}^{(2)} &= y_n + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right). && (176) \end{aligned}$$