Numerické metody a programování

Lekce 3

Úvod do numerických metod: přesnost, zaokrouhlovací chyby, stabilita

základní fakta

- čísla nelze ukládat s libovolnou přesností – v paměti zabírají daný počet bitů (dvojková číslice) nebo bytů (skupin osmi bitů)
- programátor může obvykle volit z několika možných datových typů lišících se
  - počtem použitých bitů
  - přesnou (fixed-point) / přibližnou (floating-point) reprezentací
- reprezentace celých čísel je přesná, výsledek celočíselné aritmetiky je přesný pokud
  - výsledek zůstává v mezích reprezentace
  - dělení je interpretováno jako celočíselné dělení, např. 7/5=1 apod.

"floating-point" reprezentace (IEEE standard 754)

číslo je reprezentováno

- znaménkem S
- mantisou M
- exponentem E

$S \times M \times b^{E-e}$,

- e je posuv exponentu
- normalizovaná hodnota: $M = 1.F$

64 bitová přesnost (double): mantisa 52 bitů, exponent 11 bitů, znaménko 1 bit

S	E	F	hodnota
...	1-2046	...	$(-1)^S \times 2^{E-1023} \times 1.F$
...	0	$F \neq 0$	$(-1)^S \times 2^{-1022} \times 0.F$
0	0	0	$+0$
1	0	0	-0
0	2047	0	$+\infty$
1	2047	0	$-\infty$
...	2047	$F \neq 0$	NaN ("Not a Number")

např.

$$0\ 10000000001\ 1010(+48\ \text{nul}) = +1 \times 2^{1025-1023} \times 1.1010_2 = 6.5$$

ověření reprezentace v C

```
#include<stdio.h>
main(){
   union doub{
      double d;
      char c[8];
      } u;
  int i;
  u.d=6.5;
  for(i=7; i>=0; i--) printf("%x",u.c[i]);
}
```

totéž v Matlabu/Octave

```
order=8:-1:1   # pro obraceni poradi
d=6.5
c=typecast(d,"uint8")(order)
printf("%x ",c)
```

výsledek v obou případech: **`401a000000000000`**

poznámky:

- podle způsobu ukládání do paměti buď *little–endian* nebo *big–endian*
- nenormalizované hodnoty se použijí v případě podtečení
- největší 64-bitová hodnota (`realmax`): $2 \times 2^{1023} \approx 1.798 \times 10^{308}$
- minimální normalizovaná 64-bitová hodnota (`realmin`): $2^{-1022} \approx 2.225 \times 10^{-308}$
- rozsah 32-bit. celoločíselné reprezentace je: $[-2147483648, 2147483647]$

zaokrouhlovací chyba (roundoff error)

- aritmetika ve "floating-point" reprezenaci není přesná, viz. např. sčítání velmi rozdílných čísel
- přesnost (machine accuracy) ϵ_m
  - nejmenší hodnota, taková že $1 + \epsilon_m \neq 1$ (relativní vzdálenost mezi sousedními čísly)
  - závisí na počtu bitů mantisy (hardware)
  - 64-bitová reprezentace: $\epsilon_m \approx 2.220 \times 10^{-16}$
  - každá operace způsobuje chybu alespoň ϵ_m tzv. *zaokrouhlovací chyba*
  - *N* operací ideálně vyrobí chybu řádu $\sqrt{N}\,\epsilon_m$ ("random walk")
  - mnohdy mnohem větší chyba: skládání chyb stejného znaménka, velká chyba některých operací – např. odečítání podobných čísel v řešení kvadratické rovnice

ořezávací chyba (truncation error)

- praktický výpočet – často diskrétní aproximace, výsledek závislý na počtu členů (ořezání)
- *truncation error:* chyba způsobená použitým praktickým algoritmem
- výskyt i v případě perfektního počítače
- závisí na použitém algoritmu
- oba typy chyb jsou většinou nezávislé a sčítají se

stabilita

nestabilní metoda

- algoritmus má sklon rychle zvětšovat vzniklé zaokrouhlovací chyby
- výsledek může být zcela znehodnocen

příklad:

mocniny zlatého řezu $\phi = \frac{\sqrt{5}-1}{2} \approx 0.61803398$

lze vypočítat rekurentním vztahem: $\phi^{n+1} = \phi^{n-1} - \phi^n$, kde známe $\phi^0 = 1$ a $\phi^1 = \phi$

nestabilita:

- existuje druhé řešení: $\phi' = -(\sqrt{5}+1)/2 = -1.618...$
- linearita, $|\phi'| > 1$
- zaokrouhlovací chyba vnáší ϕ' do výpočtu, chyba roste exponenciálně
- již $\phi^{\approx 38}$ je vypočteno zcela chybně (64-bit)

```
function stabilita
max=40
x=(sqrt(5)-1)/2.
x1=1
x2=x
for i=2:40
  pom=x2;
  x2=x1-x2;
  x1=pom;
  printf("%d\n%20.15f\n%20.15f\n\n",i,x2,x^i);
endfor
endfunction
```

```
37
   0.000000019824450
   0.000000018512169
38
   0.000000009317835
   0.000000011441150
```

příklad:

derivace $\frac{d^k}{dx^k}\sin(x)$ vypočtená pomocí diference k-tého řádu

```
function derivace
x0=pi/4;
zmena=1e-4;
for k=1:5
k
x=linspace(x0-zmena/2,x0+zmena/2,k+1);
y=sin(x);
dx=zmena/k;
dernum=diff(y,k)/dx^(k)
endfor
endfunction
```

```
k =  1
dernum =  0.70711
k =  2
dernum = -0.70711
k =  3
dernum = -0.71043
k =  4
dernum = -1421.1
k =  5
dernum =   -1.0408e+08
```